JNCIE:

Juniper Networks Certified Internet Expert

Study Guide

JNCIE:

Juniper™ Networks Certified Internet Expert

Study Guide

Harry Reynolds

San Francisco • London

SYBEX

Associate Publisher: Neil Edde
Acquisitions & Developmental Editor: Maureen Adams
Production Editor: Mae Lum
Technical Editors: Peter Moyer, Josef Buchsteiner
Copyeditor: Linda Stephenson
Compositors: Rozi Harris, Bill Clark, Interactive Composition Corporation
Graphic Illustrators: Rozi Harris, Bill Clark, Interactive Composition Corporation
CD Coordinator: Dan Mummert
CD Technician: Kevin Ly
Proofreaders: Emily Hsuan, Nancy Riddiough, Monique van den Berg
Indexer: Ted Laux
Book Designers: Bill Gibson, Judy Fung
Cover Designer: Archer Design
Cover Photographer: Bruce Heinemann, PhotoDisc

Library of Congress Card Number: 2003107710

ISBN: 0-7821-4069-6

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

To Our Valued Readers:

As internetworking technologies continue to pervade nearly every aspect of public and private industry worldwide, the demand grows for individuals who can demonstrate they possess the skills needed to manage these technologies. Recognizing this need, Juniper Networks—the leading provider of Internet infrastructure solutions that enable ISPs and other telecommunications companies to meet the demands of Internet growth—recently restructured its certification program to provide a clear path for the acquisition of these skills. Sybex is proud to have partnered with the Juniper Networks and worked closely with members of the Juniper Networks Technical Certification Program to develop this Official Study Guide for the Juniper Networks Certified Internetworking Associate certification.

Just as Juniper Networks is committed to establishing measurable standards for certifying those professionals who work in the cutting-edge field of internetworking, Sybex is committed to providing those professionals with the means of acquiring the skills and knowledge they need to meet those standards. It has long been Sybex's desire to help individuals acquire the technical knowledge and skills necessary to excel in the IT industry.

The authors and editors have worked hard to ensure that this Official Juniper Networks Study Guide is comprehensive, in-depth, and pedagogically sound. We're confident that this book will exceed the demanding standards of the certification marketplace and help you, the Juniper Networks certification candidate, succeed in your endeavors.

Good luck in pursuit of your Juniper Networks certification!

Neil Edde
Associate Publisher—Certification
Sybex Inc.

Software License Agreement: Terms and Conditions

The media and/or any online materials accompanying this book that are available now or in the future contain programs and/or text files (the "Software") to be used in connection with the book. SYBEX hereby grants to you a license to use the Software, subject to the terms that follow. Your purchase, acceptance, or use of the Software will constitute your acceptance of such terms.

The Software compilation is the property of SYBEX unless otherwise indicated and is protected by copyright to SYBEX or other copyright owner(s) as indicated in the media files (the "Owner(s)"). You are hereby granted a single-user license to use the Software for your personal, noncommercial use only. You may not reproduce, sell, distribute, publish, circulate, or commercially exploit the Software, or any portion thereof, without the written consent of SYBEX and the specific copyright owner(s) of any component software included on this media.

In the event that the Software or components include specific license requirements or end-user agreements, statements of condition, disclaimers, limitations or warranties ("End-User License"), those End-User Licenses supersede the terms and conditions herein as to that particular Software component. Your purchase, acceptance, or use of the Software will constitute your acceptance of such End-User Licenses.

By purchase, use or acceptance of the Software you further agree to comply with all export laws and regulations of the United States as such laws and regulations may exist from time to time.

Software Support

Components of the supplemental Software and any offers associated with them may be supported by the specific Owner(s) of that material, but they are not supported by SYBEX. Information regarding any available support may be obtained from the Owner(s) using the information provided in the appropriate read.me files or listed elsewhere on the media.

Should the manufacturer(s) or other Owner(s) cease to offer support or decline to honor any offer, SYBEX bears no responsibility. This notice concerning support for the Software is provided for your information only. SYBEX is not the agent or principal of the Owner(s), and SYBEX is in no way responsible for providing any support for the Software, nor is it liable or responsible for any support provided, or not provided, by the Owner(s).

Warranty

SYBEX warrants the enclosed media to be free of physical defects for a period of ninety (90) days after purchase. The Software is not available from SYBEX in any other form or media than that enclosed herein or posted to `www.sybex.com`. If you discover a defect in the media during this warranty period, you may obtain a replacement of identical format at no charge by sending the defective media, postage prepaid, with proof of purchase to:

SYBEX Inc.
Product Support Department
1151 Marina Village Parkway
Alameda, CA 94501
Web: `www.sybex.com`

After the 90-day period, you can obtain replacement media of identical format by sending us the defective disk, proof of purchase, and a check or money order for $10, payable to SYBEX.

Disclaimer

SYBEX makes no warranty or representation, either expressed or implied, with respect to the Software or its contents, quality, performance, merchantability, or fitness for a particular purpose. In no event will SYBEX, its distributors, or dealers be liable to you or any other party for direct, indirect, special, incidental, consequential, or other damages arising out of the use of or inability to use the Software or its contents even if advised of the possibility of such damage. In the event that the Software includes an online update feature, SYBEX further disclaims any obligation to provide this feature for any specific duration other than the initial posting.

The exclusion of implied warranties is not permitted by some states. Therefore, the above exclusion may not apply to you. This warranty provides you with specific legal rights; there may be other rights that you may have that vary from state to state. The pricing of the book with the Software by SYBEX reflects the allocation of risk and limitations on liability contained in this agreement of Terms and Conditions.

Shareware Distribution

This Software may contain various programs that are distributed as shareware. Copyright laws apply to both shareware and ordinary commercial software, and the copyright Owner(s) retains all rights. If you try a shareware program and continue using it, you are expected to register it. Individual programs differ on details of trial periods, registration, and payment. Please observe the requirements stated in appropriate files.

Copy Protection

The Software in whole or in part may or may not be copy-protected or encrypted. However, in all cases, reselling or redistributing these files without authorization is expressly forbidden except as specifically provided for by the Owner(s) therein.

As with my first book in this series, I once again dedicate this effort to my family, who persevered through my "virtual" absence as I endeavored to complete this project. Anita, I love you with all that I am, and I promise you that the extension cord's lease on our lovely home is rapidly coming due. I am happy to report to my daughters, Christina and Marissa, that their homework tutor will soon be returning to a full-time schedule. The general lack of combustion in my home office throughout this project has instilled in me a certain affinity for a special 20-amp circuit breaker that diligently stood guard over all that really matters in life—my family and my home. Ladies, you may start your hair dryers!

Acknowledgments

There are numerous people who deserve a round of thanks for assisting with this book. Special thanks go out to Jason Rogan and Patrick Ames for making the whole Juniper Networks book initiative possible. I would also like to thank Mae Lum and Maureen Adams at Sybex for keeping me on schedule and for getting the whole thing rolling, and also to my copyeditor, Linda Stephenson, who dealt mightily with my passive writing style and proclivity towards excessive comma usage. Once again, Peter Moyer and Josef Buchsteiner proved invaluable in the role of technical editors.

I would also like to thank Juniper Networks and my manager, Scott Edwards, for making this effort possible through arrangements that allowed me to access, borrow, or buy the equipment needed to build the test bed that formed the basis of this book.

—Harry Reynolds

Sybex would like to thank compositors Rozi Harris and Bill Clark, and indexer Ted Laux for their valuable contributions to this book.

Contents at a Glance

Contents

Table of Case Studies

Introduction

Greetings and welcome to the world of Juniper Networks. This introductory section serves as a location to pass on to you some pertinent information concerning the Juniper Networks Technical Certification Program. In addition, you'll find information about how the book itself is laid out and what it contains. Finally, we'll review some technical information that you should already know before reading this book.

Juniper Networks Technical Certification Program

The Juniper Networks Technical Certification Program (JNTCP) consists of two platform-specific, multitiered tracks. Each exam track allows participants to demonstrate their competence with Juniper Networks technology through a combination of written proficiency and hands-on configuration exams. Successful candidates demonstrate a thorough understanding of Internet technology and Juniper Networks platform configuration and troubleshooting skills.

The two JNTCP tracks focus on the M-series Routers & T-series Routing Platforms and the ERX Edge Routers, respectively. While some Juniper Networks customers and partners work with both platform families, it is most common to find individuals working with only one or the other platform. The two different certification tracks allow candidates to pursue specialized certifications, which focus on the platform type most pertinent to their job functions and experience. Candidates wishing to attain a certification on both platform families are welcome to do so, but are required to pass the exams from each track for their desired certification level.

This book covers the M-series & T-series track. For information on the ERX Edge Routers certification track, please visit the JNTCP website at `www.juniper.net/certification`.

M-series Routers & T-series Routing Platforms

The M-series Routers certification track consists of four tiers. They include the following:

Juniper Networks Certified Internet Associate (JNCIA) The Juniper Networks Certified Internet Associate, M-series, T-series Routers (JNCIA-M) certification does not have any prerequisites. It is administered at Prometric testing centers worldwide.

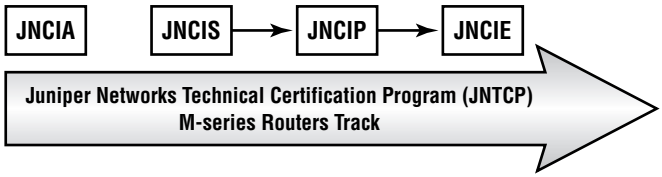
Juniper Networks Certified Internet Specialist (JNCIS) The Juniper Networks Certified Internet Specialist, M-series, T-series Routers (JNCIS-M) certification also does not have any prerequisites. Like the JNCIA-M, it is administered at Prometric testing centers worldwide.

Juniper Networks Certified Internet Professional (JNCIP) The Juniper Networks Certified Internet Professional, M-series, T-series Routers (JNCIP-M) certification requires that candidates

first obtain the JNCIS-M certification. The hands-on exam is administered at Juniper Networks offices in select locations throughout the world.

Juniper Networks Certified Internet Expert (JNCIE) The Juniper Networks Certified Internet Expert, M-series, T-series Routers (JNCIE-M) certification requires that candidates first obtain the JNCIP-M certification. The hands-on exam is administered at Juniper Networks offices in select locations throughout the world.

FIGURE 1 JNTCP M-series Routers & T-series Routing Platforms certification track



The JNTCP M-series Routers & T-series Routing Platforms certification track covers the M-series and T-series routing platforms as well as the JUNOS software configuration skills required for both platforms. The lab exams are conducted using M-series routers only.

Juniper Networks Certified Internet Associate

The JNCIA-M certification is the first of the four-tiered M-series Routers & T-series Routing Platforms track. It is the entry-level certification designed for experienced networking professionals with beginner-to-intermediate knowledge of the Juniper Networks M-series and T-series routers and the JUNOS software. The JNCIA-M (exam code JN0-201) is a computer-based, multiple-choice exam delivered at Prometric testing centers globally for U.S. $125. It is a fast-paced exam that consists of 60 questions to be completed within 60 minutes. The current passing score is set at 70 percent.

JNCIA-M exam topics are based on the content of the Introduction to Juniper Networks Routers, M-series (IJNR-M) instructor-led training course. Just as IJNR-M is the first class most students attend when beginning their study of Juniper Networks hardware and software, the JNCIA-M exam should be the first certification exam most candidates attempt. The study topics for the JNCIA-M exam include:

- System operation, configuration, and troubleshooting
- Routing protocols—BGP, OSPF, IS-IS, and RIP
- Protocol-independent routing properties
- Routing policy
- MPLS
- Multicast

70 Percent Seems Really Low!

The required score to pass an exam can be one indicator of the exam's difficulty, but not in the way that many candidates might assume. A lower pass score on an exam does *not* usually indicate an easier exam. Ironically, it often indicates the opposite—it's harder.

The JNTCP exams are extensively beta tested and reviewed. The results are then statistically analyzed based on multiple psychometric criteria. Only after this analysis is complete does the exam receive its appropriate passing score. In the case of the JNCIA-M exam, for example, requiring the passing score to be higher than 70 percent would mean that the exam's target audience would have been excluded from passing. In effect, the exam would have been more difficult to pass. Over time, as more exam statistics are collected, or the exam questions themselves are updated, the passing score may be modified to reflect the exam's new difficulty level. The end result is to ensure that the exams are passable by the members of the target audience for which they are written.

Please be aware that the JNCIA-M certification is *not* a prerequisite for further certification in the M-series Routers & T-series Routing Platforms track. The purpose of the JNCIA-M is to validate a candidate's skill set at the Associate level and it is meant to be a stand-alone certification fully recognized and worthy of pride of accomplishment. Additionally, it can be used as a stepping-stone before attempting the JNCIS-M exam.

Juniper Networks Certified Internet Specialist

The JNCIS-M was originally developed as the exam used to prequalify candidates for admittance to the practical hands-on certification exam. While it still continues to serve this purpose, this certification has quickly become a sought-after designation in its own right. Depending on the candidates' job functions, many have chosen JNCIS-M as the highest level of JNTCP certification needed to validate their skill set. Candidates also requiring validation of their hands-on configuration and troubleshooting ability on the M-series and T-series routers and the JUNOS software use the JNCIS-M as the required prerequisite to the JNCIP-M practical exam.

The JNCIS-M exam tests for a wider and deeper level of knowledge than does the JNCIA-M exam. Question content is drawn from the documentation set for the M-series routers, the T-series routers, and the JUNOS software. Additionally, on-the-job product experience and an understanding of Internet technologies and design principles are considered to be common knowledge at the Specialist level.

The JNCIS-M (exam code JN0-302) is a computer-based, multiple-choice exam delivered at Prometric testing centers globally for U.S. $125. It consists of 75 questions to be completed in 90 minutes. The current passing score is set at 70 percent.

The study topics for the JNCIS-M exam include:

- Advanced system operation, configuration, and troubleshooting
- Routing protocols—BGP, OSPF, and IS-IS
- Routing policy
- MPLS
- Multicast
- Router and network security
- Router and network management
- VPNs
- IPv6

There are no prerequisite certifications for the JNCIS-M exam. While JNCIA-M certification is a recommended stepping-stone to JNCIS-M certification, candidates are permitted to go straight to the Specialist (JNCIS-M) level.

Juniper Networks Certified Internet Professional

The JNCIP-M is the first of the two one-day practical exams in the M-series Routers & T-series Routing Platforms track of the JNTCP. The goal of this challenging exam is to validate a candidate's ability to successfully build an ISP network consisting of seven M-series routers and multiple EBGP neighbors. Over a period of eight hours, the successful candidate will perform system configuration on all seven routers, install an IGP, implement a well-designed IBGP, establish connections with all EBGP neighbors as specified, and configure the required routing policies correctly.

This certification establishes candidates' practical and theoretical knowledge of core Internet technologies and their ability to proficiently apply that knowledge in a hands-on environment. This exam is expected to meet the hands-on certification needs of the majority of Juniper Networks customers and partners. The more advanced JNCIE-M exam focuses on a set of specialized skills and addresses a much smaller group of candidates. You should carefully consider your certification goals and requirements, for you may find that the JNCIP-M exam is the highest-level certification you need.

The JNCIP-M (exam code CERT-JNCIP-M) is delivered at one of several Juniper Networks offices worldwide for U.S. $1,250. The current passing score is set at 80 percent.

The study topics for the JNCIP-M exam include:

- Advanced system operation, configuration, and troubleshooting
- Routing protocols—BGP, OSPF, IS-IS, and RIP
- Routing policy
- Routing protocol redistribution
- VLANs
- VRRP

The JNCIP-M certification is a prerequisite for attempting the JNCIE-M practical exam.

Juniper Networks Certified Internet Expert

At the pinnacle of the M-series Routers & T-series Routing Platforms track is the one-day JNCIE-M practical exam. The *E* stands for Expert and they mean it—the exam is the most challenging and respected of its type in the industry. Maintaining the standard of excellence established over two years ago, the JNCIE-M certification continues to give candidates the opportunity to distinguish themselves as the truly elite of the networking world. Only a few have dared attempt this exam, and fewer still have passed.

The new eight-hour format of the exam requires that candidates troubleshoot an existing and preconfigured ISP network consisting of 10 M-series routers. Candidates are then presented with additional configuration tasks appropriate for an expert-level engineer.

The JNCIE-M (exam code CERT-JNCIE-M) is delivered at one of several Juniper Networks offices worldwide for U.S. $1,250. The current passing score is set at 80 percent.

The study topics for the JNCIE-M exam *may* include:

- Expert-level system operation, configuration, and troubleshooting
- Routing protocols—BGP, OSPF, IS-IS, and RIP
- Routing protocol redistribution
- Advanced routing policy implementation
- Firewall filters
- Class of service
- MPLS
- VPNs
- IPv6
- IPSec
- Multicast

Since the JNCIP-M certification is a prerequisite for attempting this practical exam, all candidates who pass the JNCIE-M will have successfully completed two days of intensive practical examination.

Registration Procedures

JNTCP written exams are delivered worldwide at Prometric testing centers. To register, visit Prometric's website at `www.2test.com` (or call 1-888-249-2567 in North America) to open an account and register for an exam.

The JNTCP Prometric exam numbers are

- JNCIA-M—JN0-201
- JNCIS-M—JN0-302

JNTCP lab exams are delivered by Juniper Networks at select locations. Currently the testing locations are

- Sunnyvale, CA
- Herndon, VA
- Amsterdam, The Netherlands

Other global locations are periodically set up as testing centers based on demand. To register, send an e-mail message to Juniper Networks at `certification-testreg@juniper.net` and place one of the following exam codes in the subject field. Within the body of the message, indicate the testing center you prefer and in which month you would like to attempt the exam. You will be contacted with the available dates at your requested testing center. The JNTCP lab exam numbers are

- JNCIP-M—CERT-JNCIP-M
- JNCIE-M—CERT-JNCIE-M

Recertification Requirements

To maintain the high standards of the JNTCP certifications, and to ensure that the skills of those certified are kept current and relevant, Juniper Networks has implemented the following recertification requirements, which apply to both certification tracks of the JNTCP:

- All JNTCP certifications are valid for a period of two years.
- Certification holders who do not renew their certification within this two-year period will have their certification placed in *suspended mode*. Certifications in suspended mode are not eligible as prerequisites for further certification and cannot be applied to partner certification requirements.
- After being in suspended mode for one year, the certification is placed in *inactive mode*. At that stage, the individual is no longer certified at the JNTCP certification level that has become inactive and the individual will lose the associated certification number. For example, a JNCIP holder placed in inactive mode will be required to pass both the JNCIS and JNCIP exams in order to regain JNCIP status; such an individual will be given a new JNCIP certification number.
- Renewed certifications are valid for a period of two years from the date of passing the renewed certification exam.
- Passing an exam at a higher level renews all lower-level certifications for two years from the date of passing the higher-level exam. For example, passing the JNCIP exam will renew the JNCIS certification (and JNCIA certification if currently held) for two years from the date of passing the JNCIP exam.
- JNCIA holders must pass the current JNCIA exam in order to renew the certification for an additional two years from the most recent JNCIA pass date.

- JNCIS holders must pass the current JNCIS exam in order to renew the certification for an additional two years from the most recent JNCIS pass date.
- JNCIP and JNCIE holders must pass the current JNCIS exam in order to renew these certifications for an additional two years from the most recent JNCIS pass date.

The most recent version of the JNTCP Online Agreement must be accepted for the recertification to become effective.

JNTCP Nondisclosure Agreement

Juniper Networks considers all written and practical JNTCP exam material to be confidential intellectual property. As such, an individual is not permitted to take home, copy, or re-create the entire exam or any portions thereof. It is expected that candidates who participate in the JNTCP will not reveal the detailed content of the exams.

For written exams delivered at Prometric testing centers, candidates must accept the online agreement before proceeding with the exam. When taking practical exams, candidates are provided with a hard-copy agreement to read and sign before attempting the exam. In either case, the agreement can be downloaded from the JNTCP website for your review prior to the testing date. Juniper Networks retains all signed hard-copy nondisclosure agreements on file.

Candidates must accept the online JNTCP Online Agreement in order for their certifications to become effective and to have a certification number assigned. You can do this by going to the CertManager site at `www.certmanager.net/juniper`.

Resources for JNTCP Participants

Reading this book is a fantastic place to begin preparing for your next JNTCP exam. You should supplement the study of this volume's content with related information from various sources. The following resources are available for free and are recommended to anyone seeking to attain or maintain Juniper Networks certified status.

JNTCP Website

The JNTCP website (`www.juniper.net/certification`) is the place to go for the most up-to-date information about the program. As the program evolves, this website is periodically updated with the latest news and major announcements. Possible changes include new exams and certifications, modifications to the existing certification and recertification requirements, and information about new resources and exam objectives.

The site consists of separate sections for each of the certification tracks. The information you'll find there includes the exam number, passing scores, exam time limits, and exam topics. A special section dedicated to resources is also provided to supply you with detailed exam

topic outlines, sample written exams, and study guides. The additional resources listed next are also linked from the JNTCP website.

CertManager

The CertManager system (`www.certmanager.net/juniper`) provides you with a place to track your certification progress. The site requires a username and password for access, and you typically use the information contained on your hard-copy score report from Prometric the first time you log in. Alternatively, a valid login can be obtained by sending an e-mail message to `certification@juniper.net` with the word **certmanager** in the subject field.

Once you log in, you can view a report of all your attempted exams. This report includes the exam dates, your scores, and a progress report indicating the additional steps required to attain a given certification or recertification. This website is where you accept the online JNTCP agreement, which is a required step toward becoming certified at any level in the program. You can also use the website to request the JNTCP official certification logos to use on your business cards, resumes, and websites.

Perhaps most important, the CertManager website is where all your contact information is kept up-to-date. Juniper Networks uses this information to send you certification benefits, such as your certificate of completion, and to inform you of important developments regarding your certification status. A valid company name is used to verify a partner's compliance with certification requirements. To avoid missing out on important benefits and information, you should ensure your contact information is kept current.

Juniper Networks Training Courses

Juniper Networks training courses (`www.juniper.net/training`) are the best source of knowledge for seeking a certification and to increase your hands-on proficiency with Juniper Networks equipment and technologies. While attendance of official Juniper Networks training courses doesn't guarantee a passing score on the certification exam, it does increase the likelihood of your successfully passing it. This is especially true when you seek to attain JNCIP or JNCIE status, where hands-on experience is a vital aspect of your study plan.

Juniper Networks Technical Documentation

You should be intimately familiar with the Juniper Networks technical documentation set (`www.juniper.net/techpubs`). During the JNTCP lab exams (JNCIP and JNCIE), these documents are provided in PDF on your PC. Knowing the content, organizational structure, and search capabilities of these manuals is a key component for a successful exam attempt. At the time of this writing, hard-copy versions of the manuals are provided only for the hands-on lab exams. All written exams delivered at Prometric testing centers are closed-book exams.

Juniper Networks Solutions and Technology

To broaden and deepen your knowledge of Juniper Networks products and their applications, you can visit `www.juniper.net/techcenter`. This website contains white papers, application notes, frequently asked questions (FAQ), and other informative documents, such as customer profiles and independent test results.

Group Study

The Groupstudy mailing list and website (`www.groupstudy.com/list/juniper.html`) is dedicated to the discussion of Juniper Networks products and technologies for the purpose of preparing for certification testing. You can post and receive answers to your own technical questions or simply read the questions and answers of other list members.

Tips for Taking Your Exam

Time, or the lack thereof, is normally one of the biggest factors influencing the outcome of JNCIE-M certification attempts. Having to single-handedly configure numerous protocols and parameters on ten routers while in a somewhat stressful environment often serves as a rude wake-up call early in the JNCIE-M candidate's first attempt.

Although the product documentation is provided during the exam, you will likely run short on time if you have to refer to it more than once or twice during your exam. The successful candidate will have significant practice time with the JUNOS software CLI, and will be experienced with virtually all aspects of protocol configuration, so that commands can be entered quickly and accurately without the need for user manuals.

Although troubleshooting is not a primary component of the exam, many candidates spend a good portion of their time fault-isolating issues that result from their own configuration mistakes or that result from unanticipated interactions between the various protocols involved. Being able to quickly assess the state of the network, and to rapidly isolate and correct mistakes and omissions, are critical skills that a successful JNCIE candidate must possess.

The JNCIE-M exam is scored in a non-linear fashion—this means that a candidate can lose points for a single mistake that happens to affect multiple aspects of their network. The goal of this grading approach can be summed up as, "We grade on results, as opposed to individual configuration statements, and your grade will be determined by the *overall* operational state of your network at the end of the exam." This is a significant point, and one that needs some elaboration, because many candidates are surprised to see how many points can be lost due to a single mistake on a critical facet of the exam.

Non-linear grading The JNCIE-M exam is made up of several sections, and each section is worth a number of points. Missing too many of the criteria within one section can result in zero points being awarded for the entire section, even if the candidate configured some aspects of the task correctly! Getting zero points on a section almost always results in an insufficient number of total points for a passing grade. The goal of this grading approach is to ensure that the JNCIE candidate is able to at least get the majority of each task right. Put another way, "How can you be deemed an *Expert* if you cannot get a significant portion of your IPv6 or provider provisioned VPN configurations correct?"

Results-based grading Because of the numerous ways that JUNOS software can be configured to effect a common result and because an *Expert* should be able to configure a network that is largely operational, the JNCIE-M exam is graded based on overall results. So a serious error in a critical section of the exam can spell doom for the candidate, even if other sections of the candidate's configuration are largely correct. For example, consider the case of a candidate who makes a serious mistake in their OSPF3 configuration. With a dysfunctional IPv6 IGP, there is a high probability that the candidate's multi-protocol BGP and IPv6 related policy-related tasks will exhibit operational problems, which will result in point loss in this section, even though the BGP and policy components of their IPv6 configuration might be configured properly. The moral of this story is make sure that you periodically spot-check the operation of your network, and that you quickly identify and correct operational issues before moving on to subsequent tasks.

Here are some general tips for exam success:

- Arrive early at the exam center, so you can relax and review your study materials.
- Read the task requirements *carefully*. Don't just jump to conclusions. Make sure that you're clear about what each task requires. When in doubt, consult the proctor for clarification. Don't be shy, because the proctor is there mainly to ensure you understand what tasks you are being asked to perform.
- Because the exam is graded based on your network's overall operation, moving on to later tasks when you are "stuck" on a previous task is not always a good idea. In general, you should not move on if your network has operational problems related to a previous task. If you get stuck, you might consider "violating" the rules by deploying a static route (or something similar) in an attempt to complete the entire exam *with* an operational network. You should then plan to revisit your problem areas using any remaining time *after* you have completed all remaining requirements. The point here is that you will likely experience significant point loss if your network has operational problems, so violating some restrictions in an effort to achieve an operational network can be a sound strategy for reducing overall point loss when you are stuck on a particular task.
- Pay attention to detail! With so much work to do and so many routers to configure, many candidates make "simple" mistakes that relate to basic instructions such as the need to filter a specific route, assign prescribed names to variables, and so on.
- Use cut and paste judiciously. Cut and paste can be a real time-saver, but in many cases it can cost a candidate precious time when the configurations of the routers differ significantly or when mistakes are made because the candidate did not correctly adjust parameters before loading the configuration into the next router.
- Read each section (and perhaps the whole exam) fully before starting to type on the consoles. In many cases, the ordering of the requirements for a given section may result in the candidate having to revisit each router many times. By carefully reading all the requirements first, the candidate may be able to save time by grouping requirements so that each router needs to be configured only once.

- Know and prepare for the current test version. At the time of this writing, the production JNCIE-M exam and this book are synchronized to the same JUNOS software version. Before showing up for the exam, the candidate should determine the software version currently deployed in the JNCIE-M testing centers. If newer versions of JUNOS software are rolled out, the well-prepared candidate should study the release notes for the new software and compare any new features or functionality to the current JNCIE-M study guide and preparation road maps to ensure that exam updates will not catch them unprepared.

  It is important to note that the JNCIE-M certification requirements might not change just because a newer software version has been deployed in the lab, because there are many reasons to periodically upgrade the code used in the exam. Please also note that while the exam requirements might not change, the syntax used to establish a given level of functionality might evolve with new software releases.

JNCIE-M exam grading occurs at the end of the day. Results are provided by e-mail within ten business days.

JNCIE Study Guide

Now that you know a lot about the JNTCP, we need to provide some more information about this text. We begin with a look at some topics and information you should already be familiar with and then examine what topics are in the book. Finally, we discuss how to utilize this resource and the accompanying CD.

What You Should Know Before Starting

If you are familiar with networking books, you might be a little surprised to see that this book starts off running, rather than beginning with the Open Systems Interconnection (OSI) model common to books in our industry. We instead dive headfirst into the details of a typical JNCIE-level configuration task that involves the topology discovery (and possible fault isolation) of an internetwork comprising a link-state IGP, route redistribution, BGP, and routing policy. This philosophy of *knowing the basics* is quite ingrained in the Juniper Networks Education courseware and certification exams, so we follow that assumption.

This means that you should be knowledgeable and conversant in the following topics in the context of Juniper Networks M-series Routers or T-series Routing Platforms before attempting your JNCIE examination. Please refer to other Juniper Networks Study Guides published by Sybex for assistance in gaining this knowledge.

- Routing Policy
- Open Shortest Path First (OSPF)
- Intermediate System to Intermediate System (IS-IS)
- Border Gateway Protocol (BGP)
- Multicast
- Multiprotocol Label Switching (MPLS)

- Virtual Private Networks (VPNs)
- Class of Service
- Security and firewall filtering
- IPv6

Scope of the Book

While this book does provide the reader with a "feel" for the JNCIE-M exam, doing well on the exam also involves getting some hands-on experience with M-series and T-series routers to practice the scenarios covered in each chapter. This book serves as a guide to readers who have access to a test bed that is specifically designed for JNCIE exam preparation. However, this book was also written so that adequate preparation can be achieved when the reader combines on-the-job experience with a careful study of the tips and examples contained in this book. The bottom line is that hands-on experience is critical in gaining the proficiency and troubleshooting skills required to successfully pass the JNCIE-M exam.

This book provides the reader with sample configuration scenarios that closely parallel those used in the actual JNCIE-M exam. At the time of writing, this book completely addressed all aspects of the production JNCIE-M exam. In fact, many of the configuration scenarios actually exceed the difficulty level of the current exam so that readers may be better prepared for their certification attempt.

The operational output and configuration examples demonstrated throughout this book are based on JUNOS software version 5.6R1.3 and 5.6R2.4.

What Does This Book Cover?

This book covers design, configuration, and troubleshooting skills that are commensurate with the knowledge and skill set expected of a JNCIE-M candidate. The material closely parallels the actual JNCIE-M environment, in that each configuration example is characterized as a series of requirements and restrictions with which the resulting configuration and network behavior must comply. The reader is walked through each configuration scenario with equal emphasis placed on the correct configuration syntax and on the operational mode commands used to confirm proper operation, as defined by the restrictions placed on each configuration task. In many cases, the reader is made privy to tips and tricks that are intended to save time, avoid common pitfalls, and provide insight into how the JNCIE-M exam is graded. Knowing the techniques that are used by the exam proctors to assess the state of the candidate's network will often allow the candidate to correct his or her own mistakes before it is too late!

Each chapter begins with a list of the lab skills covered in that chapter, with the chapter body providing detailed examples of how the corresponding functionality can be quickly configured and verified. A full-blown case study typical of what the JNCIE-M candidate will encounter in the actual exam is featured near the end of each chapter. Each case study is designed to serve as a vehicle for review and as the basis for lab-based study time. Solutions to the case study configuration requirements and tips for verifying proper operation are provided at the end of

each case study. Each chapter ends with review questions to highlight (and therefore prevent) mistakes that are commonly seen when JNCIE exams are graded.

The book consists of the following material:

- Chapter 1 provides detailed coverage of a network discovery and verification task. This type of task is designed to familiarize the JNCIE candidate with a JNCIE topology that serves as the starting point for the advanced functionality and features that are added in later tasks. A network discovery task is characteristic of how the JNCIE-M candidate will usually begin their testing day.
- Chapter 2 focuses on the configuration and testing of Multiprotocol Label based Switching (MPLS) features, to include LDP, RSVP, constrained routing using Explicit Route Objects (ERO) and Constrained Shortest Path First (CSPF), routing table integration, and traffic protection. This chapter fully explores the issues with incomplete traffic engineering databases (TEDs) that result from the use of multiple area OSPF/multiple level IS-IS topologies.
- Chapter 3 explores the use of JUNOS software firewall filters for packet filtering, rate limiting, and Filter Based Forwarding (FBF). The chapter details Routing Engine (RE) and transit filtering, and also covers Prefix Specific Counters and Policers (PSCP) and interface-based policers.
- Chapter 4 details multicast configuration and testing, to include DVMRP, PIM dense and sparse modes, the bootstrap and auto-RP protocols, Anycast-RP, and interdomain multicast based on Multicast Source Discovery Protocol (MSDP).
- Chapter 5 covers the next generation of Internet protocols in the form of IPv6. The scenarios in this chapter deal with various forms of IPv6 addressing, IGP support in the form of RIPng, OSPF3, and IS0IS, as well as BGP and routing policy support for IPv6.
- Chapter 6 explores Class of Service (CoS) features made available by the 5.6 JUNOS release coupled with Enhanced FPCs (E-FPCs). These scenarios cover a typical Voice over IP (VoIP) application that involves multi-field classification at the edge and Behavior Aggregate (BA) classification in the core based on Differential Services Code Points (DSCPs). The configuration of schedulers and RED profiles that react to the loss priority of each packet is also demonstrated in this chapter.
- Chapter 7 details several Provider Provisioned Virtual Private Network (PP-VPN) scenarios that demonstrate the configuration and testing of Layer 3 and Layer 2 VPN options. These scenarios cover BGP and LDP based signaling VPNs (2547 bis, draft-Kompella, and draft-Martini), and demonstrate advanced concepts such as the OSPF domain-ID, AS-override, Internet access over non-VRF interfaces, mapping VPN traffic to particular LSPs, and obtaining IP II functionality (in other words, firewall filtering and IP address lookups) at the egress PE.

This book is written to mimic the actual JNCIE-M exam by having the reader add layers of complexity and increased functionality to a common network topology with each successive chapter. The decision to use a fixed topology allows the reader to focus on the task at hand instead of having to constantly adapt to new connectivity and address assignments. This layering approach helps to familiarize the reader with how the exam is structured, and also helps to reinforce the relationships between the various network protocols and applications that are covered.

How to Use This Book

This book can provide a solid foundation for the serious effort of preparing for the JNCIE-M exam. To best benefit from this book, we recommend the following study method:

- Read (and understand) the companion Juniper Networks Study Guides, such as the *JNCIA Study Guide*, the *JNCIS Study Guide*, and the *JNCIP Study Guide* (Sybex, 2003), which are designed to prepare you for the lab-based nature of this book.
- When possible, you should gain access to a test bed of Juniper Networks M-series and/or T-series routers—preferably one that matches the topology used throughout this book. Accessing some routers is better than none, so get your hands on as many routers as you can. This book was designed to simulate the experience of actually working with Juniper Networks routers as closely as possible, recognizing that there is a substantial cost associated with the construction of a JNCIE-M test bed. Combining on-the-job experience with a careful analysis of the examples provided in this book will prepare you for the JNCIE-M exam.
- Follow along with the chapter body configuration examples and make sure you understand how network operation is validated against the scenario's requirements through the use of operational commands.
- Do not move on to the next chapter until you are confident that you can perform the case study configuration found at the end of each chapter in the time frames suggested—without the use of manuals and without any serious operational problems in the resulting network.
- Make sure you understand the answers to all the review questions at the end of each chapter. These questions are designed to prevent common mistakes!
- Use the JUNOS software documentation set for researching related information as needed. The documentation set for JUNOS software version 5.6 is included on the accompanying CD.

To learn all the material covered in this book, you'll have to apply yourself regularly and with discipline. Try to set aside the same amount of time every day to practice router configuration and network testing, and select a comfortable and quiet place to do so. If you work hard, you will be surprised at how quickly you demonstrate an expert level of proficiency in the configuration and testing of networks based on JUNOS software and M-series/T-series platforms. Before you know it, you'll be a networking guru and well on your way to the pinnacle of achievement known as the JNCIE. Good luck, and may the force be with you!

What's on the CD?

We worked very hard to provide some really great tools to help you with your certification process. The accompanying CD contains the following:

Complete Router Configurations

The companion CD contains complete router configurations for the case studies found at the end of each chapter. The configurations are available in PDF for printing, and as plain-text files for loading into your own routers. Depending on the situation, you might need to edit the configuration to suit the specific interface types and addressing used in your test bed.

JNCIE Study Guide in PDF

Sybex is also offering the Juniper Networks Certification books on their accompanying CDs so you can read the books on your PC or laptop. The *JNCIE Study Guide* is on this CD in Adobe Acrobat format. Acrobat Reader 5.1 with Search is also included on the CD.

This will be extremely helpful to readers who travel and don't want to carry a book, as well as to readers who find it more comfortable to read from their computer.

JUNOS Software Documentation in PDF

Finally, the Juniper Networks documentation set for version 5.6 is included on the CD so that you can read these manuals on your PC or laptop. The documentation set is in Adobe Acrobat format. Acrobat Reader 5.1 with Search is also included on the CD.

About the Author and Technical Editors

Harry Reynolds, JNCIE #3, CCIE #4977, is a principal developer and a Senior Education Services Engineer at Juniper Networks Inc. He has written numerous training courses and has presented data communications and internetworking training classes for the last 15 years for a variety of organizations. His e-mail address is `h.reynolds@dr-data.net`.

Jason Rogan is a Senior Engineer with Juniper Networks Inc. and Manager of the Juniper Networks Technical Certification Program (JNTCP). He is JNCIE #8 and a Juniper Networks Authorized Instructor.

Peter Moyer is a network consultant with the Professional Services group at Juniper Networks Inc. He holds a B.S. in Computer and Information Science from the University of Maryland and is JNCIE #2 and CCIE #3286. He can be partially blamed for the construction of the industry's toughest and most valuable IP networking exam, the JNCIE.

Josef Buchsteiner is a Senior Network Support Engineer with Juniper Networks Inc. in Amsterdam, The Netherlands. He is JNCIE #38.

Chapter

1

Network Discovery and Verification

JNCIE LAB SKILLS COVERED IN THIS CHAPTER:

- ✓ Verify Out of Band (OoB) management network
- ✓ Discover and verify IGP topology and route redistribution
- ✓ Discover and verify IBGP topology
- ✓ Discover and verify EBGP topology and routing policy

In this chapter, you will be exposed to a series of network discovery and verification tasks that are indicative of those typically encountered at the beginning of the JNCIE examination. While the ability to reverse engineer an unfamiliar network is an invaluable skill that all experts should possess, the primary purpose of the discovery scenario is to allow the candidate to "become one" with the provided baseline network topology before the candidate is expected to begin modifying it during the course of subsequent configuration scenarios.

Because the JNCIE examination focuses on advanced topics such as Multi Protocol Label based Switching (MPLS), firewall filters, and VPNs, the JNCIE examination begins with a preconfigured network with regards to the OoB management network, user accounts, interface configuration, interior gateway protocol (IGP) configuration, Internal and External Border Gateway Protocol (IBGP/EBGP) configuration, and a simple set of redistribution and IBGP/EBGP policies. Though spared the need to actually configure this baseline functionality, the JNCIE candidate begins the examination by discovering the initial network topology and by verifying the overall operation of this baseline network. Because the emphasis of the JNCIE is on higher-level applications and services, the candidate might assume that their interfaces are properly configured and operational. While you will likely find nothing "fancy" about your interface configurations, it is suggested that you give the interface portion of each router's configuration a quick glance; the memory of a non-default logical unit or Virtual Router Redundancy Protocol (VRRP) group configuration may come back to prevent you from making a mistake in a later configuration task.

Although candidates are never intentionally provided with faulty equipment, you should be on guard for any symptoms of hardware malfunction during the network discovery task. In some cases, you may find that the provided configurations require some tweaking. Some versions of the JNCIE examination might require that the candidate perform fault isolation and take corrective actions during the discovery scenario.

Two sets of complete baseline configurations for all routers in the test bed are provided in this chapter. It is suggested that you load your test bed with the same baseline configuration as called out in each chapter to maximize your ability to follow along with each chapter's configuration scenarios.

To kick things off, you will need to access the routers in the test bed either using terminal server–based console attachment, or through the preconfigured Out of Band (OoB) network. Once connected, you can begin to reverse engineer and become one with your new network!

Task 1: Verify OoB Network

As described in the introduction, your JNCIE test bed consists of seven M-series routers, a terminal server, and a 100Mbps Fast Ethernet LAN segment that will serve as your network's

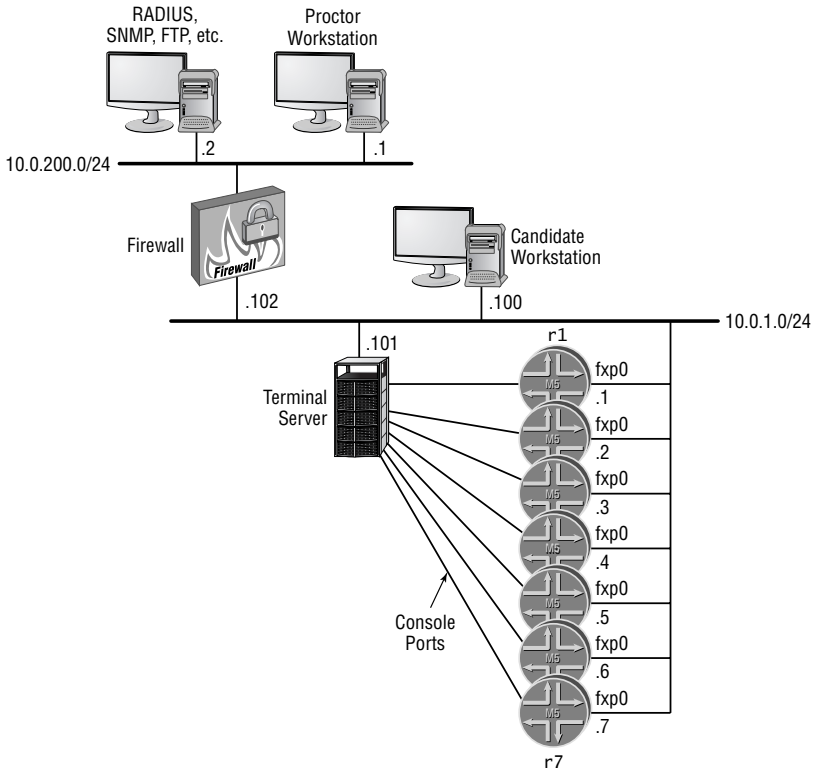
OoB management network. You will likely find that there is no need for terminal server–based attachment because the baseline network has a preconfigured OoB network.

Although you can use the router console ports for the JNCIE examination, most candidates find that it saves time to open multiple telnet sessions (one per router) using the OoB management network that is configured during the examination. You should use the terminal server whenever you are performing router maintenance (such as upgrading JUNOS software), or when routing problems cause telnet access problems.

The OoB Topology

The Out of Band management topology is illustrated in Figure 1.1. Based on this figure, you can see that the IP address of the terminal server is 10.0.1.101, and that its asynchronous interfaces are connected in ascending order to the console ports of each router that is associated with your test bed. The preconfigured fxp0 addressing is also shown in the figure.

FIGURE 1.1 The Out of Band (OoB) management network



The testing center will provide you with both user EXEC and privileged EXEC mode passwords for the terminal server (or their equivalents, should a non–Internetwork Operating System (IOS) based terminal server be in use). This chapter will focus on fxp0-based router access; please see the *JNCIP Study Guide* (Sybex, 2003) for a detailed discussion of terminal server usage.

Accessing Routers Using Telnet

Using the addressing shown earlier in Figure 1.1 and the predefined user account information provided in Table 1.1, verify that you can open a telnet connection to each router using the `lab` login. (A *`root`* login requires terminal server–based console attachment because secure shell [SSH] is not enabled by default).

TABLE 1.1 User Account Parameters

User	Password	Class/Permission	Notes
`root`	`root`	superuser	RADIUS/local password with automatic login in the event of RADIUS failure RADIUS secret is *`juniper`*
`lab`	`lab`	superuser	Same as for user `root`

A successful telnet session will be similar to this capture, which shows the telnet session to `r1` being successfully established:

```
r1 (ttyp1)

login: lab
Password:
Last login: Wed Feb  5 02:44:47 from 10.0.1.100

--- JUNOS 5.6R1.3 built 2003-01-02 20:38:33 UTC

lab@r1>
```

After opening telnet sessions to all seven routers, you quickly confirm the static routing needed to reach the proctor subnet and RADIUS/FTP server by performing some ping testing:

```
lab@r1> ping 10.0.200.2
PING 10.0.200.2 (10.0.200.2): 56 data bytes
64 bytes from 10.0.200.2: icmp_seq=0 ttl=255 time=1.228 ms
64 bytes from 10.0.200.2: icmp_seq=1 ttl=255 time=0.701 ms
^C
--- 10.0.200.2 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.701/0.964/1.228/0.264 ms
```

Verification of the OoB network is complete once you open telnet sessions to all seven routers and verify that each can ping the proctor workstation.

Task 2: Discover and Verify IGP Topology and Route Redistribution

Your next goal is to discover the provided IGP topology, and to verify that there are no operational issues in the core IGP, or in the operation of any IGP route redistribution that may be going on. Figure 1.2 details the JNCIE test bed topology that has been provided in this example. It is suggested that you mark up a copy of the network topology as you proceed in your discovery to assist you in later configuration scenarios.

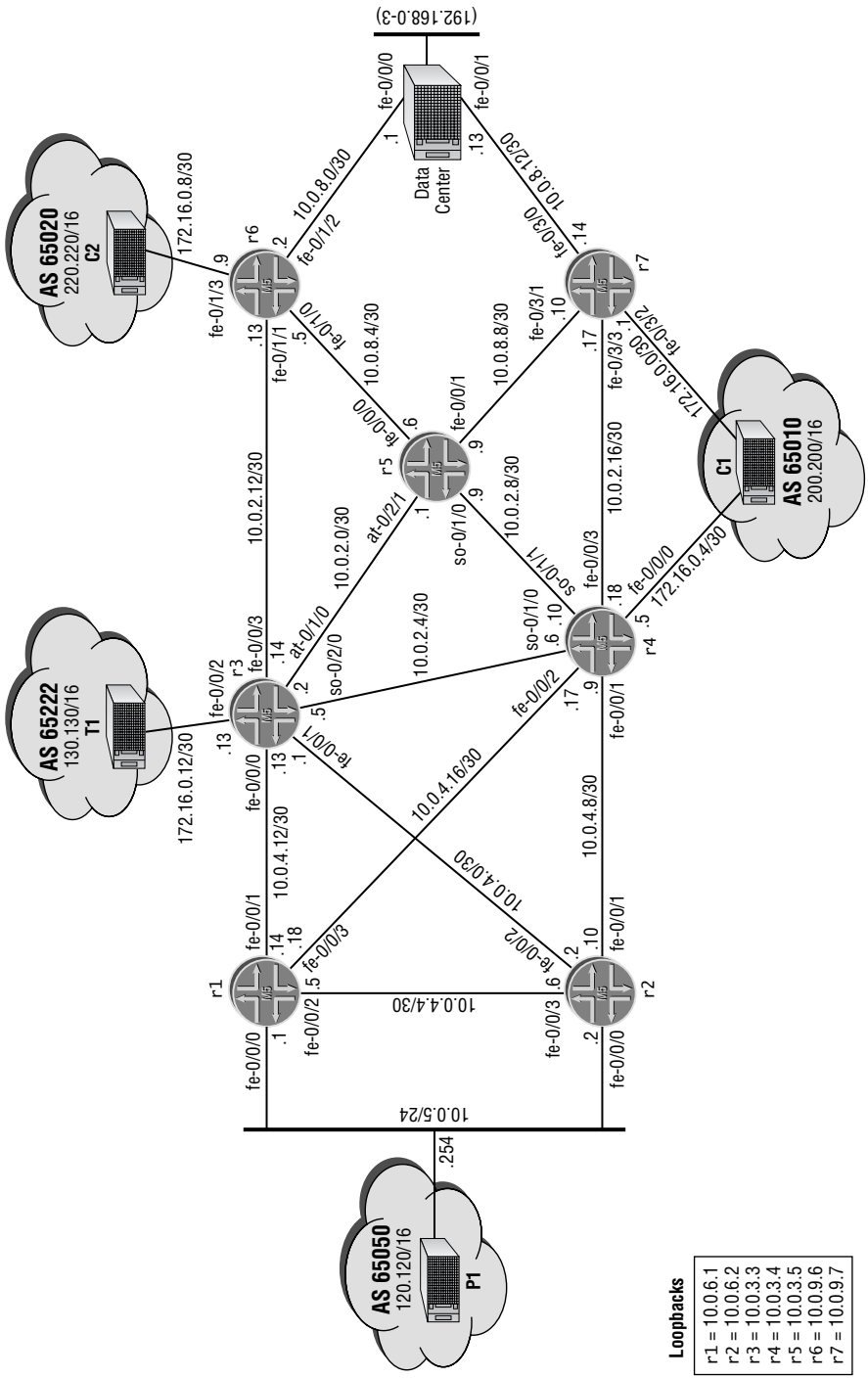
Using the IGP Operation to Verify Interface Operation

Because your IGP relies on proper interface configuration and operation, you can effectively kill two birds with one stone by starting your verification with your IGP. Proper interface operation is effectively confirmed when you have all expected IGP adjacencies and IGP routes, and when traceroute testing confirms optimal paths through the network. You should confirm interface operation when IGP problems are detected even though the IGP's configuration seems correct. It is also a good idea to note any non-default logical units in place for future reference as the JNCIE examination progresses. Note that for the IS-IS routing protocol, proper adjacency formation can occur even if errors are present in the IP configuration of the corresponding interface. Newer versions of JUNOS software, such as the 5.6 release used as the basis for this book, will not form an IS-IS adjacency when IP parameters are mismatched, as reflected in the trace output shown here:

```
lab@r2# run show log isis
Mar  5 08:04:13 Received L1 LAN IIH, source id r1 on fe-0/0/3.0
Mar  5 08:04:13     intf index 5, snpa 0:a0:c9:6f:7b:84
Mar  5 08:04:13     max area 0, circuit type l1, packet length 56
Mar  5 08:04:13     hold time 9, priority 64, circuit id r1.03
Mar  5 08:04:13     neighbor 0:a0:c9:6f:70:d (ourselves)
Mar  5 08:04:13     speaks IP
Mar  5 08:04:13     speaks IPV6
Mar  5 08:04:13     IP address 10.0.6.1
Mar  5 08:04:13     area address 49.0001 (3)
Mar  5 08:04:13     restart RR reset RA reset holdtime 0
Mar  5 08:04:13 ERROR: IIH from r1 without matching addresses,
   interface fe-0/0/3.0
```

The tracing output in this example was obtained at `r2` after the IP address was removed from `r1`'s fe-0/0/2 interface.

FIGURE 1.2 The JNCIE test bed topology



The reader who is familiar with the previous book in this series should immediately recognize numerous similarities between the JNCIP and JNCIE topologies. This level of similarity may or may not occur in the actual JNCIE examination, which is why the candidate begins the examination with a discovery scenario designed to familiarize the candidate with their "new" topology.

Figure 1.2 (shown earlier) holds a wealth of information about your test bed. From the figure, you can see that you have a mix of EBGP peers, and that route redistribution will likely be in place between `r6`, `r7`, and the data center routers. You will also note that your test bed once again consists of a mix of interface types, including Fast Ethernet, OC-3c POS, and ATM.

Discovering and Verifying Core IGP

While there are many ways in which a candidate might decide to attack the discovery of their network's IGP, this author has chosen to begin with `r3`, `r4`, and `r7`, because their central placement implies that much can be learned by examining the configuration (and operation) of their IGP. You take a deep breath and begin by displaying `r3`'s protocol stanza:

```
[edit]
lab@r3# show protocols
bgp {
    advertise-inactive;
    group int {
        type internal;
        local-address 10.0.3.3;
        export nhs;
        neighbor 10.0.6.1;
        neighbor 10.0.6.2;
        neighbor 10.0.3.4;
        neighbor 10.0.3.5;
        neighbor 10.0.9.6;
        neighbor 10.0.9.7;
    }
    group ext {
        import ebgp-in;
        export ebgp-out;
        neighbor 172.16.0.14 {
            peer-as 65222;
        }
    }
}
ospf {
    area 0.0.0.1 {
        stub default-metric 10;
```

```
        interface fe-0/0/0.0;
        interface fe-0/0/1.0;
    }
    area 0.0.0.0 {
        interface so-0/2/0.100;
        interface at-0/1/0.0;
    }
    area 0.0.0.2 {
        nssa;
        interface fe-0/0/3.0;
    }
}
```

The highlighted portion relating to r3's IGP configuration is the area of concern at this stage. From r3's IGP configuration, you can determine the following:

- The core IGP is OSPF (Open Shortest Path First).
- r3 is an Area Border Router (ABR) for areas 1 and 2.
- Area 1 is a stub network and r3 is configured to generate a default route into that area.
- Area 2 is configured as a NSSA (not-so-stubby area) network. No default route is generated by r3 into area 2.
- No address aggregation or restriction of summary LSAs is occurring at r3.
- OSPF authentication is not configured in areas 0, 1, and 2.
- r3 will run OSPF on all interfaces in the baseline topology, except its lo0 and external fe-0/0/2 interfaces.

The omission of the router's lo0 interface from the area declarations results in advertisement of the router's loopback address (the lo0 interface is the default source of the RID) in the router LSAs injected into all areas. Although not shown here, the OSPF configuration for r4 is virtually identical to that of r3. Now that you have some idea of what to expect, it makes sense to quickly assess the state of r3's adjacencies:

```
[edit]
lab@r3# run show ospf neighbor
  Address          Interface              State      ID               Pri  Dead
10.0.2.1           at-0/1/0.0             Full       10.0.3.5         128   36
10.0.2.6           so-0/2/0.100           Full       10.0.3.4         128   34
10.0.4.14          fe-0/0/0.0             Full       10.0.6.1         128   32
10.0.4.2           fe-0/0/1.0             Full       10.0.6.2         128   31
10.0.2.13          fe-0/0/3.0             Full       10.0.9.6         128   39
```

The results confirm that all five of r3's adjacencies have been correctly established. A quick look at r4's adjacencies confirms that it too has the five adjacencies one would expect,

given this topology:

```
[edit]
lab@r4# run show ospf neighbor
  Address          Interface              State      ID               Pri  Dead
10.0.2.5           so-0/1/0.100           Full       10.0.3.3         128   34
10.0.2.9           so-0/1/1.0             Full       10.0.3.5         128   39
10.0.4.10          fe-0/0/1.0             Full       10.0.6.2         128   35
10.0.4.18          fe-0/0/2.0             Full       10.0.6.1         128   35
10.0.2.17          fe-0/0/3.0             Full       10.0.9.7         128   32
```

You now quickly examine the OSPF configuration for r1 and r2:

```
[edit]
lab@r1# show protocols ospf
area 0.0.0.1 {
    stub;
    interface fe-0/0/0.0 {
        passive;
    }
    interface fe-0/0/1.0;
    interface fe-0/0/2.0;
    interface fe-0/0/3.0;
}
```

r1's configuration allows you to determine that it is configured to run a passive OSPF instance on its fe-0/0/0 interface, and that its overall configuration is commensurate with the stub area settings discovered in r3. The passive setting on its fe-0/0/0 interface will prevent adjacency formation on the P1 peering subnet, while allowing the 10.0.5/24 prefix to be carried as an OSPF internal route. With r2's configuration being virtually identical (not shown), you expect to see three OSPF adjacencies in place on both r1 and r2:

```
lab@r1# run show ospf neighbor
  Address          Interface              State      ID               Pri  Dead
10.0.4.13          fe-0/0/1.0             Full       10.0.3.3         128   34
10.0.4.6           fe-0/0/2.0             Full       10.0.6.2         128   35
10.0.4.17          fe-0/0/3.0             Full       10.0.3.4         128   34
```

As anticipated, r1 has the correct number of adjacent neighbors. With area 1 configured as a stub area, there should be no external routes in r1's link state database:

```
[edit]
lab@r2# run show ospf database extern
```

Because network summaries (LSA Type 3s) are not being filtered at the ABR (r3), you expect to see OSPF routes to the loopback addresses of all routers making up your test bed. Some

creative CLI (command-line interface) work makes this determination a snap:

```
[edit]
lab@r2# run show route protocol ospf | match /32
10.0.3.3/32        *[OSPF/10] 00:12:37, metric 1
10.0.3.4/32        *[OSPF/10] 01:52:14, metric 1
10.0.3.5/32        *[OSPF/10] 00:12:37, metric 2
10.0.6.1/32        *[OSPF/10] 01:52:14, metric 1
10.0.9.6/32        *[OSPF/10] 00:12:37, metric 2
10.0.9.7/32        *[OSPF/10] 01:52:14, metric 2
224.0.0.5/32       *[OSPF/10] 03:32:36, metric 1
```

The highlighted output generated by r2 confirms that the loopback addresses of the other six routers are being learned through the OSPF protocol. As a final check, you confirm the presence of a default route in accordance with the configuration found on ABR r3:

```
[edit]
lab@r2# run show route

inet.0: 118111 destinations, 118118 routes (118111 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[OSPF/10] 00:47:14, metric 11
                    > to 10.0.4.9 via fe-0/0/1.0
                      to 10.0.4.1 via fe-0/0/2.0
. . .
```

The default route is present in area 1, and the two viable next hops listed indicate that both r3 and r4 are sourcing a default route into the stub area. So far, things are looking pretty good for the operation of the test bed's IGP!

Discovering and Verifying IGP Redistribution

Having confirmed the overall operation of the OSPF protocol for r1 through r4, you next examine the OSPF configuration at r5:

```
[edit]
lab@r5# show protocols ospf
area 0.0.0.0 {
    interface at-0/2/1.0;
    interface so-0/1/0.0;
}
area 0.0.0.2 {
    nssa;
    interface fe-0/0/0.0;
```

```
    interface fe-0/0/1.0;
}
```

The output indicates that r5 is an ABR interconnecting area 2 and the backbone. You also note that, like r3, r5 considers area 2 to be a NSSA. The lack of the default metric keyword indicates that r5 will not generate a default route into area 2. With the same finding made at r3 and r4, you conclude that the NSSA will not have an OSPF derived default route. You quickly confirm your suspicions regarding the absence of a default route in area 2 using the following command on r6:

```
[edit]
lab@r6# run show route | match 0.0.0.0/0

[edit]
lab@r6#
```

Considering that nothing you have uncovered so far can be considered "broken," you simply note the lack of a default route in the NSSA, and you move on with your discovery task.

Why Is There No Default Route in the NSSA?

You may find it odd that none of area 2's ABRs are configured to generate a default route into the NSSA. Because network summaries are permitted in the NSSA, and because there are no OSPF AS-externals (LSA Type 5s) being generated in areas 0 or 1, the lack of a default route in the NSSA may not represent a problem. If all external routing information associated with areas 0 and 1 is carried in BGP, for example, the routers in area 2 should not have trouble reaching external destinations.

You expect to find four OSPF adjacencies in place at r5. The output from r5 quickly confirms your expectations on this front:

```
[edit]
lab@r5# run show ospf neighbor
  Address          Interface              State       ID               Pri  Dead
10.0.2.2           at-0/2/1.0             Full        10.0.3.3         128    32
10.0.2.10          so-0/1/0.0             Full        10.0.3.4         128    38
10.0.8.5           fe-0/0/0.0             Full        10.0.9.6         128    39
10.0.8.10          fe-0/0/1.0             Full        10.0.9.7         128    37
```

With r5's IGP configuration analyzed, you move on to r7 to inspect its IGP configuration:

```
[edit]
lab@r7# show interfaces
fe-0/3/0 {
    unit 0 {
```

```
        family inet {
            address 10.0.8.14/30;
        }
        family iso;
    }
}
fe-0/3/1 {
    unit 0 {
        family inet {
            address 10.0.8.10/30;
        }
    }
}
fe-0/3/2 {
    unit 0 {
        family inet {
            address 172.16.0.1/30;
        }
    }
}
fe-0/3/3 {
    unit 0 {
        family inet {
            address 10.0.2.17/30;
        }
    }
}
fxp0 {
    unit 0 {
        family inet {
            address 10.0.1.7/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 10.0.9.7/32;
        }
        family iso {
            address 49.0002.7777.7777.7777.00;
```

```
        }
    }
}
[edit]
lab@r7# show protocols
bgp {
    group int {
        type internal;
        local-address 10.0.9.7;
        export nhs;
        neighbor 10.0.6.1;
        neighbor 10.0.6.2;
        neighbor 10.0.3.3;
        neighbor 10.0.3.4;
        neighbor 10.0.3.5;
        neighbor 10.0.9.6;
    }
    group c1 {
        type external;
        export ebgp-out;
        neighbor 172.16.0.2 {
            peer-as 65010;
        }
    }
}
isis {
    export ospf-isis;
    level 2 disable;
    level 1 external-preference 149;
    interface fe-0/3/0.0;
    interface lo0.0;
}
ospf {
    export isis-ospf;
    area 0.0.0.2 {
        nssa;
        interface fe/0/3/1.0;
        interface fe-0/3/0.0 {
            passive;
        }
```

```
        interface fe-0/3/3.0;
    }
}
```

Once again, the IGP related portions of the router's configuration are called out with highlights. Though not shown here, the configuration of `r6` is virtually identical to that shown for `r7`. The presence of both OSPF and IS-IS stanzas tells you that `r7` is most likely acting as a redistribution source for the 192.168.0/22 routes associated with the data center. You also note the following:

- `r7` is configured to operate as a Level 1 IS-IS router on its fe-0/3/0 interface, which implies that the DC router is running IS-IS Level 1.
- The global preference for IS-IS Level 1 external routes has been modified to ensure that the IS-IS routes are preferred over their OSPF equivalents when they are redistributed into OSPF as NSSA externals, which have a default preference of 150.
- `r7` has been set to run a passive OSPF instance on its fe-0/3/0 interface; this will result in advertisement of the 10.0.8.12/30 subnet as an OSPF internal route while also guarding against unwanted OSPF adjacencies to the DC router.
- Export policy is in place for both the OSPF and IS-IS IGPs.

You start by quickly accessing the state of IGP adjacencies at `r6` or `r7`; based on the configuration displayed, you expect a total of three adjacencies, two of which should be OSPF and one that is IS-IS Level 1:

```
[edit]
lab@r6# run show ospf neighbor
  Address          Interface              State      ID              Pri   Dead
10.0.8.6           fe-0/1/0.0             Full       10.0.3.5        128   36
10.0.2.14          fe-0/1/1.0             Full       10.0.3.3        128   32
```

The display confirms the expected number of OSPF adjacencies at `r6`. You next confirm its IS-IS adjacency status:

```
[edit]
lab@r6# run show isis adjacency
Interface             System           L State       Hold (secs)  SNPA
fe-0/1/2.0               dc              1 Up                   7  0:a0:c9:69:c5:27
```

Excellent! All expected IGP adjacencies are established at `r6`. You now display the *`ospf-isis`* export policy to get a handle on what routes should be redistributed from OSPF to the DC router:

```
[edit]
lab@r6# show policy-options policy-statement ospf-isis
term 1 {
    from {
        protocol ospf;
        route-filter 0.0.0.0/0 exact;
    }
```

```
    then accept;
}
```

The *ospf-isis* export policy is designed to redistribute a default route from OSPF into IS-IS. Most likely, this default route is intended to provide the data center router with reachability to internal and external prefixes, as it is assumed that a DC router will not be running BGP. But wait—a previous test confirmed that there was no default route in area 2! You quickly re-verify that no OSPF-based default route exists at r6:

```
[edit]
lab@r6# run show route protocol ospf | match 0.0.0.0
```

No default route, OSPF or otherwise. This makes the *ospf-isis* policy more than a bit moot, and this may represent an operational problem. A quick telnet hop to the DC router confirms the magnitude of the situation:

```
lab@dc> show route protocol isis

inet.0: 16 destinations, 16 routes (16 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.9.6/32        *[IS-IS/15] 02:00:35, metric 10
                    > to 10.0.8.2 via fe-0/0/0.0
10.0.9.7/32        *[IS-IS/15] 00:49:24, metric 10
                    > to 10.0.8.14 via fe-0/0/1.0

iso.0: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)
```

The output confirms that the only IS-IS routes being advertised to the data center router from r6 and r7 are the prefixes associated with their loopback addresses. Further testing confirms serious reachability problems at the data center when a traceroute to r5 fails:

```
lab@dc> traceroute 10.0.3.5
traceroute to 10.0.3.5 (10.0.3.5), 30 hops max, 40 byte packets
traceroute: sendto: No route to host
 1 traceroute: wrote 10.0.3.5 40 chars, ret=-1
^C
```

In light of the *ospf-isis* policies in place on r6 and r7, and the fact that reachability problems have been confirmed in the data center, it now seems that NSSA area 2 is "broken" by virtue of there being no OSPF-based default route available for redistribution into the data center. Before making any changes to the baseline network, you should document your findings and bring them to the attention of the proctor. In this example, the proctor confirms the need for a default route in area 2 and authorizes the necessary changes on the ABRs that serve the NSSA. The following command is entered on r3, r4, and r5, which configures them to generate a default route into area 2:

```
[edit protocols ospf]
lab@r3# set area 2 nssa default-lsa default-metric 10
```

There is no need to specify an LSA Type 7 for the default route in this case, as summary LSAs are permitted in the NSSA. After the change is committed on r3, the results are confirmed at r6:

```
[edit]
lab@r6# run show route protocol ospf | match 0.0.0.0/0
0.0.0.0/0          *[OSPF/150] 00:00:23, metric 11, tag 0
```

Great, the default route is now present and active as an OSPF route. Before proceeding, you should verify that all three of the NSSA's ABRs are now configured to source a default route into area 2. When correctly configured, both r6 and r7 will display two viable next hops for the OSPF default route. The data center router should now be receiving the default route from both r6 and r7. After telnetting to the data center router, the presence of a default route pointing to r6 and r7 as the next hops is confirmed, as is the data center router's ability to reach various 10.0/16 destinations:

```
lab@dc> show route

inet.0: 17 destinations, 17 routes (17 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[IS-IS/160] 00:00:05, metric 21
                      to 10.0.8.2 via fe-0/0/0.0
                    > to 10.0.8.14 via fe-0/0/1.0
```

The default route correctly lists both r6 and r7 as viable next hops; this proves that the *ospf-isis* export policy is now functional on both r6 and r7. With the default route present, traceroutes originated at the data center now succeed:

```
lab@dc> traceroute 10.0.3.3
traceroute to 10.0.3.3  (10.0.3.3), 30 hops max, 40 byte packets
 1  10.0.8.14 (10.0.8.14)  0.377 ms  0.221 ms  0.155 ms
 2  10.0.8.9  (10.0.8.9)   0.435 ms  0.391 ms  0.388 ms
 3  10.0.3.3  (10.0.3.3)   0.815 ms  1.120 ms  1.071 ms

lab@dc> traceroute 10.0.6.2
traceroute to 10.0.6.2 (10.0.6.2), 30 hops max, 40 byte packets
 1  10.0.8.14 (10.0.8.14)  0.263 ms  0.185 ms  0.155 ms
 2  10.0.2.18 (10.0.2.18)  0.435 ms  0.374 ms  0.388 ms
 3  10.0.6.2  (10.0.6.2)   0.288 ms  0.285 ms  0.262 ms
```

Both of the traceroutes complete normally, but the reliance on a default route with two equal-cost next hops has resulted in a less than optimal forwarding path to some destinations, because the data center router has installed r7 as the default route's current next hop as this is being written. This situation can be considered normal, so for now you simply note the issue and move on with your network discovery actions.

With OSPF to IS-IS redistribution now confirmed, you examine the *isis-ospf* policy to determine the redistribution behavior expected in the IS-IS to OSPF direction:

```
[edit]
lab@r7# show policy-options policy-statement isis-ospf
term 1 {
    from {
        protocol isis;
        route-filter 192.168.0.0/22 longer;
    }
    then accept;
}
```

The *isis-ospf* policy seems pretty straightforward. Routes learned from IS-IS matching the `192.168.0/22 longer` route filter declaration should be redistributed into area 2 using an LSA Type 7 in accordance with the area's NSSA settings.

You begin verification of the IS-IS to OSPF redistribution aspects of the baseline network by confirming that both `r6` and `r7` have installed the IS-IS versions of the 192.168.0/22 data center routes as active. Recall that the configuration in `r6` and `r7` has adjusted the default preference of IS-IS Level 1 externals from 160 to 149, to ensure that they will be preferred to the versions being redistributed into OSPF by the other router:

```
[edit]
lab@r7# run show route 192.168.0/22

inet.0: 125015 destinations, 125029 routes (125015 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

192.168.0.0/24     *[IS-IS/149] 00:26:16, metric 10
                    > to 10.0.8.13 via fe-0/3/0.0
                    [OSPF/150] 00:25:52, metric 10, tag 0
                    > to 10.0.8.9 via fe-0/3/1.0
                    [BGP/170] 00:25:53, MED 10, localpref 100, from 10.0.9.6
                      AS path: I
                    > to 10.0.8.9 via fe-0/3/1.0
192.168.0.1/32     *[IS-IS/15] 00:26:16, metric 10
                    > to 10.0.8.13 via fe-0/3/0.0
                    [OSPF/150] 00:25:52, metric 10, tag 0
                    > to 10.0.8.9 via fe-0/3/1.0
                    [BGP/170] 00:25:53, MED 10, localpref 100, from 10.0.9.6
                      AS path: I
                    > to 10.0.8.9 via fe-0/3/1.0
```

```
192.168.1.0/24     *[IS-IS/149] 00:26:16, metric 20
                    > to 10.0.8.13 via fe-0/3/0.0
                    [OSPF/150] 00:25:52, metric 20, tag 0
                    > to 10.0.8.9 via fe-0/3/1.0
                    [BGP/170] 00:25:52, MED 20, localpref 100, from 10.0.9.6
                      AS path: I
                    > to 10.0.8.9 via fe-0/3/1.0
192.168.2.0/24     *[IS-IS/149] 00:26:16, metric 20
                    > to 10.0.8.13 via fe-0/3/0.0
                    [OSPF/150] 00:25:52, metric 20, tag 0
                    > to 10.0.8.9 via fe-0/3/1.0
                    [BGP/170] 00:25:52, MED 20, localpref 100, from 10.0.9.6
                      AS path: I
                    > to 10.0.8.9 via fe-0/3/1.0
192.168.3.0/24     *[IS-IS/149] 00:26:16, metric 20
                    > to 10.0.8.13 via fe-0/3/0.0
                    [OSPF/150] 00:25:52, metric 20, tag 0
                    > to 10.0.8.9 via fe-0/3/1.0
                    [BGP/170] 00:25:52, MED 20, localpref 100, from 10.0.9.6
                      AS path: I
                    > to 10.0.8.9 via fe-0/3/1.0
```

The output confirms that r7 has selected the IS-IS versions of the 192.168.0/22 routes as active. You can also determine from this display that r6 has redistributed the 192.168.0/22 routes into both OSPF and IBGP. These points help to confirm the correct operation of r6's redistribution policies. Though not shown, the same command is issued on r6 to confirm that it displays a similar view of the 192.168.0/22 routes. The presence of the data center routes are next confirmed in the backbone area with the following command entered on r3:

```
lab@r3# run show route 192.168.1/24

inet.0: 118083 destinations, 118105 routes (118083 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

192.168.1.0/24     *[OSPF/150] 00:12:59, metric 20, tag 0
                    > to 10.0.2.13 via fe-0/0/3.0
                    [BGP/170] 00:12:59, MED 20, localpref 100, from 10.0.9.6
                      AS path: I
                    > to 10.0.2.13 via fe-0/0/3.0
                    [BGP/170] 00:12:59, MED 20, localpref 100, from 10.0.9.7
                      AS path: I
```

```
> via at-0/1/0.0
  via so-0/2/0.100
```

Good, the routes are present in the OSPF backbone as both OSPF and BGP routes; the presence of two BGP next hops for the DC routes further confirms that both r6 and r7 are redistributing the 192.168.0/22 routes into BGP. Before considering your OSPF discovery exercise complete, you should take a few moments to trace routes to various internal destinations to verify there are no forwarding oddities at play in the baseline network. For example, the layout of area 2 results in packets taking an extra hop when r3 or r4 forwards packets to the loopback address of r7 or r6, respectively. This behavior is to be expected, because in this topology r4 learns r6's loopback address from a router LSA in area 2 (as flooded by r7) and from a network summary flooded into the backbone area by r5. Because an OSPF router always prefers internal (intra area) routes over a network summary, r4 forwards through r7 to reach the loopback address of r6. The same behavior is observed when r3 forwards to r7's loopback address, as shown here:

```
lab@r3# run traceroute 10.0.9.7
traceroute to 10.0.9.7 (10.0.9.7), 30 hops max, 40 byte packets
 1  10.0.2.13 (10.0.2.13)  0.776 ms  0.556 ms  0.426 ms
 2  10.0.8.6 (10.0.8.6)  0.700 ms  9.111 ms  0.648 ms
 3  10.0.9.7 (10.0.9.7)  0.577 ms  0.553 ms  0.514 ms
```

This situation can be considered par for the course, or could be corrected with an additional link between r6 and r7, with a static route, or with a redefinition of the OSPF area boundaries. In this case, you are informed that the baseline network is "operating as designed" so no corrective actions are taken at this time. With the completion of your traceroute testing, your operational analysis of area 2 is complete!

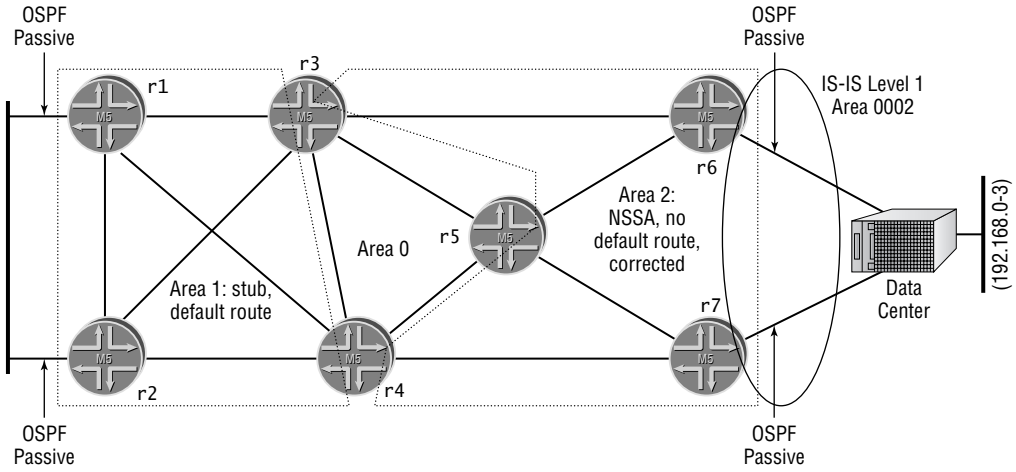
Summary of IGP Discovery

Your discovery activities have resulted in the determination that the baseline network consists of a multi-area OSPF IGP with mutual route redistribution occurring between the network core and data center locations. In this example, you were provided with an IGP that was largely functional and, for the most part, properly configured. The notable exception would be the missing OSPF default route in area 2 that led to connectivity problems for the data center.

Your findings have confirmed that all OSPF (and IS-IS) adjacencies are in place and that, with a few exceptions, optimal forwarding paths are in place. The exceptions you have noted include the data center router, which uses a 0/0 default route with two viable next hops to reach various destinations, and the extra hops incurred by r3 and r4 due to the specific layout of area 2.

Documenting your discovery findings is a good idea. Being able to refresh your memory with an accurate picture of the network that you have inherited may prevent mistakes in subsequent configuration tasks. Figure 1.3 provides an example of the key points that you have discovered in your JNCIE test bed so far.

FIGURE 1.3 Summary of IGP discovery findings



Notes:

Loopback addresses have not been assigned to specific areas (lo0 address advertised in Router LSA in all areas).

Passive OSPF interfaces on P1 and data center segments.

No authentication or route summarization in effect; summaries (LSA type 3) allowed in all areas.

Redistribution of OSPF default route to data center from both `r6` and `r7` was broken. Fixed with default-metric command on `r3`, `r4`, and `r5`.

Data center router running IS-IS, Level 1. `r6` and `r7` compatibly configured and adjacent.

Redistribution of 192.168.0/24 through 192.168.3/24 into OSPF from IS-IS by both `r6` and `r7`.

Adjustment to IS-IS level 1 external preference to ensure `r6` and `r7` always prefer IS-IS Level 1 externals over OSPF externals.

All adjacencies up and full reachability confirmed.

Sub-optimal routing detected at the data center router for some locations, and when `r3` and `r4` forward to some Area 2 addresses. This is the result of random nexthop choice for its default route and Area 2 topology specifics. Considered to be working as designed; no action taken.

Task 3: IBGP Discovery and Verification

With your network's IGPs and route redistribution confirmed as operational, it is time to take things up a notch by analyzing the network's IBGP configuration and operation. Once again, you begin your analysis on a backbone router:

```
[edit]
lab@r5# show protocols bgp
group int {
    type internal;
```

```
        local-address 10.0.3.5;
        neighbor 10.0.6.1;
        neighbor 10.0.6.2;
        neighbor 10.0.3.3;
        neighbor 10.0.3.4;
        neighbor 10.0.9.6;
        neighbor 10.0.9.7;
    }
    [edit]
    lab@r5# show routing-options
    static {
        route 10.0.200.0/24 {
            next-hop 10.0.1.102;
            no-readvertise;
        }
    }
    autonomous-system 65412;
```

Well, there certainly appears to be nothing fancy going on here! You now know that your test bed is (once again) using Autonomous System Number (ASN) *65412*. Further, the IBGP configuration at `r5` indicates that you have been provided with a full mesh of IBGP sessions using lo0-based peering. A quick glance at the status of `r5`'s IBGP sessions confirms that all six of its IBGP sessions have been correctly established:

```
[edit]
lab@r5# run show bgp summary
Groups: 1 Peers: 6 Down peers: 0
Table    Tot Paths  Act Paths Suppressed    History Damp State    Pending
inet.0      125100     125088          0          0          0          0
Peer      AS    InPkt OutPkt OutQ Flaps  Last Up/Dwn State|#Active/Received/Damped...
10.0.3.3  65412 24421    166    0     0     1:21:54 125085/125085/0      0/0/0
10.0.3.4  65412   168    168    0     0     1:22:46 1/1/0                0/0/0
10.0.6.1  65412   165    167    0     0     1:22:02 1/1/0                0/0/0
10.0.6.2  65412   164    166    0     0     1:21:58 0/1/0                0/0/0
10.0.9.6  65412   167    166    0     0     1:21:52 1/6/0                0/0/0
10.0.9.7  65412   167    167    0     0     1:22:04 0/6/0                0/0/0
```

Seeing that all of `r5`'s loopback-based IBGP sessions are in the established state provides an additional check of your network's IGP, as the IGP is needed to route between the loopback addresses of the routers in the test bed. You also note that `r5` has received at least one route from each IBGP peer, and that it has received a whole bunch of routes from `r3`; you note that `r3` in turn EBGP peers with a transit provider T1, so these findings make a fair bit of sense. Your attention now shifts to the analysis of `r7`'s configuration. You note that the presence of an

EBGP peer in the form of C1 will make r7's configuration differ from that observed at r5:

```
[edit]
lab@r7# show protocols bgp
group int {
    type internal;
    local-address 10.0.9.7;
    export nhs;
    neighbor 10.0.6.1;
    neighbor 10.0.6.2;
    neighbor 10.0.3.3;
    neighbor 10.0.3.4;
    neighbor 10.0.3.5;
    neighbor 10.0.9.6;
}
group c1 {
    type external;
    export ebgp-out;
    neighbor 172.16.0.2 {
        peer-as 65010;
    }
}
```

The IBGP configuration of r7 is similar to that shown for r5, with the exception of the highlighted *nhs* export policy statement and the presence of EBGP-related configuration for the C1 peering session. The *nhs* export policy is displayed on r7:

```
[edit]
lab@r7# show policy-options policy-statement nhs
term 1 {
    from {
        protocol bgp;
        neighbor 172.16.0.2;
    }
    then {
        next-hop self;
    }
}
term 2 {
    from {
        route-filter 192.168.0.0/22 longer;
    }
    then accept;
}
```

The first term in the *nhs* policy resets the BGP next hop for routes learned from C1 to r7's RID. This eliminates the need to carry the various 172.16.0/30 EBGP link addresses in your IGP. The second term in the *nhs* policy results in the data center routes being redistributed into IBGP, presumably so that they can in turn be re-advertised to your network's EBGP peers by the other routers in the test bed. Note that r1 and r2 do not receive the data center routes via OSPF external LSAs due to a stub area's inability to carry external routing information.

The IBGP configuration on the remaining routers is similar to that shown for r7, with the following exceptions noted.

The advertise-inactive option has been set on r3 and r4 as highlighted:

```
[edit]
lab@r4# show protocols bgp
advertise-inactive;
group int {
    type internal;
    local-address 10.0.3.4;
    export nhs;
    neighbor 10.0.6.1;
    neighbor 10.0.6.2;
    neighbor 10.0.3.3;
    neighbor 10.0.3.5;
    neighbor 10.0.9.6;
    neighbor 10.0.9.7;
}
group c1 {
    type external;
    export ebgp-out;
    neighbor 172.16.0.6 {
        peer-as 65010;
    }
```

The presence of active OSPF routes for the data center on r3 and r4 will prevent their advertisement into EBGP without the use of some type of OSPF-to-BGP export policy. The advertise-inactive option on r3 and r4 alleviates this problem in the most expedient way possible with no policy modifications needed. The advertise-inactive option is not needed on r1 and r2 because they do not receive the OSPF advertisements for the DC's routes, thus making the IBGP versions of these routes active and therefore eligible for export using the default BGP policy.

The lack of a next hop self-policy on r1 and r2 is noted, but is not considered an issue at this time. Resetting the EBGP next hop is not needed on these routers because the 10.0.5/24 peering subnet is being advertised into OSPF due to the passive IGP setting on their fe-0/0/0 interfaces. Having the 10.0.5/24 subnet carried in OSPF makes P1's 10.0.5.254 EBGP next hop reachable by all routers in your AS.

As a final check on your network's IBGP operation, you verify that the data center's routes are present in both r1 and r2, and that each router displays two viable BGP next hops, as this will confirm that r1 and r2 are correctly receiving the 192.168.0/22 routes from both r6 and r7:

```
[edit]
lab@r2# run show route 192.168.2/24

inet.0: 118098 destinations, 118113 routes (118098 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

192.168.2.0/24     *[BGP/170] 01:27:12, MED 20, localpref 100, from 10.0.9.6
                      AS path: I
                    > to 10.0.4.1 via fe-0/0/2.0
                    [BGP/170] 01:27:12, MED 20, localpref 100, from 10.0.9.7
                      AS path: I
                    > to 10.0.4.9 via fe-0/0/1.0
```

Before moving into the EBGP and policy verification task, you should review each router's IBGP export policy, and you should quickly confirm that all IBGP session are established on all routers. You can assume that in this example all IBGP sessions are established and that no IBGP-related operational problems were detected.

Task 4: EBGP and Routing Policy Discovery

Having verified that your network's overall IGP and IBGP operation are sound, it is time to move on to your final network discovery task—namely the discovery and verification of your test bed's EBGP topology and its related routing policy.

P1 Peering

You begin the EBGP and policy discovery process on r1 by verifying its EBGP session status to P1:

```
[edit]
lab@r1# run show bgp neighbor 10.0.5.254
Peer: 10.0.5.254+179  AS 65050 Local: 10.0.5.1+1544   AS 65412
  Type: External    State: Established    Flags: <>
  Last State: OpenConfirm   Last Event: RecvKeepAlive
  Last Error: None
  Export: [ ebgp-out ]
  Options: <Preference HoldTime PeerAS Refresh>
```

```
  Holdtime: 90 Preference: 170
  Number of flaps: 0
  Peer ID: 120.120.0.1      Local ID: 10.0.6.1         Active Holdtime: 90
  Keepalive Interval: 30
  Local Interface: fe-0/0/0.0
  NLRI advertised by peer: inet-unicast
  NLRI for this session: inet-unicast
  Peer supports Refresh capability (2)
  Table inet.0 Bit: 10000
    RIB State: BGP restart is complete
    Send state: in sync
    Active prefixes:            1
    Received prefixes:          1
    Suppressed due to damping:  0
  Last traffic (seconds): Received 23   Sent 2    Checked 2
  Input messages:  Total 884    Updates 16      Refreshes 0     Octets 17434
  Output messages: Total 894    Updates 23      Refreshes 0     Octets 17960
  Output Queue[0]: 0
```

The output confirms that the EBGP session to P1 is in the established state, and that one prefix has been received and installed as an active route over this session. The EBGP sessions' established state is an auspicious beginning, so you decide to analyze the EBGP configuration stanza on r1:

```
lab@r1# show protocols bgp
group int {
    type internal;
    local-address 10.0.6.1;
    neighbor 10.0.6.2;
    neighbor 10.0.3.3;
    neighbor 10.0.3.4;
    neighbor 10.0.3.5;
    neighbor 10.0.9.6;
    neighbor 10.0.9.7;
}
group p1 {
    type external;
    export ebgp-out;
    neighbor 10.0.5.254 {
        peer-as 65050;
    }
}
```

The highlighted EBGP portion of the configuration reveals a rather vanilla setup. There is no EBGP import policy in place, and a single export policy called, conveniently enough, *ebgp-out* has been applied. You display the *ebgp-out* policy to determine the expected EBGP advertisement behavior from r1 to P1:

```
[edit]
lab@r1# show policy-options policy-statement ebgp-out
term 1 {
    from {
        protocol aggregate;
        route-filter 10.0.0.0/16 exact;
    }
    then accept;
}
term 2 {
    from community transit;
    then reject;
}
[edit]
lab@r1# show routing-options aggregate
route 10.0.0.0/16;

[edit]
lab@r1# show policy-options community transit
members 65412:420;
```

The first term in the policy results in the advertisement of a locally defined aggregate route encompassing the addressing space of your AS; the aggregate route is also confirmed as present and active on r1 (not shown). The second term in the *ebgp-out* policy serves to block the advertisement of routes with the *transit* community attached. With the default policy left in place for all remaining BGP routes, you expect to see r1 advertise all remaining (and active) BGP routes to the P1 router. Assuming for the moment that the routes learned from transit peer T1 are being tagged with the *transit* community, you expect to see your AS's 10.0/16 aggregate, the data center routes, and both sets of customer routes being sent to P1.

A series of commands are now issued at r1 to confirm the advertisement of the expected routes to P1. These commands also serve to provide an ongoing check of the overall operations of your test bed, as the lack of advertisement for a given set of EBGP routes may constitute cause for further investigation:

```
lab@r1> show route advertising-protocol bgp 10.0.5.254 10.0/16

inet.0: 118092 destinations, 118107 routes (118092 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
* 10.0.0.0/16             Self                                    I
```

The aggregate for your AS is correctly being advertised to P1. This should allow P1 to respond to pings and traceroutes issued from within your AS.

The presence of a locally defined 10.0/16 aggregate is not causing reachability problems on r1 and r2 due to the presence of network summary (LSA Type 3) in their stub area. If network summaries were blocked by the area's ABRs, this aggregate definition would result in a black hole for internal destinations outside of area 1. This situation was documented, and solved, in the *JNCIP Study Guide* (Sybex, 2003).

The next command confirms that the data center routes are being advertised to P1:

```
lab@r1> show route advertising-protocol bgp 10.0.5.254 192.168.0/22

inet.0: 118092 destinations, 118107 routes (118092 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
* 192.168.0.0/24          Self                                    I
* 192.168.0.1/32          Self                                    I
* 192.168.1.0/24          Self                                    I
* 192.168.2.0/24          Self                                    I
* 192.168.3.0/24          Self                                    I
```

The next set of commands confirms that both sets of customer routes are being sent to P1:

```
lab@r1> show route advertising-protocol bgp 10.0.5.254 200.200/16

inet.0: 118093 destinations, 118108 routes (118093 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
* 200.200.0.0/16          Self                                    65010 I

lab@r1> show route advertising-protocol bgp 10.0.5.254 220.220/16

inet.0: 118094 destinations, 118109 routes (118094 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
* 220.220.0.0/16          Self                                    65020 I
```

The output (or lack thereof) from the last command in this series confirms that the 130.130/16 routes, as received from EBGP peer T1, are not being sent to the P1 router in accordance with the *ebgp-out* export policy's rejection of routes with the *transit* community:

```
lab@r1> show route advertising-protocol bgp 10.0.5.254 130.130/16
```

The results shown here indicate that all is well with the r1-P1 EBGP peering session and its related routing policy. Although not shown here, the same verification steps are also performed on r2 and similar results are obtained. These findings confirm that EBGP peering to the P1 router is operational.

T1 Peering

You next analyze the EBGP peering session to the T1 router using an approach similar to that demonstrated for the P1 peering session. Once again, you begin by verifying the EBGP session status to T1:

```
[edit]
lab@r3# run show bgp summary
Groups: 2 Peers: 7 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
inet.0            125079     125067          0          0          0          0
Peer         AS    InPkt OutPkt OutQ Flaps Last Up/Dwn State|#Active/Received/Damped...
172.16.0.14  65222  23868  24684  0     0     1:35:16 125064/125064/0     0/0/0
10.0.3.4     65412  214    24730  0     0     1:46:51 1/1/0               0/0/0
10.0.3.5     65412  215    24748  0     0     1:46:49 0/0/0               0/0/0
10.0.6.1     65412  215    24765  0     0     1:46:59 1/1/0               0/0/0
10.0.6.2     65412  215    24765  0     0     1:46:55 0/1/0               0/0/0
10.0.9.6     65412  218    24765  0     0     1:46:54 1/6/0               0/0/0
10.0.9.7     65412  217    24765  0     0     1:46:58 0/6/0               0/0/0
```

The highlighted entry confirms that the EBGP session between `r3` and P1 has been correctly established, and that some 125,000 routes have been received over this peering session.

As was the case with the JNCIP examination, making "simple" mistakes when you are dealing with a full BGP routing table can have a significant impact on your network's health and general state of well-being. Extra care should be taken when BGP-related redistribution policies are placed into service with this many routes floating about!

Displaying the EBGP-related configuration on `r3` reveals the following settings:

```
[edit]
lab@r3# show protocols bgp
advertise-inactive;
group int {
    type internal;
    local-address 10.0.3.3;
    export nhs;
    neighbor 10.0.6.1;
    neighbor 10.0.6.2;
    neighbor 10.0.3.4;
    neighbor 10.0.3.5;
    neighbor 10.0.9.6;
```

```
    neighbor 10.0.9.7;
}
group ext {
    import ebgp-in;
    export ebgp-out;
    neighbor 172.16.0.14 {
        peer-as 65222;
    }
}
```

The highlighted entries represent another rather basic EBGP peering configuration. Worth noting is the use of `advertise-inactive` to allow the export of the data center routes despite the fact that the routes are active as OSPF routes. Using this option avoids the need for some type of OSPF-to-EBGP export policy for the data center's routes. You also note the presence of group-level import and export policy, the contents of which are displayed next:

```
[edit]
lab@r3# show policy-options policy-statement ebgp-in
term 1 {
    from {
        protocol bgp;
        neighbor 172.16.0.14;
    }
    then {
        community add transit;
    }
}
[edit]
lab@r3# show policy-options community transit
members 65412:420;

[edit]
lab@r3# show policy-options policy-statement ebgp-out
term 1 {
    from {
        protocol aggregate;
        route-filter 10.0.0.0/16 exact;
    }
    then accept;
}
```

The *ebgp-in* policy functions to tag routes received from the T1 peer with the *transit* community. Recall that `r1` and `r2` are filtering routes with this community when sending EBGP

updates to the P1 router. The *ebgp-out* policy causes the advertisement of a locally defined aggregate route representing your AS's addressing space. Based on these findings, you can conclude that all active BGP routes will be sent from r3 to T1, as well as a locally defined 10.0/16 aggregate route. Further, the presence of advertise-inactive will result in the advertisement of the best BGP routes that are currently not active due to protocol preference, which means that in this case, the 192.168.0/22 data center routes should also be advertised to the T1 router.

As with the P1 peering, you now issue a series of **show route advertising-protocol bgp** commands to confirm if r3's EBGP route advertisements to T1 match your predictions:

```
lab@r3> show route advertising-protocol bgp 172.16.0.14 10.0/16

inet.0: 118054 destinations, 118076 routes (118054 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
* 10.0.0.0/16             Self                                    I

lab@r3> show route advertising-protocol bgp 172.16.0.14 120.120/16

inet.0: 125150 destinations, 125164 routes (125150 active, 0 holddown, 0 hidden)
  Prefix          Nexthop        MED  Lclpref  AS path
* 120.120.0.0/16  Self                        65050 I                 65050 I

lab@r3> show route advertising-protocol bgp 172.16.0.14 192.168.0/22

inet.0: 118054 destinations, 118076 routes (118054 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
  192.168.0.0/24          Self                                    I
  192.168.0.1/32          Self                                    I
  192.168.1.0/24          Self                                    I
  192.168.2.0/24          Self                                    I
  192.168.3.0/24          Self                                    I
```

The output from the commands confirm all your predictions regarding the EBGP advertisement behavior at the r3–T1 EBGP peering. Note that the 192.168.0/22 data center routes are being advertised despite the lack of active route indication (there is no * next to them). Though not shown, you may assume that the 200.200/16 and 220.220/16 routes, owned by C1 and C2 respectively, have also been confirmed in r3's EBGP advertisements to the T1 peer. These results indicate that the r3–T1 EBGP peering session is working as expected.

Customer Peering

The next check of your network's EBGP and routing policy operation involves the discovery and verification of the EBGP peering to customer sites. In this example, the EBGP configuration

and routing policy configurations for both customer sites are virtually identical, so discovery and verification steps will be demonstrated only for the C1 peering points at r4 and r7.

r7 to C1 EBGP Peering

You begin your customer peering analysis and discovery with router r7, with the confirmation that the r7–C1 peering session is in the established state:

```
[edit]
lab@r7# run show bgp summary
Groups: 2 Peers: 6 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
inet.0            118032     118022          0          0          0          0
Peer         AS    InPkt OutPkt OutQ Flaps Last Up/Dwn State|#Active/Received/Damped...
172.16.0.2   65010   51182  26385  0     0     2:19:24 1/1/0               0/0/0
10.0.3.3     65412   26308    278  0     0     2:16:29 118012/118012/0     0/0/0
10.0.3.4     65412     274    277  0     0     2:16:22 0/1/0               0/0/0
10.0.3.5     65412     274    277  0     0     2:16:10 0/0/0               0/0/0
10.0.6.1     65412     275    277  0     0     2:16:23 1/1/0               0/0/0
10.0.6.2     65412     275    277  0     0     2:16:12 0/1/0               0/0/0
```

With r7's EBGP session to C1 confirmed as operational, you move on to the inspection of r7's EBGP configuration:

```
[edit]
lab@r7# show protocols bgp
group int {
    type internal;
    local-address 10.0.9.7;
    export nhs;
    neighbor 10.0.6.1;
    neighbor 10.0.6.2;
    neighbor 10.0.3.3;
    neighbor 10.0.3.4;
    neighbor 10.0.3.5;
}
group c1 {
    type external;
    export ebgp-out;
    neighbor 172.16.0.2 {
        peer-as 65010;
    }
}
```

Nothing of note here, except the presence of an *ebgp-out* export policy, the contents of which are displayed next:

```
[edit]
lab@r7# show policy-options policy-statement ebgp-out
term 1 {
    from {
        protocol aggregate;
        route-filter 10.0.0.0/16 exact;
    }
    then accept;
}
term 2 {
    from {
        route-filter 192.168.0.0/22 upto /24;
    }
    then accept;
}
```

The first term in r7's *ebgp-out* export policy functions to advertise a local 10.0/16 aggregate to EBGP peer C1. As with the routers in area 1, the presence of the local aggregate does not cause operational problems in area 2 due to the presence of network summaries (LSA Type 3s). The second policy term results in the redistribution of the data center routes from IS-IS into EBGP. Though not shown in this capture, you should recall that r6 and r7 also redistribute the same routes into IBGP so that r1 and r2 can in turn advertise the DC routes to the P1 router.

The analysis of r7's EBGP peering configuration indicates that C1 should be receiving the 10.0/16 aggregate, the 192.168.0/22 data center routes, C2's routes, T1's routes, and the routes learned from the P1 router. The same set of commands demonstrated for the T1 and P1 peering points are now issued to confirm your analysis. Although not shown here, you can assume that in this example all expected routes are confirmed as present in r7's EBGP advertisements to the C1 router.

r4 to C1 EBGP Peering

You now shift your attention to the C1 peering point at r4. After verifying that the EBGP session is established (not shown), you move onto the inspection of r4's EBGP configuration and routing policy:

```
[edit]
lab@r4# show protocols bgp
advertise-inactive;
group int {
    type internal;
    local-address 10.0.3.4;
```

```
    export nhs;
    neighbor 10.0.6.1;
    neighbor 10.0.6.2;
    neighbor 10.0.3.3;
    neighbor 10.0.3.5;
    neighbor 10.0.9.6;
    neighbor 10.0.9.7;
}
group c1 {
    type external;
    export ebgp-out;
    neighbor 172.16.0.6 {
        peer-as 65010;
    }
}
```

The highlighted entries in the output relate to the C1 EBGP peering, and are virtually identical to the settings shown for `r3`. Once again, the `advertise-inactive` option is being used to allow the export of the data center routes via EBGP when the BGP versions of these routes are not active due to global preference settings. The *`ebgp-out`* policy is now displayed:

```
[edit]
lab@r4# show policy-options policy-statement ebgp-out
term 1 {
    from {
        protocol aggregate;
        route-filter 10.0.0.0/16 exact;
    }
    then accept;
}
```

Based on the contents of the *`ebgp-out`* policy, you conclude that `r4` will advertise the same set of routes to C1 as was described for the `r7`–C1 peering. You now issue a series of **`show route advertising-protocol bgp`** commands on `r4` to confirm the advertisement of the 10.0/16 aggregate, the data center's 192.168.0/22 routes, and the routes learned from the T1, P1, and C2 EBGP peerings:

```
lab@r4> show route advertising-protocol bgp 172.16.0.6 10.0/16

inet.0: 125146 destinations, 125160 routes (125146 active, 5 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
* 10.0.0.0/16             Self                                    I
lab@r4> show route advertising-protocol bgp 172.16.0.6 192.168.0/22

inet.0: 125147 destinations, 125161 routes (125147 active, 5 holddown, 0 hidden)
```

```
  Prefix                  Nexthop              MED     Lclpref    AS path
  192.168.0.0/24          Self                                    I
  192.168.0.1/32          Self                                    I
  192.168.1.0/24          Self                                    I
  192.168.2.0/24          Self                                    I
  192.168.3.0/24          Self                                    I
```

This output confirms that the 10.0/16 aggregate and data center routes are correctly advertised from r4 to C1. The next set of commands verifies the remaining routes, all of which have been learned from the various EBGP peerings in your baseline network:

```
lab@r4> show route advertising-protocol bgp 172.16.0.6 130.130/16

inet.0: 125146 destinations, 125160 routes (125146 active, 5 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
* 130.130.0.0/16          Self                                    65222 I

lab@r4> show route advertising-protocol bgp 172.16.0.6 120.120/16

inet.0: 125146 destinations, 125160 routes (125146 active, 5 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
* 120.120.0.0/16          Self                                    65050 I

lab@r4> show route advertising-protocol bgp 172.16.0.6 220.220/16

inet.0: 125147 destinations, 125161 routes (125147 active, 5 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
* 220.220.0.0/16          Self                                    65020 I
```

The results indicate that the r4–C1 EBGP peering and routing policies are fully operational.

Final EBGP and Policy Checks

Before blessing the EBGP and policy operation of the baseline network that you have been lucky enough to inherit, it is a good idea to check for hidden routes and to confirm reachability and forwarding paths to all EBGP peers. You really should inspect all routers in the test bed for hidden routes but, because r5 has no EBGP peerings, any problems with next hop reachability will most likely manifest themselves at r5. The following command is used to determine hidden route status at r5:

```
[edit]
lab@r5# run show route hidden

inet.0: 125144 destinations, 125158 routes (125144 active, 0 holddown, 0 hidden)

[edit]
```

The lack of output from r5 indicates that none of the 125,000 or so routes that it has received are hidden. The absence of hidden routes provides an additional indication that your network's EBGP, IBGP, and IGP protocols are functioning correctly. You now issue a series of traceroute commands from r5 to verify external prefix reachability and to validate the forwarding paths to external destinations:

```
lab@r5> traceroute 120.120.0.1
traceroute to 120.120.0.1 (120.120.0.1), 30 hops max, 40 byte packets
 1  10.0.2.10 (10.0.2.10)  0.994 ms  0.765 ms  0.629 ms
 2  10.0.4.10 (10.0.4.10)  0.533 ms  0.529 ms  0.491 ms
 3  120.120.0.1 (120.120.0.1)  0.641 ms  0.610 ms  0.580 ms

lab@r5> traceroute 130.130.0.1
traceroute to 130.130.0.1 (130.130.0.1), 30 hops max, 40 byte packets
 1  10.0.2.2 (10.0.2.2)  1.295 ms  1.029 ms  1.136 ms
 2  130.130.0.1 (130.130.0.1)  1.078 ms  1.024 ms  1.171 ms

lab@r5> traceroute 200.200.0.1
traceroute to 200.200.0.1 (200.200.0.1), 30 hops max, 40 byte packets
 1  10.0.2.10 (10.0.2.10)  0.834 ms  0.680 ms  0.603 ms
 2  200.200.0.1 (200.200.0.1)  0.532 ms  0.540 ms  0.504 ms

lab@r5> traceroute 220.220.0.1
traceroute to 220.220.0.1 (220.220.0.1), 30 hops max, 40 byte packets
 1  10.0.8.5 (10.0.8.5)  0.724 ms  0.535 ms  0.464 ms
 2  220.220.0.1 (220.220.0.1)  0.575 ms  0.586 ms  0.543 ms
```

The traceroute commands all succeed, which provides confirmation that all EBGP peers are receiving the 10.0/16 aggregate for your AS. The indication that packets take optimal forwarding paths to external destinations provides further validation that all aspects of your baseline network are now operational. Before moving on to the first configuration scenario, it is advisable that you repeat your traceroutes testing from the data center router, taking care to source the packets from one of its 192.168.0/22 prefixes, as doing so will validate the operation of the default route used by the data center router while also confirming that all EBGP peers are receiving advertisements for the data center's routes. Although not shown, you can assume that all traceroute testing from the data center router succeeds in this example.

Summary of EBGP and Policy Discovery

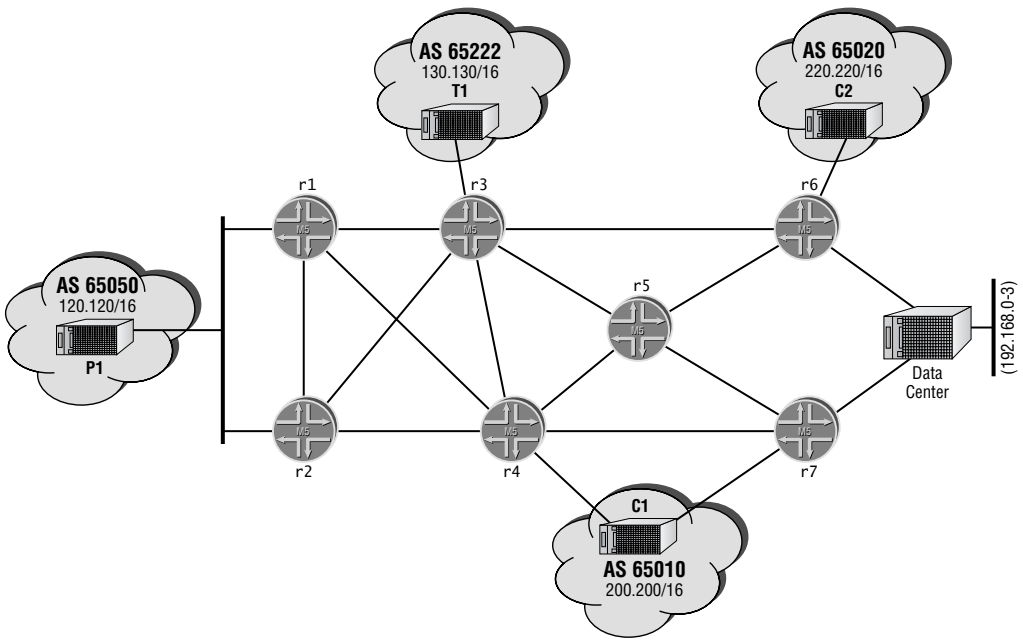
Once again, it is suggested that you take a few moments to document the results of your network discovery for future reference. After all, trying to lay down advanced services such as MPLS on top of a network that you are not intimately familiar with is akin to running with scissors, only more dangerous. Being able to jog your memory with the notes and documentation you make during a discovery scenario can make all the difference in later configuration tasks. A

summary of your IBGP, EBGP, and BGP-related routing policy is provided here:

- Full IBGP mesh between loopback addresses with all IBGP sessions established.
- Next hop self-policies on `r3`, `r4`, `r6`, and `r7`. Not needed on `r1` and `r2`.
- Data center routes redistributed into IBGP at `r6` and `r7`.
- All EBGP sessions established with no hidden routes.
- All active BGP routes being sent to all peers, with the exception of transit routes, which are not advertised to the P1 router.
- Local 10.0/16 aggregate advertised to all peers.
- Data center routes advertised to all peers; using `advertise-inactive` at `r3` and `r4`.
- No Martian filtering is in place.
- Connectivity and forwarding paths confirmed to all EBGP peers.

Figure 1.4 details your BGP-related findings in the context of a simplified topology map.

FIGURE 1.4 EBGP and policy discovery example



Notes:

Full IBGP mesh, all IBGP sessions established. EBGP peering to physical addresses, all EGBP sessions established.

10.0/16 aggregate, and data center routes confirmed to all EBGP peers. Local 10.0/16 aggregate is not black holing due to the presence of network summaries in all areas.

All active BGP routes sent to all EBGP peers, except T1 routes, which are tagged with a transit community and filtered from P1 at `r1` and `r2`.

Advertise inactive at `r3` and `r4`. `r6` and `r7` redistributing data center routes into both IGP and IBGP.

No operational issues detected. Trace routes to EGBP peers are successful and follow optimal paths. No hidden routes detected.

Complete Configurations for OSPF Baseline Network

Listings 1.1 through 1.7 provide the complete baseline configurations for all seven routers in the test bed as they existed at the conclusion of the network discovery and validation techniques demonstrated in the body of this chapter. You might need to modify the specifics to suit your hardware environment before loading the configurations into your test bed, but try to maintain as much similarity as possible. The baseline configuration will serve as the building block for the advanced topics covered in later chapters.

Listing 1.1: *r1* OSPF Baseline Configuration

```
[edit]
lab@r1# show | no-more
version 5.6R1.3;
system {
    host-name r1;
    authentication-order [ radius password ];
    ports {
        console type vt100;
    }
    root-authentication {
        encrypted-password "$1$RTyGDGYG$ukqr37VGRgtohedSlruOk/"; # SECRET-DATA
    }
    radius-server {
        10.0.1.201 secret "$9$jvkmT69pRhrz3hrev7Nik."; # SECRET-DATA
    }
    login {
        user lab {
            uid 2000;
            class superuser;
            authentication {
                encrypted-password "$1$L6ZKKWYI$GxEI/7YzXes2JXDcHJvz7/";
                   # SECRET-DATA
            }
        }
    }
    services {
        ssh;
        telnet;
    }
    syslog {
```

```
        user * {
            any emergency;
        }
        file messages {
            any notice;
            authorization info;
        }
        file r1-cli {
            interactive-commands any;
            archive files 5;
        }
    }
}
interfaces {
    fe-0/0/0 {
        unit 0 {
            family inet {
                address 10.0.5.1/24;
            }
        }
    }
    fe-0/0/1 {
        unit 0 {
            family inet {
                address 10.0.4.14/30;
            }
        }
    }
    fe-0/0/2 {
        unit 0 {
            family inet {
                address 10.0.4.5/30;
            }
        }
    }
    fe-0/0/3 {
        unit 0 {
            family inet {
                address 10.0.4.18/30;
            }
        }
    }
```

```
    fxp0 {
        unit 0 {
            family inet {
                address 10.0.1.1/24;
            }
        }
    }
    lo0 {
        unit 0 {
            family inet {
                address 10.0.6.1/32;
            }
        }
    }
}
routing-options {
    static {
        route 10.0.200.0/24 {
            next-hop 10.0.1.102;
            no-readvertise;
        }
    }
    aggregate {
        route 10.0.0.0/16;
    }
    autonomous-system 65412;
}
protocols {
    bgp {
        group int {
            type internal;
            local-address 10.0.6.1;
            neighbor 10.0.6.2;
            neighbor 10.0.3.3;
            neighbor 10.0.3.4;
            neighbor 10.0.3.5;
            neighbor 10.0.9.6;
            neighbor 10.0.9.7;
        }
        group p1 {
            type external;
            export ebgp-out;
```

```
            neighbor 10.0.5.254 {
                peer-as 65050;
            }
        }
    }
    ospf {
        area 0.0.0.1 {
            stub;
            interface fe-0/0/0.0 {
                passive;
            }
            interface fe-0/0/1.0;
            interface fe-0/0/2.0;
            interface fe-0/0/3.0;
        }
    }
}
policy-options {
    policy-statement ebgp-out {
        term 1 {
            from {
                protocol aggregate;
                route-filter 10.0.0.0/16 exact;
            }
            then accept;
        }
        term 2 {
            from community transit;
            then reject;
        }
    }
    community transit members 65412:420;
}
```

Listing 1.2: *r2* OSPF Baseline Configuration

```
[edit]
lab@r2# show | no-more
version 5.6R1.3;
system {
    host-name r2;
    authentication-order [ radius password ];
    ports {
```

```
        console type vt100;
    }
    root-authentication {
        encrypted-password "$1$RTyGDGYG$ukqr37VGRgtohedSlruOk/"; # SECRET-DATA
    }
    radius-server {
        10.0.1.201 secret "$9$jvkmT69pRhrz3hrev7Nik."; # SECRET-DATA
    }
    login {
        user lab {
            uid 2000;
            class superuser;
            authentication {
                encrypted-password "$1$L6ZKKWYI$GxEI/7YzXes2JXDcHJvz7/";
                   # SECRET-DATA
            }
        }
    }
    services {
        ssh;
        telnet;
    }
    syslog {
        user * {
            any emergency;
        }
        file messages {
            any notice;
            authorization info;
        }
        file r2-cli {
            interactive-commands any;
            archive files 5;
        }
    }
}
interfaces {
    fe-0/0/0 {
        unit 0 {
            family inet {
                address 10.0.5.2/24;
```

```
            }
        }
    }
    fe-0/0/1 {
        unit 0 {
            family inet {
                address 10.0.4.10/30;
            }
        }
    }
    fe-0/0/2 {
        speed 100m;
        unit 0 {
            family inet {
                address 10.0.4.2/30;
            }
        }
    }
    fe-0/0/3 {
        unit 0 {
            family inet {
                address 10.0.4.6/30;
            }
        }
    }
    fxp0 {
        unit 0 {
            family inet {
                address 10.0.1.2/24;
            }
        }
    }
    lo0 {
        unit 0 {
            family inet {
                address 10.0.6.2/32;
            }
        }
    }
}
```

```
routing-options {
    static {
        route 10.0.200.0/24 {
            next-hop 10.0.1.102;
            no-readvertise;
        }
    }
    aggregate {
        route 10.0.0.0/16;
    }
    autonomous-system 65412;
}
protocols {
    bgp {
        group int {
            type internal;
            local-address 10.0.6.2;
            neighbor 10.0.6.1;
            neighbor 10.0.3.3;
            neighbor 10.0.3.4;
            neighbor 10.0.3.5;
            neighbor 10.0.9.6;
            neighbor 10.0.9.7;
        }
        group p1 {
            type external;
            export ebgp-out;
            neighbor 10.0.5.254 {
                peer-as 65050;
            }
        }
    }
    ospf {
        area 0.0.0.1 {
            stub;
            interface fe-0/0/0.0 {
                passive;
            }
            interface fe-0/0/1.0;
            interface fe-0/0/2.0;
```

```
            interface fe-0/0/3.0;
        }
    }
}
policy-options {
    policy-statement ebgp-out {
        term 1 {
            from {
                protocol aggregate;
                route-filter 10.0.0.0/16 exact;
            }
            then accept;
        }
        term 2 {
            from community transit;
            then reject;
        }
    }
    community transit members 65412:420;
}
```

Listing 1.3: *r3* OSPF Baseline Configuration (with Highlighted Corrections)

```
[edit]
lab@r3# show | no-more
version 5.6R1.3;
system {
    host-name r3;
    authentication-order [ radius password ];
    ports {
        console type vt100;
    }
    root-authentication {
        encrypted-password "$1$RTyGDGYG$ukqr37VGRgtohedSlruOk/"; # SECRET-DATA
    }
    radius-server {
        10.0.1.201 secret "$9$jvkmT69pRhrz3hrev7Nik."; # SECRET-DATA
    }
    login {
        user lab {
            uid 2000;
            class superuser;
```

```
            authentication {
                encrypted-password "$1$L6ZKKWYI$GxEI/7YzXes2JXDcHJvz7/";
                   # SECRET-DATA
            }
        }
    }
    services {
        ssh;
        telnet;
    }
    syslog {
        user * {
            any emergency;
        }
        file messages {
            any notice;
            authorization info;
        }
        file r3-cli {
            interactive-commands any;
            archive files 5;
        }
    }
}
interfaces {
    fe-0/0/0 {
        unit 0 {
            family inet {
                address 10.0.4.13/30;
            }
        }
    }
    fe-0/0/1 {
        unit 0 {
            family inet {
                address 10.0.4.1/30;
            }
        }
    }
    fe-0/0/2 {
```

```
        unit 0 {
            family inet {
                address 172.16.0.13/30;
            }
        }
    }
    fe-0/0/3 {
        unit 0 {
            family inet {
                address 10.0.2.14/30;
            }
        }
    }
    at-0/1/0 {
        atm-options {
            vpi 0 {
                maximum-vcs 64;
            }
        }
        unit 0 {
            point-to-point;
            vci 50;
            family inet {
                address 10.0.2.2/30;
            }
        }
    }
    so-0/2/0 {
        dce;
        encapsulation frame-relay;
        unit 100 {
            dlci 100;
            family inet {
                address 10.0.2.5/30;
            }
        }
    }
    fxp0 {
        unit 0 {
            family inet {
                address 10.0.1.3/24;
```

```
            }
        }
    }
    lo0 {
        unit 0 {
            family inet {
                address 10.0.3.3/32;
            }
        }
    }
}
routing-options {
    static {
        route 10.0.200.0/24 {
            next-hop 10.0.1.102;
            no-readvertise;
        }
    }
    aggregate {
        route 10.0.0.0/16;
    }
    autonomous-system 65412;
}
protocols {
    bgp {
        advertise-inactive;
        group int {
            type internal;
            local-address 10.0.3.3;
            export nhs;
            neighbor 10.0.6.1;
            neighbor 10.0.6.2;
            neighbor 10.0.3.4;
            neighbor 10.0.3.5;
            neighbor 10.0.9.6;
            neighbor 10.0.9.7;
        }
        group ext {
            import ebgp-in;
            export ebgp-out;
            neighbor 172.16.0.14 {
```

```
                peer-as 65222;
            }
        }
    }
    ospf {
        area 0.0.0.1 {
            stub default-metric 10;
            interface fe-0/0/0.0;
            interface fe-0/0/1.0;
        }
        area 0.0.0.0 {
            interface so-0/2/0.100;
            interface at-0/1/0.0;
        }
        area 0.0.0.2 {
            nssa {
                default-lsa default-metric 10;
            }
            interface fe-0/0/3.0;
        }
    }
}
policy-options {
    policy-statement nhs {
        term 1 {
            from {
                protocol bgp;
                neighbor 172.16.0.14;
            }
            then {
                next-hop self;
            }
        }
    }
    policy-statement ebgp-out {
        term 1 {
            from {
                protocol aggregate;
                route-filter 10.0.0.0/16 exact;
            }
            then accept;
```

```
        }
    }
    policy-statement ebgp-in {
        term 1 {
            from {
                protocol bgp;
                neighbor 172.16.0.14;
            }
            then {
                community add transit;
            }
        }
    }
    community transit members 65412:420;
}
```

Note that r3, r4, and r5 had their configurations modified (as highlighted) to resolve a problem with a missing default route in area 2.

Listing 1.4: *r4* OSPF Baseline Configuration (with Highlighted Corrections)

```
[edit]
lab@r4# show | no-more
version 5.6R1.3;
system {
    host-name r4;
    authentication-order [ radius password ];
    ports {
        console type vt100;
    }
    root-authentication {
        encrypted-password "$1$RTyGDGYG$ukqr37VGRgtohedSlruOk/"; # SECRET-DATA
    }
    radius-server {
        10.0.1.201 secret "$9$jvkmT69pRhrz3hrev7Nik."; # SECRET-DATA
    }
    login {
        user lab {
            uid 2000;
            class superuser;
            authentication {
                encrypted-password "$1$L6ZKKWYI$GxEI/7YzXes2JXDcHJvz7/";
                   # SECRET-DATA
            }
```

```
        }
    }
    services {
        ssh;
        telnet;
    }
    syslog {
        user * {
            any emergency;
        }
        file messages {
            any notice;
            authorization info;
        }
        file r4-cli {
            interactive-commands any;
            archive files 5;
        }
    }
}
interfaces {
    fe-0/0/0 {
        unit 0 {
            family inet {
                address 172.16.0.5/30;
            }
        }
    }
    fe-0/0/1 {
        unit 0 {
            family inet {
                address 10.0.4.9/30;
            }
        }
    }
    fe-0/0/2 {
        unit 0 {
            family inet {
                address 10.0.4.17/30;
            }
        }
```

```
    }
    fe-0/0/3 {
        unit 0 {
            family inet {
                address 10.0.2.18/30;
            }
        }
    }
    so-0/1/0 {
        encapsulation frame-relay;
        unit 100 {
            dlci 100;
            family inet {
                address 10.0.2.6/30;
            }
        }
    }
    so-0/1/1 {
        encapsulation ppp;
        unit 0 {
            family inet {
                address 10.0.2.10/30;
            }
        }
    }
    fxp0 {
        unit 0 {
            family inet {
                address 10.0.1.4/24;
            }
        }
    }
    lo0 {
        unit 0 {
            family inet {
                address 10.0.3.4/32;
            }
        }
    }
}
routing-options {
```

```
    static {
        route 10.0.200.0/24 {
            next-hop 10.0.1.102;
            no-readvertise;
        }
    }
    aggregate {
        route 10.0.0.0/16;
    }
    autonomous-system 65412;
}
protocols {
    bgp {
        advertise-inactive;
        group int {
            type internal;
            local-address 10.0.3.4;
            export nhs;
            neighbor 10.0.6.1;
            neighbor 10.0.6.2;
            neighbor 10.0.3.3;
            neighbor 10.0.3.5;
            neighbor 10.0.9.6;
            neighbor 10.0.9.7;
        }
        group c1 {
            type external;
            export ebgp-out;
            neighbor 172.16.0.6 {
                peer-as 65010;
            }
        }
    }
    ospf {
        area 0.0.0.1 {
            stub default-metric 10;
            interface fe-0/0/1.0;
            interface fe-0/0/2.0;
        }
        area 0.0.0.0 {
```

```
            interface so-0/1/0.100;
            interface so-0/1/1.0;
        }
        area 0.0.0.2 {
            nssa {
                default-lsa default-metric 10;
            }
            interface fe-0/0/3.0;
        }
    }
}
policy-options {
    policy-statement ebgp-out {
        term 1 {
            from {
                protocol aggregate;
                route-filter 10.0.0.0/16 exact;
            }
            then accept;
        }
    }
    policy-statement nhs {
        term 1 {
            from {
                protocol bgp;
                neighbor 172.16.0.6;
            }
            then {
                next-hop self;
            }
        }
    }
}
```

Note that r3, r4, and r5 had their configurations modified (as highlighted) to resolve a problem with a missing default route in area 2.

Listing 1.5: *r5* OSPF Baseline Configuration (with Highlighted Corrections)

```
lab@r5# show | no-more
version 5.6R1.3;
system {
    host-name r5;
```

```
    authentication-order [ radius password ];
    ports {
        console type vt100;
    }
    root-authentication {
        encrypted-password "$1$RTyGDGYG$ukqr37VGRgtohedSlruOk/"; # SECRET-DATA
    }
    radius-server {
        10.0.1.201 secret "$9$jvkmT69pRhrz3hrev7Nik."; # SECRET-DATA
    }
    login {
        user lab {
            uid 2000;
            class superuser;
            authentication {
                encrypted-password "$1$L6ZKKWYI$GxEI/7YzXes2JXDcHJvz7/";
                   # SECRET-DATA
            }
        }
    }
    services {
        ssh;
        telnet;
    }
    syslog {
        user * {
            any emergency;
        }
        file messages {
            any notice;
            authorization info;
        }
        file r5-cli {
            interactive-commands any;
            archive files 5;
        }
    }
}
interfaces {
    fe-0/0/0 {
        unit 0 {
```

```
        family inet {
            address 10.0.8.6/30;
        }
    }
}
fe-0/0/1 {
    unit 0 {
        family inet {
            address 10.0.8.9/30;
        }
    }
}
so-0/1/0 {
    encapsulation ppp;
    unit 0 {
        family inet {
            address 10.0.2.9/30;
        }
    }
}
at-0/2/1 {
    atm-options {
        vpi 0 {
            maximum-vcs 64;
        }
    }
    unit 0 {
        point-to-point;
        vci 50;
        family inet {
            address 10.0.2.1/30;
        }
    }
}
fxp0 {
    unit 0 {
        family inet {
            address 10.0.1.5/24;
        }
    }
}
```

```
    lo0 {
        unit 0 {
            family inet {
                address 10.0.3.5/32;
            }
        }
    }
}
routing-options {
    static {
        route 10.0.200.0/24 {
            next-hop 10.0.1.102;
            no-readvertise;
        }
    }
    autonomous-system 65412;
}
protocols {
    bgp {
        group int {
            type internal;
            local-address 10.0.3.5;
            neighbor 10.0.6.1;
            neighbor 10.0.6.2;
            neighbor 10.0.3.3;
            neighbor 10.0.3.4;
            neighbor 10.0.9.6;
            neighbor 10.0.9.7;
        }
    }
    ospf {
        area 0.0.0.0 {
            interface at-0/2/1.0;
            interface so-0/1/0.0;
        }
        area 0.0.0.2 {
            nssa {
                default-lsa default-metric 10;
            }
            interface fe-0/0/0.0;
```

```
            interface fe-0/0/1.0;
        }
    }
}
```

Note that r3, r4, and r5 had their configurations modified (as highlighted) to resolve a problem with a missing default route in area 2.

Listing 1.6: *r6* OSPF Baseline Configuration

```
[edit]
lab@r6# show | no-more
version 5.6R1.3;
system {
    host-name r6;
    authentication-order [ radius password ];
    ports {
        console type vt100;
    }
    root-authentication {
        encrypted-password "$1$RTyGDGYG$ukqr37VGRgtohedSlruOk/"; # SECRET-DATA
    }
    radius-server {
        10.0.1.201 secret "$9$jvkmT69pRhrz3hrev7Nik."; # SECRET-DATA
    }
    login {
        user lab {
            uid 2000;
            class superuser;
            authentication {
                encrypted-password "$1$L6ZKKWYI$GxEI/7YzXes2JXDcHJvz7/";
                   # SECRET-DATA
            }
        }
    }
    services {
        ssh;
        telnet;
    }
    syslog {
        user * {
            any emergency;
        }
```

```
        file messages {
            any notice;
            authorization info;
        }
        file r6-cli {
            interactive-commands any;
            archive files 5;
        }
    }
}
interfaces {
    fe-0/1/0 {
        unit 0 {
            family inet {
                address 10.0.8.5/30;
            }
        }
    }
    fe-0/1/1 {
        unit 0 {
            family inet {
                address 10.0.2.13/30;
            }
        }
    }
    fe-0/1/2 {
        unit 0 {
            family inet {
                address 10.0.8.2/30;
            }
            family iso;
        }
    }
    fe-0/1/3 {
        unit 0 {
            family inet {
                address 172.16.0.9/30;
            }
        }
    }
    fxp0 {
```

```
        unit 0 {
            family inet {
                address 10.0.1.6/24;
            }
        }
    }
    lo0 {
        unit 0 {
            family inet {
                address 10.0.9.6/32;
            }
            family iso {
                address 49.0002.6666.6666.6666.00;
            }
        }
    }
}
routing-options {
    static {
        route 10.0.200.0/24 {
            next-hop 10.0.1.102;
            no-readvertise;
        }
    }
    aggregate {
        route 10.0.0.0/16;
    }
    autonomous-system 65412;
}
protocols {
    bgp {
        group int {
            type internal;
            local-address 10.0.9.6;
            export ibgp;
            neighbor 10.0.6.1;
            neighbor 10.0.6.2;
            neighbor 10.0.3.3;
            neighbor 10.0.3.4;
            neighbor 10.0.3.5;
            neighbor 10.0.9.7;
```

```
        }
        group c2 {
            type external;
            export ebgp-out;
            neighbor 172.16.0.10 {
                peer-as 65020;
            }
        }
    }
    isis {
        export ospf-isis;
        level 2 disable;
        level 1 external-preference 149;
        interface fe-0/1/2.0;
        interface lo0.0;
    }
    ospf {
        export isis-ospf;
        area 0.0.0.2 {
            nssa;
            interface fe-0/1/0.0;
            interface fe-0/1/2.0 {
                passive;
            }
            interface fe-0/1/1.0;
        }
    }
}
policy-options {
    policy-statement ospf-isis {
        term 1 {
            from {
                protocol ospf;
                route-filter 0.0.0.0/0 exact;
            }
            then accept;
        }
    }
    policy-statement isis-ospf {
        term 1 {
```

```
            from {
                protocol isis;
                route-filter 192.168.0.0/22 longer;
            }
            then accept;
        }
    }
    policy-statement ebgp-out {
        term 1 {
            from {
                protocol aggregate;
                route-filter 10.0.0.0/16 exact;
            }
            then accept;
        }
        term 2 {
            from {
                route-filter 192.168.0.0/22 upto /24;
            }
            then accept;
        }
    }
    policy-statement ibgp {
        term 1 {
            from {
                protocol bgp;
                neighbor 172.16.0.10;
            }
            then {
                next-hop self;
            }
        }
        term 2 {
            from {
                route-filter 192.168.0.0/22 longer;
            }
            then accept;
        }
    }
}
```

Listing 1.7: *r7* OSPF Baseline Configuration

```
[edit]
lab@r7# show | no-more
version 5.6R1.3;
system {
    host-name r7;
    authentication-order [ radius password ];
    ports {
        console type vt100;
    }
    root-authentication {
        encrypted-password "$1$RTyGDGYG$ukqr37VGRgtohedSlruOk/"; # SECRET-DATA
    }
    radius-server {
        10.0.1.201 secret "$9$jvkmT69pRhrz3hrev7Nik."; # SECRET-DATA
    }
    login {
        user lab {
            uid 2000;
            class superuser;
            authentication {
                encrypted-password "$1$L6ZKKWYI$GxEI/7YzXes2JXDcHJvz7/";
                   # SECRET-DATA
            }
        }
    }
    services {
        ssh;
        telnet;
    }
    syslog {
        user * {
            any emergency;
        }
        file messages {
            any notice;
            authorization info;
        }
        file r7-cli {
            interactive-commands any;
            archive files 5;
```

```
        }
    }
}
interfaces {
    fe-0/3/0 {
        unit 0 {
            family inet {
                address 10.0.8.14/30;
            }
            family iso;
        }
    }
    fe-0/3/1 {
        unit 0 {
            family inet {
                address 10.0.8.10/30;
            }
        }
    }
    fe-0/3/2 {
        unit 0 {
            family inet {
                address 172.16.0.1/30;
            }
        }
    }
    fe-0/3/3 {
        unit 0 {
            family inet {
                address 10.0.2.17/30;
            }
        }
    }
    fxp0 {
        unit 0 {
            family inet {
                address 10.0.1.7/24;
            }
        }
    }
    lo0 {
```

```
        unit 0 {
            family inet {
                address 10.0.9.7/32;
            }
            family iso {
                address 49.0002.7777.7777.7777.00;
            }
        }
    }
}
routing-options {
    static {
        route 10.0.200.0/24 {
            next-hop 10.0.1.102;
            no-readvertise;
        }
    }
    aggregate {
        route 10.0.0.0/16;
    }
    autonomous-system 65412;
}
protocols {
    bgp {
        group int {
            type internal;
            local-address 10.0.9.7;
            export nhs;
            neighbor 10.0.6.1;
            neighbor 10.0.6.2;
            neighbor 10.0.3.3;
            neighbor 10.0.3.4;
            neighbor 10.0.3.5;
            neighbor 10.0.9.6;
        }
        group c1 {
            type external;
            export ebgp-out;
            neighbor 172.16.0.2 {
                peer-as 65010;
            }
```

```
        }
    }
    isis {
        export ospf-isis;
        level 2 disable;
        level 1 external-preference 149;
        interface fe-0/3/0.0;
        interface lo0.0;
    }
    ospf {
        export isis-ospf;
        area 0.0.0.2 {
            nssa;
            interface fe-0/3/1.0;
            interface fe-0/3/0.0 {
                passive;
            }
            interface fe-0/3/3.0;
        }
    }
}
policy-options {
    policy-statement ospf-isis {
        term 1 {
            from {
                protocol ospf;
                route-filter 0.0.0.0/0 exact;
            }
            then accept;
        }
    }
    policy-statement isis-ospf {
        term 1 {
            from {
                protocol isis;
                route-filter 192.168.0.0/22 longer;
            }
            then accept;
        }
    }
    policy-statement ebgp-out {
```

```
        term 1 {
            from {
                protocol aggregate;
                route-filter 10.0.0.0/16 exact;
            }
            then accept;
        }
        term 2 {
            from {
                route-filter 192.168.0.0/22 upto /24;
            }
            then accept;
        }
    }
    policy-statement nhs {
        term 1 {
            from {
                protocol bgp;
                neighbor 172.16.0.2;
            }
            then {
                next-hop self;
            }
        }
        term 2 {
            from {
                route-filter 192.168.0.0/22 longer;
            }
            then accept;
        }
    }
}
```

Summary

This chapter provided you with a set of network discovery and verification tasks representative of those encountered by candidates as they begin their JNCIE examination. Because all of the topics addressed in this chapter were thoroughly tested in the prerequisite JNCIP examination, a JNCIE candidate is provided with a preconfigured network intended to serve as the starting point for the advanced configuration aspects that are the focus of the JNCIE examination proper. The goal of the network discovery scenario is twofold. The primary purpose is to provide

the candidate with a chance to assimilate and understand the operation of a network that he or she has never seen, before the candidate is asked to begin adding advanced services to the test bed. The secondary goal of the network discovery scenario is to provide a sanity check of the overall operation of the baseline network on the off chance that hardware errors (or configuration errors) are present.

Note that the specific nature of a given discovery scenario may vary over time to keep things interesting. The bottom line is that the prepared JNCIE candidate will possess the skills and protocol understanding needed to quickly reverse engineer an inherited network, and to rapidly access its overall operational state. It is expected that a JNCIE candidate will be able to detect and repair any configuration problems that may be uncovered during network discovery, with the exception of hardware-related failures that require proctor intervention.

Case Study: IS-IS Network Discovery and Validation Techniques

This case study presents a sample IGP discovery and validation scenario designed to demonstrate techniques and commands that are particularly useful when reverse engineering and validating an IS-IS based IGP. The focus of the case study is placed on IGP discovery because the functionality of IBGP, EBGP, and BGP-related routing policy is largely independent of the particular IGP in use. In fact, very few changes have been made to the IBGP, EBGP, and policy configuration documented in the course of the chapter body. Please refer back to Figure 1.2 for the case study topology.

The criteria for the case study are as follows:

- Discover and document the configuration of an IS-IS based network with mutual route redistribution.
- Verify correct IGP operation while documenting and correcting any IGP-related configuration problems encountered.
- Confirm that IBGP, EBGP, and routing policy are operational.

Although a number of discovery approaches can be used to reverse engineer an IGP, this author has found that it is normally more expedient to begin your analysis in the IGP core. The commands and techniques shown here are similar to the approach demonstrated in the chapter body; after all, if something works, why mess with it? You begin on `r3` with the inspection of its IGP-related configuration:

```
[edit]
lab@r3# show protocols
bgp {
    advertise-inactive;
    group int {
        type internal;
        local-address 10.0.3.3;
```

```
        export nhs;
        neighbor 10.0.6.1;
        neighbor 10.0.6.2;
        neighbor 10.0.3.4;
        neighbor 10.0.3.5;
        neighbor 10.0.9.6;
        neighbor 10.0.9.7;
    }
    group ext {
        import ebgp-in;
        export ebgp-out;
        neighbor 172.16.0.14 {
            peer-as 65222;
        }
    }
}
isis {
    interface fe-0/0/0.0 {
        level 2 disable;
    }
    interface fe-0/0/1.0 {
        level 2 disable;
    }
    interface fe-0/0/3.0 {
        level 1 disable;
    }
    interface at-0/1/0.0 {
        level 1 disable;
    }
    interface so-0/2/0.100 {
        level 1 disable;
    }
    interface lo0.0 {
        level 1 disable;
    }
}
```

The highlighted IGP portion of the configuration leads to the following observations:

- That `r3` is an attached router with a mix of L1 and L2 interfaces
- That authentication and route leaking are not configured for its L1 and L2 areas
- That `r3`'s lo0 address will not be injected into the Level 1 area due to the lo0 interface being disabled at IS-IS Level 1

Displaying the IS-IS interface status on r3 yields the following output:

```
[edit]
lab@r3# run show isis interface
IS-IS interface database:
Interface             L CirID Level 1 DR       Level 2 DR       L1/L2 Metric
at-0/1/0.0            2   0x1 Disabled         Point to Point        10/10
fe-0/0/0.0            1   0x3 r1.02            Disabled              10/10
fe-0/0/1.0            1   0x4 r2.03            Disabled              10/10
fe-0/0/3.0            2   0x2 Disabled         r6.03                 10/10
lo0.0                 0   0x1 Disabled         Passive                0/0
so-0/2/0.100          2   0x1 Disabled         Point to Point        10/10
```

The display indicates that all of r3's interfaces, except its fxp0- and T1-facing fe-0/0/2 interfaces are considered ISO interfaces. This confirms that the iso family has been properly configured on their various logical units. Note that the lo0 interface is also listed as a passive IS-IS Level 2 interface; with some JUNOS software versions, it is critical that you actually run IS-IS on the interface that serves as the source of the ISO NET, making the presence of lo0 in the IS-IS interface display a good thing. The omission of r3's lo0 interface from the Level 1 area is intentional in this case. The intent is to prevent extra hops (through the Level 1 area) when r4 forwards packets to r3's lo0 address. Keeping the lo0 addresses of r3 and r4 from the Level 1 area is necessary if we want r3 and r4 to use their Level 2 link when forwarding to each other's loopback addresses, because an IS-IS router will always prefer a L1 internal route over the same route in Level 2, regardless of preference settings or route metrics. Traceroute testing confirms that r3 and r4 have optimal forwarding between loopback addresses as a result of this configuration:

```
[edit]
lab@r3# run traceroute 10.0.3.4
traceroute to 10.0.3.4 (10.0.3.4), 30 hops max, 40 byte packets
 1  10.0.3.4 (10.0.3.4)  1.061 ms  0.890 ms  0.795 ms
```

You next display r3's lo0 interface configuration. The output displays the ISO NET configured for r3. Based on the display, you are able to determine that r1, r2, r3, and r4 should all be in area 0001, because an IS-IS L1 adjacency will form only between routers that share a common area ID. You also note that the SYS-ID coding is based on the assigned router number. This coding approach should not pose a problem because the resulting SYS-IDs will have the uniqueness required for proper IS-IS operation:

```
[edit]
lab@r3# show interfaces lo0
unit 0 {
    family inet {
        address 10.0.3.3/32;
    }
    family iso {
```

```
        address 49.0001.3333.3333.3333.00;
    }
}
```

With the inspection of r3's IGP configuration completed, it makes sense to ascertain the state of its adjacencies, as you now have some idea of what to expect:

```
[edit]
lab@r3# run show isis adjacency
Interface             System         L State       Hold (secs)  SNPA
at-0/1/0.0            r5             2 Up                   21
fe-0/0/0.0            r1             1 Up                    6  0:a0:c9:6f:7b:3e
fe-0/0/1.0            r2             1 Up                    7  0:a0:c9:6f:7a:ff
fe-0/0/3.0            r6             2 Up                    7  0:d0:b7:3f:af:73
so-0/2/0.100          r4             2 Up                   21
```

The output shows that r3 has the five adjacencies one would expect, given the test bed topology and r3's IS-IS configuration. The output makes it easy to confirm what interfaces are running IS-IS, and at what IS-IS level. The following command quickly determines if the backbone area is correctly receiving LSPs from all of the routers in the test bed:

```
[edit]
lab@r3# run show isis hostname
IS-IS hostname database:
System ID      Hostname                                        Type
1111.1111.1111 r1                                              Dynamic
2222.2222.2222 r2                                              Dynamic
3333.3333.3333 r3                                              Static
4444.4444.4444 r4                                              Dynamic
5555.5555.5555 r5                                              Dynamic
6666.6666.6666 r6                                              Dynamic
7777.7777.7777 r7                                              Dynamic
```

The results could not be any better! You know that IS-IS is working overall, in that the backbone has received LSPs from all routers in the test bed. The final check at r3 determines correct IS-IS functionality with regard to the advertisement of IP routes by verifying that IS-IS routes for the loopback addresses of all routers in the test bed are present:

```
[edit]
lab@r3# run show route protocol isis | match \32
10.0.3.4/32        *[IS-IS/15] 02:49:44, metric 20
10.0.3.5/32        *[IS-IS/18] 00:07:08, metric 20
10.0.6.1/32        *[IS-IS/15] 03:11:24, metric 10
10.0.6.2/32        *[IS-IS/15] 02:49:44, metric 10
10.0.9.6/32        *[IS-IS/18] 00:33:06, metric 10
10.0.9.7/32        *[IS-IS/18] 00:07:08, metric 20
```

The results confirm that r3 has learned routes through IS-IS for the loopback addresses of all remote routers (r3's loopback address is not learned through IS-IS, and is therefore not listed). You can assume that similar results are obtained when the same commands are issued on r4. A quick look at r2's IS-IS configuration returns the following:

```
[edit]
lab@r2# show protocols isis
level 2 disable;
interface fe-0/0/0.0 {
    passive;
}
interface fe-0/0/1.0;
interface fe-0/0/2.0;
interface fe-0/0/3.0;
interface lo0.0;
```

r2's IS-IS configuration is pretty basic; the only real item to note is the fact that the router is running a passive IS-IS instance on its fe-0/0/0 interface. As with the OSPF example in the chapter body, the passive interface setting ensures that the 10.0.5/25 subnet will be reachable as an IS-IS internal route without chancing IGP adjacency formation to peer P1. Although not shown, the same passive interface observation is made when inspecting r1's IS-IS stanza. You next confirm IS-IS adjacency status, at r2:

```
[edit]
lab@r2# run show isis adjacency
Interface             System         L State       Hold (secs)  SNPA
fe-0/0/1.0            r4             1 Up                   21  0:90:69:6b:30:1
fe-0/0/2.0            r3             1 Up                   23  0:90:69:6d:98:1
fe-0/0/3.0            r1             1 Up                    6  0:a0:c9:6f:7b:84
```

With r2 displaying the expected number and types of IS-IS adjacencies, things are looking good for IS-IS functionality in area 0001. You decide to shift your discovery activities to r5 as a result:

```
lab@r5# show
export l1-l2;
interface fe-0/0/0.0 {
    level 2 disable;
}
interface fe-0/0/1.0 {
    level 2 disable;
}
interface so-0/1/0.0 {
    level 1 {
        passive;
```

```
    }
}
interface at-0/2/1.0 {
    level 1 {
        passive;
    }
}
interface lo0.0;
```

As with r3 and r4, r5's configuration indicates that it is a L1/L2 (attached) router by virtue of the mix of L1 and L2 interface statements in its IS-IS stanza. Unlike r3 and r4, r5 is running IS-IS (passively) at both Level 1 and Level 2 on its lo0 interface; this will result in the advertisement of its lo0 address in both its L1 and L2 LSPs. The passive configuration on r5's core facing interfaces prevents inefficient routing in area 0002, by having r5 advertise the 10.0.2.0/30 and 10.0.2.8/30 prefixes in the L1 LSP it sends into area 0002. Disabling Level 1 on r5's core facing interfaces will result in r6 and r7 incurring extra hops when forwarding to these prefixes, as their only routes to these destinations would be learned through the L2 LSPs generated by r3 and r4, respectively. A quick traceroute at r7 confirms proper forwarding paths to core prefixes:

```
[edit]
lab@r7# run traceroute 10.0.2.1
traceroute to 10.0.2.1 (10.0.2.1), 30 hops max, 40 byte packets
 1  10.0.2.1 (10.0.2.1)  0.758 ms  0.558 ms  0.454 ms

[edit]
lab@r7# run traceroute 10.0.2.9
traceroute to 10.0.2.9 (10.0.2.9), 30 hops max, 40 byte packets
 1  10.0.2.9 (10.0.2.9)  0.644 ms  0.489 ms  0.436 ms
```

The presence of an *l1-l2* export policy is also noted and the contents displayed:

```
lab@r5# show policy-options policy-statement l1-l2
term 1 {
    from {
        protocol isis;
        level 1;
        route-filter 192.168.0.0/22 longer;
    }
    to level 2;
    then accept;
}
```

The *l1-l2* policy instructs r5 to leak L1 external routes matching the route filter statement into the backbone (Level 2) area. Note that L1 internals are leaked into L2 areas by default, but policy is needed for the leaking of L1 externals. From this, you surmise that the routes

associated with the data center router are being redistributed by r6 and r7 into IS-IS as Level 1 externals. You next display r5's IS-IS interface status along with its ISO NET:

```
[edit]
lab@r5# run show isis interface
IS-IS interface database:
Interface             L CirID Level 1 DR        Level 2 DR        L1/L2 Metric
at-0/2/1.0            2   0x1 Passive           Point to Point         10/10
fe-0/0/0.0            1   0x2 r6.02             Disabled               10/10
fe-0/0/1.0            1   0x3 r5.03             Disabled               10/10
lo0.0                 0   0x1 Passive           Passive                 0/0
so-0/1/0.0            2   0x1 Passive           Point to Point         10/10

[edit]
lab@r5# show interfaces lo0
unit 0 {
    family inet {
        address 10.0.3.5/32;
    }
    family iso {
        address 49.0002.5555.5555.5555.00;
    }
}
```

Based on the display, you conclude that r5, r6, and r7 are attached to IS-IS area 49.0002, and that r5 should have L2 adjacencies on its core facing interfaces and L1 adjacencies on the Fast Ethernet links to r6 and r7. The IS-IS adjacency status is now verified on r5:

```
[edit]
lab@r5# run show isis adjacency
Interface           System       L State      Hold (secs)      SNPA
at-0/2/1.0          r3           2 Up                  21
fe-0/0/0.0          r6           1 Up                   6      0:d0:b7:3f:af:f
fe-0/0/1.0          r7           1 Up                  21      0:60:94:51:c4:27
so-0/1/0.0          r4           2 Up                  26
```

The results confirm that r5 has the expected number of adjacencies (4), and further validates that area 0002 is a Level 1 area. With r5's configuration and operation looking good, your attention shifts to r7:

```
[edit]
lab@r7# show protocols
bgp {
    group int {
        type internal;
```

```
        local-address 10.0.9.7;
        export nhs;
        neighbor 10.0.6.1;
        neighbor 10.0.6.2;
        neighbor 10.0.3.3;
        neighbor 10.0.3.4;
        neighbor 10.0.3.5;
        neighbor 10.0.9.6;
    }
    group c1 {
        type external;
        export ebgp-out;
        neighbor 172.16.0.2 {
            peer-as 65010;
        }
    }
}
isis {
    export rip-isis;
    interface fe-0/3/0.0 {
        level 2 disable;
        level 1 passive;
    }
    interface fe-0/3/1.0 {
        level 2 disable;
    }
    interface fe-0/3/3.0 {
        level 1 disable;
    }
    interface lo0.0;
}
rip {
    group dc {
        export static-rip;
        neighbor fe-0/3/0.0;
    }
}
```

The highlighted output indicates that `r7` is running both IS-IS and RIP. From this, you surmise that the data center router must now be configured to advertise the 192.168.0/22 routes to `r6` and `r7` using RIP. The *`rip-isis`* export policy is now displayed. The output

confirms that r7 is configured to redistribute the data center's routes from RIP into IS-IS:

```
[edit]
lab@r7# show policy-options policy-statement rip-isis
term 1 {
    from {
        protocol rip;
        route-filter 192.168.0.0/22 longer;
    }
    then accept;
}
```

Displaying r7's *isis-rip* policy tells you that r7 should be sending a statically defined default route to the data center router:

```
[edit]
lab@r7# show policy-options policy-statement static-rip
term 1 {
    from {
        protocol static;
        route-filter 0.0.0.0/0 exact;
    }
    then accept;
}
```

The static route definition is also confirmed:

```
[edit]
lab@r7# show routing-options static
route 10.0.200.0/24 {
    next-hop 10.0.1.102;
    no-readvertise;
}
route 0.0.0.0/0 reject;
```

Why a Static Default Route?

The astute reader is likely wondering why a static default route has been defined on r6 and r7, especially considering that a Level 1 router normally installs an IS-IS based default route when the presence of an Attached router is detected in a Level 1 area through the setting of the Attached bit in Level 1 LSPs. The key here is the observation that area 0002 has *only* Attached routers, in that r5, r6, and r7 are all L2 and L1 Attached. Because an Attached router will not install a default route based on the presence of the Attached bit in the LSPs received from other Attached routers, a static default (or generated route) route is defined on r6 and r7.

Now that you know what to expect, you confirm r7's adjacency status and the presence of the data center's route RIP routes:

```
[edit]
lab@r7# run show isis adjacency
Interface             System         L State       Hold (secs)   SNPA
fe-0/3/1.0            r5               1 Up                 7    0:90:69:69:70:1
fe-0/3/3.0            r4               2 Up                 8    0:90:69:6b:30:3
```

Good! All of the expected adjacencies are up. You move on to confirm the presence of RIP routes at r7:

```
[edit]
lab@r7# run show route 192.168.0/22

inet.0: 118137 destinations, 118138 routes (118137 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

192.168.0.0/24     *[RIP/100] 01:16:43, metric 2, tag 0
                    > to 10.0.8.13 via fe-0/3/0.0
192.168.1.0/24     *[RIP/100] 01:16:43, metric 2, tag 0
                    > to 10.0.8.13 via fe-0/3/0.0
192.168.2.0/24     *[RIP/100] 01:16:43, metric 2, tag 0
                    > to 10.0.8.13 via fe-0/3/0.0
192.168.3.0/24     *[RIP/100] 01:16:43, metric 2, tag 0
                    > to 10.0.8.13 via fe-0/3/0.0
```

The output from r7 confirms it is correctly learning the 192.168.0/22 DC routes through the RIP protocol. Though the results are not shown here, you can assume that similar results were obtained when inspecting r6. All of the results obtained thus far in your IS-IS IGP discovery case study have been positive. You should now be able to document your IGP discovery findings on a copy of the test bed topology. However, before calling it quits, you wisely opt to further validate the operation of your IGP through some traceroute testing:

```
[edit]
lab@r6# run traceroute 10.0.3.4
traceroute to 10.0.3.4 (10.0.3.4), 30 hops max, 40 byte packets
 1  10.0.2.14 (10.0.2.14)  0.437 ms  0.352 ms  0.270 ms
 2  10.0.3.4 (10.0.3.4)  0.508 ms  0.464 ms  0.436 ms

[edit]
lab@r6# run traceroute 10.0.3.3
traceroute to 10.0.3.3 (10.0.3.3), 30 hops max, 40 byte packets
 1  10.0.3.3 (10.0.3.3)  0.592 ms  0.461 ms  0.422 ms
```

The traceroutes to attached routers r3 and r4 succeed, but things are not so positive for the traceroute to r1:

```
[edit]
lab@r6# run traceroute 10.0.6.1
traceroute to 10.0.6.1 (10.0.6.1), 30 hops max, 40 byte packets
 1  10.0.2.14 (10.0.2.14)  0.390 ms  0.287 ms  0.243 ms
 2  * * *
 3  * * *
 4  * * *
 5  * * *
 6  * * *
```

The timeouts are not expected, so you move to r3, which is the first and only hop shown in the traceroute, to determine what is amiss:

```
[edit]
lab@r3# run traceroute 10.0.6.1
traceroute to 10.0.6.1 (10.0.6.1), 30 hops max, 40 byte packets
 1  10.0.6.1 (10.0.6.1)  0.743 ms  0.553 ms  0.475 ms
```

The traceroute from r3 to r1 is successful, so your attention shifts to r1 itself:

```
[edit]
lab@r1# run traceroute 10.0.3.5
traceroute to 10.0.3.5 (10.0.3.5), 30 hops max, 40 byte packets
traceroute: sendto: No route to host
 1 traceroute: wrote 10.0.3.5 40 chars, ret=-1
^C
[edit]
lab@r1# run show route 10.0.3.5

inet.0: 18 destinations, 19 routes (18 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.0.0/16         *[Aggregate/130] 00:13:15
                      Reject
```

D'oh! The local definition of a 10.0/16 aggregate has created a black hole on r1 and r2 for all 10.0/16 destinations outside of their Level 1 area. Note that the presence of the same 10.0/16 aggregate route had no impact on the operation of the OSPF IGP, as demonstrated in the chapter body, because neither the stub nor the NSSA areas were blocking network summaries. Recalling that an IS-IS L1 area functions much as an OSPF stub/NSSA area with no-summaries, the problems caused by the local aggregate make perfect sense. This problem is not occurring on r6 and r7, because they are attached routers with full routing knowledge of the IS-IS domain.

After noting the problem and obtaining permission from the proctor to make baseline configuration changes, you decide to correct the issue by adjusting the IBGP export policy on r3

and r4 to effect the advertisement of the 10.0/16 aggregate to r1 and r2 through IBGP. You must be careful that your policy advertises a next hop for the aggregate route that is *within* L1 area 0001. The default behavior will be to set the route's next hop to the lo0 address of the advertising router, which will cause the 10.0/16 route to be hidden on r1 and r2 due to their inability to resolve the advertised BGP next hop (10.0.3.3 or 10.0.3.4) through the 10.0/16 route itself. (A BGP route cannot have its next hop resolved through itself because this can result in recursion problems.)

Advertising the aggregate to r1 and r2 through IBGP allows for the removal of the local 10.0/16 aggregate route definition from r1 and r2, while still providing them with the ability to advertise the 10.0/16 route to their EBGP peer P1. The highlighted entries show the modifications that were made to the IBGP portion of r4's configuration to evoke per-neighbor IBGP export policy (similar changes were also made at r3):

```
[edit]
lab@r4# show protocols bgp
advertise-inactive;
group int {
    type internal;
    local-address 10.0.3.4;
    export nhs;
    neighbor 10.0.6.1 {
        export r1;
    }
    neighbor 10.0.6.2 {
        export r2;
    }
    neighbor 10.0.3.3;
    neighbor 10.0.3.5;
    neighbor 10.0.9.6;
    neighbor 10.0.9.7;
}
group c1 {
    type external;
    export ebgp-out;
    neighbor 172.16.0.6 {
        peer-as 65010;
    }
}
```

And the new *r1* and *r2* policies are displayed with the next hop settings for the aggregate route highlighted:

```
[edit]
lab@r4# show policy-options policy-statement r1
```

```
term 1 {
    from {
        protocol aggregate;
        route-filter 10.0.0.0/16 exact;
    }
    then {
        next-hop 10.0.4.17;
        accept;
    }
}
term 2 {
    from {
        protocol bgp;
        neighbor 172.16.0.6;
    }
    then {
        next-hop self;
    }
}

[edit]
lab@r4# show policy-options policy-statement r2
term 1 {
    from {
        protocol aggregate;
        route-filter 10.0.0.0/16 exact;
    }
    then {
        next-hop 10.0.4.9;
        accept;
    }
}
term 2 {
    from {
        protocol bgp;
        neighbor 172.16.0.6;
    }
    then {
```

```
        next-hop self;
    }
}
```

The first term in the *r1* and *r2* export policies results in r3 and r4 advertising their 10.0/16 aggregate to r1 and r2 with the BGP next hop set to an IP address that exists within their Level 1 area. Note that the next hop advertised to r1 differs from that sent to r2 in an effort to help promote optimal forwarding paths wherever possible. Because the next hop for the 10.0/16 now resolves through a more specific route (as opposed to the 10.0/16 route itself), the route is no longer hidden and is therefore eligible for export to the P1 router by r1 and r2. The second policy term functions to set next hop self on the routes being learned from C1 peering. Though not shown, r3 now has similar *r1* and *r2* policies in place.

After deleting the local aggregate at r1 and r2, further testing confirms that all is well:

```
[edit]
lab@r1# run show route 10.0/16

inet.0: 118087 destinations, 118094 routes (118087 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.0.0/16        *[BGP/170] 00:10:31, localpref 100, from 10.0.3.3
                      AS path: I
                    > to 10.0.4.13 via fe-0/0/1.0
                    [BGP/170] 00:10:34, localpref 100, from 10.0.3.4
                      AS path: I
                    > to 10.0.4.17 via fe-0/0/4.0
. . .
```

The 10.0/16 aggregate, as received from r3 and r4, is confirmed with the output just shown. You next verify that the aggregate is being correctly sent on to the P1 router:

```
[edit]
lab@r1# run show route advertising-protocol bgp 10.0.5.254 10.0/16

inet.0: 118167 destinations, 118182 routes (118167 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
* 10.0.0.0/16             Self
```

The output confirms the advertisement of the 10.0/16 aggregate to router P1. The next set of commands verifies reachability and forwarding paths from Level 1 router r1 to various internal and external destinations:

```
[edit]
lab@r1# run traceroute 10.0.9.7
traceroute to 10.0.9.7 (10.0.9.7), 30 hops max, 40 byte packets
 1  10.0.4.13 (10.0.4.13)  0.409 ms  0.341 ms  0.266 ms
```

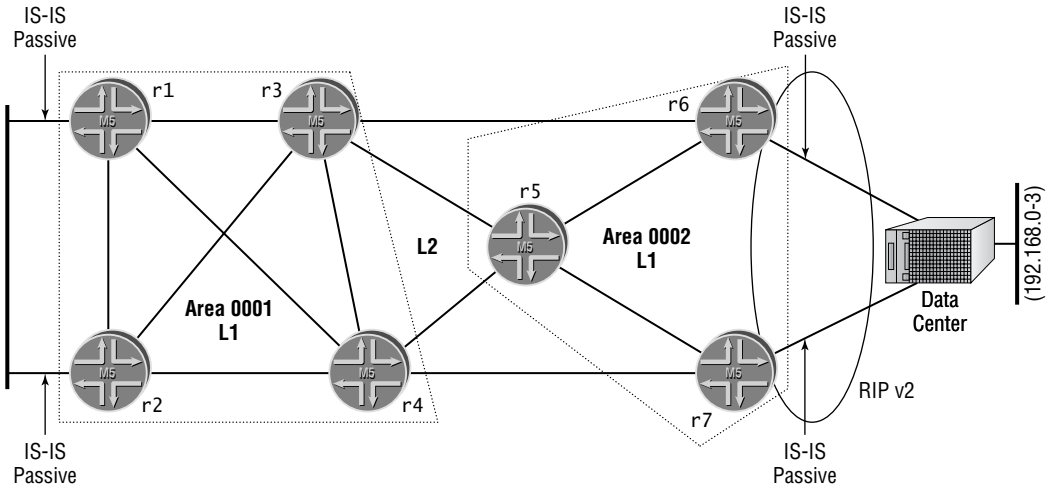
```
 2  10.0.2.1 (10.0.2.1)  0.811 ms  1.054 ms  0.798 ms
 3  10.0.9.7 (10.0.9.7)  0.705 ms  0.648 ms  0.405 ms

[edit]
lab@r1# run traceroute 192.168.0.1
traceroute to 192.168.0.1 (192.168.0.1), 30 hops max, 40 byte packets
 1  10.0.4.13 (10.0.4.13)  0.400 ms  0.293 ms  0.250 ms
 2  10.0.2.13 (10.0.2.13)  0.157 ms  0.153 ms  0.133 ms
 3  192.168.0.1 (192.168.0.1)  0.260 ms  0.231 ms  0.209 ms

[edit]
lab@r1# run traceroute 130.130.0.1
traceroute to 130.130.0.1 (130.130.0.1), 30 hops max, 40 byte packets
 1  10.0.4.13 (10.0.4.13)  0.392 ms  0.289 ms  0.248 ms
 2  130.130.0.1 (130.130.0.1)  0.172 ms  0.166 ms  0.144 ms

[edit]
lab@r1# run traceroute 220.220.0.1
traceroute to 220.220.0.1 (220.220.0.1), 30 hops max, 40 byte packets
 1  10.0.4.13 (10.0.4.13)  0.388 ms  0.300 ms  0.247 ms
 2  10.0.2.13 (10.0.2.13)  0.160 ms  0.152 ms  0.130 ms
 3  220.220.0.1 (220.220.0.1)  0.246 ms  0.228 ms  0.208 ms

[edit]
lab@r1# run traceroute 200.200.0.1
traceroute to 200.200.0.1 (200.200.0.1), 30 hops max, 40 byte packets
 1  10.0.4.13 (10.0.4.13)  0.408 ms  0.291 ms  0.248 ms
 2  10.0.2.6 (10.0.2.6)  0.309 ms  0.276 ms  0.253 ms
 3  200.200.0.1 (200.200.0.1)  0.178 ms  0.180 ms  0.154 ms
```

The forwarding paths shown have all been optimal, with the exception of the extra hop through r3 that occurs when r1 traces routes to C1's destinations. Closer inspection reveals that the extra hop is the result of C1's 200.200/16 route resolving through the 10.0/16 aggregate, coupled with the fact that both r1 and r2 prefer the 10.0/16 advertisement from r3 to that learned from r4, due to r3's lower RID. Because extra hops are sometimes inevitable when relying on aggregate or default routing, this condition is considered par for the course and, other than simply noting the condition, no additional actions are taken. With full reachability and optimal forwarding confirmed to all internal and external destinations, you have finished the validation aspects of the IS-IS based IGP discovery case study.

Although not shown here, you should quickly confirm that all IBGP and EBGP sessions are correctly established, and that no hidden route problems exist, before considering your baseline network operational. You can assume that there are no operational problems in the test bed at this time. To complete the IGP discovery case study, you must document your findings.

Figure 1.5 provides a summary of your IGP discovery case study findings. The figure also notes the operational issues that were discovered, and rectified, in this case study.

FIGURE 1.5 Results from IGP discovery case study



Notes:

Multi-level IS-IS, Areas 0001 and 0002 with ISO NET based on router number.

lo0 address of `r3` and `r4` not injected into Area 0001 to ensure optimal forwarding between 10.0.3.3 and 10.0.3.4.

Passive setting on `r5`'s core interfaces for optimal Area 0002-to-core routing.

No authentication or route summarization. Routing policy at `r5` to leak L1 externals (DC routes) to L2.

Redistribution of static default route to data center from both `r6` and `r7`. Redistribution of 192.168.0/24 through 192.168.3/24 routes from RIP into IS-IS by both `r6` and `r7`.

All adjacencies are up, reachability problem discovered at `r1` and `r2` caused by local aggregate definition. Corrected through IBGP policy to effect 10.0/16 route advertisement from `r3` and `r4` to `r1` and `r2`; removed local aggregate from `r1` and `r2`.

Suboptimal routing detected at the data center and at `r1`/`r2` for some locations. This is the result of random nexthop choice for data center's default, and the result of `r1` and `r2`'s preference for `r3`'s RID over `r4` with regard to the 10.0/16 route. This is considered normal behavior, so no corrective actions are taken.

Network Discovery Case Study Configuration

The complete case study configuration for all routers in the test bed is provided next. To keep things interesting, the configuration examples shown in subsequent chapters may use either the OSPF or the IS-IS baseline configuration. Differences between the chapter body and case study configurations, and any changes needed to provide proper IGP operation, are called out with highlights in Listings 1.8 through 1.14.

Listing 1.8: *r1* IS-IS Baseline Configuration

```
lab@r1# show | no-more
version 5.6R1.3;
system {
```

```
    host-name r1;
    authentication-order [ radius password ];
    ports {
        console type vt100;
    }
    root-authentication {
        encrypted-password "$1$RTyGDGYG$ukqr37VGRgtohedSlruOk/"; # SECRET-DATA
    }
    radius-server {
        10.0.1.201 secret "$9$jvkmT69pRhrz3hrev7Nik."; # SECRET-DATA
    }
    login {
        user lab {
            uid 2000;
            class superuser;
            authentication {
                encrypted-password "$1$L6ZKKWYI$GxEI/7YzXes2JXDcHJvz7/";
                   # SECRET-DATA
            }
        }
    }
    services {
        ssh;
        telnet;
    }
    syslog {
        user * {
            any emergency;
        }
        file messages {
            any notice;
            authorization info;
        }
        file r1-cli {
            interactive-commands any;
            archive files 5;
        }
    }
}
interfaces {
    fe-0/0/0 {
        unit 0 {
```

```
        family inet {
            address 10.0.5.1/24;
        }
        family iso;
    }
}
fe-0/0/1 {
    unit 0 {
        family inet {
            address 10.0.4.14/30;
        }
        family iso;
    }
}
fe-0/0/2 {
    unit 0 {
        family inet {
            address 10.0.4.5/30;
        }
        family iso;
    }
}
fe-0/0/3 {
    unit 0 {
        family inet {
            address 10.0.4.18/30;
        }
        family iso;
    }
}
fxp0 {
    unit 0 {
        family inet {
            address 10.0.1.1/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 10.0.6.1/32;
```

```
            }
            family iso {
                address 49.0001.1111.1111.1111.00;
            }
        }
    }
}
routing-options {
    static {
        route 10.0.200.0/24 {
            next-hop 10.0.1.102;
            no-readvertise;
        }
    }
    autonomous-system 65412;
}
protocols {
    bgp {
        group int {
            type internal;
            local-address 10.0.6.1;
            neighbor 10.0.6.2;
            neighbor 10.0.3.3;
            neighbor 10.0.3.4;
            neighbor 10.0.3.5;
            neighbor 10.0.9.6;
            neighbor 10.0.9.7;
        }
        group p1 {
            type external;
            export ebgp-out;
            neighbor 10.0.5.254 {
                peer-as 65050;
            }
        }
    }
    isis {
        level 2 disable;
        interface fe-0/0/0.0 {
            passive;
        }
```

```
        interface fe-0/0/1.0;
        interface fe-0/0/2.0;
        interface fe-0/0/3.0;
        interface lo0.0;
    }
}
policy-options {
    policy-statement ebgp-out {
        term 1 {
            from {
                protocol aggregate;
                route-filter 10.0.0.0/16 exact;
            }
            then accept;
        }
        term 2 {
            from community transit;
            then reject;
        }
    }
    community transit members 65412:420;
}
```

Note that the 10.0/16 local aggregate has been deleted from the `routing-options` stanza on `r1`, and that the first term in the *`ebgp-out`* policy is no longer needed; the term has been left in place because it is causing no harm.

Listing 1.9: *r2* IS-IS Baseline Configuration

```
[edit]
lab@r2# show | no-more
version 5.6R1.3;
system {
    host-name r2;
    authentication-order [ radius password ];
    ports {
        console type vt100;
    }
    root-authentication {
        encrypted-password "$1$RTyGDGYG$ukqr37VGRgtohedSlruOk/"; # SECRET-DATA
    }
    radius-server {
        10.0.1.201 secret "$9$jvkmT69pRhrz3hrev7Nik."; # SECRET-DATA
    }
```

```
    login {
        user lab {
            uid 2000;
            class superuser;
            authentication {
                encrypted-password "$1$L6ZKKWYI$GxEI/7YzXes2JXDcHJvz7/";
                   # SECRET-DATA
            }
        }
    }
    services {
        ssh;
        telnet;
    }
    syslog {
        user * {
            any emergency;
        }
        file messages {
            any notice;
            authorization info;
        }
        file r2-cli {
            interactive-commands any;
            archive files 5;
        }
    }
}
interfaces {
    fe-0/0/0 {
        unit 0 {
            family inet {
                address 10.0.5.2/24;
            }
            family iso;
        }
    }
    fe-0/0/1 {
        unit 0 {
            family inet {
                address 10.0.4.10/30;
```

```
            }
            family iso;
        }
    }
    fe-0/0/2 {
        speed 100m;
        unit 0 {
            family inet {
                address 10.0.4.2/30;
            }
            family iso;
        }
    }
    fe-0/0/3 {
        unit 0 {
            family inet {
                address 10.0.4.6/30;
            }
            family iso;
        }
    }
    fxp0 {
        unit 0 {
            family inet {
                address 10.0.1.2/24;
            }
        }
    }
    lo0 {
        unit 0 {
            family inet {
                address 10.0.6.2/32;
            }
            family iso {
                address 49.0001.2222.2222.2222.00;
            }
        }
    }
}
routing-options {
    static {
```

```
        route 10.0.200.0/24 {
            next-hop 10.0.1.102;
            no-readvertise;
        }
    }
    autonomous-system 65412;
}
protocols {
    bgp {
        group int {
            type internal;
            local-address 10.0.6.2;
            neighbor 10.0.6.1;
            neighbor 10.0.3.3;
            neighbor 10.0.3.4;
            neighbor 10.0.3.5;
            neighbor 10.0.9.6;
            neighbor 10.0.9.7;
        }
        group p1 {
            type external;
            export ebgp-out;
            neighbor 10.0.5.254 {
                peer-as 65050;
            }
        }
    }
    isis {
        level 2 disable;
        interface fe-0/0/0.0 {
            passive;
        }
        interface fe-0/0/1.0;
        interface fe-0/0/2.0;
        interface fe-0/0/3.0;
        interface lo0.0;
    }
}
policy-options {
    policy-statement ebgp-out {
        term 1 {
```

```
            from {
                protocol aggregate;
                route-filter 10.0.0.0/16 exact;
            }
            then accept;
        }
        term 2 {
            from community transit;
            then reject;
        }
    }
    community transit members 65412:420;
}
```

Note that the 10.0/16 local aggregate has been deleted from the `routing-options` stanza on `r2`, and that the first term in the *`ebgp-out`* policy is no longer needed; the term has been left in place because it is causing no harm.

Listing 1.10: *r3* IS-IS Baseline Configuration

```
[edit]
lab@r3# show | no-more
version 5.6R1.3;
system {
    host-name r3;
    authentication-order [ radius password ];
    ports {
        console type vt100;
    }
    root-authentication {
        encrypted-password "$1$RTyGDGYG$ukqr37VGRgtohedSlruOk/"; # SECRET-DATA
    }
    radius-server {
        10.0.1.201 secret "$9$jvkmT69pRhrz3hrev7Nik."; # SECRET-DATA
    }
    login {
        user lab {
            uid 2000;
            class superuser;
            authentication {
                encrypted-password "$1$L6ZKKWYI$GxEI/7YzXes2JXDcHJvz7/";
                   # SECRET-DATA
            }
```

```
        }
    }
    services {
        ssh;
        telnet;
    }
    syslog {
        user * {
            any emergency;
        }
        file messages {
            any notice;
            authorization info;
        }
        file r3-cli {
            interactive-commands any;
            archive files 5;
        }
    }
}
interfaces {
    fe-0/0/0 {
        unit 0 {
            family inet {
                address 10.0.4.13/30;
            }
            family iso;
        }
    }
    fe-0/0/1 {
        unit 0 {
            family inet {
                address 10.0.4.1/30;
            }
            family iso;
        }
    }
    fe-0/0/2 {
        unit 0 {
            family inet {
```

```
            address 172.16.0.13/30;
        }
    }
}
fe-0/0/3 {
    unit 0 {
        family inet {
            address 10.0.2.14/30;
        }
        family iso;
    }
}
at-0/1/0 {
    atm-options {
        vpi 0 {
            maximum-vcs 64;
        }
    }
    unit 0 {
        point-to-point;
        vci 50;
        family inet {
            address 10.0.2.2/30;
        }
        family iso;
    }
}
so-0/2/0 {
    dce;
    encapsulation frame-relay;
    unit 100 {
        dlci 100;
        family inet {
            address 10.0.2.5/30;
        }
        family iso;
    }
}
fxp0 {
    unit 0 {
        family inet {
```

```
                address 10.0.1.3/24;
            }
        }
    }
    lo0 {
        unit 0 {
            family inet {
                address 10.0.3.3/32;
            }
            family iso {
                address 49.0001.3333.3333.3333.00;
            }
        }
    }
}
routing-options {
    static {
        route 10.0.200.0/24 {
            next-hop 10.0.1.102;
            no-readvertise;
        }
    }
    aggregate {
        route 10.0.0.0/16;
    }
    autonomous-system 65412;
}
protocols {
    bgp {
        advertise-inactive;
        group int {
            type internal;
            local-address 10.0.3.3;
            export nhs;
            neighbor 10.0.6.1 {
                export r1;
            }
            neighbor 10.0.6.2 {
                export r2;
            }
            neighbor 10.0.3.4;
```

```
            neighbor 10.0.3.5;
            neighbor 10.0.9.6;
            neighbor 10.0.9.7;
        }
        group ext {
            import ebgp-in;
            export ebgp-out;
            neighbor 172.16.0.14 {
                peer-as 65222;
            }
        }
    }
    isis {
        interface fe-0/0/0.0 {
            level 2 disable;
        }
        interface fe-0/0/1.0 {
            level 2 disable;
        }
        interface fe-0/0/3.0 {
            level 1 disable;
        }
        interface at-0/1/0.0 {
            level 1 disable;
        }
        interface so-0/2/0.100 {
            level 1 disable;
        }
        interface lo0.0 {
            level 1 disable;
        }
    }
}
policy-options {
    policy-statement nhs {
        term 1 {
            from {
                protocol bgp;
                neighbor 172.16.0.14;
            }
            then {
```

```
            next-hop self;
        }
    }
}
policy-statement ebgp-out {
    term 1 {
        from {
            protocol aggregate;
            route-filter 10.0.0.0/16 exact;
        }
        then accept;
    }
}
policy-statement ebgp-in {
    term 1 {
        from {
            protocol bgp;
            neighbor 172.16.0.14;
        }
        then {
            community add transit;
        }
    }
}
policy-statement r1 {
    term 1 {
        from {
            protocol aggregate;
            route-filter 10.0.0.0/16 exact;
        }
        then {
            next-hop 10.0.4.13;
            accept;
        }
    }
    term 2 {
        from {
            protocol bgp;
            neighbor 172.16.0.14;
        }
        then {
```

```
                next-hop self;
            }
        }
    }
    policy-statement r2 {
        term 1 {
            from {
                protocol aggregate;
                route-filter 10.0.0.0/16 exact;
            }
            then {
                next-hop 10.0.4.1;
                accept;
            }
        }
        term 2 {
            from {
                protocol bgp;
                neighbor 172.16.0.14;
            }
            then {
                next-hop self;
            }
        }
    }
    community transit members 65412:420;
}
```

Note that IBGP export policy changes were made on `r3` to allow advertisement of the 10.0/16 aggregate to `r1` and `r2` through IBGP with the next hop set to an area 0001 address.

Listing 1.11: *r4* IS-IS Baseline Configuration

```
[edit]
lab@r4# show | no-more
version 5.6R1.3;
system {
    host-name r4;
    authentication-order [ radius password ];
    ports {
        console type vt100;
    }
    root-authentication {
        encrypted-password "$1$RTyGDGYG$ukqr37VGRgtohedSlruOk/"; # SECRET-DATA
```

```
    }
    radius-server {
        10.0.1.201 secret "$9$jvkmT69pRhrz3hrev7Nik."; # SECRET-DATA
    }
    login {
        user lab {
            uid 2000;
            class superuser;
            authentication {
                encrypted-password "$1$L6ZKKWYI$GxEI/7YzXes2JXDcHJvz7/";
                   # SECRET-DATA
            }
        }
    }
    services {
        ssh;
        telnet;
    }
    syslog {
        user * {
            any emergency;
        }
        file messages {
            any notice;
            authorization info;
        }
        file r4-cli {
            interactive-commands any;
            archive files 5;
        }
    }
}
interfaces {
    fe-0/0/0 {
        unit 0 {
            family inet {
                address 172.16.0.5/30;
            }
        }
    }
```

```
fe-0/0/1 {
    unit 0 {
        family inet {
            address 10.0.4.9/30;
        }
        family iso;
    }
}
fe-0/0/2 {
    unit 0 {
        family inet {
            address 10.0.4.17/30;
        }
        family iso;
    }
}
fe-0/0/3 {
    unit 0 {
        family inet {
            address 10.0.2.18/30;
        }
        family iso;
    }
}
so-0/1/0 {
    encapsulation frame-relay;
    unit 100 {
        dlci 100;
        family inet {
            address 10.0.2.6/30;
        }
        family iso;
    }
}
so-0/1/1 {
    encapsulation ppp;
    unit 0 {
        family inet {
            address 10.0.2.10/30;
        }
```

```
            family iso;
        }
    }
    fxp0 {
        unit 0 {
            family inet {
                address 10.0.1.4/24;
            }
        }
    }
    lo0 {
        unit 0 {
            family inet {
                address 10.0.3.4/32;
            }
            family iso {
                address 49.0001.4444.4444.4444.00;
            }
        }
    }
}
routing-options {
    static {
        route 10.0.200.0/24 {
            next-hop 10.0.1.102;
            no-readvertise;
        }
    }
    aggregate {
        route 10.0.0.0/16;
    }
    autonomous-system 65412;
}
protocols {
    bgp {
        advertise-inactive;
        group int {
            type internal;
            local-address 10.0.3.4;
            export nhs;
```

```
            neighbor 10.0.6.1 {
                export r1;
            }
            neighbor 10.0.6.2 {
                export r2;
            }
            neighbor 10.0.3.3;
            neighbor 10.0.3.5;
            neighbor 10.0.9.6;
            neighbor 10.0.9.7;
        }
        group c1 {
            type external;
            export ebgp-out;
            neighbor 172.16.0.6 {
                peer-as 65010;
            }
        }
    }
    isis {
        interface fe-0/0/1.0 {
            level 2 disable;
        }
        interface fe-0/0/2.0 {
            level 2 disable;
        }
        interface fe-0/0/3.0 {
            level 1 disable;
        }
        interface so-0/1/0.100 {
            level 1 disable;
        }
        interface so-0/1/1.0 {
            level 1 disable;
        }
        interface lo0.0 {
            level 1 disable;
        }
    }
}
policy-options {
```

```
policy-statement ebgp-out {
    term 1 {
        from {
            protocol aggregate;
            route-filter 10.0.0.0/16 exact;
        }
        then accept;
    }
}
policy-statement nhs {
    term 1 {
        from {
            protocol bgp;
            neighbor 172.16.0.6;
        }
        then {
            next-hop self;
        }
    }
}
policy-statement r1 {
    term 1 {
        from {
            protocol aggregate;
            route-filter 10.0.0.0/16 exact;
        }
        then {
            next-hop 10.0.4.17;
            accept;
        }
    }
    term 2 {
        from {
            protocol bgp;
            neighbor 172.16.0.6;
        }
        then {
            next-hop self;
        }
    }
}
```

```
    policy-statement r2 {
        term 1 {
            from {
                protocol aggregate;
                route-filter 10.0.0.0/16 exact;
            }
            then {
                next-hop 10.0.4.9;
                accept;
            }
        }
        term 2 {
            from {
                protocol bgp;
                neighbor 172.16.0.6;
            }
            then {
                next-hop self;
            }
        }
    }
}
```

Note that IBGP export policy changes were made on `r4` to allow advertisement of the 10.0/16 aggregate to `r1` and `r2` through IBGP with the next hop set to an area 0001 address.

Listing 1.12: *r5* IS-IS Baseline Configuration

```
[edit]
lab@r5# show | no-more
version 5.6R1.3;
system {
    host-name r5;
    authentication-order [ radius password ];
    ports {
        console type vt100;
    }
    root-authentication {
        encrypted-password "$1$RTyGDGYG$ukqr37VGRgtohedSlruOk/"; # SECRET-DATA
    }
    radius-server {
        10.0.1.201 secret "$9$jvkmT69pRhrz3hrev7Nik."; # SECRET-DATA
    }
```

```
    login {
        user lab {
            uid 2000;
            class superuser;
            authentication {
                encrypted-password "$1$L6ZKKWYI$GxEI/7YzXes2JXDcHJvz7/";
                   # SECRET-DATA
            }
        }
    }
    services {
        ssh;
        telnet;
    }
    syslog {
        user * {
            any emergency;
        }
        file messages {
            any notice;
            authorization info;
        }
        file r5-cli {
            interactive-commands any;
            archive files 5;
        }
    }
}
interfaces {
    fe-0/0/0 {
        unit 0 {
            family inet {
                address 10.0.8.6/30;
            }
            family iso;
        }
    }
    fe-0/0/1 {
        unit 0 {
            family inet {
```

```
            address 10.0.8.9/30;
        }
        family iso;
    }
}
so-0/1/0 {
    encapsulation ppp;
    unit 0 {
        family inet {
            address 10.0.2.9/30;
        }
        family iso;
    }
}
at-0/2/1 {
    atm-options {
        vpi 0 {
            maximum-vcs 64;
        }
    }
    unit 0 {
        point-to-point;
        vci 50;
        family inet {
            address 10.0.2.1/30;
        }
        family iso;
    }
}
fxp0 {
    unit 0 {
        family inet {
            address 10.0.1.5/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 10.0.3.5/32;
```

```
            }
            family iso {
                address 49.0002.5555.5555.5555.00;
            }
        }
    }
}
routing-options {
    static {
        route 10.0.200.0/24 {
            next-hop 10.0.1.102;
            no-readvertise;
        }
    }
    autonomous-system 65412;
}
protocols {
    bgp {
        group int {
            type internal;
            local-address 10.0.3.5;
            neighbor 10.0.6.1;
            neighbor 10.0.6.2;
            neighbor 10.0.3.3;
            neighbor 10.0.3.4;
            neighbor 10.0.9.6;
            neighbor 10.0.9.7;
        }
    }
    isis {
        export l1-l2;
        interface fe-0/0/0.0 {
            level 2 disable;
        }
        interface fe-0/0/1.0 {
            level 2 disable;
        }
        interface so-0/1/0.0 {
            level 1 passive;
        }
```

```
        interface at-0/2/1.0 {
            level 1 passive;
        }
        interface lo0.0;
    }
}
policy-options {
    policy-statement l1-l2 {
        term 1 {
            from {
                protocol isis;
                level 1;
                route-filter 192.168.0.0/22 longer;
            }
            to level 2;
            then accept;
        }
    }
}
```

Listing 1.13: *r6* IS-IS Baseline Configuration

```
[edit]
lab@r6# show | no-more
version 5.6R1.3;
system {
    host-name r6;
    authentication-order [ radius password ];
    ports {
        console type vt100;
    }
    root-authentication {
        encrypted-password "$1$RTyGDGYG$ukqr37VGRgtohedSlruOk/"; # SECRET-DATA
    }
    radius-server {
        10.0.1.201 secret "$9$jvkmT69pRhrz3hrev7Nik."; # SECRET-DATA
    }
    login {
        user lab {
            uid 2000;
            class superuser;
            authentication {
```

```
                encrypted-password "$1$L6ZKKWYI$GxEI/7YzXes2JXDcHJvz7/";
                   # SECRET-DATA
            }
        }
    }
    services {
        ssh;
        telnet;
    }
    syslog {
        user * {
            any emergency;
        }
        file messages {
            any notice;
            authorization info;
        }
        file r6-cli {
            interactive-commands any;
            archive files 5;
        }
    }
}
interfaces {
    fe-0/1/0 {
        unit 0 {
            family inet {
                address 10.0.8.5/30;
            }
            family iso;
        }
    }
    fe-0/1/1 {
        unit 0 {
            family inet {
                address 10.0.2.13/30;
            }
            family iso;
        }
    }
```

```
    fe-0/1/2 {
        unit 0 {
            family inet {
                address 10.0.8.2/30;
            }
            family iso;
        }
    }
    fe-0/1/3 {
        unit 0 {
            family inet {
                address 172.16.0.9/30;
            }
        }
    }
    fxp0 {
        unit 0 {
            family inet {
                address 10.0.1.6/24;
            }
        }
    }
    lo0 {
        unit 0 {
            family inet {
                address 10.0.9.6/32;
            }
            family iso {
                address 49.0002.6666.6666.6666.00;
            }
        }
    }
}
routing-options {
    static {
        route 0.0.0.0/0 reject;
        route 10.0.200.0/24 {
            next-hop 10.0.1.102;
            no-readvertise;
        }
    }
```

```
    aggregate {
        route 10.0.0.0/16;
    }
    autonomous-system 65412;
}
protocols {
    bgp {
        group int {
            type internal;
            local-address 10.0.9.6;
            export nhs;
            neighbor 10.0.6.1;
            neighbor 10.0.6.2;
            neighbor 10.0.3.3;
            neighbor 10.0.3.4;
            neighbor 10.0.3.5;
            neighbor 10.0.9.7;
        }
        group c2 {
            type external;
            export ebgp-out;
            neighbor 172.16.0.10 {
                peer-as 65020;
            }
        }
    }
    isis {
        export rip-isis;
        interface fe-0/1/0.0 {
            level 2 disable;
        }
        interface fe-0/1/1.0 {
            level 1 disable;
        }
        interface fe-0/1/2.0 {
            level 2 disable;
            level 1 passive;
        }
        interface lo0.0;
    }
```

```
    rip {
        group dc {
            export static-rip;
            neighbor fe-0/1/2.0;
        }
    }
}
policy-options {
    policy-statement static-rip {
        term 1 {
            from {
                protocol static;
                route-filter 0.0.0.0/0 exact;
            }
            then accept;
        }
    }
    policy-statement rip-isis {
        term 1 {
            from {
                protocol rip;
                route-filter 192.168.0.0/22 longer;
            }
            then accept;
        }
    }
    policy-statement ebgp-out {
        term 1 {
            from {
                protocol aggregate;
                route-filter 10.0.0.0/16 exact;
            }
            then accept;
        }
        term 2 {
            from {
                route-filter 192.168.0.0/22 upto /24;
            }
            then accept;
        }
    }
```

```
    policy-statement nhs {
        term 1 {
            from {
                protocol bgp;
                neighbor 172.16.0.10;
            }
            then {
                next-hop self;
            }
        }
        term 2 {
            from {
                route-filter 192.168.0.0/22 longer;
            }
            then accept;
        }
    }
}
```

Listing 1.14: *r7* IS-IS Baseline Configuration

```
[edit]
lab@r7# show | no-more
version 5.6R1.3;
system {
    host-name r7;
    authentication-order [ radius password ];
    ports {
        console type vt100;
    }
    root-authentication {
        encrypted-password "$1$RTyGDGYG$ukqr37VGRgtohedSlruOk/"; # SECRET-DATA
    }
    radius-server {
        10.0.1.201 secret "$9$jvkmT69pRhrz3hrev7Nik."; # SECRET-DATA
    }
    login {
        user lab {
            uid 2000;
            class superuser;
            authentication {
                encrypted-password "$1$L6ZKKWYI$GxEI/7YzXes2JXDcHJvz7/";
                   # SECRET-DATA
```

```
            }
        }
    }
    services {
        ssh;
        telnet;
    }
    syslog {
        user * {
            any emergency;
        }
        file messages {
            any notice;
            authorization info;
        }
        file r7-cli {
            interactive-commands any;
            archive files 5;
        }
    }
}
interfaces {
    fe-0/3/0 {
        unit 0 {
            family inet {
                address 10.0.8.14/30;
            }
            family iso;
        }
    }
    fe-0/3/1 {
        unit 0 {
            family inet {
                address 10.0.8.10/30;
            }
            family iso;
        }
    }
    fe-0/3/2 {
        unit 0 {
```

```
            family inet {
                address 172.16.0.1/30;
            }
        }
    }
    fe-0/3/3 {
        unit 0 {
            family inet {
                address 10.0.2.17/30;
            }
            family iso;
        }
    }
    fxp0 {
        unit 0 {
            family inet {
                address 10.0.1.7/24;
            }
        }
    }
    lo0 {
        unit 0 {
            family inet {
                address 10.0.9.7/32;
            }
            family iso {
                address 49.0002.7777.7777.7777.00;
            }
        }
    }
}
routing-options {
    static {
        route 0.0.0.0/0 reject;
        route 10.0.200.0/24 {
            next-hop 10.0.1.102;
            no-readvertise;
        }
    }
    aggregate {
```

```
        route 10.0.0.0/16;
    }
    autonomous-system 65412;
}
protocols {
    bgp {
        group int {
            type internal;
            local-address 10.0.9.7;
            export nhs;
            neighbor 10.0.6.1;
            neighbor 10.0.6.2;
            neighbor 10.0.3.3;
            neighbor 10.0.3.4;
            neighbor 10.0.3.5;
            neighbor 10.0.9.6;
        }
        group c1 {
            type external;
            export ebgp-out;
            neighbor 172.16.0.2 {
                peer-as 65010;
            }
        }
    }
    isis {
        export rip-isis;
        interface fe-0/3/0.0 {
            level 2 disable;
            level 1 passive;
        }
        interface fe-0/3/1.0 {
            level 2 disable;
        }
        interface fe-0/3/3.0 {
            level 1 disable;
        }
        interface lo0.0;
    }
    rip {
        group dc {
```

```
            export static-rip;
            neighbor fe-0/3/0.0;
        }
    }
}
policy-options {
    policy-statement static-rip {
        term 1 {
            from {
                protocol static;
                route-filter 0.0.0.0/0 exact;
            }
            then accept;
        }
    }
    policy-statement rip-isis {
        term 1 {
            from {
                protocol rip;
                route-filter 192.168.0.0/22 longer;
            }
            then accept;
        }
    }
    policy-statement ebgp-out {
        term 1 {
            from {
                protocol aggregate;
                route-filter 10.0.0.0/16 exact;
            }
            then accept;
        }
        term 2 {
            from {
                route-filter 192.168.0.0/22 upto /24;
            }
            then accept;
        }
    }
    policy-statement nhs {
```

```
        term 1 {
            from {
                protocol bgp;
                neighbor 172.16.0.2;
            }
            then {
                next-hop self;
            }
        }
        term 2 {
            from {
                route-filter 192.168.0.0/22 longer;
            }
            then accept;
        }
    }
}
```

Spot the Issues: Review Questions

1. Referring back to Figure 1.5, and the configuration snippet below, describe the number and type of adjacencies that you expect to find on r5. You may assume that r3, r4, r6, and r7 have a similar configuration:

```
[edit protocols isis]
lab@r5# show
export l1-l2;
level 1 disable;
interface fe-0/0/0.0;
interface fe-0/0/1.0;
interface so-0/1/0.0;
interface at-0/2/1.0;
interface lo0.0;
```

2. r5 has no IS-IS adjacencies. Can you spot the problems from the following configuration snippets?

```
[edit]
lab@r5# run show isis interface
IS-IS interface database:
Interface       L CirID Level 1 DR         Level 2 DR        L1/L2 Metric
at-0/2/1.0      2   0x1 Disabled           Point to Point         10/10
fe-0/0/0.0      1   0x2 0000.0000.0000.02  Disabled               10/10
fe-0/0/1.0      1   0x3 0000.0000.0000.03  Disabled               10/10
lo0.0           0   0x1 Passive            Passive                 0/0
so-0/1/0.0      2   0x1 Disabled           Point to Point         10/10

[edit]
lab@r5# run show isis adjacency

[edit]
lab@r5# show interfaces
fe-0/0/0 {
    unit 0 {
        family inet {
            address 10.0.8.6/30;
        }
        family iso;
    }
}
```

```
fe-0/0/1 {
    unit 0 {
        family inet {
            address 10.0.8.9/30;
        }
        family iso;
    }
}
so-0/1/0 {
    encapsulation ppp;
    unit 0 {
        family inet {
            address 10.0.2.9/30;
        }
        family iso;
    }
}
at-0/2/1 {
    atm-options {
        vpi 0 {
            maximum-vcs 64;
        }
    }
    unit 0 {
        point-to-point;
        vci 50;
        family inet {
            address 10.0.2.1/30;
        }
        family iso;
    }
}
fxp0 {
    unit 0 {
        family inet {
            address 10.0.1.5/24;
        }
    }
}
lo0 {
```

```
    unit 0 {
        family inet {
            address 10.0.3.5/32;
        }
        family iso;
    }
}
[edit]
lab@r5# show protocols isis
export l1-l2;
interface fe-0/0/0.0 {
    level 2 disable;
}
interface fe-0/0/1.0 {
    level 2 disable;
}
interface so-0/1/0.0 {
    level 1 disable;
}
interface at-0/2/1.0 {
    level 1 disable;
}
interface lo0.0;
```

3. Based on the topology demonstrated in this chapter and the output shown next, do you expect that all traffic generated by the data center router will follow an optimal path?

```
lab@dc> show route 0/0

inet.0: 16 destinations, 16 routes (16 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[RIP/100] 01:23:29, metric 2, tag 0
                      to 10.0.8.2 via fe-0/0/0.0
                    > to 10.0.8.14 via fe-0/0/1.0
```

4. Explain why the locally defined 10.0/16 aggregate on `r1` and `r2` worked fine with OSPF but not with IS-IS in the baseline topology.

5. Make at least three observations from the OSPF stanza shown next:

```
[edit protocols ospf]
lab@r3# show
area 0.0.0.0 {
```

```
        area-range 10.0.2.0/23;
        area-range 10.0.3.3/32 restrict;
        authentication-type md5; # SECRET-DATA
        interface at-0/1/0.0 {
            authentication-key "$9$zKS-n9peK8X7V"; # SECRET-DATA
        }
        interface lo0.0;
    }
    area 0.0.0.1 {
        nssa;
        interface all;
    }
```

Spot the Issues: Answers to Review Questions

1. Because Level 1 has been disabled in the IS-IS instance, you should expect to see a total of four Level 2 adjacencies at `r5`.

2. The problem with `r5`'s IS-IS configuration is the lack of a configured ISO NET. Although lo0 is considered an IS-IS interface, and is defined in the IS-IS stanza, a NET is required for proper IS-IS operation.

3. No. With the DC router receiving two equal-cost next hops for the default route, you should expect to see that some traffic incurs extra hops through either `r6` or `r7`, depending on which next hop is installed at any given time.

4. The key to the differing behaviors lies in the fact that network summaries (LSA Type 3s) were permitted in the OSPF stub area. The network summaries resulted in `r1` and `r2` learning about specific routes to all in-use 10.0/16 addresses. The 10.0/16 aggregate was never used due to the presence of the more specific routes, and therefore the presence of the aggregate did not cause any black holes. IS-IS, on the other hand, does not support the concept of network summaries, which makes an IS-IS Level 1 area function like an OSPF stub area with no summaries. Lacking summary routes, the 10.0/16 aggregate became the longest match for destinations outside of the Level 1 area. The reject next hop associated with the 10.0/16 aggregate route therefore resulted in a black hole.

5. Based on this OSPF stanza, you can determine the following:

   - `r3` is an ABR serving areas 0 and 1.
   - MD5-based authentication is in place in area 0 but not area 1.
   - Area 0 routes matching 10.0.2.0/23 will be presented to area 1 as a single network summary.
   - Area 1 will not have a summary route for the 10.0.3.3 loopback address of `r3` due to the `restrict` keyword.
   - All interfaces on `r3`, except the at-0/1/0 and the lo0 interfaces, have been placed into area 1.
   - `r3` will not generate a default route into the NSSA.

Chapter

2

MPLS and Traffic Engineering

JNCIE LAB SKILLS COVERED IN THIS CHAPTER:

- ✓ **LDP signaled LSPs**
- ✓ **RSVP signaled LSPs**
- ✓ **Constrained routing**
  - Explicit Route Objects
  - Constrained Shortest Path First
- ✓ **Routing table integration**
  - Prefix installation
  - Traffic engineering shortcuts
  - Prefix mapping
- ✓ **Traffic protection**
  - Primary and secondary paths
  - Fast Reroute and link protection
  - Preemption
- ✓ **Miscellaneous MPLS capabilities and features**

This chapter exposes you to a variety of JNCIE-level Multiple Protocol Label based Switching (MPLS) and traffic engineering (TE) configuration scenarios.

MPLS technology allows for the establishment of Label Switched Paths (LSPs) through an IP network in a manner similar to ATM or Frame Relay Virtual Circuits (VCs). Once these paths are established, ingress routers *push* a fixed-length MPLS label onto packets as they enter an LSP. Transit routers act only on the MPLS label, performing *swap* functions as they switch the labeled packet from port to port. The egress router normally receives an unlabeled packet as the result of the penultimate router performing a label *pop* in what is known as Penultimate Hop Popping (PHP). The unlabeled packet is then routed by the egress node, which performs a conventional, longest-match IP route lookup.

LSPs can be established through manual intervention or through the use of a signaling protocol. Manually established LSPs are similar to ATM PVCs, and are referred to as *static* in the JUNOS software. Signaled LSPs are similar in concept to ATM Demand Connections (DCs, or SVCs), whereas a signaling protocol is used to establish the LSP's state through the network in a dynamic fashion. MPLS signaling options supported by JUNOS software include the Label Distribution Protocol (LDP) and the Resource Reservation Protocol (RSVP).

Because the routing of the MPLS LSP can be controlled by factors other than the IGP's view of the shortest path, MPLS allows for the engineering of paths through an IP network that do not follow the IGP's view of the shortest path. The ability to force certain traffic over an LSP, which in turn follows an arbitrary path, is referred to as traffic engineering (TE). Traffic engineering is normally achieved through the use of RSVP in combination with Constrained Shortest Path First (CSPF) online path calculation, and/or the use of Explicit Route Objects (EROs), which are similar to source routing an IP packet.

MPLS can support the handling of non-IP traffic, or non-routable IP packets such as those using RFC 1918 private addressing, because transit Label Switching Routers (LSRs) switch the traffic based strictly on the fixed-length MPLS header; in other words, there is no requirement that the payload of an MPLS packet must contain a globally routable IP packet. The protocol/addressing agnostic nature of MPLS makes it an ideal candidate for the support of both Layer 2 and Layer 3 VPNs. Provider-provisioned VPNs are covered in a later chapter.

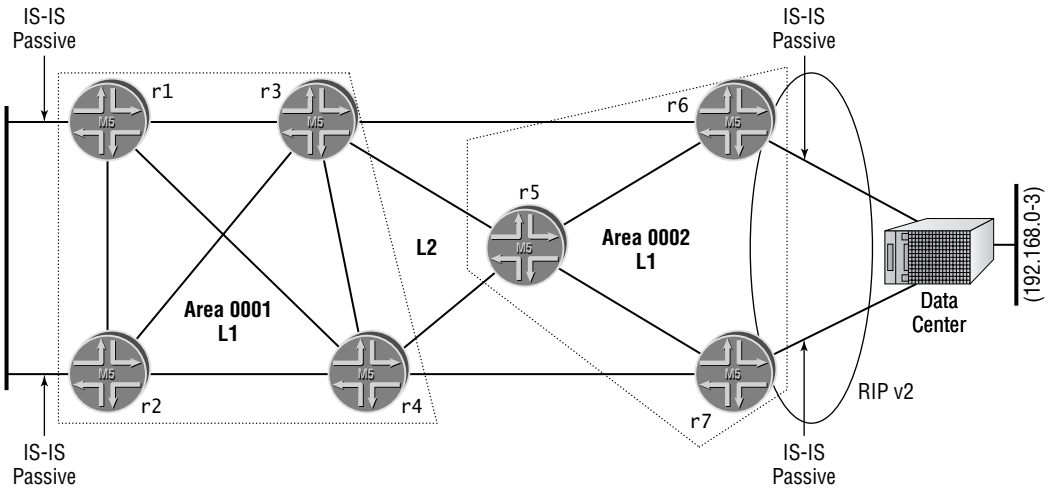
This chapter will demonstrate LSP signaling, routing table integration, and traffic protection options that are supported in the 5.6 release of the JUNOS software. Along the way, various MPLS troubleshooting and verification techniques are demonstrated. While JUNOS software supports the establishment of statically defined LSPs in a manner akin to an ATM PVC, this capability is rarely used in service provider networks due to a static LSP's propensity toward being misconfigured, and the inability to ascertain the end-to-end operational status of a static LSP. This chapter does not address statically defined LSPs because their disadvantages

make it very unlikely that you will need to deploy statically defined LSP during the JNCIE examination.

This chapter concludes with a case study designed to closely approximate a typical JNCIE MPLS and traffic engineering configuration scenario. The results of key operational mode commands are provided in the case study analysis section so that you can also compare the behavior of your network to a known good example. Example router configurations that meet all case study requirements are provided at the end of the case study for comparison with your own configurations.

The examples demonstrated in the chapter body are based on the IS-IS baseline topology left in place at the end of Chapter 1's case study. If you are unsure as to the state of your test bed, you should take a few moments to load up and confirm the IS-IS baseline configuration before proceeding. You should review your findings from the IS-IS IGP discovery case study before beginning your MPLS configuration. Figure 2.1 summarizes the results of your IS-IS IGP findings.

FIGURE 2.1 Summary of IS-IS discovery findings



Notes:

Multi-level IS-IS, Areas 0001 and 0002 with ISO NET based on router number.

lo0 address of `r3` and `r4` not injected into Area 0001 to ensure optimal forwarding between 10.0.3.3 and 10.0.3.4.

Passive setting on `r5`'s core interfaces for optimal Area 0002-to-core routing.

No authentication or route summarization. Routing policy at `r5` to leak L1 externals (DC routes) to L2.

Redistribution of static default route to data center from both `r6` and `r7`. Redistribution of 192.168.0/24 through 192.168.3/24 routes from RIP into IS-IS by both `r6` and `r7`.

All adjacencies are up, reachability problem discovered at `r1` and `r2` caused by local aggregate definition. Corrected through IBGP policy to effect 10.0/16 route advertisement from `r3` and `r4` to `r1` and `r2`; removed local aggregate from `r1` and `r2`.

Suboptimal routing detected at the data center and at `r1`/`r2` for some locations. This is the result of random nexthop choice for data center's default, and the result of `r1` and `r2`'s preference for `r3`'s RID over `r4` with regard to the 10.0/16 route. This is considered normal behavior, so no corrective actions are taken.

LDP Signaled LSPs

You begin your MPLS configuration scenario by establishing an LDP signaled LSP between r6 and r7 for the purpose of carrying traffic between customer locations C1 and C2. The criteria for this scenario are as follows:

- Configure LDP on r5, r6, and r7 to establish an LSP between r6 and r7.
- Ensure that traffic between C1 and C2 uses the LDP signaled LSP when the ingress router is r6 or r7.
- Use a 5-second keepalive interval, and ensure that LDP state is maintained should a routing restart occur.
- Collect traffic statistics in a file called *ldp-stats* every 90 seconds.

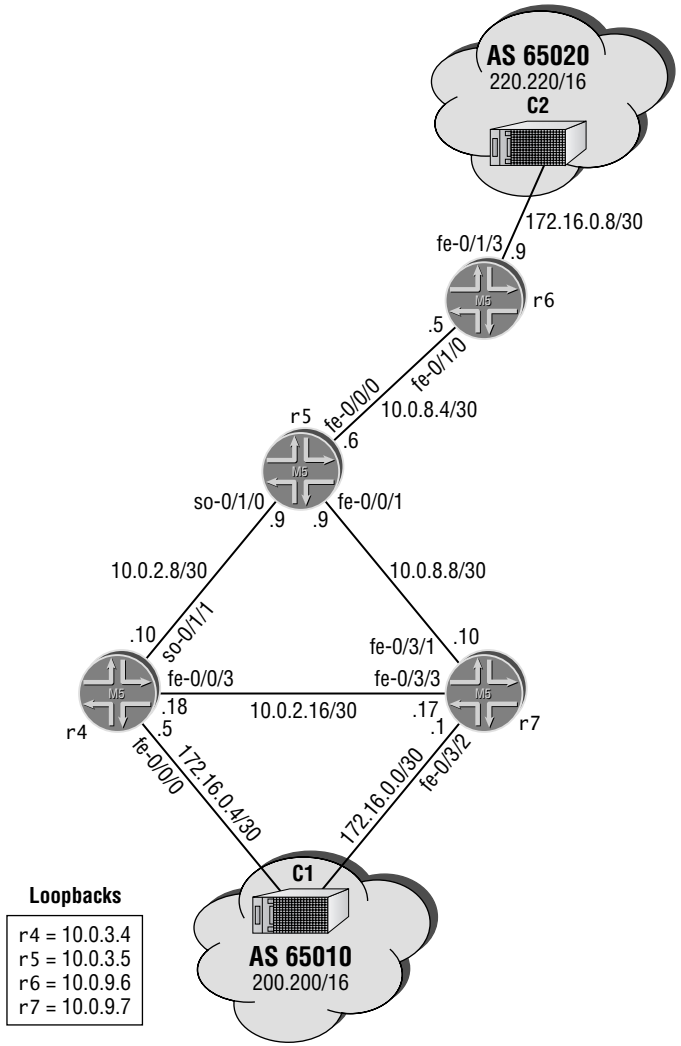
Configuring Interfaces for MPLS Support

You begin this configuration task by enabling the mpls family on the logical units of the internal facing transit interfaces in use at r5, r6, and r7. Refer to Figure 2.2 for the topology specifics needed to complete this configuration scenario. Although r4 is part of the overall LDP configuration topology, the configuration requirements indicated that LDP will not be configured on r4 in this scenario.

Adding the mpls family to a given interface enables labeled packet processing for that interface. You do not need to include the mpls family on the lo0 interface to meet the requirements of this configuration example. The following commands correctly add the mpls family to the transit interfaces on r5. The mpls family is added to all of r5's interfaces at this time under the assumption that r5's ATM and Packet Over SONET (POS) interfaces will require MPLS support in the near future, and because the presence of the mpls family will cause no harm in the event that your assumption does not pan out:

```
[edit]
lab@r5# edit interfaces

[edit interfaces]
lab@r5# set fe-0/0/0 unit 0 family mpls

[edit interfaces]
lab@r5# set fe-0/0/1 unit 0 family mpls

[edit interfaces]
lab@r5# set at-0/2/1 unit 0 family mpls

[edit interfaces]
lab@r5# set so-0/1/0 unit 0 family mpls
```

FIGURE 2.2 LDP signaled LSPs



Similar commands are now entered on `r6` and `r7` to add the `mpls` family to their internal facing transit interfaces, because they are expected to require MPLS support. The `mpls` family is not specified on external interfaces (EBGP or data center–facing interfaces), in accordance with the requirements of this scenario; note that adding the `mpls` family to these interfaces should not break anything in this example. The completed interface configuration for `r6` is shown with the MPLS-related changes highlighted. Though not shown here, `r7` has a similar configuration at this time:

```
[edit]
lab@r6# show interfaces
```

```
fe-0/1/0 {
    unit 0 {
        family inet {
            address 10.0.8.5/30;
        }
        family iso;
        family mpls;
    }
}
fe-0/1/1 {
    unit 0 {
        family inet {
            address 10.0.2.13/30;
        }
        family iso;
        family mpls;
    }
}
fe-0/1/2 {
    unit 0 {
        family inet {
            address 10.0.8.2/30;
        }
        family iso;
    }
}
fe-0/1/3 {
    unit 0 {
        family inet {
            address 172.16.0.9/30;
        }
    }
}
fxp0 {
    unit 0 {
        family inet {
            address 10.0.1.6/24;
        }
    }
}
```

```
lo0 {
    unit 0 {
        family inet {
            address 10.0.9.6/32;
        }
        family iso {
            address 49.0002.6666.6666.6666.00;
        }
    }
}
```

With the interfaces of r5, r6, and r7 set to support the mpls family, you commit your changes and issue a **show mpls interface** command to check your work:

```
[edit interfaces]
lab@r5# run show mpls interface
MPLS not configured

[edit interfaces]
```

The (lack of) output indicates that the mere presence of the mpls family is not sufficient for the interface to be enabled for MPLS operation. This condition is addressed in the following section, "Enable MPLS Processing on the Router." Before moving on, you issue a **show interfaces terse** command to confirm the presence of the mpls family on the internal facing transit interfaces at r5:

```
[edit interfaces]
lab@r5# run show interfaces terse
Interface               Admin Link Proto Local                 Remote
fe-0/0/0                up    up
fe-0/0/0.0              up    up   inet  10.0.8.6/30
                                  iso
                                  mpls
fe-0/0/1                up    up
fe-0/0/1.0              up    up   inet  10.0.8.9/30
                                  iso
                                  mpls
fe-0/0/2                up    down
fe-0/0/3                up    down
so-0/1/0                up    up
so-0/1/0.0              up    up   inet  10.0.2.9/30
                                  iso
                                  mpls
```

```
so-0/1/1                up    down
so-0/1/2                up    down
so-0/1/3                up    down
at-0/2/0                up    down
at-0/2/1                up    up
at-0/2/1.0              up    up   inet  10.0.2.1/30
                                   iso
                                   mpls
dsc                     up    up
fxp0                    up    up
fxp0.0                  up    up   inet  10.0.1.5/24
fxp1                    up    up
fxp1.0                  up    up   tnp   4
gre                     up    up
ipip                    up    up
lo0                     up    up
lo0.0                   up    up   inet  10.0.3.5            --> 0/0
                                   iso   49.0002.5555.5555.5555.00
lsi                     up    up
mtun                    up    up
pimd                    up    up
pime                    up    up
tap                     up    up
```

As with the addition of any protocol family, care must be taken to ensure that the correct logical unit is specified. In the examples shown thus far, all interfaces are using the default logical unit value 0, but this will not always be the case!

Enable MPLS Processing on the Router

The previous section ended with the determination that no interfaces are considered "MPLS capable" despite the addition of the `mpls` family to their logical units. This condition is the result of not enabling the MPLS process on the *router* itself. The `mpls` family allows interface-level processing of labeled packets, but without an MPLS process on the router to back this up, the MPLS capabilities of an interface are moot. The following command, entered on `r7`, enables the MPLS process on the router for all interfaces associated with the `mpls` family. If desired, you can specify each interface explicitly. The use of the `all` keyword is quite safe,

however, because only interfaces with the mpls family provisioned will be considered an MPLS interface:

```
[edit]
lab@r7# set protocols mpls interface all

[edit]
lab@r7# show protocols mpls
interface all;

[edit]
lab@r7# commit
commit complete
```

After the change is committed, you once again display the list of MPLS interfaces:

```
[edit]
lab@r7# run show mpls interface
Interface        State       Administrative groups
fe-0/3/1.0        Up         <none>
fe-0/3/3.0        Up         <none>
```

The output confirms that r7 now considers its fe-0/3/1 and fe-0/3/3 interfaces as MPLS capable. The lack of administrative groups is normal, considering the CSPF-related link coloring has not been configured yet. Before moving on to the next section, you should confirm that r5 and r6 also list the appropriate interfaces as MPLS enabled. The following capture confirms that r5 is correctly configured, and that all of its MPLS-enabled interfaces are functional:

```
[edit]
lab@r5# run show mpls interface
Interface        State       Administrative groups
fe-0/0/0.0       Up         <none>
fe-0/0/1.0       Up         <none>
so-0/1/0.0       Up         <none>
at-0/2/1.0       Up         <none>
```

The show mpls interface command is very useful when the goal is to quickly diagnose interface and MPLS instance–related problems. If an interface is absent from the command's output, it is normally because that interface does not have the mpls family configured, or because the interface is not listed under the [edit protocols mpls] instance. A Dn indication in the State column tells you that the corresponding interface is in the down state, while also indicating that the interface has the mpls family configured and that it is listed in the mpls instance.

Enabling the LDP Instance

Your next set of commands creates the LDP instance on r7, configures the modified keepalive interval, and enables the collection of traffic statistics, in accordance with the requirements of this example:

```
[edit]
lab@r7# set protocols ldp interface fe-0/3/1

[edit]
lab@r7# set protocols ldp interface fe-0/3/3

[edit]
lab@r7# set protocols ldp keepalive-interval 5

[edit]
lab@r7# set protocols ldp traffic-statistics file ldp-stats

[edit]
lab@r7# set protocols ldp traffic-statistics interval 90
```

The LDP configuration at r7 is committed and displayed:

```
[edit]
lab@r7# show protocols ldp
traffic-statistics {
    file ldp-stats;
    interval 90;
}
keepalive-interval 5;
interface fe-0/3/1.0;
interface fe-0/3/3.0;
```

To meet the "maintain LDP state in the event of routing restart" aspects of this scenario, you must enable the graceful-restart feature under the main routing instance's routing-options stanza. The wording of this requirement intentionally avoids using the "graceful restart" term to test the candidate's understanding of what the feature does without giving away the actual keyword used to configure it. Failing to configure graceful-restart under routing-options results in the LDP instance operating in helper mode only, which will not meet the requirements posed in this example:

```
[edit]
lab@r7# set routing-options graceful-restart
```

After committing the changes at r7, you issue commands to determine LDP neighbor and session status. Because r7 is currently the only router with an LDP instance, no LDP

neighbors or sessions are currently displayed at r7:

```
lab@r7> show ldp session

lab@r7> show ldp neighbor
```

No neighbors or sessions are in place, but the correct LDP interface listing is returned, which is an auspicious start:

```
lab@r7> show ldp interface
Interface            Label space ID        Nbr count   Next hello
fe-0/3/1.0           10.0.9.7:0               0           4
fe-0/3/3.0           10.0.9.7:0               0           4
```

The lack of entries in the inet.3 routing table further confirms that no LDP LSPs have yet been established:

```
lab@r7> show route table inet.3

lab@r7>
```

The lack of LDP signaled LSPs is to be expected, considering that r7 has not detected any LDP neighbors or established any LDP sessions. With r7 apparently ready to go, you enable the LDP instance on r5 and r6 using commands similar to those demonstrated for r7. The resulting LDP configuration for r5 and r6 is shown next:

```
[edit]
lab@r5# show protocols ldp
traffic-statistics {
    file ldp-stats;
    interval 90;
}
keepalive-interval 5;
interface fe-0/0/0.0;
interface fe-0/0/1.0;
 [edit]
lab@r5# show routing-options
graceful-restart;
static {
    route 10.0.200.0/24 {
        next-hop 10.0.1.102;
        no-readvertise;
    }
}

[edit]
```

Note that r5 does not have its so-0/1/0 interface listed under the LDP stanza, which is in keeping with r4 not running LDP. The LDP settings at r6 are examined next:

```
lab@r6# show protocols ldp
traffic-statistics {
    file ldp-stats;
    interval 90;
}
keepalive-interval 5;
interface fe-0/1/0.0;
interface fe-0/1/1.0;

[edit]
lab@r6# show routing-options
graceful-restart;
static {
    route 0.0.0.0/0 reject;
    route 10.0.200.0/24 {
        next-hop 10.0.1.102;
        no-readvertise;
    }
}
aggregate {
    route 10.0.0.0/16;
}
autonomous-system 65412;
```

After committing the changes on all routers, your attention shifts to verification of the LDP signaled LSP, as documented in the next section.

Verifying LDP Signaled LSPs

The LDP protocol strives to automatically detect neighbors and establish LDP sessions, which are then used to advertise one or more forwarding equivalency classes (FECs) with MPLS labels, such that it is difficult *not* to automatically establish LSPs between the loopback addresses of all LDP-enabled routers. LDP is sometimes characterized as "RIP on steroids" for this automatic neighbor-to-neighbor FEC advertisement behavior. By default, the JUNOS software implementation of LDP advertises a FEC for /32 interface routes only. The use of LDP will normally result in each router having an LDP signaled LSP to the loopback address of all other routers running LDP as a result of this default behavior. The use of import and export policy can alter this default behavior, but such policy is not called for in this scenario.

You begin your LDP signaled LSP verification by confirming that r5 now lists both r6 and r7 as LDP neighbors:

```
lab@r5> show ldp neighbor
Address            Interface          Label space ID         Hold time
10.0.8.5           fe-0/0/0.0         10.0.9.6:0               14
10.0.8.10          fe-0/0/1.0         10.0.9.7:0               11
```

With both r6 and r7 detected as LDP neighbors, you expect to see that a TCP-based LDP session has been established between r5 and routers r6 and r7:

```
lab@r5> show ldp session
  Address            State        Connection     Hold time
10.0.9.6           Operational  Open             25
10.0.9.7           Operational  Open             29
```

As expected, an LDP session has been established to both r6 and r7 from r5. Note that, by default, the session is associated with the remote router's loopback address/RID. This behavior can be modified with the use of the `transport-address` keyword in the `ldp` stanza if needed. Adding the `detail` keyword to the previous command allows for verification of the modified keepalive interval and the support of both graceful restart and helper mode:

```
[edit]
lab@r5# run show ldp session 10.0.9.7 detail
Address: 10.0.9.7, State: Operational, Connection: Open, Hold time: 28
  Session ID: 10.0.3.5:0--10.0.9.7:0
  Next keepalive in 3 seconds
  Passive, Maximum PDU: 4096, Hold time: 30, Neighbor count: 1
  Keepalive interval: 5, Connect retry interval: 1
  Local address: 10.0.3.5, Remote address: 10.0.9.7
  Up for 00:00:30
  Local - Restart: enabled, Helper mode: enabled, Reconnect time: 60000
  Remote - Restart: enabled, Helper mode: enabled, Reconnect time: 60000
  Local maximum recovery time: 120000 msec
  Next-hop addresses received:
    10.0.8.10
    10.0.2.17
```

The output generated by r5 indicates that you have correctly set the keepalive interval and graceful restart aspects of the LDP protocol. The next command confirms that r5 has two LDP signaled LSPs that are associated with the loopback addresses of r6 and r7, respectively:

```
lab@r5> show route table inet.3

inet.3: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)
Restart Complete
+ = Active Route, - = Last Active, * = Both
```

```
10.0.9.6/32        *[LDP/9] 00:04:05, metric 1
                    > to 10.0.8.5 via fe-0/0/0.0
10.0.9.7/32        *[LDP/9] 00:04:24, metric 1
                    > to 10.0.8.10 via fe-0/0/1.0
```

As hoped for, two LDP signaled LSPs have been established at r5. These LSPs were established as a result of r6 and r7 using these LDP sessions to advertise their lo0-based FEC along with their choice of MPLS label to r5. The lack of label-related actions (swap, push, pop, etc.) for the LDP signaled LSPs in the previous display is an indication that Penultimate Hop Popping (PHP) is in place. Because r5 is the penultimate hop for the LSPs that have been established to r6 and r7, no label operation occurs when packets are sourced at r5 and targeted at the loopback address of either r6 or r7. Put another way, no labels are pushed, popped, or swapped on an LSP that consists of a single hop. A quick look at the LDP database on r5 confirms that both r6 and r7 have signaled a desire for PHP behavior by virtue of their advertising reserved label 3 in conjunction with their lo0 FECs:

```
lab@r5> show ldp database
Input label database, 10.0.3.5:0--10.0.9.6:0
  Label     Prefix
 100002     10.0.3.5/32
      3     10.0.9.6/32
 100003     10.0.9.7/32

Output label database, 10.0.3.5:0--10.0.9.6:0
  Label     Prefix
      3     10.0.3.5/32
 100004     10.0.9.6/32
 100003     10.0.9.7/32

Input label database, 10.0.3.5:0--10.0.9.7:0
  Label     Prefix
 100004     10.0.3.5/32
 100005     10.0.9.6/32
      3     10.0.9.7/32

Output label database, 10.0.3.5:0--10.0.9.7:0
  Label     Prefix
      3     10.0.3.5/32
 100004     10.0.9.6/32
 100003     10.0.9.7/32
```

The display can be confusing, owing to the fact that LDP does not implement split horizon, and therefore re-advertises the FEC it receives from a given peer back to that peer. This is normal behavior for LDP, as LDP's reliance on the IGP to prevent loops means there is no need

to implement some form of split horizon. Focusing on the highlighted portions of r5's LDP database, we can see that:

- r7 has advertised a 10.0.9.7/32 FEC with label value 3. This entry appears in r5's input database for the LDP session between r5 and r7. This 10.0.9.7 database entry results in the creation of an LSP between r5 and r7's loopback address. The presence of this LSP is indicated by the 10.0.9.7 entry in r5's inet.3 routing table (shown previously).
- r7 has sent the label bindings that it received from r5 back to r5, which accounts for r5's input database indicating FECs for 10.0.3.5 and 10.0.9.6 for the 10.0.3.5:0--10.0.9.7:0 LDP session.
- The output database on r5 indicates that r7 should have two LDP signaled LSPs: one with no label operation (PHP is signaled with label value 3) that is associated with r5 itself, and another that will push label 100004 onto packets destined for r6's 10.0.9.6 loopback address.

To confirm this prediction, you analyze r7's inet.3 routing table:

```
lab@r7> show route table inet.3

inet.3: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)
Restart Complete
+ = Active Route, - = Last Active, * = Both

10.0.3.5/32        *[LDP/9] 00:17:52, metric 1
                    > to 10.0.8.9 via fe-0/3/1.0
10.0.9.6/32        *[LDP/9] 00:17:52, metric 1
                    > to 10.0.8.9 via fe-0/3/1.0, Push 100004
```

As predicted, two LDP signaled LSPs are present at r7. The highlights call out the label push operation associated with the 10.0.9.6/32 address. A quick display of the route to 10.0.9.6 confirms that r7 now has both an IGP entry in inet.0 and an LDP entry in inet.3 for this prefix:

```
lab@r7> show route 10.0.9.6

inet.0: 125622 destinations, 125628 routes (125622 active, 0 holddown, 0 hidden)
Restart Complete
+ = Active Route, - = Last Active, * = Both

10.0.9.6/32        *[IS-IS/15] 03:06:21, metric 20
                    > to 10.0.8.9 via fe-0/3/1.0

inet.3: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)
Restart Complete
+ = Active Route, - = Last Active, * = Both

10.0.9.6/32        *[LDP/9] 00:20:30, metric 1
                    > to 10.0.8.9 via fe-0/3/1.0, Push 100004
```

So far, your LDP confirmation checks have indicated that all aspects of the LDP signaling configuration scenario are working as required. The final check of LDP operation is to verify that traffic flowing between customer locations uses the LDP signaled LSPs. Note that LDP signaled LSPs are not displayed in the output of the `show mpls lsp` command. The fact that only RSVP signaled LSPs are listed in the output of this command has been known to cause candidates to believe that their LDP signaled LSPs are broken!

```
[edit]
lab@r7# run show mpls lsp
Ingress LSP: 0 sessions
Total 0 displayed, Up 0, Down 0

Egress LSP: 0 sessions
Total 0 displayed, Up 0, Down 0

Transit LSP: 0 sessions
Total 0 displayed, Up 0, Down 0
```

Candidates are often confused about when traffic will, or will not, be mapped to an LSP. As a general rule, only the traffic associated with BGP next hops that resolve through `inet.3` will be transported over an LSP. Later portions of this chapter will detail MPLS routing table integration, but for now it is sufficient to state that with default settings, traceroutes from `r7` to 10.0.9.6 will *not* use the LSP, while traceroutes from `r7` to BGP destinations that resolve to 10.0.9.6 as the BGP next hop *will* use the LSP.

You next verify that traffic between C1 and C2 is forwarded over the LDP signaled LSP between `r6` and `r7` by conducting traceroute testing from either the customer locations or the LSP endpoints themselves. The former approach is shown here, with this capture taken from C1:

```
lab@c1> traceroute 220.220.0.1 source 200.200.0.1
traceroute to 220.220.0.1 (220.220.0.1) from 200.200.0.1, 30 hops max, 40 byte
    packets
 1  172.16.0.5 (172.16.0.5)  0.385 ms  0.303 ms  0.295 ms
 2  10.0.2.5 (10.0.2.5)  0.342 ms  0.322 ms  0.315 ms
 3  10.0.2.13 (10.0.2.13)  0.266 ms  0.227 ms  0.223 ms
 4  220.220.0.1 (220.220.0.1)  0.342 ms  0.320 ms  0.319 ms
```

Though the traceroute from C1 to C2 succeeds, nothing in the output confirms the presence of LSP-based forwarding. Fortunately you happen to notice that the first hop in the traceroute points to `r4`, as opposed to `r7`. A forwarding decision such as this at C1 accounts for the lack of LSP hops in the traceroute. Failing to make this observation could lead to a candidate messing around with the LDP configurations at `r5`, `r6`, and `r7`, when the problem has nothing at all to do with LDP and MPLS! A `show route` command confirms your suspicions, by indicating

that r4's 220.220/16 advertisement is preferred over r7's due to their respective router IDs:

```
lab@c1> show route 220.220.0.1 detail

inet.0: 125561 destinations, 251118 routes (125561 active, 0 holddown, 2 hidden)
220.220.0.0/16 (2 entries, 1 announced)
        *BGP    Preference: 170/-101
                Source: 172.16.0.5
                Nexthop: 172.16.0.5 via fe-0/0/0.0, selected
                State: <Active Ext>
                Local AS: 65010 Peer AS: 65412
                Age: 36:39
                Task: BGP_65412.172.16.0.5+179
                Announcement bits (2): 0-KRT 1-BGP.0.0.0.0+179
                AS path: 65412 65020 I
                Localpref: 100
                Router ID: 10.0.3.4
         BGP    Preference: 170/-101
                Source: 172.16.0.1
                Nexthop: 172.16.0.1 via fe-0/0/1.0, selected
                State: <NotBest Ext>
                Inactive reason: Router ID
                Local AS: 65010 Peer AS: 65412
                Age: 34:30
                Task: BGP_65412.172.16.0.1+2759
                AS path: 65412 65020 I
                Localpref: 100
                Router ID: 10.0.9.7
```

While this situation could be fixed with the manipulation of BGP attributes such as MED, a more direct solution is presented here:

```
lab@c1> traceroute 220.220.0.1 source 200.200.0.1 bypass-routing gateway
   172.16.0.1
traceroute to 220.220.0.1 (172.16.0.1) from 200.200.0.1, 30 hops max, 48 byte
   packets
 1  172.16.0.1 (172.16.0.1)  0.254 ms  0.158 ms  0.148 ms
 2  10.0.8.9 (10.0.8.9)  0.656 ms  0.577 ms  0.582 ms
     MPLS Label=100004 CoS=0 TTL=1 S=1
 3  10.0.8.5 (10.0.8.5)  0.350 ms  0.322 ms  0.320 ms
 4  220.220.0.1 (220.220.0.1)  0.436 ms  0.420 ms  0.416 ms
```

Excellent! The use of bypass-routing along with the specification of r7's EBGP peering address as the gateway has resulted in C1 forwarding packets addressed to 220.220.0.1 through r7. Further, the presence of MPLS-based forwarding between r7 and r6 is clearly indicated by the 100004 label value shown at hop 2 in the traceroute output. Recall that

previous confirmation steps have determined that r7 should push label 10004 onto any packets that are associated with a BGP next hop of 10.0.9.6, and this is exactly what you are seeing!

You can also verify proper LSP forwarding from the LSP endpoints, as in this example taken from r6:

```
lab@r6> traceroute 10.0.9.7
traceroute to 10.0.9.7 (10.0.9.7), 30 hops max, 40 byte packets
 1  10.0.8.6 (10.0.8.6)  0.561 ms  0.351 ms  0.268 ms
 2  10.0.9.7 (10.0.9.7)  0.160 ms  0.162 ms  0.131 ms
```

The output from a traceroute to 10.0.9.7 confirms that, by default, traffic destined to the LSP endpoint is not actually subjected to LSP forwarding. By altering the traceroute target to target one of the routes being advertised by C1, the presence of LSP forwarding is again confirmed:

```
lab@r6> traceroute 200.200.0.1
traceroute to 200.200.0.1 (200.200.0.1), 30 hops max, 40 byte packets
 1  10.0.8.6 (10.0.8.6)  0.623 ms  0.504 ms  0.450 ms
     MPLS Label=100003 CoS=0 TTL=1 S=1
 2  10.0.8.10 (10.0.8.10)  0.154 ms  0.154 ms  0.135 ms
 3  200.200.0.1 (200.200.0.1)  0.222 ms  0.206 ms  0.187 ms
```

The final verification task is to confirm that LDP statistics gathering is functioning in accordance with the scenario's parameters. The following commands confirm that statistics gathering is enabled, and that the correct file is being used to house these statistics:

```
lab@r6> show ldp traffic-statistics
FEC                 Type            Packets          Bytes       Shared
 10.0.3.5/32        Transit               0              0           No
                    Ingress               0              0           No
 10.0.9.7/32        Transit               0              0           No
                    Ingress              21           1032           No
```

LDP statistics are being accumulated, so correct log file usage is confirmed next:

```
lab@r6> show log ldp-stats
Feb 13 03:53:02 trace_on: Tracing to "/var/log/ldp-stats" started
 FEC                 Type            Packets          Bytes       Shared
 10.0.3.5/32        Transit               0              0           No
                    Ingress               0              0           No
 10.0.9.7/32        Transit               0              0           No
                    Ingress               0              0           No
 Feb 13 03:54:33, read statistics for 2 FECs in 00:00:01 seconds (5 queries)
```

The output from r6 confirms that you have correctly configured LDP statistics gathering based on the specified criteria. You should confirm that r5 and r7 generate similar LDP statistics output before deciding to move on.

The results of your traceroute testing, taken in conjunction with the LDP neighbor and session status output shown previously, confirm that you have met all requirements for the LDP signaled LSP configuration scenario.

LDP Summary

The LDP protocol is used to dynamically signal MPLS LSPs. You must configure `family mpls` support on your router interfaces, and enable a MPLS instance on the router before you can dynamically signal LSPs. The configuration and operation of LDP is pretty straightforward. At a minimum, you must configure an LDP instance and associate one or more MPLS-enabled interfaces with the LDP instance before LDP signaling can begin. Once configured, LDP will automatically detect neighbors and establish a TCP-based session to each neighbor for the purpose of establishing LSPs through the exchange of FECs. Heck, the hardest part about LDP is *preventing* the automatic establishment of LSPs to all other LDP-enabled routers! LDP relies on the IGP for loop detection and forwarding, which means that the LDP does not support traffic engineering in that the forwarding path of an LDP signaled LSP will always match that used by the IGP.

By default, JUNOS software will automatically advertise a FEC for /32 interface routes, which in most cases means that each router will advertise a FEC for its loopback address/RID only.

Various commands are available to monitor the operation of LDP and to view the resulting label database. LDP signaled LSPs are placed into the `inet.3` routing table by default. Because entries in `inet.3` are used only for BGP next hop resolution, traffic sent to internal addresses, or the tunnel endpoint itself, will not be forwarded over the signaled LSP.

RSVP Signaled LSPs

This section covers various configuration topics that relate to the use of RSVP signaling for LSP establishment. You begin by adding basic RSVP functionality to the test bed; as the section progresses, you will be adding routing constraints in the form of Explicit Route Objects (EROs) and/or online path calculation using the Constrained Shortest Path First (CSPF) algorithm.

You begin your RSVP configuration scenario with the addition of an RSVP signaled LSP to the test bed. This LSP must adhere to these criteria:

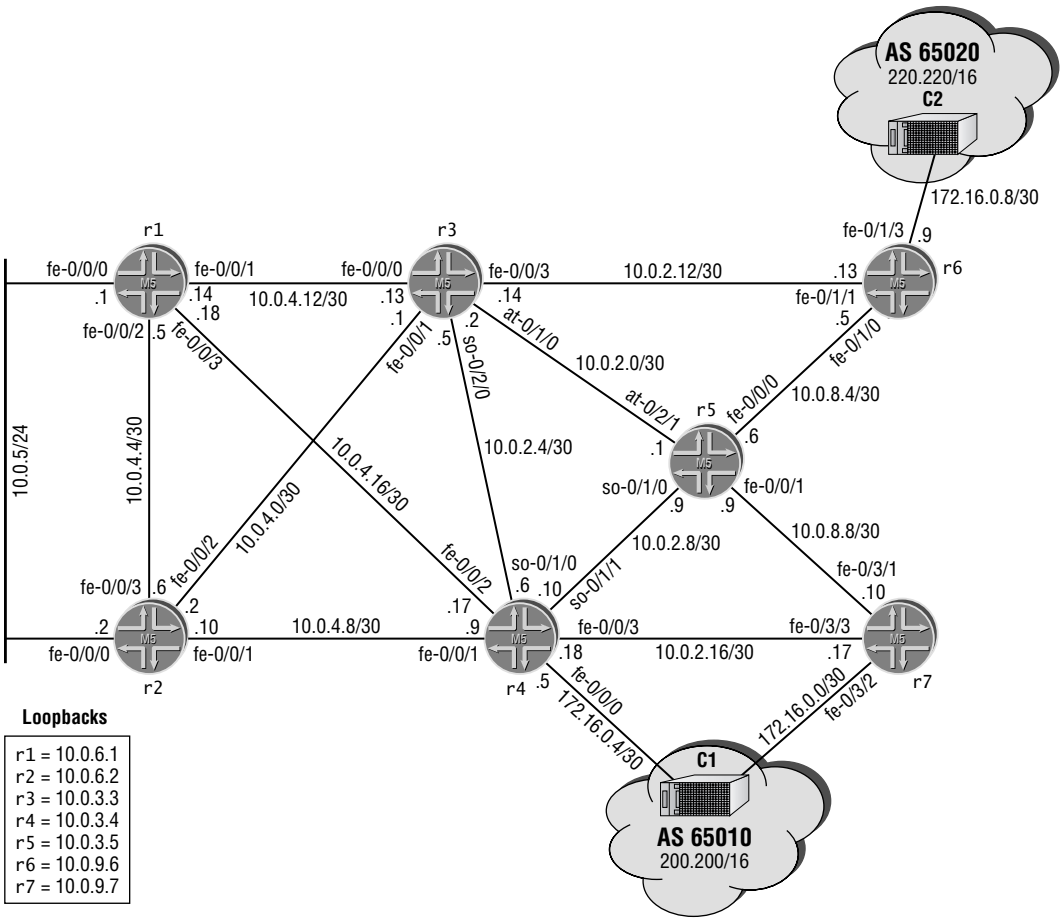
- Use RSVP signaling to establish LSP *r1–r7*.
- Ensure that the LSP follows the IGP's shortest path between `r1` and `r7`.
- Configure the LSP to reserve 10Mbps of bandwidth.
- Configure `r1` and `r3` to authenticate their RSVP signaling exchanges using key *jnx*.

Configuring Baseline MPLS Support on Remaining Routers

Before adding RSVP signaling, it is suggested that you configure all routers in the test bed with baseline MPLS functionality. This baseline functionality can be added to each router as needed, but this author has found that all too often these basic configuration settings are overlooked when LSPs are later being added to your network, which can lead to lost time as you find

yourself troubleshooting every LSP you try to establish. Refer to Figure 2.3 for the topology specifics needed to complete this scenario.

FIGURE 2.3 RSVP signaling topology



You begin this configuration task by adding baseline MPLS functionality to all routers in the test bed. This is accomplished by configuring the `mpls` family on all remaining internal facing transit interfaces and by creating an MPLS instance on each router in the test bed. The command sequence shown next correctly adds the `mpls` family to `r1`'s internal facing transit interfaces. Note that MPLS is added to `r1`'s fe-0/0/0 interface at this time because the 10.0.5/24 subnet is considered an internal prefix:

```
[edit]
lab@r1# edit interfaces
```

```
[edit interfaces]
lab@r1# set fe-0/0/0 unit 0 family mpls

[edit interfaces]
lab@r1# set fe-0/0/1 unit 0 family mpls

[edit interfaces]
lab@r1# set fe-0/0/2 unit 0 family mpls

[edit interfaces]
lab@r1# set fe-0/0/3 unit 0 family mpls
```

With protocol family support correctly configured on `r1`, you now define `r1`'s MPLS instance; be sure that you list all transit interfaces either explicitly, or implicitly, through the use of an `interface all` statement, as shown in this example:

```
[edit]
lab@r1# set protocols mpls interface all
```

The modified configuration for `r1` is shown with the MPLS-related changes highlighted.

```
[edit]
lab@r1# show interfaces
fe-0/0/0 {
    unit 0 {
        family inet {
            address 10.0.5.1/24;
        }
        family iso;
        family mpls;
    }
}
fe-0/0/1 {
    unit 0 {
        family inet {
            address 10.0.4.14/30;
        }
        family iso;
        family mpls;
    }
}
fe-0/0/2 {
    unit 0 {
        family inet {
```

```
            address 10.0.4.5/30;
        }
        family iso;
        family mpls;
    }
}
fe-0/0/3 {
    unit 0 {
        family inet {
            address 10.0.4.18/30;
        }
        family iso;
        family mpls;
    }
}
fxp0 {
    unit 0 {
        family inet {
            address 10.0.1.1/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 10.0.6.1/32;
        }
        family iso {
            address 49.0001.1111.1111.1111.00;
        }
    }
}

[edit]
lab@r1# show protocols
mpls {
    interface all;
}
bgp {
    group int {
```

```
        type internal;
        local-address 10.0.6.1;
        neighbor 10.0.6.2;
        neighbor 10.0.3.3;
        neighbor 10.0.3.4;
        neighbor 10.0.3.5;
        neighbor 10.0.9.6;
        neighbor 10.0.9.7;
    }
    group p1 {
        type external;
        export ebgp-out;
        neighbor 10.0.5.254 {
            peer-as 65050;
        }
    }
}
isis {
    level 2 disable;
    interface fe-0/0/0.0 {
        passive;
    }
    interface fe-0/0/1.0;
    interface fe-0/0/2.0;
    interface fe-0/0/3.0;
    interface lo0.0;
}
```

With the modifications committed, operational output from `r1` confirms that all internal facing transit interfaces have been correctly enabled for MPLS:

```
[edit]
lab@r1# run show mpls interface
Interface        State       Administrative groups
fe-0/0/0.0       Up          <none>
fe-0/0/1.0       Up          <none>
fe-0/0/2.0       Up          <none>
fe-0/0/3.0       Up          <none>
```

Before proceeding to the next section, you should configure baseline MPLS support on `r2`, `r3`, and `r4` using similar commands. Be sure to verify your work by listing the router's MPLS interfaces after you commit your changes. You should not enable MPLS support on EBGP-facing interfaces at `r3`, `r4`, or `r6` at this time.

r4's configuration is now displayed with the MPLS-related additions highlighted. Note the correct specification of `unit 100` for its so-0/1.0 interface declaration in the `mpls` stanza; specifying the correct interface units is critical for proper operation.

```
[edit]
lab@r4# show protocols mpls
interface so-0/1/0.100;
interface so-0/1/1.0;
interface fe-0/0/1.0;
interface fe-0/0/2.0;
interface fe-0/0/3.0;

[edit]
lab@r4# show interfaces
fe-0/0/0 {
    unit 0 {
        family inet {
            address 172.16.0.5/30;
        }
    }
}
fe-0/0/1 {
    unit 0 {
        family inet {
            address 10.0.4.9/30;
        }
        family iso;
        family mpls;
    }
}
fe-0/0/2 {
    unit 0 {
        family inet {
            address 10.0.4.17/30;
        }
        family iso;
        family mpls;
    }
}
fe-0/0/3 {
    unit 0 {
        family inet {
```

```
            address 10.0.2.18/30;
        }
        family iso;
        family mpls;
    }
}
so-0/1/0 {
    encapsulation frame-relay;
    unit 100 {
        dlci 100;
        family inet {
            address 10.0.2.6/30;
        }
        family iso;
        family mpls;
    }
}
so-0/1/1 {
    encapsulation ppp;
    unit 0 {
        family inet {
            address 10.0.2.10/30;
        }
        family iso;
        family mpls;
    }
}
fxp0 {
    unit 0 {
        family inet {
            address 10.0.1.4/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 10.0.3.4/32;
        }
        family iso {
            address 49.0001.4444.4444.4444.00;
```

```
        }
    }
}
```

To confirm baseline MPLS functionality at r4, a **show mpls interface** command is issued:

```
[edit]
lab@r4# run show mpls interface
Interface        State       Administrative groups
fe-0/0/1.0       Up          <none>
fe-0/0/2.0       Up          <none>
fe-0/0/3.0       Up          <none>
so-0/1/0.100     Up          <none>
so-0/1/1.0       Up          <none>
```

The output indicates that r4's interfaces and baseline MPLS functionality are properly configured for the current JNCIE test bed topology. You should ensure that all seven routers in the test bed have similar baseline MPLS functionality before proceeding to the next section.

Enabling RSVP Signaling

In order to successfully signal an LSP with RSVP, all routers in the LSP's path must be configured to support the RSVP protocol. While you can choose to enable RSVP on an *as needed* basis, it is recommended that you take the time to set up RSVP on all routers in the test bed *before* defining your first RSVP signaled LSP. Such preconfiguration often means that you will not have to spend time troubleshooting issues relating to the lack of RSVP support with each new LSP that is later added to the test bed.

Before deciding to add a given functionality to all routers in the test bed, you should verify that such actions are not outside the parameters of your particular configuration scenario. Because no RSVP-related restrictions are defined in this case, you decide to enable RSVP on all routers. You begin on r3 by configuring all of r3's internal-facing transit interfaces as being RSVP capable. The keyword all could also be used in lieu of explicit interface listing. When using the all keyword, it is recommended that you explicitly disable RSVP support on the router's fxp0 interface.

```
[edit]
lab@r3# set protocols rsvp interface fe-0/0/0

[edit]
lab@r3# set protocols rsvp interface fe-0/0/1

[edit]
lab@r3# set protocols rsvp interface fe-0/0/3
```

```
[edit]
lab@r3# set protocols rsvp interface at-0/1/0

[edit]
lab@r3# set protocols rsvp interface so-0/2/0.100
```

Note that the default logical unit value of 0 is implied on all of r3's RSVP interface declarations with the exception of the specification of the non-default `unit 100` for r3's so-0/2/0 interface. The changes are committed, and the RSVP-enabled interfaces are displayed to validate your work:

```
[edit]
lab@r3# commit
commit complete

[edit]
lab@r3# run show rsvp interface
RSVP interface: 5 active
                    Active                Static       Available  Reserved Highwater
Interface     State resv    Subscription  BW           BW         BW       mark
fe-0/0/0.0    Up    0              100%  100Mbps      100Mbps    0bps     0bps
fe-0/0/1.0    Up    0              100%  100Mbps      100Mbps    0bps     0bps
fe-0/0/3.0    Up    0              100%  100Mbps      100Mbps    0bps     0bps
at-0/1/0.0    Up    0              100%  155.52Mbps   155.52Mbps 0bps     0bps
so-0/2/0.100  Up    0              100%  155.52Mbps   155.52Mbps 0bps     0bps
```

The display correctly lists all of r3's internal-facing transit interfaces as being RSVP capable. Note that all of the interfaces are listed as `Up`, and that by default all interfaces will allow 100 percent of their bandwidth to be reserved by RSVP. You should enter similar commands on all remaining routers and verify that the correct interfaces are listed as `Up` in the `show rsvp interfaces` display before proceeding to the next section. The following capture shows a functional configuration for r2; note the use of the `all` keyword in this example, and the explicit disabling of the fxp0 interface to make sure no RSVP signaling exchanges can occur over the OoB network segment:

```
[edit]
lab@r2# show protocols rsvp
interface all;
interface fxp0.0 {
    disable;
}
```

After the changes are committed, the correct RSVP interface listing is confirmed. Note the absence of the fxp0 interface in the resulting output:

```
[edit]
lab@r2# run show rsvp interface
RSVP interface: 4 active
```

```
                    Active                  Static    Available  Reserved  Highwater
Interface   State   resv    Subscription    BW        BW         BW        mark
fe-0/0/0.0  Up      0       100%            100Mbps   100Mbps    0bps      0bps
fe-0/0/1.0  Up      0       100%            100Mbps   100Mbps    0bps      0bps
fe-0/0/2.0  Up      0       100%            100Mbps   100Mbps    0bps      0bps
fe-0/0/3.0  Up      0       100%            100Mbps   100Mbps    0bps      0bps
```

With all routers confirmed as supporting RSVP signaling on the correct interfaces, you decide to issue a RSVP neighbor command on r3:

```
[edit]
lab@r3# run show rsvp neighbor
RSVP neighbor: 0 learned

[edit]
lab@r3#
```

The lack of detected RSVP neighbors on r3 could lead the uninitiated to believe that something is wrong with their baseline MPLS and RSVP configurations. In reality, the lack of neighbors is not a symptom of problems at all, because RSVP neighbors are detected only *when* LSPs are actually signaled. For now, the lack of RSVP neighbors is considered par for the course, so you proceed to the next section.

Configuring RSVP Authentication

With RSVP signaling now enabled on all routers in the test bed, you could try to establish the LSP from r1 to r7 *before* adding additional constraints such as the bandwidth limits, and the need for authentication between r1 and r3. The upside to this approach is that many aspects of your baseline MPLS and RSVP configurations can be quickly validated by the ability to establish such a bare-bones RSVP signaled LSP. The downside to the "start basic and add restrictions later" approach is that, many times, a candidate will simply forget to revisit the specifics once the main aspects of the task are operational.

In this example, you decide to revisit the RSVP configuration of r1 and r3 now so that you do not forget to add RSVP authentication later. You start with r1 by specifying that authentication is to be used on the fe-0/0/1 interface:

```
[edit protocols rsvp]
lab@r1# set interface fe-0/0/1 authentication-key jnx

[edit protocol rsvp]
lab@r1# show
interface all;
interface fxp0.0 {
    disable;
}
interface fe-0/0/1.0 {
    authentication-key "$9$MKZL7Vji.mT3"; # SECRET-DATA
}
```

You confirm that authentication is in effect on the correct interface(s) by adding the `detailed` switch to the `show rsvp interface` command:

```
[edit]
lab@r1# run show rsvp interface detail | find fe-0/0/1
fe-0/0/1.0  Index 3, State Ena/Up
  Authentication, NoAggregate, NoReliable, NoLinkProtection
  HelloInterval 9(second)
  Address 10.0.4.14
  ActiveResv 0, PreemptionCnt 0, Update threshold 10%
  Subscription 100%, StaticBW 100Mbps, AvailableBW 100Mbps
                         Total                   Last 5 seconds
  PacketType       Sent      Received        Sent            Received
  Path                0             0           0                   0
  PathErr             0             0           0                   0
  PathTear            0             0           0                   0
  Resv                0             0           0                   0
  ResvErr             0             0           0                   0
  ResvTear            0             0           0                   0
  Hello               0             0           0                   0
  Ack                 0             0           0                   0
  Srefresh            0             0           0                   0
  EndtoEnd RSVP       0             0           0                   0
```

You now add similar authentication to `r3`'s fe-0/0/0 interface:

```
[edit]
lab@r3# set protocols rsvp interface fe-0/0/0.0 authentication-key jnx
```

You commit the authentication-related changes to `r3` and proceed to the next section.

Configuring and Verifying RSVP Signaled LSP

With all routers in the test bed ready to support RSVP signaling, and RSVP authentication in place between `r1` and `r3`, it is now time to define the *r1–r7* LSP in accordance with the requirements of this configuration scenario. The following commands are entered at `r1`, the LSP ingress node, to define the LSP:

```
[edit protocols mpls]
lab@r1# set label-switched-path r1-r7 to 10.0.9.7

[edit protocols mpls]
lab@r1# set label-switched-path r1-r7 bandwidth 10m
```

The resulting LSP configuration is shown. Note that the `m` suffix is added to the bandwidth argument to correctly request *10Mbps* of bandwidth as opposed to *10bps*:

```
[edit protocols mpls]
lab@r1# show
```

```
label-switched-path r1-r7 {
    to 10.0.9.7;
    bandwidth 10m;
}
interface all;
```

After committing the change, the status of the new LSP is determined:

```
lab@r1> show rsvp session
Ingress RSVP: 0 sessions
Total 0 displayed, Up 0, Down 0

Egress RSVP: 0 sessions
Total 0 displayed, Up 0, Down 0

Transit RSVP: 0 sessions
Total 0 displayed, Up 0, Down 0
```

The output is somewhat of a buzzkill, in that the display indicates that the *r1–r7* LSP has failed to be established. You should wait at least a minute or so before taking diagnostic or corrective actions, owing to RSVP's default 30-second retry timer. After waiting another 30 seconds or so, you confirm that things have not improved, so you issue a **show mpls lsp extensive** command to get maximum information about what is, or is not, happening with your new RSVP signaled LSP:

```
lab@r1> show mpls lsp extensive
Ingress LSP: 1 sessions

10.0.9.7
  From: 0.0.0.0, State: Dn, ActiveRoute: 0, LSPname: r1-r7
  ActivePath: (none)
  LoadBalance: Random
  Encoding type: Packet, Switching type: Packet, GPID: IPv4
  Primary                    State: Dn
    Bandwidth: 10Mbps
    Will be enqueued for recomputation in 28 second(s).
    1 Feb 14 05:05:09  CSPF failed: no route toward 10.0.9.7[12 times]
  Created: Fri Feb 14 04:54:22 2003
Total 1 displayed, Up 0, Down 1

Egress LSP: 0 sessions
Total 0 displayed, Up 0, Down 0

Transit LSP: 0 sessions
Total 0 displayed, Up 0, Down 0
```

The highlighted fields in the output provide a veritable silver bullet for your problem. It seems that `r1` is unable to complete its online path calculation using the Constrained Shortest Path First (CSPF) algorithm, and therefore RSVP has never been notified that it should even attempt to signal

the LSP. In this case, the CSPF algorithm fails due to the lack of entries in the Traffic Engineering Database (TED) for routers outside of the Level 1 IS-IS area 0001. This is a normal occurrence for a Multi-Level IS-IS topology in that traffic engineering TLVs are not leaked between IS-IS levels.

In this case, CSPF is not even needed, because the requirements state that the *r1–r7* LSP should be routed according to the IGP's shortest path anyway. Given the current situation, you opt to disable CSPF with the following command:

```
[edit protocols mpls]
lab@r1# set label-switched-path r1-r7 no-cspf

[edit protocols mpls]
lab@r1# commit
commit complete
```

After the changes are committed, the status of the new LSP is confirmed:

```
[edit protocols mpls]
lab@r1# run show rsvp session detail
Ingress RSVP: 1 sessions

10.0.9.7
  From: 10.0.6.1, LSPstate: Up, ActiveRoute: 0
  LSPname: r1-r7, LSPpath: Primary
  Suggested label received: -, Suggested label sent: -
  Recovery label received: -, Recovery label sent: 100000
  Resv style: 1 FF, Label in: -, Label out: 100000
  Time left:    -,  Since: Fri Feb 14 05:14:41 2003
  Tspec: rate 10Mbps size 10Mbps peak Infbps m 20 M 1500
  Port number: sender 2 receiver 11897 protocol 0
  PATH rcvfrom: localclient
  PATH sentto: 10.0.4.13 (fe-0/0/1.0) 4 pkts
  RESV rcvfrom: 10.0.4.13  (fe-0/0/1.0) 4 pkts
  Record route: <self>  10.0.4.13  10.0.2.1  10.0.8.10
Total 1 displayed, Up 1, Down 0

Egress RSVP: 0 sessions
Total 0 displayed, Up 0, Down 0

Transit RSVP: 0 sessions
Total 0 displayed, Up 0, Down 0
```

Excellent! The LSP from `r1` to `r7` has been successfully established, as indicated by the highlighted portions of the capture. A quick display of `r1`'s `inet.3` routing table confirms the presence of an RSVP signaled LSP to 10.0.9.7:

```
[edit]
lab@r1# run show route table inet.3
```

```
inet.3: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.9.7/32        *[RSVP/7] 00:30:33, metric 10
                    > to 10.0.4.17 via fe-0/0/1.0, label-switched-path r1-r7
```

Although the reservation of 10Mbps was confirmed in the `show rsvp session detail` display at `r1` shown earlier, another quick check is performed on transit LSR `r5` to provide definitive confirmation that all is well with the *r1–r7* LSP:

```
[edit]
lab@r5# run show rsvp interface
RSVP interface: 4 active
                    Active                Static      Available   Reserved    Highwater
Interface    State  resv   Subscription   BW          BW          BW          mark
fe-0/0/0.0   Up        0           100%   100Mbps     100Mbps     0bps        0bps
fe-0/0/1.0   Up        1           100%   100Mbps     90Mbps      10Mbps      10Mbps
so-0/1/0.0   Up        0           100%   155.52Mbps  155.52Mbps  0bps        0bps
at-0/2/1.0   Up        0           100%   155.52Mbps  155.52Mbps  0bps        0bps
```

The highlighted entry confirms that `r5` has a single RSVP reservation that consumes 10Mbps of bandwidth on its fe-0/0/1 interface. These results indicate that you have met the requirements of the RSVP signaled LSP configuration scenario.

Many candidates are surprised to find that a bandwidth constrained LSP can be established without the use of a Traffic Engineering Database (TED) and the CSPF algorithm. Because RSVP was intended to support Quality of Service (QoS) over the Internet, it always had built-in support for session-based bandwidth reservations. Because RSVP lacks a network-wide view of the current state of network resources, there is a chance that no path in the network can honor the reservation, which causes the RSVP session to fail somewhere between ingress and egress routers. Note that the look-ahead capability of CSPF will prevent the transmission of a futile RSVP Path message because no ERO will be provided to RSVP for signaling when CSPF cannot find an end-to-end path that honors the user's constraints.

Real World Scenario

RSVP Troubleshooting

In this troubleshooting case study, the unconstrained RSVP LSP from `r1` to `r7` is being routed through `r4` instead of `r3`; in this case the alteration in the LSP's path is a function of deactivating `r1`'s fe-0/0/1 interface (not shown). Note that the LSP was being routed over `r1`'s fe-0/0/1

interface to r3 before the interface deactivation. You begin with a determination that the *r1–r7* LSP is down, and that r4 is now the first hop LSR:

```
[edit]
lab@r1# run show rsvp session detail
Ingress RSVP: 1 sessions

10.0.9.7
  From: 10.0.6.1, LSPstate: Dn, ActiveRoute: 0
  LSPname: r1-r7, LSPpath: Primary
  Suggested label received: -, Suggested label sent: -
  Recovery label received: -, Recovery label sent: -
  Resv style: 0 -, Label in: -, Label out: -
  Time left:    -,  Since: Fri Feb 14 05:21:19 2003
  Tspec: rate 10Mbps size 10Mbps peak Infbps m 20 M 1500
  Port number: sender 3 receiver 11897 protocol 0
  PATH rcvfrom: localclient
  PATH sentto: 10.0.4.17 (fe-0/0/3.0) 53 pkts
  Record route: <self>  ...incomplete
Total 1 displayed, Up 0, Down 1

Egress RSVP: 0 sessions
Total 0 displayed, Up 0, Down 0

Transit RSVP: 0 sessions
Total 0 displayed, Up 0, Down 0
```

The highlights confirm that the LSP has not been established, and that the routing of the new LSP made it as far as r4's 10.0.4.17 address. In situations like these, RSVP tracing often provides valuable clues as to the nature of the problem. Armed with the knowledge that things seemed fine until the path message hit r4, you configure RSVP tracing on r4, as shown here:

```
[edit]
lab@r4# show protocols rsvp
traceoptions {
    file rsvp;
    flag error detail;
    flag path detail;
    flag pathtear detail;
}
interface all;
interface fxp0.0 {
    disable;
}
```

A few moments after you commit the tracing changes and begin monitoring the *rsvp* log file, the following output is observed:

```
[edit]
lab@r4# Feb 13 22:04:36 RSVP recv Path 10.0.6.1->10.0.9.7 Len=188 fe-0/0/2.0
Feb 13 22:04:36   Session7 Len 16 10.0.9.7(port/tunnel ID 11897) Proto 0
Feb 13 22:04:36   Hop      Len 12 10.0.4.18/0x0c044330
Feb 13 22:04:36   Time     Len  8 30000 ms
Feb 13 22:04:36   SessionAttribute Len 16 Prio (7,0) flag 0x0 "r1-r7"
Feb 13 22:04:36   Sender7  Len 12 10.0.6.1(port/lsp ID  3)
Feb 13 22:04:36   Tspec    Len 36 rate 10Mbps size 10Mbps peak Infbps m 20 M 1500
```

```
Feb 13 22:04:36   ADspec   Len 48
Feb 13 22:04:36   LabelRequest Len  8 EtherType 0x800
Feb 13 22:04:36   Properties Len 12 Primary path
Feb 13 22:04:36   RecRoute Len 12  10.0.4.18
Feb 13 22:04:37 RSVP send Path 10.0.6.1->10.0.9.7 Len=196 fe-0/0/3.0
Feb 13 22:04:37   Session7 Len 16 10.0.9.7(port/tunnel ID 11897) Proto 0
Feb 13 22:04:37   Hop      Len 12 10.0.2.18/0x0a778330
Feb 13 22:04:37   Time     Len  8 30000 ms
Feb 13 22:04:37   SessionAttribute Len 16 Prio (7,0) flag 0x0 "r1-r7"
Feb 13 22:04:37   Sender7  Len 12 10.0.6.1(port/lsp ID  3)
Feb 13 22:04:37   Tspec    Len 36 rate 10Mbps size 10Mbps peak Infbps m 20 M 1500
Feb 13 22:04:37   ADspec   Len 48
Feb 13 22:04:37   LabelRequest Len  8 EtherType 0x800
Feb 13 22:04:37   Properties Len 12 Primary path
Feb 13 22:04:37   RecRoute Len 20  10.0.2.18 10.0.4.18
```

The trace output makes it clear that r4 correctly received the path request from r1. The output goes on to indicate that the path request was sent on to r7 over r4's fe-0/0/3 interface. With this information, the focus shifts to r7 as the problem, because we know the path message was correctly sent to r7 over its fe-0/3/3 interface, but all indications are that r7 has simply ignored the RSVP message.

After placing a similar RSVP tracing configuration into effect on r7, you note that nothing is displayed, even after several minutes. The lack of RSVP trace output should mean one of two things. Either r7 never received the path message due to a communications error, or r7 has silently discarded the path message due to a lack of RSVP support on the related interface—that is, policed discards are occurring. After successfully pinging r4's fe-0/0/3 interface, communication over the 10.0.2.16/30 subnet is confirmed to be operational, so attention shifts to the RSVP configuration on r7. After displaying r7's RSVP stanza, the problem becomes apparent: r7 is not running RSVP on its fe-0/3/3 interface after all!

```
[edit]
lab@r7# run ping 10.0.2.18 rapid count 5
PING 10.0.2.18 (10.0.2.18): 56 data bytes
!!!!!
--- 10.0.2.18 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.422/0.517/0.722/0.117 ms

[edit]
lab@r7# show protocols rsvp
traceoptions {
    file rsvp;
    flag path detail;
    flag error detail;
}
interface fe-0/3/1
interface fe-0/3/2
```

You decide to use the rename function to change the erroneous fe-0/3/2 statement to the correct value of fe-0/3/3. After committing your changes, the *r1–r7* RSVP signaled LSP is correctly established over the alternative path through r4. In this example, the trace-related configuration is removed when the problem has been resolved. Leaving a tracing configuration in effect is

usually pretty safe, but even so, it is suggested that you remove any configuration elements that are no longer required, just to reduce configuration clutter, if for no other reason. In certain cases, such as when you forget to remove an old OSPF tracing stanza *while* you also mistakenly redistribute an *entire* BGP table into your IGP, you may well wish you had deleted that unnecessary tracing setup after all. Here, the added tracing burden can only serve to tax an already strained platform, and although there should be no "chunk emission," things are not made better by the presence of an ever-churning RSVP trace file.

```
[edit protocols rsvp]
lab@r7# delete traceoptions

[edit protocols rsvp]
lab@r7# rename interface fe-0/3/2 to interface fe-0/3/3

[edit protocols rsvp]
lab@r7# commit

[edit protocols rsvp]
lab@r7# run show rsvp session
Ingress RSVP: 0 sessions
Total 0 displayed, Up 0, Down 0

Egress RSVP: 1 sessions
To              From            State Rt Style Labelin Labelout LSPname
10.0.9.7        10.0.6.1        Up     0  1 FF       3         - r1-r7
Total 1 displayed, Up 1, Down 0

Transit RSVP: 0 sessions
Total 0 displayed, Up 0, Down 0
```

Constrained Routing

The LSPs that you have created so far have all followed the shortest path contours dictated by your IGP's view of the shortest path between any two points. In the case of LDP, you have no other choice, because LDP does not support traffic engineering. However, with RSVP signaling, you have the ability to force the routing of the path message, and therefore the routing of the LSP itself, using either Explicit Routing Objects (EROs), CSPF-based online calculation, or a combination of both. This section provides several JNCIE-level configuration scenarios that incorporate TE through constrained LSP routing.

Note that ERO-based routing constraints do not involve a Traffic Engineering Database (TED), or any type of SPF calculation. The use of EROs does not require the IGP extensions that are needed to build and maintain the TED, nor do they have issues with multiple TED domains, as are present in the current Multi-Level IS-IS network. On the other hand, successful ERO-based constraints are predicated on the operator specifying a valid routing path for the LSP. For example, LSP establishment will fail if an incorrect strict hop is specified, or if a poorly specified ERO list results in the LSP wrapping back over its existing path.

Troubleshooting CSPF-related failures requires a detailed understanding of the TED and the data structures that it contains. This is because the typical "no route to host" error message resulting from a failed CSPF run provides little in the way of clues as to what has gone wrong. Sometimes, disabling CSPF computations can help you troubleshoot because removing the online path calculation component generally results in the ingress node being able to at least begin signaling the LSP by sending a path message toward the LSP's egress node. While the non-CSPF LSP is being routed toward the egress, you may see RSVP error messages and operational displays that help you locate the trouble spot in your network. You can always turn CSPF processing back on after you determine that the LSP can be successfully established without the use of CSPF.

Keep in mind that the result of a CSPF run is the failure to find either a path or a complete set of strict EROs that define the path chosen by the CSPF calculation. This means that when CSPF "fails," there is nothing for RSVP to signal, which in turn means that commands such as `show rsvp session` are virtually useless when the problem relates to CSPF failures. CSPF failures can result from IGP TED support problems, the presence of multiple TED domains, errors in configuration with regard to bandwidth and link coloring, or simply having a network that cannot support the set of constraints you have decided to place on a given LSP. In some cases, it is a good idea to try relaxing the set of constraints when you suspect TED problems. After all, you are brought a bit closer to the nature of the problem when you discover that you can establish a 10Mbps LSP but not a 10,000Mbps LSP, especially when the test bed happens to be composed entirely of Fast Ethernet technology!

ERO Constraints

In this section, you will use Explicit Route Objects (EROs) to constrain the routing of an LSP. EROs function in a manner similar to a source routed packet, in that the presence of EROs in the RSVP Path message forces the path message to visit the list of nodes in the sequence in which they are specified. EROs can be either *loose* or *strict*. A loose hop allows the IGP to route from the current node to the loosely specified target node any way it wants. A strict hop, on the other hand, does not permit an IGP recursive lookup and will not allow the presence of intervening hops between the current node and the strictly specified target node.

When listing a node, you can use either a physical or a loopback address. Note that loopback addresses should not be used in conjunction with strict hops, because by definition a strict hop should be directly connected. Although some JUNOS software versions allow strict hops that pointed to a neighbor's loopback address, the 5.6R1 version used in this test bed does not support loopback addresses as strict hops for non-CSPF LSPs. Because the CSPF algorithm generates a complete list of EROs that consists of interface addresses, you can use a strict hop ERO pointing to a neighbor's loopback interface when CSPF is used to compute the LSP's path.

To complete this configuration scenario, you must establish an LSP meeting these criteria:

- Establish LSP *r2–r6*, with `r2` as the ingress and `r6` as the egress.
- The LSP must transit both `r7` and `r5`.

Configuring the ERO Constrained LSP

You begin by defining the new LSP on `r2`:

```
[edit protocols mpls]
lab@r2# set label-switched-path r2-r6 to 10.0.9.6 primary visit-r7-r5
```

The use of primary in this command indicates that a path definition named *visit-r7-r5* exists, and that the primary LSP should be routed according to the EROs in the specified path definition.

Once again, CSPF is disabled on the new LSP. This is because the Multi-Level IS-IS topology in the current test bed results in the lack of a domain-wide TED, which in the case of an LSP from r2 to r6 will result in CSPF failures. Note that the restrictions in this configuration task did not explicitly prohibit the use of CSPF, nor do they explicitly state that ERO-based constraints should be used to constrain the LSP's routing. The intent of this wording is to allow the candidate to sink into a mire of CSPF- and TED-related analysis, if they are so inclined. After all, a *true* expert will know when CSPF can, and cannot, be used:

```
[edit protocols mpls]
lab@r2# set label-switched-path r2-r6 no-cspf
```

You now define the *visit-r7-r5* path, taking care that the EROs specified will not cause the LSP to visit the same link or node twice, because doing so will cause the LSP's routing to fail due to Record Route Object (RRO)–based loop detection. In this example, you opt to use a combination of strict and loose hops that force the LSP to visit r4, then r7, then r5, and then the egress node, r6. There is no need to specify r6 in the ERO, because the egress node becomes the last "loose hop" in the routing of the RSVP Path message when the ERO list runs out. r4's loopback address does not exist in IS-IS Level 1 area 0001, so a strict hop pointing at its fe-0/0/1 interface starts the ERO list; note that EROs are considered strict by default. The last two hops are specified as loose to accommodate the indirect routing associated with loopback interfaces:

```
lab@r2# show
label-switched-path r2-r6 {
    to 10.0.9.6;
    no-cspf;
    primary visit-r7-r5;
}
path visit-r7-r5 {
    10.0.4.9;
    10.0.9.7 loose;
    10.0.3.5 loose;
}
interface all;
```

Verifying the ERO Constrained LSP

A few moments after the changes are committed at r2, the status of the LSP is displayed:

```
[edit protocols mpls]
lab@r2# run show mpls lsp detail
Ingress LSP: 1 sessions

10.0.9.6
  From: 10.0.6.2, State: Up, ActiveRoute: 1, LSPname: r2-r6
  ActivePath: visit-r7-r5 (primary)
  LoadBalance: Random
```

```
  Encoding type: Packet, Switching type: Packet, GPID: IPv4
 *Primary   visit-r7-r5      State: Up
    Received RRO:
         10.0.4.9 10.0.2.17 10.0.8.9 10.0.8.5
Total 1 displayed, Up 1, Down 0

Egress LSP: 0 sessions
Total 0 displayed, Up 0, Down 0

Transit LSP: 0 sessions
Total 0 displayed, Up 0, Down 0
```

The output confirms that the *r2–r6* LSP is up, and that it transits `r4`, `r7`, and `r5` on its way to `r6`, as required by the scenario's configuration criteria. The output from the `show rsvp session` command provides an interesting contrast in the form of the user-provided ERO vs. the contents of the Record Route Object (RRO) as contained in the RSVP Path and Reservation (RESV) messages themselves. Although they might seem different to a casual observer, both of the address listings indicate that the same series of nodes are crossed by the LSP:

```
[edit protocols mpls]
lab@r2# run show rsvp session detail
Ingress RSVP: 1 sessions

10.0.9.6
  From: 10.0.6.2, LSPstate: Up, ActiveRoute: 1
  LSPname: r2-r6, LSPpath: Primary
  Suggested label received: -, Suggested label sent: -
  Recovery label received: -, Recovery label sent: 100004
  Resv style: 1 FF, Label in: -, Label out: 100004
  Time left:    -,  Since: Fri Feb 14 11:21:57 2003
  Tspec: rate 0bps size 0bps peak Infbps m 20 M 1500
  Port number: sender 5 receiver 64567 protocol 0
  PATH rcvfrom: localclient
  PATH sentto: 10.0.4.9 (fe-0/0/1.0) 9 pkts
  RESV rcvfrom: 10.0.4.9  (fe-0/0/1.0) 9 pkts
  Explct route: 10.0.4.9 10.0.9.7 10.0.3.5
  Record route: <self>  10.0.4.9  10.0.2.17  10.0.8.9  10.0.8.5
Total 1 displayed, Up 1, Down 0

Egress RSVP: 0 sessions
Total 0 displayed, Up 0, Down 0

Transit RSVP: 0 sessions
Total 0 displayed, Up 0, Down 0
```

The output confirms that the *r2-r6* LSP has been successfully signaled, as indicated by the `Up` indication for the `LSPstate`. Further, the display confirms that the LSP's routing complies with the user-provided ERO constraints; the added highlights call out the contents of the user-provided ERO and the contents of the RRO.

As a final check, the LSP's forwarding plane is tested by tracing the route to a BGP destination whose next hop resolves to the 10.0.9.6 egress address of the *r2–r6* LSP. The 220.220/16 routes coming from C2 should be just the ticket in this case:

```
lab@r2> show route 220.220/16

inet.0: 125480 destinations, 125487 routes (125480 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

220.220.0.0/16     *[BGP/170] 08:12:49, MED 0, localpref 100, from 10.0.9.6
                      AS path: 65020 I
                    > to 10.0.4.9 via fe-0/0/1.0, label-switched-path r2-r6
```

As expected, the next hop self actions of `r6`'s IBGP export policy result in a BGP next hop of 10.0.9.6 for the 220.220/16 prefix advertisement within your AS. Because this next hop resolves through the `inet.3` routing table on `r2`, the *r2–r6* LSP has automatically been installed as the next hop for 220.220/16 destination. LSP forwarding is now confirmed:

```
lab@r2> traceroute 220.220.0.1
traceroute to 220.220.0.1 (220.220.0.1), 30 hops max, 40 byte packets
 1  10.0.4.9 (10.0.4.9)  0.712 ms  0.541 ms  0.433 ms
     MPLS Label=100004 CoS=0 TTL=1 S=1
 2  10.0.2.17 (10.0.2.17)  0.272 ms  0.194 ms  0.156 ms
     MPLS Label=100009 CoS=0 TTL=1 S=1
 3  10.0.8.9 (10.0.8.9)  0.565 ms  0.524 ms  0.489 ms
     MPLS Label=100005 CoS=0 TTL=1 S=1
 4  10.0.8.5 (10.0.8.5)  0.206 ms  0.199 ms  0.180 ms
 5  220.220.0.1 (220.220.0.1)  0.292 ms  0.274 ms  0.258 ms
```

The LSP's presence in the forwarding path is clearly indicated by the label-related output associated with hops 1 through 3. With the success of the traceroute, all aspects of the ERO constrained LSP scenario have been met.

Constrained Shortest Path First

In this section, you will use the CSPF algorithm to compute LSP paths through your network based on a variety of constraints. As mentioned previously, CSPF functions to locate optimal paths (Shortest Path First) after paths that fail to meet one or more user-provided constraints have been pruned from the SPF tree (constrained). As with link state routing protocols, the CSPF algorithm relies on a database that is shared among all routers in the TE domain. This database, which is known as the TED, is built through link state routing protocol extensions that allow for the flooding of information regarding available link bandwidth, link coloring, etc. In essence, the TED has everything contained in the OSPF or IS-IS database, and more!

Worthy of note is that the IS-IS routing protocol defaults to TE extension support while the OSPF protocol does not. Put another way, if you run IS-IS, you already have a TED, unless you have consciously decided to disable TE extensions. With OSPF, the exact opposite is true: you must issue a **`set protocols ospf traffic-engineering`** statement to instruct OSPF to build a TED.

The difference in default behavior stems from the nature of the two protocols, and the fact that OSPF requires a special LSA (opaque LSA type 10) to support TE extensions while IS-IS simply uses additional TLVs. Also of note is that you will be unable to use CSPF to calculate a full, end-to-end path, when the network contains multiple TE domains—that is, more than one TED view. Such a condition can arise when a network has multiple routing domains, or in the case of a Multi-Level or Multi-Area IS-IS or OSPF network, respectively, because TE information is not leaked between IS-IS levels or OSPF areas. A network that runs islands of OSPF and IS-IS, with BGP route redistribution between these islands, is an example of a network with multiple routing domains. In a network such as this, no router will have a TED that completely describes an end-to-end path through the network! A router can have only one TED; it is possible to use both OSPF and IS-IS to populate the same TED, however.

When building LSPs that cross multiple TE domains, your best bet is to disable CSPF and use plain old EROs. Note that JUNOS software does offer support for *LSP stitching*. This feature functions to "glue" together an end-to-end LSP that comprises multiple TE domain segments, each of which was established by an independent CSPF process.

To be effective with CSPF-based path calculation, you must be familiar with the contents of the TED, and you must be able to effectively perform CSPF-based tracing. This is because most network problems will simply manifest themselves as a local CSPF failure regarding the inability to locate a route to the host, which simply means that no path meeting the constraints set forth for the LSP in question can be found in the router's TED.

To resolve these problems effectively, your best bet involves TED analysis in an effort to locate the node that *should* meet the constraint in question, yet from the view of the TED, does not. Once so identified, you can concentrate your troubleshooting efforts on why that node is not being represented accurately in the TED.

A key point can be made here regarding the distinct roles played by RSVP, the IGP, and CSPF, in the context of bandwidth reservations. First, the IGP has no idea about the *actual* vs. *reservable* bandwidth supported by the various nodes and links in your network. RSVP, on the other hand, can request bandwidth reservations and control the level of oversubscription or undersubscription, but RSVP has no idea about the overall state of the network with regard to end-to-end reservable bandwidth.

Barring additional ERO constraints, RSVP will generate a path message requesting a given bandwidth reservation, and this request will propagate hop-by-hop according to the IGP's view of the shortest path, until the reservation either succeeds or a node/link is encountered that does not have sufficient reservable bandwidth to honor the reservation. In the latter case, LSP setup fails and RSVP will try to re-signal it after expiration of the retry timer. The use of CSPF in conjunction with the network state maintained by the TED would have resulted in no RSVP signaling attempts in this example. This is because the CSPF algorithm's ability to look at the entire network state, as represented by its local copy of the TED, will simply result in the ingress node's determination that there is "no route to host."

In many cases, the use of CSPF will be optional. This is to say that constrained LSP routing can be performed with EROs, as previously demonstrated. Features and capabilities that require

CSPF include the computation of Fast Reroute detours, link bypass, LSP re-optimization, and the use of link coloring. Link colors are sometimes referred to as *affinities* or *administration groups*. In many cases, the configuration requirements given to a candidate will not mandate the use of CSPF, such that the candidate is free to decide how they prefer to constrain the routing of their RSVP signaled LSPs.

With the review of CSPF and online path calculation complete, it is time to get back to work. To complete the CSPF configuration scenario, you must establish an LSP that conforms to the following stipulations:

- Establish LSP *r4–r3*, with `r4` as the ingress and `r3` as the egress.
- Color the 10.0.2.4/30 link between `r4` and `r3` as *`blue`*.
- Ensure that the *r4–r3* LSP never uses blue links.

Configuring Link Coloring

The requirement that LSP routing be constrained by link coloring indicates that you must use CSPF for this configuration scenario. Note that the LSP being established is confined to the L2 backbone, which represents a single TE domain. Confining the LSP to backbone routers will therefore allow the CSPF algorithm to succeed.

You begin by establishing a link coloring plan. In this case, you decide to color the 10.0.2.8/30 and 10.0.2.0/30 links as *`red`* while making the 10.0.2.4/30 link *`blue`* in accordance with the restrictions imposed. The addition of *`red`* coloring is not strictly necessary in this example; you could merely color the 10.0.2.4/30 link as *`blue`* and then exclude *`blue`* links to force the LSP to transit `r5`. In this example, you will use a combination of `exclude` and `include` statements to play it extra safe. You must also decide on the set of numeric values that will mapped to the user-friendly *`blue`* and *`red`* mnemonics. Use care to ensure that the color-to-numeric mappings are entered consistently on all routers to avoid problems and confusion down the road!

Although the actual spelling of the mnemonic color is a purely local matter, it is highly recommended that you use consistent spelling and case to minimize confusion should you later find yourself analyzing the contents of the TED. Figure 2.4 details the topology for the CSPF configuration scenario.

FIGURE 2.4 CSPF topology and link coloring

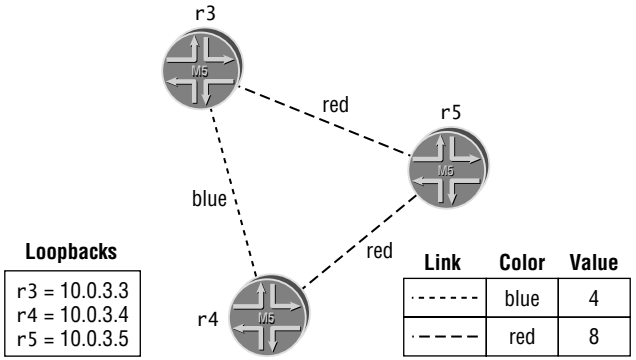


Table 2.1 shows your choice of numeric-to-mnemonic color mappings:

TABLE 2.1 Numeric to Color Mappings

Color	Numeric Value	Comment
blue	4	Color for the direct link between r4 and r3
red	8	Color for remaining core links

Note that the numeric values that are assigned to represent the link colors are based on a bit-mask. This is to say that *blue* will be 0010 while *red* is 0100. Note that the bit 0 is not available for use in link coloring, such that all binary values are offset by 1, which makes a decimal "4" equal to a binary "10," as opposed to "100." If a link were to be colored as both *red and blue*, the resulting bit-mask would be 0110. The following commands correctly define the color-to-numeric mappings at r3:

```
[edit protocols mpls]
lab@r3# set admin-groups blue 4

[edit protocols mpls]
lab@r3# set admin-groups red 8
```

The new administration groups are now displayed:

```
[edit protocols mpls]
lab@r3# show
admin-groups {
    blue 4;
    red 8;
}
interface at-0/1/0.0;
interface fe-0/0/0.0;
interface fe-0/0/1.0;
interface fe-0/0/3.0;
interface so-0/2/0.100;
```

With the groups defined on r3, you now associate each of its core interfaces with the correct administrative group:

```
[edit protocols mpls]
lab@r3# set interface so-0/2/0.100 admin-group blue

[edit protocols mpls]
lab@r3# set interface at-0/1/0.0 admin-group red

[edit protocols mpls]
lab@r3# show
```

```
admin-groups {
    blue 4;
    red 8;
}
interface at-0/1/0.0 {
    admin-group red;
}
interface fe-0/0/0.0;
interface fe-0/0/1.0;
interface fe-0/0/3.0;
interface so-0/2/0.100 {
    admin-group blue;
}
```

After committing the changes, you can easily check your interface to link color associations, as shown here:

```
[edit protocols mpls]
lab@r3# run show mpls interface
Interface        State       Administrative groups
fe-0/0/0.0       Up          <none>
fe-0/0/1.0       Up          <none>
fe-0/0/3.0       Up          <none>
at-0/1/0.0       Up           red
so-0/2/0.100     Up           blue
```

Before moving on, you decide to inspect the TED on `r3` to verify that the link coloring is accurately reported:

```
[edit protocols mpls]
lab@r3# run show ted database extensive r3.00
TED database: 14 ISIS nodes 7 INET nodes
NodeID: r3.00(10.0.3.3)
  Type: Rtr, Age: 219 secs, LinkIn: 5, LinkOut: 5
  Protocol: IS-IS(2)
    To: r5.00(10.0.3.5), Local: 10.0.2.2, Remote: 10.0.2.1
      Color: 0x100 red
      Metric: 10
      Static BW: 155.52Mbps
      Reservable BW: 155.52Mbps
      Available BW [priority] bps:
       [0] 145.52Mbps   [1] 145.52Mbps   [2] 145.52Mbps   [3] 145.52Mbps
       [4] 145.52Mbps   [5] 145.52Mbps   [6] 145.52Mbps   [7] 145.52Mbps
      Interface Switching Capability Descriptor(1):
        Switching type: Packet
        Encoding type: Packet
```

```
      Maximum LSP BW [priority] bps:
       [0] 145.52Mbps   [1] 145.52Mbps   [2] 145.52Mbps   [3] 145.52Mbps
       [4] 145.52Mbps   [5] 145.52Mbps   [6] 145.52Mbps   [7] 145.52Mbps
  To: r6.03, Local: 10.0.2.14, Remote: 0.0.0.0
    Color: 0 <none>
    Metric: 10
    Static BW: 100Mbps
    Reservable BW: 100Mbps
    Available BW [priority] bps:
     [0] 100Mbps      [1] 100Mbps      [2] 100Mbps      [3] 100Mbps
     [4] 100Mbps      [5] 100Mbps      [6] 100Mbps      [7] 100Mbps
    Interface Switching Capability Descriptor(1):
      Switching type: Packet
      Encoding type: Packet
      Maximum LSP BW [priority] bps:
       [0] 100Mbps      [1] 100Mbps      [2] 100Mbps      [3] 100Mbps
       [4] 100Mbps      [5] 100Mbps      [6] 100Mbps      [7] 100Mbps
  To: r4.00(10.0.3.4), Local: 10.0.2.5, Remote: 10.0.2.6
    Color: 0x10 blue
    Metric: 10
    Static BW: 155.52Mbps
    Reservable BW: 155.52Mbps
    Available BW [priority] bps:
     [0] 155.52Mbps   [1] 155.52Mbps   [2] 155.52Mbps   [3] 155.52Mbps
     [4] 155.52Mbps   [5] 155.52Mbps   [6] 155.52Mbps   [7] 155.52Mbps
    Interface Switching Capability Descriptor(1):
      Switching type: Packet
      Encoding type: Packet
      Maximum LSP BW [priority] bps:
       [0] 155.52Mbps   [1] 155.52Mbps   [2] 155.52Mbps   [3] 155.52Mbps
       [4] 155.52Mbps   [5] 155.52Mbps   [6] 155.52Mbps   [7] 155.52Mbps
Protocol: IS-IS(1)
  To: r2.03, Local: 10.0.4.1, Remote: 0.0.0.0
    Color: 0 <none>
    Metric: 10
    Static BW: 100Mbps
    Reservable BW: 100Mbps
    Available BW [priority] bps:
     [0] 100Mbps      [1] 100Mbps      [2] 100Mbps      [3] 100Mbps
     [4] 100Mbps      [5] 100Mbps      [6] 100Mbps      [7] 100Mbps
    Interface Switching Capability Descriptor(1):
      Switching type: Packet
```

```
      Encoding type: Packet
      Maximum LSP BW [priority] bps:
       [0] 100Mbps      [1] 100Mbps      [2] 100Mbps      [3] 100Mbps
       [4] 100Mbps      [5] 100Mbps      [6] 100Mbps      [7] 100Mbps
  To: r1.02, Local: 10.0.4.13, Remote: 0.0.0.0
    Color: 0 <none>
    Metric: 10
    Static BW: 100Mbps
    Reservable BW: 100Mbps
    Available BW [priority] bps:
     [0] 100Mbps      [1] 100Mbps      [2] 100Mbps      [3] 100Mbps
     [4] 100Mbps      [5] 100Mbps      [6] 100Mbps      [7] 100Mbps
    Interface Switching Capability Descriptor(1):
      Switching type: Packet
      Encoding type: Packet
      Maximum LSP BW [priority] bps:
       [0] 100Mbps      [1] 100Mbps      [2] 100Mbps      [3] 100Mbps
       [4] 100Mbps      [5] 100Mbps      [6] 100Mbps      [7] 100Mbps
```

The display confirms that r3's contribution to the TED is accurate with regard to its link coloring values. You should take a few minutes to marvel at the wealth of information contained in the TED entry for a given node. With r3's operation confirmed, similar changes are now made to r4 and r5. The changes made to r5's configuration are highlighted here:

```
[edit protocols mpls]
lab@r5# show
admin-groups {
    blue 4;
    red 8;
}
interface all;
interface at-0/2/1.0 {
    admin-group red;
}
interface so-0/1/0.0 {
    admin-group red;
}
```

You confirm that all core links at r3, r4, and r5 are correctly colored before proceeding to the next section.

Configuring and Verifying a CSPF Constrained LSP

With administrative group coloring in place, you now define the color-constrained LSP on r4:

```
[edit protocols mpls]
lab@r4# set label-switched-path r4-r3 to 10.0.3.3
```

```
[edit protocols mpls]
lab@r4# set label-switched-path r4-r3 admin-group include red

[edit protocols mpls]
lab@r4# set label-switched-path r4-r3 admin-group exclude blue
```

The first command in this sequence defines the LSP's name and egress point. The next two commands associate this LSP with the need to include *red* links while also excluding *blue* links. The resulting LSP configuration is displayed next:

```
[edit protocols mpls]
lab@r4# show label-switched-path r4-r3
to 10.0.3.3;
admin-group {
    include red;
    exclude blue;
}
```

After a commit, the RSVP session status for the new LSP is displayed:

```
lab@r4> show rsvp session ingress detail
Ingress RSVP: 1 sessions

10.0.3.3
  From: 10.0.3.4, LSPstate: Up, ActiveRoute: 125496
  LSPname: r4-r3, LSPpath: Primary
  Suggested label received: -, Suggested label sent: -
  Recovery label received: -, Recovery label sent: 100006
  Resv style: 1 FF, Label in: -, Label out: 100006
  Time left:    -,  Since: Fri Feb 14 19:19:18 2003
  Tspec: rate 0bps size 0bps peak Infbps m 20 M 1500
  Port number: sender 1 receiver 42938 protocol 0
  PATH rcvfrom: localclient
  PATH sentto: 10.0.2.9 (so-0/1/1.0) 9 pkts
  RESV rcvfrom: 10.0.2.9  (so-0/1/1.0) 8 pkts
  Explct route: 10.0.2.9 10.0.2.2
  Record route: <self>  10.0.2.9  10.0.2.2
Total 1 displayed, Up 1, Down 0
```

The output confirms LSP establishment, and that the LSP's routing correctly avoids the forbidden *blue* link between `r3` and `r4`. Additional confirmation regarding color-based constraints is provided in the output of the `show mpls lsp extensive` command:

```
lab@r4> show mpls lsp extensive name r4-r3
Ingress LSP: 1 sessions

10.0.3.3
  From: 10.0.3.4, State: Up, ActiveRoute: 125510, LSPname: r4-r3
```

```
  ActivePath:  (primary)
  LoadBalance: Random
  Encoding type: Packet, Switching type: Packet, GPID: IPv4
*Primary                    State: Up
    Include: red    Exclude: blue
    Computed ERO (S [L] denotes strict [loose] hops): (CSPF metric: 20)
          10.0.2.9 S 10.0.2.2 S
    Received RRO:
          10.0.2.9 10.0.2.2
    5 Feb 14 19:19:18  Selected as active path
    4 Feb 14 19:19:18  Record Route:  10.0.2.9 10.0.2.2
    3 Feb 14 19:19:18  Up
    2 Feb 14 19:19:18  Originate Call
    1 Feb 14 19:19:18  CSPF: computation result accepted
  Created: Fri Feb 14 19:13:50 2003
Total 1 displayed, Up 1, Down 0

Egress LSP: 0 sessions
Total 0 displayed, Up 0, Down 0

Transit LSP: 1 sessions
Total 0 displayed, Up 0, Down 0
```

The operational output shown in this section confirms that all aspects of the CSPF configuration scenario have been met.

Real World Scenario

CSPF Troubleshooting

This real world scenario will demonstrate techniques that are useful when troubleshooting CSPF-related problems. To simulate a configuration error, the link coloring has been removed from r5's at-0/2/1 interface. The result is a CSPF failure at r4 because there is no longer a path between r4 and r3 including the *red* color. You confirm that r4 is unable to reestablish the *r4–r3* LSP after clearing all ingress LSPs at r4:

```
lab@r4> clear mpls lsp

lab@r4> show mpls lsp extensive name r4-r3
Ingress LSP: 1 sessions

10.0.3.3
  From: 0.0.0.0, State: Dn, ActiveRoute: 0, LSPname: r4-r3
  ActivePath: (none)
  LoadBalance: Random
```

```
    Encoding type: Packet, Switching type: Packet, GPID: IPv4
    Primary                     State: Dn
      Include: red    Exclude: blue
      Will be enqueued for recomputation in 27 second(s).
      8 Feb 14 19:31:00  CSPF failed: no route toward 10.0.3.3
      7 Feb 14 19:31:00  Clear Call
      6 Feb 14 19:31:00  Deselected as active
      5 Feb 14 19:19:18  Selected as active path
      4 Feb 14 19:19:18  Record Route:  10.0.2.9 10.0.2.2
      3 Feb 14 19:19:18  Up
      2 Feb 14 19:19:18  Originate Call
      1 Feb 14 19:19:18  CSPF: computation result accepted
    Created: Fri Feb 14 19:19:00 2003
  Total 1 displayed, Up 0, Down 1

  Egress LSP: 0 sessions
  Total 0 displayed, Up 0, Down 0

  Transit LSP: 1 sessions
  Total 0 displayed, Up 0, Down 0
```

Note that the CSPF failure is conveyed in the form of a rather generic message that simply states there is `no route toward 10.0.3.3`. The use of CSPF tracing on the ingress node often provides you with an indication of the problem. A typical CSPF tracing configuration is shown here, along with the trace output for a failed CSPF run.

```
[edit protocols mpls]
lab@r4# show traceoptions
file cspf;
flag cspf;
flag cspf-link;
flag cspf-node;
lab@r4# run monitor start cspf

lab@r4# run clear mpls lsp

[edit protocols mpls]
lab@r4#
*** cspf ***
Feb 14 19:46:06       CSPF adding path r4-r3(primary ) to CSPF queue 1
Feb 14 19:46:06       CSPF creating CSPF job
Feb 14 19:46:06       CSPF job starting
Feb 14 19:46:06       CSPF for path r4-r3(primary ), starting at r4.00
Feb 14 19:46:06         path include: 0x00000100
Feb 14 19:46:06         path exclude: 0x00000010
Feb 14 19:46:06         bandwidth: 0bps; setup priority: 0; random
Feb 14 19:46:06       CSPF final destination 10.0.3.3
Feb 14 19:46:06       CSPF starting from r4.00 (10.0.3.4) to 10.0.3.3, hoplimit 254
Feb 14 19:46:06         constrains include 0x00000100
Feb 14 19:46:06         constrains exclude 0x00000010
Feb 14 19:46:06       Node r4.00 (10.0.3.4) metric 0, hops 0,
                       avail 32000 32000 32000 32000
Feb 14 19:46:06       Link 10.0.2.6->10.0.2.5(r3.00/10.0.3.3) metric 10 color
                         0x00000010 bw 155.52Mbps
Feb 14 19:46:06       Reverse Link for 10.0.2.6->10.0.2.5 is 10.0.2.5->10.0.2.6
Feb 14 19:46:06         link fails include 0x00000100
Feb 14 19:46:06       Link 10.0.2.10->10.0.2.9(r5.00/10.0.3.5) metric 10 color
                         0x00000100 bw 155.52Mbps
```

```
Feb 14 19:46:06       Reverse Link for 10.0.2.10->10.0.2.9 is 10.0.2.9->10.0.2.10
Feb 14 19:46:06         link's interface switch capability descriptor #1
Feb 14 19:46:06           encoding: Packet, switching: Packet
Feb 14 19:46:06         link passes constraints
Feb 14 19:46:06       Link 10.0.4.17->0.0.0.0(r1.04/0.0.0.0) metric 10 color
                         0x00000000 bw 100Mbps
Feb 14 19:46:06       Reverse Link for 10.0.4.17->0.0.0.0 is 0.0.0.0->0.0.0.0
Feb 14 19:46:06         link fails include 0x00000100
Feb 14 19:46:06       Link 10.0.2.18->0.0.0.0(r4.04/0.0.0.0) metric 10 color
                         0x00000000 bw 100Mbps
Feb 14 19:46:06       Reverse Link for 10.0.2.18->0.0.0.0 is 0.0.0.0->0.0.0.0
Feb 14 19:46:06         link fails include 0x00000100
Feb 14 19:46:06       Link 10.0.4.9->0.0.0.0(r2.02/0.0.0.0) metric 10 color
                         0x00000000 bw 100Mbps
Feb 14 19:46:06       Reverse Link for 10.0.4.9->0.0.0.0 is 0.0.0.0->0.0.0.0
Feb 14 19:46:06         link fails include 0x00000100
Feb 14 19:46:06     Node r5.00 (10.0.3.5) metric 10, hops 1,
                       avail 32000 32000 32000 32000
Feb 14 19:46:06       Link 10.0.2.9->10.0.2.10(r4.00/10.0.3.4) metric 10 color
                         0x00000100 bw 155.52Mbps
Feb 14 19:46:06         skipped: end point already visited
Feb 14 19:46:06       Link 10.0.2.1->10.0.2.2(r3.00/10.0.3.3) metric 10 color
                         0x00000000 bw 155.52Mbps
Feb 14 19:46:06       Reverse Link for 10.0.2.1->10.0.2.2 is 10.0.2.2->10.0.2.1
Feb 14 19:46:06         link fails include 0x00000100
Feb 14 19:46:06 CSPF completed in 0.001409s
Feb 14 19:46:06 CSPF couldn't find a route to 10.0.3.3
Feb 14 19:46:06 CSPF job done!

*** monitor and syslog output disabled, press ESC-Q to enable ***
```

The highlights in the CSPF tracing output call out that the 10.0.2.8/30 link between r4 and r5 passes the administrative group constraints, while the 10.0.2.0/30 link between r5 and r3 does not (this is the link with no color assigned). Though not highlighted, you can also see that the 10.0.2.4/30 link between r4 and r3 does not meet the `include` constraint either. Another useful approach for troubleshooting TED problems involves the selective filtering of the TED's contents, which is designed to allow you to focus on what is missing. The following example makes heavy use of JUNOS CLI matching functionality to achieve this goal. Note that this command can be entered on any router in the same TE domain because the contents of the TED should be consistent among all routers within a given TE domain.

```
lab@r3> show ted database extensive r5.00 | match "(NodeID|To:|Color)"
NodeID: r5.00(10.0.3.5)
    To: r3.00(10.0.3.3), Local: 10.0.2.1, Remote: 10.0.2.2
      Color: 0 <none>
    To: r4.00(10.0.3.4), Local: 10.0.2.9, Remote: 10.0.2.10
      Color: 0x100 red
```

The filtered output, which was obtained at r3, clearly indicates that only one of r5's core interfaces is correctly associated with a link color. It is interesting to note that r5's Level 1 interfaces are not represented in r3's TED. This is due to the fact that TE extensions are not leaked between IS-IS levels, making r5's Level 1 interfaces unknown to the L2 TED. Before moving on, r5's link coloring configuration is restored, allowing the reestablishment of the color constrained LSP.

RSVP Summary

This section provided details on the configuration and verification of constrained and unconstrained RSVP signaled LSPs. In this section, you learned how to create a RSVP instance, how to enable RSVP support on a particular interface, and how to define a LSP that reserves a specified amount of bandwidth along its path. Subsequent examples demonstrated how to use EROs, and CSPF-related administrative groups, to constrain the routing of a RSVP signaled LSP.

You were exposed to a variety of operational mode commands that are useful in determining the operational status of a RSVP LSP. This section also demonstrated how RSVP and CSPF tracing can be used to assist in troubleshooting RSVP-related control plane problems.

Routing Table Integration

This section provides several configuration scenarios designed to demonstrate common LSP routing table integration options. To begin, it should be noted that the default LSP routing table integration rules serve to make LSPs invisible for internal traffic because internal traffic is aware only of the `inet.0` routing table and signaled LSPs are placed into the `inet.3` table by default. Only BGP is aware of the `inet.3` routing table, and BGP uses the `inet.3` table only for purposes of resolving BGP next hops. The default LSP routing table integration behavior has been demonstrated previously in this chapter; in summary, it can be stated that only traffic associated with a BGP next hop, which resolves through the `inet.3` table, will be aware of, and therefore be able to make use of, signaled LSPs.

When multiple equal-cost LSPs exist to the same destination, the default behavior is to randomly balance traffic over the group of LSPs on a per-prefix basis. This section also demonstrates how this default behavior can be modified.

Installing Prefixes

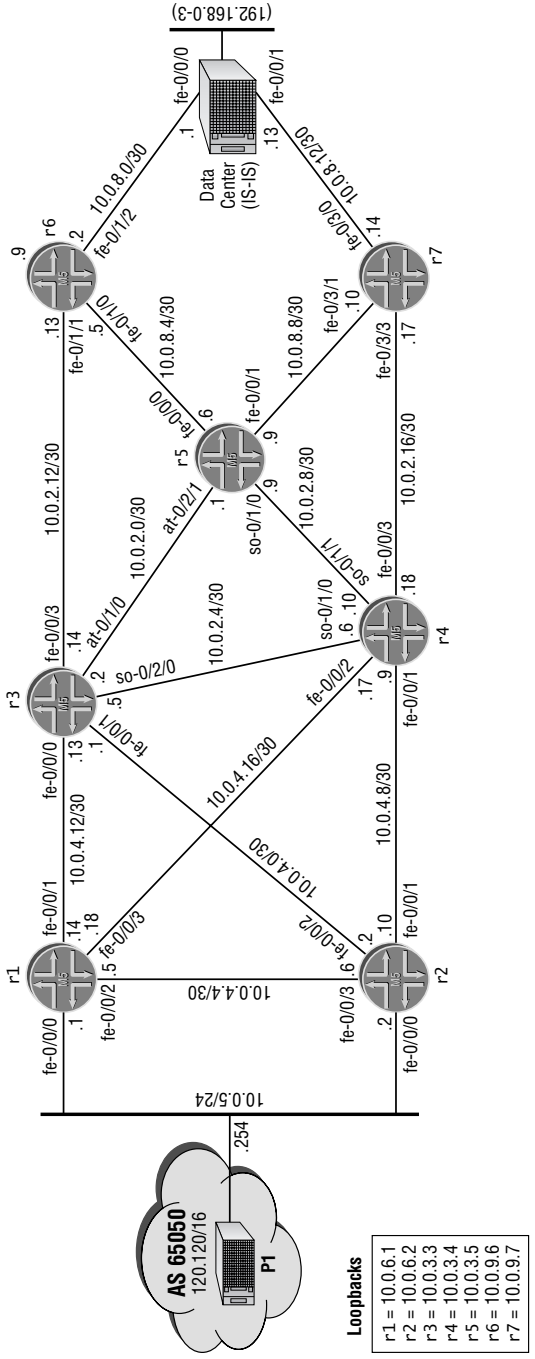
By default, the only prefix installed in the `inet.3` routing table is the /32 address associated with the LSP endpoint. You can add additional prefixes to the `inet.3` table using the `install` keyword when defining the LSP. Once such a prefix has been placed into the `inet.3` routing table, it becomes available for use in BGP next hop resolution. Including the `active` keyword when installing a prefix results in the prefix-to-LSP mapping being installed in the main `inet.0` routing table, where it can be used for both external (BGP) and internal (IGP) destinations. The following configuration task demonstrates the former, "BGP use only," scenario.

To complete this task, you must configure an LSP that meets these requirements:

- Build LSP *r6–r1*, with `r6` as the ingress that transits `r5`.
- Ensure that traffic to 120.120/16 destinations uses the *r6–r1* LSP.
- You must not add a next hop self policy at `r1` to meet the LSP forwarding requirements.

Refer to Figure 2.5 for topology details as needed.

FIGURE 2.5 Routing table integration



The tricky aspect of this task relates to the fact that your LSP terminates on r1, but the BGP next hop associated with the 120.120/16 routes is not being overwritten to reflect r1's RID. This means that the 10.0.5.254 BGP next hop associated with the 120.120/16 route cannot be resolved through the inet.3 table at r6, and therefore traffic to P1 will not be forwarded through the *r6–r1* LSP. To meet the requirements posed, you will have to find a way to get P1's BGP next hop into r6's inet.3 table because you are not able to rewrite the BGP next hop at r1.

You begin by defining a baseline LSP at r6; no prefixes are installed at this time to better demonstrate the *before* and *after* operation:

```
[edit protocols mpls]
lab@r6# set label-switched-path r6-r1 to 10.0.6.1 no-cspf

[edit protocols mpls]
lab@r6# set label-switched-path r6-r1 primary use-r5

[edit protocols mpls]
lab@r6# set path use-r5 10.0.3.5 loose
```

The LSP configuration is now displayed. Note that CSPF has been disabled (CSPF would be condemned to fail due to the presence of multiple TE domains in the current test bed) and that the routing of the LSP is constrained through an ERO to transit r5:

```
[edit protocols mpls]
lab@r6# show
label-switched-path r6-r1 {
    to 10.0.6.1;
    no-cspf;
    primary use-r5;
}
path use-r5 {
    10.0.3.5 loose;
}
interface all;
```

Once committed, the *r6–r1* LSP is confirmed operational:

```
[edit protocols mpls]
lab@r6# run show mpls lsp ingress
Ingress LSP: 1 sessions
To              From            State Rt ActivePath       P     LSPname
10.0.6.1        10.0.9.6        Up     0                  *     r6-r1
Total 1 displayed, Up 1, Down 0
```

The contents of the inet.3 table is displayed at r6:

```
[edit protocols mpls]
lab@r6# run show route table inet.3
```

```
inet.3: 3 destinations, 3 routes (3 active, 0 holddown, 0 hidden)
Restart Complete
+ = Active Route, - = Last Active, * = Both

10.0.3.5/32        *[LDP/9] 00:03:44, metric 1
                    > to 10.0.8.6 via fe-0/1/0.0
10.0.6.1/32        *[RSVP/7] 00:04:10, metric 20
                    > to 10.0.2.14 via fe-0/1/1.0, label-switched-path r6-r1
10.0.9.7/32        *[LDP/9] 00:03:44, metric 1
                    > to 10.0.8.6 via fe-0/1/0.0, Push 100001
```

As expected, the *r6–r1* LSP has been installed in r6's `inet.3` routing table for use in resolving BGP next hops that match the LSP's egress address of 10.0.6.1. Inspection of the BGP next hop currently attached to the 120.120/16 route, as advertised into your AS by r1, clearly indicates why `inet.3`-based BGP next hop resolution for this route cannot succeed at r6:

```
[edit protocols mpls]
lab@r6# run show route 120.120/16 detail

inet.0: 125587 destinations, 125593 routes (125587 active, 0 holddown, 0 hidden)
Restart Complete
120.120.0.0/16 (2 entries, 1 announced)
        *BGP    Preference: 170/-101
                Source: 10.0.6.1
                Next hop: 10.0.2.14 via fe-0/1/1.0, selected
                Protocol next hop: 10.0.5.254 Indirect next hop: 84cfc78 69
                State: <Active Int Ext>
                Local AS: 65412 Peer AS: 65412
                Age: 4:24:31    Metric: 0       Metric2: 30
                Task: BGP_65412.10.0.6.1+179
                Announcement bits (4): 0-KRT 1-LDP 6-BGP.0.0.0.0+179 7-Resolve
                   inet.0
                AS path: 65050 I
                Localpref: 100
                Router ID: 10.0.6.1
         BGP    Preference: 170/-101
                Source: 10.0.6.2
                Next hop: 10.0.2.14 via fe-0/1/1.0, selected
                Protocol next hop: 10.0.5.254 Indirect next hop: 84cfc78 69
                State: <NotBest Int Ext>
                Inactive reason: Router ID
                Local AS: 65412 Peer AS: 65412
                Age: 4:24:23    Metric: 0       Metric2: 30
```

```
                Task: BGP_65412.10.0.6.2+179
                AS path: 65050 I
                Localpref: 100
                Router ID: 10.0.6.2
```

The 10.0.5.254 protocol next hop in the command's output reflects the fact that r1 is not overwriting the BGP next hop advertised to it from the P1 router. The absence of the 10.0.5.254 prefix in r6's inet.3 table means that the *r6–r1* LSP will not be used when forwarding traffic to P1 destinations. This lack of LSP forwarding is confirmed at r6:

```
[edit protocols mpls]
lab@r6# run traceroute 120.120.0.1
traceroute to 120.120.0.1 (120.120.0.1), 30 hops max, 40 byte packets
 1  10.0.2.14 (10.0.2.14)  0.591 ms  0.372 ms  0.283 ms
 2  10.0.4.14 (10.0.4.14)  0.160 ms  0.160 ms  0.128 ms
 3  120.120.0.1 (120.120.0.1)  0.240 ms  0.220 ms  0.207 ms
```

As predicted, traffic to 120.120/16 is not mapped to the *r6–r1* LSP. You now modify r6's configuration to effect the installation of the 10.0.5.254/32 prefix into its inet.3 routing table:

```
[edit protocols mpls]
lab@r6# set label-switched-path r6-r1 install 10.0.5.254

[edit]
lab@r6# show protocols mpls
label-switched-path r6-r1 {
    to 10.0.6.1;
    install 10.0.5.254/32;
    no-cspf;
    primary use-r5;
}
path use-r5 {
    10.0.3.5 loose;
}
interface all;
```

Proper operation is confirmed after committing the changes on r6:

```
[edit protocols mpls]
lab@r6# run show route table inet.3

inet.3: 4 destinations, 4 routes (4 active, 0 holddown, 0 hidden)
Restart Complete
+ = Active Route, - = Last Active, * = Both

10.0.3.5/32        *[LDP/9] 00:11:36, metric 1
                    > to 10.0.8.6 via fe-0/1/0.0
```

```
10.0.5.254/32      *[RSVP/7] 00:00:43, metric 20
                    > to 10.0.2.14 via fe-0/1/0.0, label-switched-path r6-r1
10.0.6.1/32        *[RSVP/7] 00:00:43, metric 20
                    > to 10.0.2.14 via fe-0/1/0.0, label-switched-path r6-r1
10.0.9.7/32        *[LDP/9] 00:11:36, metric 1
                    > to 10.0.8.6 via fe-0/1/0.0, Push 100001
```

Excellent! The 10.0.5.254 BGP next hop, as associated with P1's 120.120/16 route, is now resolvable through r6's inet.3 table, which should result in r6 using the *r6–r1* LSP when forwarding to 120.120/16 destinations. Traceroute testing now confirms the required LSP-forwarding behavior:

```
[edit protocols mpls]
lab@r6# run traceroute 120.120.0.1
traceroute to 120.120.0.1 (120.120.0.1), 30 hops max, 40 byte packets
 1  10.0.8.6 (10.0.8.6)  0.736 ms  0.562 ms  0.438 ms
     MPLS Label=100026 CoS=0 TTL=1 S=1
 2  10.0.2.10 (10.0.2.10)  0.657 ms  0.500 ms  0.444 ms
     MPLS Label=100031 CoS=0 TTL=1 S=1
 3  10.0.4.18 (10.0.4.18)  0.158 ms  0.152 ms  0.137 ms
 4  120.120.0.1 (120.120.0.1)  0.245 ms  0.242 ms  0.215 ms
```

Installing Prefixes as Active

You will now modify r6's configuration so that the *r6–r1* LSP is used for both external (120.120/16) and internal (10.0.5/24) destinations, according to this additional criterion:

- Ensure that traffic sent from r6 to the 10.0.5/25 subnet also takes the *r6–r1* LSP.

To meet this requirement, you must install the *r6–r1* LSP into the main inet.0 routing table so that the LSP can be used for both non-BGP traffic *and* for BGP next hop resolution (BGP resolves its next hops through both inet.3 and inet.0, with the preference being inet.3-based resolutions). The next statements result in the 10.0.5/24 entry being moved from inet.3 into the inet.0 table:

```
[edit protocols mpls]
lab@r6# delete label-switched-path r6-r1 install 10.0.5.254

[edit protocols mpls]
lab@r6# set label-switched-path r6-r1 install 10.0.5/24 active

[edit]
lab@r6# show protocols mpls
label-switched-path r6-r1 {
    to 10.0.6.1;
    install 10.0.5.0/24 active;
```

```
    no-cspf;
    primary use-r5;
}
path use-r5 {
    10.0.3.5 loose;
}
interface all;
```

Verifying Active Prefixes

After the changes are committed, the presence of the *r6–r1* LSP is confirmed in r6's `inet.0` table:

```
[edit]
lab@r6# run show route 10.0.5/24

inet.0: 125624 destinations, 125631 routes (125624 active, 0 holddown, 0 hidden)
Restart Complete
+ = Active Route, - = Last Active, * = Both

10.0.5.0/24         *[RSVP/7] 00:01:26, metric 20
                     > to 10.0.8.6 via fe-0/1/0.0, label-switched-path r6-r1
                    [IS-IS/18] 00:14:52, metric 30
                     > to 10.0.2.14 via fe-0/1/1.0
```

To provide final verification that all requirements have been met, traceroutes are conducted to 10.0.5/24 destinations as well as to the EBGP destinations that are associated with P1:

```
lab@r6> traceroute 10.0.5.1
traceroute to 10.0.5.1 (10.0.5.1), 30 hops max, 40 byte packets
 1  10.0.8.6 (10.0.8.6)  0.802 ms  0.610 ms  0.490 ms
     MPLS Label=100026 CoS=0 TTL=1 S=1
 2  10.0.2.10 (10.0.2.10)  0.663 ms  0.525 ms  0.449 ms
     MPLS Label=100031 CoS=0 TTL=1 S=1
 3  10.0.5.1 (10.0.5.1)  0.169 ms  0.163 ms  0.139 ms

lab@r6> traceroute 120.120.0.1
traceroute to 120.120.0.1 (120.120.0.1), 30 hops max, 40 byte packets
 1  10.0.8.6 (10.0.8.6)  0.638 ms  0.476 ms  0.453 ms
     MPLS Label=100026 CoS=0 TTL=1 S=1
 2  10.0.2.10 (10.0.2.10)  0.521 ms  0.467 ms  0.622 ms
     MPLS Label=100031 CoS=0 TTL=1 S=1
 3  10.0.4.18 (10.0.4.18)  0.170 ms  0.170 ms  0.140 ms
 4  120.120.0.1 (120.120.0.1)  0.253 ms  0.237 ms  0.219 ms
```

The presence of LSP forwarding for the specified internal and external destinations confirms that you have met the requirements of this configuration scenario.

Use care when configuring traffic engineering for internal destinations. In some cases, forwarding internal traffic over an LSP can cause things to break. Although not a problem with the JUNOS software version 5.6R1 used to develop this book, installing a router's loopback address as `active` has been known to break RSVP signaling in previous software versions. In these earlier versions, another router's RSVP Path message that identifies `r1` as the egress node could be forwarded through the *r6–r1* LSP (due to LSPs being preferred over an equivalent IGP route). However, the lack of valid RRO entries in the resulting path message will result in RSVP signaling failures.

Traffic Engineering Shortcuts

Traffic engineering shortcuts provide a functionality that is similar to that of using *`install`*. The main difference is that TE shortcuts result in the *automatic* installation of all prefixes that are considered *downstream* of the LSP egress point into the `inet.3` or `inet.0` routing table based on the specifics of the configuration. Put another way, you might analogize `install` with the use of a scalpel that provides precise control over what routes are added to `inet.3` or `inet.0`, while TE shortcuts are more akin to using a chain saw.

The automatic installation of downstream prefixes relies on the presence of a link state routing protocol with complete knowledge of the routing domain. This is because downstream prefixes are computed by determining what routes would be reached on a SPF tree, when the LSP is considered as a point-to-point link between the ingress and egress nodes. Keep an eye out for networks with partial Link State Databases (LSDBs), such as in the case of the Multi-Level IS-IS topology currently in use, because their lack of complete topology knowledge will likely result in problems with TE shortcuts.

To complete this scenario, you must modify the configuration of `r6` according to these stipulations:

- Do not use `install` at `r6`.
- Ensure that traffic sent to the 10.0.5/24 *and* 120.120/16 destinations is forwarded over an LSP that transits `r5`.

The restriction on the use of `install` means that you will need to use TE shortcuts to meet the requirements of this scenario. The fact that `r6` does not receive IS-IS L1 LSPs from `r1` makes this task problematic. With the current *r6–r1* LSP, TE shortcuts will have no effect because `r6` does not see `r1` as a node on its shortest path tree. This is demonstrated in the output shown next:

```
lab@r6# run show isis database r1
IS-IS level 1 link-state database:
  0 LSPs

IS-IS level 2 link-state database:
  0 LSPs
```

```
[edit]
lab@r6# run show isis database r3
IS-IS level 1 link-state database:
  0 LSPs

IS-IS level 2 link-state database:
LSP ID                  Sequence Checksum Lifetime Attributes
r3.00-00                    0x27   0xeea5      995 L1 L2
  1 LSPs
```

To achieve the goals of this scenario, you need to redefine the LSP's egress to be r3, because r3 does appear in r6's shortest path tree. You begin by removing the *r1–r6* LSP configuration at r6:

```
[edit protocols mpls]
lab@r6# delete label-switched-path r6-r1
```

A new LSP that terminates on r3 is now defined. Note that this LSP still references the existing *use-r5* path:

```
[edit protocols mpls]
lab@r6# set label-switched-path r6-r3 to 10.0.3.3 no-cspf

[edit protocols mpls]
lab@r6# set label-switched-path r6-r3 primary use-r5
```

The new LSP configuration is displayed:

```
[edit protocols mpls]
lab@r6# show
label-switched-path r6-r3 {
    to 10.0.3.3;
    no-cspf;
    primary use-r5;
}
path use-r5 {
    10.0.3.5 loose;
}
interface all;
```

After committing the changes, the inet.3 routing table is displayed at r6 to confirm establishment of the *r6–r3* LSP:

```
[edit]
lab@r6# run show route table inet.3

inet.3: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)
Restart Complete
```

```
+ = Active Route, - = Last Active, * = Both

10.0.3.3/32         *[RSVP/7] 00:00:07, metric 10
                     > to 10.0.8.6 via fe-0/1/0.0, label-switched-path r6-r3
10.0.3.5/32         *[LDP/9] 00:28:38, metric 1
                     > to 10.0.8.6 via fe-0/1/0.0
```

To enable TE shortcut computations, the IS-IS protocol instance at `r6` is modified as shown next:

```
[edit protocols isis]
lab@r5# set traffic-engineering shortcuts
```

When the `inet.3` table is again displayed, the effect of `traffic-engineering shortcuts` becomes obvious, even to the most casual of observers:

```
[edit]
lab@r6# run show route table inet.3

inet.3: 14 destinations, 15 routes (14 active, 0 holddown, 0 hidden)
Restart Complete
+ = Active Route, - = Last Active, * = Both

10.0.2.4/30         *[IS-IS/18] 00:17:25, metric 20
                     > to 10.0.8.6 via fe-0/1/0.0, label-switched-path r6-r3
10.0.2.16/30        *[IS-IS/18] 00:17:25, metric 30
                     > to 10.0.8.6 via fe-0/1/0.0, label-switched-path r6-r3
10.0.3.3/32         *[RSVP/7] 00:19:21, metric 10
                     > to 10.0.8.6 via fe-0/1/0.0, label-switched-path r6-r3
                     [IS-IS/18] 00:17:25, metric 10
                     > to 10.0.8.6 via fe-0/1/0.0, label-switched-path r6-r3
10.0.3.4/32         *[IS-IS/18] 00:17:25, metric 20
                     > to 10.0.8.6 via fe-0/1/0.0, label-switched-path r6-r3
10.0.3.5/32         *[LDP/9] 00:19:04, metric 1
                     > to 10.0.8.6 via fe-0/1/0.0
10.0.4.0/30         *[IS-IS/18] 00:17:25, metric 20
                     > to 10.0.8.6 via fe-0/1/0.0, label-switched-path r6-r3
10.0.4.4/30         *[IS-IS/18] 00:17:25, metric 30
                     > to 10.0.8.6 via fe-0/1/0.0, label-switched-path r6-r3
10.0.4.8/30         *[IS-IS/18] 00:17:25, metric 30
                     > to 10.0.8.6 via fe-0/1/0.0, label-switched-path r6-r3
10.0.4.12/30        *[IS-IS/18] 00:17:25, metric 20
                     > to 10.0.8.6 via fe-0/1/0.0, label-switched-path r6-r3
10.0.4.16/30        *[IS-IS/18] 00:17:25, metric 30
                     > to 10.0.8.6 via fe-0/1/0.0, label-switched-path r6-r3
```

```
10.0.5.0/24        *[IS-IS/18] 00:17:25, metric 30
                    > to 10.0.8.6 via fe-0/1/0.0, label-switched-path r6-r3
10.0.6.1/32        *[IS-IS/18] 00:17:25, metric 20
                    > to 10.0.8.6 via fe-0/1/0.0, label-switched-path r6-r3
10.0.6.2/32        *[IS-IS/18] 00:17:25, metric 20
                    > to 10.0.8.6 via fe-0/1/0.0, label-switched-path r6-r3
10.0.9.7/32        *[LDP/9] 00:15:16, metric 1
                    > to 10.0.8.6 via fe-0/1/0.0, Push 100029
```

The display indicates that most of the prefixes associated with the L2 backbone and L1 area 0001 have been installed into r6's `inet.3` table with the *r6–r3* LSP as the next hop. While closer to your goal, internal traffic will not take the LSP, because only BGP looks into `inet.3`. This is confirmed with a quick traceroute:

```
[edit]
lab@r6# run traceroute 10.0.5.1
traceroute to 10.0.5.1 (10.0.5.1), 30 hops max, 40 byte packets
 1  10.0.2.14 (10.0.2.14)  0.410 ms  0.299 ms  0.244 ms
 2  10.0.5.1 (10.0.5.1)  0.165 ms  0.151 ms  0.132 ms
```

To meet all the requirements of this scenario, your TE shortcuts must be moved into the r6's `inet.0` table. This is achieved with the addition of a `bgp-igp` statement issued at the `edit protocols mpls` hierarchy, as shown here:

```
[edit]
lab@r6# set protocols mpls traffic-engineering bgp-igp

[edit]
lab@r6# show protocols mpls
traffic-engineering bgp-igp;
label-switched-path r6-r3 {
    to 10.0.3.3;
    no-cspf;
    primary use-r5;
}
path use-r5 {
    10.0.3.5 loose;
}
interface all;
```

After the commit, the results are confirmed:

```
[edit]
lab@r6# run show route table inet.3
```

```
[edit]
lab@r6#
```

The `inet.3` table is now empty because the `bgp-igp` statement has caused the contents of `inet.3` to be moved into the `inet.0` routing table, as confirmed here:

```
[edit]
lab@r6# run show route 10.0.5/24

inet.0: 125636 destinations, 125645 routes (125636 active, 0 holddown, 0 hidden)
Restart Complete
+ = Active Route, - = Last Active, * = Both

10.0.5.0/24         *[IS-IS/18] 00:00:17, metric 30
                     > to 10.0.8.6 via fe-0/1/0.0, label-switched-path r6-r3
```

The final verification involves some traceroute testing to the 10.0.5/24 internal destination, as well as to the 120.120/16 external route:

```
[edit]
lab@r6# run traceroute 10.0.5.1
traceroute to 10.0.5.1 (10.0.5.1), 30 hops max, 40 byte packets
 1  10.0.8.6 (10.0.8.6)  0.812 ms  0.552 ms  0.506 ms
     MPLS Label=100027 CoS=0 TTL=1 S=1
 2  10.0.2.2 (10.0.2.2)  0.487 ms  0.499 ms  0.441 ms
 3  10.0.5.1 (10.0.5.1)  0.350 ms  0.227 ms  0.442 ms

[edit]
lab@r6# run traceroute 120.120.0.1
traceroute to 120.120.0.1 (120.120.0.1), 30 hops max, 40 byte packets
 1  10.0.8.6 (10.0.8.6)  0.638 ms  0.489 ms  0.431 ms
     MPLS Label=100027 CoS=0 TTL=1 S=1
 2  10.0.2.2 (10.0.2.2)  0.656 ms  0.634 ms  0.410 ms
 3  10.0.4.14 (10.0.4.14)  0.305 ms  0.679 ms  0.407 ms
 4  120.120.0.1 (120.120.0.1)  0.508 ms  0.701 ms  0.404 ms
```

The traceroute output confirms that LSP forwarding is now in place between `r6` and the specified prefixes. The heavy-handed nature of TE shortcuts results in various other destinations also being mapped to the *r6–r3* LSP, however. This is one of the reasons why TE shortcuts can be problematic, making the use of `install` generally preferred. Another side effect of TE shortcuts on `r6` is the fact that the LDP signaled LSP that terminates on `r7` has also been moved into the `inet.0` routing table, as shown next:

```
[edit]
lab@r6# run show route 10.0.9.7
```

```
inet.0: 125637 destinations, 125646 routes (125637 active, 0 holddown, 0 hidden)
Restart Complete
+ = Active Route, - = Last Active, * = Both

10.0.9.7/32         *[LDP/9] 00:11:10, metric 1
                     > to 10.0.8.6 via fe-0/1/0.0, Push 100029
                     [IS-IS/15] 00:11:07, metric 20
                     > to 10.0.8.6 via fe-0/1/0.0

[edit]
lab@r6# run traceroute 10.0.9.7
traceroute to 10.0.9.7 (10.0.9.7), 30 hops max, 40 byte packets
 1  10.0.8.6 (10.0.8.6)  0.649 ms  3.877 ms  0.445 ms
     MPLS Label=100029 CoS=0 TTL=1 S=1
 2  10.0.9.7 (10.0.9.7)  0.165 ms  0.155 ms  0.133 ms
```

The traceroute test confirms that internal traffic is now able to use r6's LDP and RSVP signaled LSPs. Due to the inherent sloppiness of TE shortcuts, especially when combined with `traffic-engineering bgp-igp`, the *r6–r3* LSP and its related TE shortcut configuration are removed from r6 before proceeding to the next configuration task:

```
[edit]
lab@r6# delete protocols mpls label-switched-path r6-r3

[edit]
lab@r6# delete protocols mpls path use-r5

[edit]
lab@r6# delete protocols mpls traffic-engineering

[edit]
lab@r6# delete protocols isis traffic-engineering shortcuts
```

Prefix Mapping

When multiple, equal-cost LSPs exist between ingress and egress nodes, the default JUNOS software behavior is to randomly load-balance the associated traffic over each such LSP. In some cases, it may be desirable to map specific prefixes to specific LSPs, such as in the case of a multi-level service offering that is achieved through diverse LSP routing that results in differing QoS levels.

Generally speaking, there are two main ways to map traffic to a particular LSP, at least when Filter Based Forwarding (FBF) is not a consideration (FBF is covered in Chapter 3, "Firewall Filter and Traffic Sampling"). The most common prefix-to-LSP mapping technique involves

forwarding table policy at the LSP ingress node (the receiver of the routes), typically making use of route filter or community-based match conditions to map traffic to a particular LSP. The other common approach involves BGP next hop manipulation at the LSP egress node (the advertiser of the routes) such that the default BGP next hop resolution behavior at the LSP ingress results in the mapping of a given prefix to the desired LSP.

To complete the LSP prefix mapping scenario, you must configure your network to meet these criteria:

- Establish LSP *r7–r3*, ensuring that the LSP transits r5.
- Establish LSP *r7–r3–prime*, ensuring that the LSP transits r5 and r6.
- Map all routes with a netmask *less* than /25 to the *r7–r3* LSP; map all remaining routes to the *r7–r3–prime* LSP.

In this example, traffic will be mapped to each LSP based on the LSP ingress policy, as opposed to BGP next hop manipulation at the LSP's egress.

Configuring LSP-to-Prefix Mapping

You begin by establishing the two LSPs from r7 to r3, taking care to ensure that both of them meet the stipulated routing constraints. While CSPF could be used to control LSP routing, this author believes that it is more expedient to simply use ERO-based routing constraints in this case. The following commands define the *r7–r3* LSP and create the *use-r5* path that will constrain the LSP's routing:

```
[edit protocols mpls]
lab@r7# set label-switched-path r7-r3 to 10.0.3.3 no-cspf

[edit protocols mpls]
lab@r7# set label-switched-path r7-r3 primary use-r5

[edit protocols mpls]
lab@r7# set path use-r5 10.0.3.5 loose
```

r7's modified configuration is shown next, with the recent changes highlighted:

```
[edit protocols mpls]
lab@r7# show
label-switched-path r7-r3 {
    to 10.0.3.3;
    no-cspf;
    primary use-r5;
}
path use-r5 {
    10.0.3.5 loose;
}
interface all;
```

The next set of commands defines the *r7–r3–prime* LSP, and the corresponding *use–r5–r6* path that will force the LSP's routing through r5 and r6 as required by the specifics of this example:

```
[edit protocols mpls]
lab@r7# set label-switched-path r7-r3-prime to 10.0.3.3 no-cspf

[edit protocols mpls]
lab@r7# set label-switched-path r7-r3-prime primary use-r5-r6

[edit protocols mpls]
lab@r7# set path use-r5-r6 10.0.3.5 loose

[edit protocols mpls]
lab@r7# set path use-r5-r6 10.0.9.6 loose
```

The modified configuration at r7 is shown next with the recent changes highlighted:

```
[edit protocols mpls]
lab@r7# show
label-switched-path r7-r3 {
    to 10.0.3.3;
    no-cspf;
    primary use-r5;
}
label-switched-path r7-r3-prime {
    to 10.0.3.3;
    no-cspf;
    primary use-r5-r6;
}
path use-r5 {
    10.0.3.5 loose;
}
path use-r5-r6 {
    10.0.3.5 loose;
    10.0.9.6 loose;
}
interface all;
```

Before concerning yourself with prefix mapping, you commit the initial changes and verify the correct establishment of both the LSPs between r7 and r3:

```
[edit protocols mpls]
lab@r7# run show rsvp session ingress detail
Ingress RSVP: 2 sessions
```

```
10.0.3.3
  From: 10.0.9.7, LSPstate: Up, ActiveRoute: 63126
  LSPname: r7-r3, LSPpath: Primary
  Suggested label received: -, Suggested label sent: -
  Recovery label received: -, Recovery label sent: 100009
  Resv style: 1 FF, Label in: -, Label out: 100009
  Time left:    -,  Since: Mon Feb 17 06:08:28 2003
  Tspec: rate 0bps size 0bps peak Infbps m 20 M 1500
  Port number: sender 1 receiver 58625 protocol 0
  PATH rcvfrom: localclient
  PATH sentto: 10.0.8.9 (fe-0/3/1.0) 18 pkts
  RESV rcvfrom: 10.0.8.9  (fe-0/3/1.0) 19 pkts
  Explct route: 10.0.8.9 10.0.3.5
  Record route: <self>  10.0.8.9  10.0.2.2

10.0.3.3
  From: 10.0.9.7, LSPstate: Up, ActiveRoute: 62349
  LSPname: r7-r3-prime, LSPpath: Primary
  Suggested label received: -, Suggested label sent: -
  Recovery label received: -, Recovery label sent: 100010
  Resv style: 1 FF, Label in: -, Label out: 100010
  Time left:    -,  Since: Mon Feb 17 06:18:57 2003
  Tspec: rate 0bps size 0bps peak Infbps m 20 M 1500
  Port number: sender 1 receiver 58629 protocol 0
  PATH rcvfrom: localclient
  PATH sentto: 10.0.8.9 (fe-0/3/1.0) 1 pkts
  RESV rcvfrom: 10.0.8.9  (fe-0/3/1.0) 1 pkts
  Explct route: 10.0.8.9 10.0.3.5 10.0.9.6
  Record route: <self>  10.0.8.9  10.0.8.5  10.0.2.14
Total 2 displayed, Up 2, Down 0
```

The highlights call out the fact that both LSPs have been established and that their routing is in accordance with the restrictions posed for this scenario. Also of note is the number of active prefixes shown for the two LSPs; the count values of 63126 and 62349 displayed for the *r7–r3* and *r7–r3–prime* LSPs, respectively, represent a nearly ideal split of the 125,000 or so BGP routes currently being advertised by r3. Some fancy CLI footwork affirms the count values and provides some additional detail:

```
lab@r7> show route source-gateway 10.0.3.3

inet.0: 125529 destinations, 125535 routes (125529 active, 0 holddown, 0 hidden)
Restart Complete
+ = Active Route, - = Last Active, * = Both
```

```
3.0.0.0/8          *[BGP/170] 00:02:46, localpref 100, from 10.0.3.3
                      AS path: 65222 10458 14203 701 7018 80 I
                    > to 10.0.8.9 via fe-0/3/1.0, label-switched-path r7-r3
                      to 10.0.8.9 via fe-0/3/1.0, label-switched-path r7-r3-
                      prime
4.0.0.0/8          *[BGP/170] 01:25:16, localpref 100, from 10.0.3.3
                      AS path: 65222 10458 14203 3561 1 I
                    > to 10.0.8.9 via fe-0/3/1.0, label-switched-path r7-r3
                      to 10.0.8.9 via fe-0/3/1.0, label-switched-path r7-r3-prime
6.1.0.0/16         *[BGP/170] 01:25:16, localpref 100, from 10.0.3.3
                      AS path: 65222 10458 14203 3561 701 668 7170 1455 I
                    > to 10.0.8.9 via fe-0/3/1.0, label-switched-path r7-r3
                      to 10.0.8.9 via fe-0/3/1.0, label-switched-path r7-r3-
                      prime
. . .
```

The output confirms the presence of two equal-cost LSPs available for load balancing to BGP prefixes learned from 10.0.3.3.

By default, RSVP LSPs are assigned a metric that is equal to the IGP's best route to that destination. You can manually set the metric of an RSVP signaled LSP with the `metric` keyword under the `edit protocols mpls label-switched-path` *`path-name`* hierarchy. LDP LSPs are assigned a metric of 1 by default. Use the `track-igp-metric` command under the `edit protocols ldp` hierarchy to have LDP signaled LSPs track the IGP metric to a given destination.

In the case of this truncated output, the routes shown have all been mapped to the *r7–r3* LSP. Armed with the specifics of the display syntax, CLI-matching functions are now used to verify the default prefix-to-LSP mapping counts:

```
lab@r7> show route source-gateway 10.0.3.3 | match > | match "(r7-r3$)" | count
Count: 63135 lines
lab@r7>
```

In this example, the active forwarding next hops for the routes learned from 10.0.3.3 (matching is based on the presence of the > character) are piped to a regular expression (regx) that matches only on *r7–r3*. In this example, the lines that match the regx are then piped to the CLI `count` function, yielding the value of 63135. Adjusting the syntax to match on the second LSP yields the following output:

```
lab@r7> show route source-gateway 10.0.3.3 | match > | match r7-r3-prime | count
Count: 62338 lines

lab@r7>
```

With knowledge of the default LSP load balancing behavior at r7, you move on to the prefix mapping aspects of this configuration task. You begin by creating the first term in the new *lsp-map* policy:

```
[edit]
lab@r7# edit policy-options policy-statement lsp-map

[edit policy-options policy-statement lsp-map]
lab@r7# set term 1 from protocol bgp

[edit policy-options policy-statement lsp-map]
lab@r7# set term 1 from neighbor 10.0.3.3

[edit policy-options policy-statement lsp-map]
lab@r7# set term 1 from route-filter 0/0 upto /23

[edit policy-options policy-statement lsp-map]
lab@r7# set term 1 then install-nexthop lsp r7-r3

[edit policy-options policy-statement lsp-map]
lab@r7# set term 1 then accept
```

The first term in the *lsp-map* policy is now displayed:

```
[edit policy-options policy-statement lsp-map]
lab@r7# show term 1
from {
    protocol bgp;
    neighbor 10.0.3.3;
    route-filter 0.0.0.0/0 upto /24;
}
then {
    install-nexthop lsp r7-r3;
    accept;
}
```

The match conditions in the first term ensure that only BGP routes learned from r3 (10.0.3.3) will be subjected to the effects of the LSP mapping policy. This level of matching selectivity is not strictly necessary here, as BGP routes learned from other peerings will not resolve to the BGP next hop of 10.0.3.3, therefore making them ineligible for LSP forwarding over either the *r7–r3* or the *r7–r3–prime* LSPs. The route filter line matches only on the prefix length, as the 0.0.0.0/0 initial match conditions indicate a "do not care" for the actual prefix value. The highlighted accept action is critical for proper mapping operation. Without the terminating action, routes matching the first term will continue to be evaluated by your forwarding table policy, and this can result in remapping the prefixes to another LSP.

Despite indications to the contrary, use of `from neighbor 10.0.3.3` as part of the policy's match condition did not result in the desired behavior. According to the documentation set, the `neighbor` keyword allows the specification of a directly, or indirectly, connected BGP peer when used as part of a `from` match condition. While it is unclear at the time of this writing whether the problem relates to incorrect software behavior, or a less-than-perfect documentation set, the main point is that you should never take the result of any configuration for granted! The successful JNCIE candidate will always take a moment to confirm the correct operation of their configurations.

You now add a second policy term to the *lsp-map* policy, which functions to match on all routes that were not accepted by the first term for mapping to the *r7-r3-prime* LSP. In this case, the route filter statement is not strictly necessary (the term's goal, after all, is to match on all remaining routes); it is included here for general consistency with the first term. Note that the ordering of the terms is significant in this example. Listing the second term first would cause *all* routes to be mapped to the *r7-r3-prime* LSP. The `accept` action in the second term is also important. Without it, routes matching term 2 would fall through to the default policy, where they would end up being balanced between the two LSPs.

The completed *lsp-map* policy is now displayed:

```
[edit policy-options policy-statement lsp-map]
lab@r7# show
term 1 {
    from {
        protocol bgp;
        neighbor 10.0.3.3;
        route-filter 0.0.0.0/0 upto /24;
    }
    then {
        install-nexthop lsp r7-r3;
        accept;
    }
}
term 2 {
    from {
        protocol bgp;
        neighbor 10.0.3.3;
        route-filter 0.0.0.0/0 upto /32;
    }
    then {
        install-nexthop lsp r7-r3-prime;
        accept;
    }
}
```

You must apply your prefix mapping policy to the main routing instance's forwarding table in order for it to take effect. The following command correctly applies the *lsp-map* policy as export:

```
[edit routing-options]
lab@r7# set forwarding-table export lsp-map

[edit routing-options]
lab@r7# show
graceful-restart;
static {
    route 10.0.200.0/24 {
        next-hop 10.0.1.102;
        no-readvertise;
    }
    route 0.0.0.0/0 reject;
}
aggregate {
    route 10.0.0.0/16;
}
autonomous-system 65412;
forwarding-table {
    export lsp-map;
}
```

Verifying LSP-to-Prefix Mapping

After committing the changes related to the *lsp-map* policy, you verify its effect using the same commands demonstrated previously when the default prefix mapping behavior was explored. You start by inspecting the number of routes now shown as active for each LSP in the output of the show rsvp session ingress detail command:

```
[edit]
lab@r7# run show rsvp session ingress detail
Ingress RSVP: 2 sessions

10.0.3.3
  From: 10.0.9.7, LSPstate: Up, ActiveRoute: 118034
  LSPname: r7-r3, LSPpath: Primary
  Suggested label received: -, Suggested label sent: -
  Recovery label received: -, Recovery label sent: 100009
  Resv style: 1 FF, Label in: -, Label out: 100009
  Time left:    -,  Since: Mon Feb 17 06:08:28 2003
  Tspec: rate 0bps size 0bps peak Infbps m 20 M 1500
```

```
  Port number: sender 1 receiver 58625 protocol 0
  PATH rcvfrom: localclient
  PATH sentto: 10.0.8.9 (fe-0/3/1.0) 159 pkts
  RESV rcvfrom: 10.0.8.9  (fe-0/3/1.0) 159 pkts
  Explct route: 10.0.8.9 10.0.3.5
  Record route: <self>  10.0.8.9  10.0.2.2

10.0.3.3
  From: 10.0.9.7, LSPstate: Up, ActiveRoute: 7447
  LSPname: r7-r3-prime, LSPpath: Primary
  Suggested label received: -, Suggested label sent: -
  Recovery label received: -, Recovery label sent: 100010
  Resv style: 1 FF, Label in: -, Label out: 100010
  Time left:    -,  Since: Mon Feb 17 06:18:57 2003
  Tspec: rate 0bps size 0bps peak Infbps m 20 M 1500
  Port number: sender 1 receiver 58629 protocol 0
  PATH rcvfrom: localclient
  PATH sentto: 10.0.8.9 (fe-0/3/1.0) 144 pkts
  RESV rcvfrom: 10.0.8.9  (fe-0/3/1.0) 144 pkts
  Explct route: 10.0.8.9 10.0.3.5 10.0.9.6
  Record route: <self>  10.0.8.9  10.0.8.5  10.0.2.14
Total 2 displayed, Up 2, Down 0
```

The highlights call out the dramatic change in the distribution of prefix-to-LSP mappings for the BGP routes learned from `r3`. The JUNOS software CLI-matching function is used to provide some additional spot checks of the *`lsp-map`* policy's operation:

```
[edit]
lab@r7# run show route source-gateway 10.0.3.3 | match "(/|>)"
3.0.0.0/8          *[BGP/170] 00:04:26, localpref 100, from 10.0.3.3
                    > to 10.0.8.9 via fe-0/3/1.0, label-switched-path r7-r3
4.0.0.0/8          *[BGP/170] 03:20:22, localpref 100, from 10.0.3.3
                    > to 10.0.8.9 via fe-0/3/1.0, label-switched-path r7-r3
6.1.0.0/16         *[BGP/170] 03:20:22, localpref 100, from 10.0.3.3
                    > to 10.0.8.9 via fe-0/3/1.0, label-switched-path r7-r3
6.2.0.0/22         *[BGP/170] 03:20:22, localpref 100, from 10.0.3.3
                    > to 10.0.8.9 via fe-0/3/1.0, label-switched-path r7-r3
6.3.0.0/18         *[BGP/170] 03:20:22, localpref 100, from 10.0.3.3
                    > to 10.0.8.9 via fe-0/3/1.0, label-switched-path r7-r3
6.4.0.0/16         *[BGP/170] 03:20:22, localpref 100, from 10.0.3.3
                    > to 10.0.8.9 via fe-0/3/1.0, label-switched-path r7-r3
6.5.0.0/19         *[BGP/170] 03:20:22, localpref 100, from 10.0.3.3
                    > to 10.0.8.9 via fe-0/3/1.0, label-switched-path r7-r3
```

```
6.8.0.0/20          *[BGP/170] 03:20:22, localpref 100, from 10.0.3.3
                     > to 10.0.8.9 via fe-0/3/1.0, label-switched-path r7-r3
6.9.0.0/20          *[BGP/170] 03:20:22, localpref 100, from 10.0.3.3
                     > to 10.0.8.9 via fe-0/3/1.0, label-switched-path r7-r3
6.10.0.0/15         *[BGP/170] 03:20:22, localpref 100, from 10.0.3.3
                     > to 10.0.8.9 via fe-0/3/1.0, label-switched-path r7-r3
. . .
```

All of the routes shown in this truncated display have prefix lengths equal to, or less than, the cutoff length of /24, and all are correctly shown as being mapped to the *r7–r3* LSP in this example. The next command counts the routes that are learned from 10.0.3.3 with mask lengths in the range of 25–32:

```
[edit]
lab@r7# run show route source-gateway 10.0.3.3 | match "
   (/25|/26|/27|/28|/29|/30|/31|/32)" | count
Count: 7449 lines
```

The results confirm that the prefix mapping policy at `r7` is working in accordance with all criteria specified for this scenario.

Summary of Routing Table Integration

By default, signaled LSPs are placed into the `inet.3` routing table where they can be used for BGP next hop resolution only.

Although not shown in this chapter, you should note that a statically defined LSP is placed into the inet.0 routing table where it can be used by internal and external traffic.

This default behavior results in LSP-based forwarding for external prefixes only. Note that failing to set the BGP next hop to `self` on routers with external peers normally results in IGP forwarding (as opposed to LSP forwarding), due to the EBGP peer's next hop not being present in the `inet.3` routing table.

This section provided an example of how the default behavior can be modified using the `install` option to manually place prefix-to-LSP entries into the `inet.3` table, and also showed how the `active` keyword can be used to place these manually defined entries into the `inet.0` routing table where they can be used to evoke LSP forwarding for both internal and external destinations.

The use of `traffic-engineering shortcuts`, both with the default `bgp` and with the `bgp-igp` switch, was also demonstrated. TE shortcuts rely on a link state IGP with knowledge of the prefixes that are reachable *downstream* of the LSP egress point. By default, TE shortcuts install all downstream prefixes into the `inet.3` table where they can be used for BGP next hop resolution. Configuring `bgp-igp` causes the contents of the `inet.3` table to be moved into `inet.0`, which makes the LSP visible to both internal and external prefixes.

Note that TE shortcuts pose problems when you need to be selective about what prefixes should, or should not, be mapped to a particular LSP, especially when you have an ingress node with a less-than-complete knowledge of the link state topology. Incomplete knowledge of the link state topology occurs in a multi-level IS-IS network and in a multi-area OSPF network.

Traffic Protection

JUNOS software supports a variety of mechanisms that can be used to "protect" traffic associated with an LSP. Traffic protection options include secondary paths, Fast Reroute (FRR), link protection, fate sharing, and preemption.

This section will provide various configuration scenarios that demonstrate key MPLS protection features and capabilities.

Secondary Paths

Secondary LSP paths are used to provide a backup for a primary LSP path. You can define as many secondary paths as you like, but only one secondary LSP path will be established at any given time. You can have only one primary path definition, however. When multiple secondary definitions exist, the secondary LSP paths are signaled according to their setup priority and, in the event of a priority tie, the order in which they are listed.

A secondary path can be placed in a *standby* state, whereas the LSP is established *before* the primary path fails. The default behavior is to signal and establish a secondary path *after* the primary path is detected as down. Traffic will automatically revert back to the primary path when it is reestablished and remains established for at least 60 seconds. To prevent traffic from reverting back to a path that has previously failed, define only secondary LSPs.

When CSPF is used to calculate LSP paths, the routing of a secondary path will automatically avoid using path elements that are common to the corresponding primary LSP when possible. When diverse routing is mandated, you should make use of EROs to ensure that the primary and secondary paths will not share particular elements. This is especially true when CSPF is not in use.

To complete this configuration task, you must modify the configuration of `r1` to meet these requirements:

- Protect the *r1–r7* LSP with a secondary path.
- Ensure that both the primary and secondary paths are established.
- Make both LSPs reserve 70Mbps of bandwidth while making sure that their bandwidth reservations are not double-counted.
- You must not use EROs to control the routing of the LSPs.

Based on the criteria specified, you must configure a secondary path that will be placed in the `standby` state. Diverse routing for the two paths is not required, and the use of EROs has been prohibited. This is significant, because attempting to use CSPF will result in LSP setup failure, due to the lack of a domain-wide TED in `r1`.

The following commands delete the existing *r1–r7* LSP definition at r1 and redefine the new primary path:

```
[edit protocols mpls]
lab@r1# delete label-switched-path r1-r7

[edit protocols mpls]
lab@r1# set label-switched-path r1-r7 no-cspf to 10.0.9.7 primary r1-r7

[edit protocols mpls]
lab@r1# set label-switched-path r1-r7 bandwidth 70M
```

The bandwith parameter is specified at the LSP level in this case, because both the primary and secondary paths are expected to reserve the same amount of bandwidth. The same logic holds for the no-cspf specification, which is also defined at the LSP level. You now define a null *r1–r7* path to allow your configuration to commit. You must define a named path when setting a LSP as primary or secondary because these keywords must be followed by the specification of a named path. The restriction on EROs in this example forces the definition of an empty path:

```
[edit protocols mpls]
lab@r1# set path r1-r7
```

The modified LSP configuration is now shown on r1:

```
lab@r1# show
label-switched-path r1-r7 {
    to 10.0.9.7;
    bandwidth 70m;
    no-cspf;
    primary r1-r7;
}
path r1-r7;
interface all;
```

Before adding the secondary LSP, you commit your changes and verify establishment of the *r1–r7* primary LSP:

```
[edit]
lab@r1# run show mpls lsp ingress extensive
Ingress LSP: 1 sessions

10.0.9.7
  From: 10.0.6.1, State: Up, ActiveRoute: 0, LSPname: r1-r7
  ActivePath: r1-r7 (primary)
  LoadBalance: Random
  Encoding type: Packet, Switching type: Packet, GPID: IPv4
```

```
 *Primary   r1-r7              State: Up
    Bandwidth: 70Mbps
    Received RRO:
          10.0.4.13 10.0.2.6 10.0.2.17
    4 Feb 19 04:30:36  Selected as active path
    3 Feb 19 04:30:36  Record Route:  10.0.4.13 10.0.2.6 10.0.2.17
    2 Feb 19 04:30:36  Up
    1 Feb 19 04:30:36  Originate Call
  Created: Wed Feb 19 04:27:50 2003
Total 1 displayed, Up 1, Down 0
```

The output confirms that the new path has been established with the required reservation of 70Mbps. The primary path indication is also highlighted. A quick look at the RSVP interface state further confirms the presence of reserved bandwidth:

```
[edit]
lab@r1# run show rsvp interface
RSVP interface: 4 active
                    Active                  Static    Available   Reserved    Highwater
Interface   State   resv    Subscription    BW        BW          BW          mark
fe-0/0/0.0    Up      0            100%   100Mbps   100Mbps     0bps        0bps
fe-0/0/1.0    Up      1            100%   100Mbps   30Mbps      70Mbps      70Mbps
fe-0/0/2.0    Up      0            100%   100Mbps   100Mbps     0bps        0bps
fe-0/0/3.0    Up      0            100%   100Mbps   100Mbps     0bps        20Mbps
```

With the primary path correctly established, you move on to configure the secondary path:

```
[edit protocols mpls]
lab@r1# set label-switched-path r1-r7 secondary r1-r7-prime standby
```

Note the use of the `standby` keyword in the secondary path's definition. This causes the router to try to establish the secondary path, regardless of the operational state of the primary path; standby secondary behavior is required by the criteria in this example. As with the primary named path, you also define a null named path for the secondary:

```
[edit protocols mpls]
lab@r1# set path r1-r7-prime
```

The additions to `r1`'s configuration are now displayed, with highlights added to the entries relating to the secondary LSP:

```
[edit protocols mpls]
lab@r1# show
label-switched-path r1-r7 {
    to 10.0.9.7;
    bandwidth 70m;
    no-cspf;
    primary r1-r7;
```

```
    secondary r1-r7-prime {
        standby;
    }
}
path r1-r7;
path r1-r7-prime;
interface all;
```

Verifying Secondary Path

After committing the changes on r1, you display the RSVP session status to determine if both primary and secondary paths have been established:

```
[edit protocols mpls]
lab@r1# run show rsvp session ingress detail
Ingress RSVP: 2 sessions

10.0.9.7
  From: 10.0.6.1, LSPstate: Up, ActiveRoute: 0
  LSPname: r1-r7, LSPpath: Primary
  Suggested label received: -, Suggested label sent: -
  Recovery label received: -, Recovery label sent: 100002
  Resv style: 1 FF, Label in: -, Label out: 100002
  Time left:    -,  Since: Wed Feb 19 04:48:48 2003
  Tspec: rate 70Mbps size 70Mbps peak Infbps m 20 M 1500
  Port number: sender 1 receiver 53606 protocol 0
  PATH rcvfrom: localclient
  PATH sentto: 10.0.4.13 (fe-0/0/1.0) 6 pkts
  RESV rcvfrom: 10.0.4.13  (fe-0/0/1.0) 6 pkts
  Record route: <self>  10.0.4.13  10.0.2.6  10.0.2.17

10.0.9.7
  From: 10.0.6.1, LSPstate: Dn, ActiveRoute: 0
  LSPname: r1-r7, LSPpath: Secondary
  Suggested label received: -, Suggested label sent: -
  Recovery label received: -, Recovery label sent: -
  Resv style: 0 -, Label in: -, Label out: -
  Time left:    -,  Since: Wed Feb 19 04:49:17 2003
  Tspec: rate 70Mbps size 70Mbps peak Infbps m 20 M 1500
  Port number: sender 2 receiver 53607 protocol 0
  PATH rcvfrom: localclient
  PATH sentto: 10.0.4.17 (fe-0/0/3.0) 5 pkts
  Record route: <self>  ...incomplete
Total 2 displayed, Up 1, Down 1
```

Not good. While the display correctly indicates that two ingress RSVP sessions to 10.0.9.7 are defined at r1, the display also makes it clear that the secondary path has not been successfully established. The output from a `show mpls lsp` command provides a critical clue to the nature of the problem. Generally speaking, when troubleshooting RSVP and/or CSPF problems you should try the `show mpls lsp` command with the `extensive` switch:

```
[edit protocols mpls]
lab@r1# run show mpls lsp ingress extensive
Ingress LSP: 1 sessions

10.0.9.7
  From: 10.0.6.1, State: Up, ActiveRoute: 0, LSPname: r1-r7
  ActivePath: r1-r7 (primary)
  LoadBalance: Random
  Encoding type: Packet, Switching type: Packet, GPID: IPv4
 *Primary   r1-r7            State: Up
    Bandwidth: 70Mbps
    Received RRO:
          10.0.4.13 10.0.2.6 10.0.2.17
    4 Feb 19 04:48:48  Selected as active path
    3 Feb 19 04:48:48  Record Route:  10.0.4.13 10.0.2.6 10.0.2.17
    2 Feb 19 04:48:48  Up
    1 Feb 19 04:48:48  Originate Call
  Standby   r1-r7-prime      State: Dn
    Bandwidth: 70Mbps
    2 Feb 19 04:57:08  10.0.4.17: Requested bandwidth unavailable[15 times]
    1 Feb 19 04:49:17  Originate Call
  Created: Wed Feb 19 04:39:53 2003
Total 1 displayed, Up 1, Down 0
```

The highlighted entries make it clear that the secondary path cannot be established due to insufficient bandwidth being available for the reservation at 10.0.4.17 (r4). Considering that the Fast Ethernet link between r1 and r4 operates at 100Mbps, and that 2×70Mbps equals 140Mbps, this error should not be too surprising. While you could adjust the RSVP subscription percentage on the other routers in the test bed to allow RSVP oversubscription, this would be in violation of the requirement that the bandwidth for the two LSPs not be double-counted.

The following command changes the RSVP reservation style from the default Fixed Filter (FF) to a Shared Explicit (SE) style; SE style reservations allow resources to be shared among multiple LSPs that share a common Session Object parameter. Note that the `adaptive` keyword is specified at the LSP level in this case, because your goal is to evoke an SE style reservation for both the primary and secondary paths:

```
[edit protocols mpls]
lab@r1# set label-switched-path r1-r7 adaptive
```

After the changes are committed, the status of the secondary path is once again displayed:

```
[edit protocols mpls]
lab@r1# run show mpls lsp ingress extensive
Ingress LSP: 1 sessions

10.0.9.7
  From: 10.0.6.1, State: Up, ActiveRoute: 0, LSPname: r1-r7
  ActivePath: r1-r7 (primary)
  LoadBalance: Random
  Encoding type: Packet, Switching type: Packet, GPID: IPv4
 *Primary   r1-r7              State: Up
    Bandwidth: 70Mbps
    Received RRO:
          10.0.4.13 10.0.2.6 10.0.2.17
    4 Feb 19 05:21:38  Selected as active path
    3 Feb 19 05:21:38  Record Route:  10.0.4.13 10.0.2.6 10.0.2.17
    2 Feb 19 05:21:38  Up
    1 Feb 19 05:21:38  Originate Call
  Standby   r1-r7-prime       State: Up
    Bandwidth: 70Mbps
    Received RRO:
          10.0.4.13 10.0.2.1 10.0.8.10
    3 Feb 19 05:22:08  Record Route:  10.0.4.13 10.0.2.1 10.0.8.10
    2 Feb 19 05:22:08  Up
    1 Feb 19 05:22:08  Originate Call
  Created: Wed Feb 19 05:21:03 2003
Total 1 displayed, Up 1, Down 0
```

The output now indicates that both LSPs have been successfully established, and that each has correctly reserved 70Mbps of bandwidth. To provide added confirmation that the bandwidth reservation is being shared, you verify the use of a SE style reservation, and that a single 70Mbps bandwidth reservation is supporting the needs of both LSPs. The following commands are entered on r3, but they could be entered anywhere along the path of the LSPs:

```
[edit]
lab@r3# run show rsvp session transit
Transit RSVP: 2 sessions
To              From            State Rt Style Labelin Labelout LSPname
10.0.9.7        10.0.6.1        Up     1  1 SE  100003   100009 r1-r7
10.0.9.7        10.0.6.1        Up     1  1 SE  100004   100008 r1-r7
Total 2 displayed, Up 2, Down 0
```

The highlights confirm the presence of two transit LSPs at r3 with common Session Object attributes (the LSP name and egress point), and that both LSPs were signaled to use a SE style of reservation.

```
[edit]
lab@r3# run show rsvp interface
RSVP interface: 5 active
                    Active               Static       Available  Reserved  Highwater
Interface    State resv   Subscription    BW           BW         BW        mark
fe-0/0/0.0   Up          0          100%  100Mbps      100Mbps    0bps      0bps
fe-0/0/1.0   Up          0          100%  100Mbps      100Mbps    0bps      0bps
fe-0/0/3.0   Up          0          100%  100Mbps      100Mbps    0bps      0bps
at-0/1/0.0   Up          1          100%  155.52Mbps   85.52Mbps  70Mbps    70Mbps
so-0/2/0.100Up           1          100%  155.52Mbps   85.52Mbps  70Mbps    70Mbps
```

The RSVP reservation state at r3 indicates that diverse routing of the LSPs has occurred due to the presence of two equal-cost routes from r3 to 10.0.9.7. To confirm that bandwidth will not be double-counted, you can temporarily down one of the interfaces at r3 to force the LSPs into taking a common path. Alternatively, you can also display the RSVP interface state at r1, assuming that it routes both LSPs over the same interface, which is the case in this example. In this case, the "temporarily down an interface" approach is demonstrated:

```
[edit]
lab@r3# deactivate interfaces so-0/2/0

[edit]
lab@r3# commit
commit complete
```

After deactivating r3's so-0/2/0 interface, you confirm that both LSPs are correctly reestablished:

```
[edit]
lab@r3# run show mpls lsp transit
Transit LSP: 2 sessions
To              From            State Rt Style Labelin Labelout LSPname
10.0.9.7        10.0.6.1        Up     1  1 SE  100005   100009 r1-r7
10.0.9.7        10.0.6.1        Up     1  1 SE  100004   100008 r1-r7
Total 2 displayed, Up 2, Down 0
```

Both LSPs are up, so RSVP interface status is displayed to confirm that bandwidth is correctly shared among the primary and secondary paths:

```
[edit]
lab@r3# run show rsvp interface
RSVP interface: 4 active
```

```
                    Active                  Static     Available   Reserved  Highwater
Interface   State   resv     Subscription   BW         BW          BW        mark
fe-0/0/0.0  Up           0             100%  100Mbps    100Mbps     0bps      0bps
fe-0/0/1.0  Up           0             100%  100Mbps    100Mbps     0bps      0bps
fe-0/0/3.0  Up           0             100%  100Mbps    100Mbps     0bps      0bps
at-0/1/0.0  Up           1             100%  155.52Mbps 85.52Mbps   70Mbps    70Mbps
```

The presence of a single reservation using a total of 70Mbps of bandwidth confirms that you have met the requirements of the secondary LSP configuration scenario. Before proceeding to the next section, be sure to reactive r3's so-0/2/0 interface:

```
[edit]
lab@r3# rollback 1
load complete

[edit]
lab@r3# commit
commit complete
```

Fast Reroute and Link Protection

Fast Reroute (FRR) provides a mechanism by which nodes use information contained in their TED, along with the CSPF algorithm, to attempt to compute detours around a LSP's downstream link and its associated downstream node. In the event of a LSP failure, a node with an established FRR path can divert traffic over the detour while the primary LSP is reestablished. The JUNOS software implementation of Fast Reroute is proprietary.

Fast Reroute can protect against failures that may occur along the entire path of the LSP, excepting the catastrophic failure of either the ingress or the egress nodes, of course. Note that CSPF does not have to be used to compute the LSP's path at the ingress node to support Fast Reroute functionality; intervening nodes can use their TED to locate detour paths, when available, regardless of whether the ingress node has used CSPF or not.

In contrast, link protection strives to protect a specific interface, and any appropriately designated LSPs that happen to make use of that interface, by establishing a LSP that bypasses the protected interface in the event of its failure. Link protection guards against the failure of interfaces that have been explicitly configured for link protection. For an individual LSP to take advantage of a protected interface, you must explicitly configure the ingress LSR to identify that LSP as one that should be subjected to link protection.

Link protection also makes use of a TED and the CSPF algorithm—in this case, to compute a shortest path around the protected interface back to the adjacent node. Unlike Fast Reroute, link protection allows the specification of EROs that can be used to influence the outcome of the CSPF process. Link protection is not proprietary to Juniper Networks, so it can be deployed in a multi-vendor environment.

A key point to consider when dealing with both Fast Reroute and link protection is the need for a TED with sufficient information to allow a successful CSPF calculation of the Fast Reroute detour or the bypass LSP. Keep this point in mind as you proceed through this configuration

example, as the Multi-Level IS-IS topology currently in play sports multiple TED views, depending on the router in question. For example, r5's TED consists of the Level 1 information in area 0002, as well as the Level 2 information from the backbone. In contrast, r4 does not receive the L1 LSPs from area 0002, and therefore its TED view of r7 does not include the 10.0.8.8/20 link, as shown here:

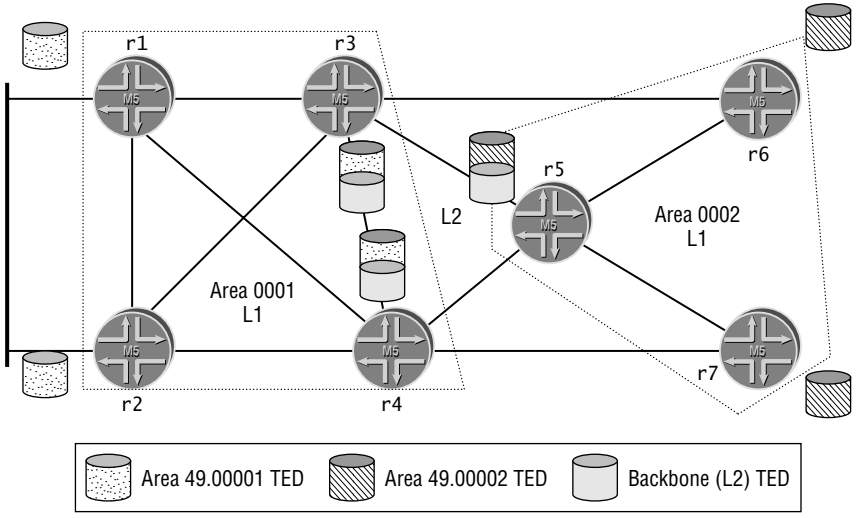
```
[edit]
lab@r4# run show ted database detail r7.0
TED database: 14 ISIS nodes 7 INET nodes
NodeID: r7.00(10.0.9.7)
  Type: Rtr, Age: 456 secs, LinkIn: 1, LinkOut: 1
  Protocol: IS-IS(2)
    To: r4.04, Local: 10.0.2.17, Remote: 0.0.0.0
```

The same command issued on r5 clearly shows that the two routers have differing TED views of r7:

```
[edit]
lab@r5# run show ted database detail r7.0
TED database: 9 ISIS nodes 5 INET nodes
NodeID: r7.00(10.0.9.7)
  Type: Rtr, Age: 552 secs, LinkIn: 2, LinkOut: 2
  Protocol: IS-IS(2)
    To: r4.04, Local: 10.0.2.17, Remote: 0.0.0.0
  Protocol: IS-IS(1)
```

Figure 2.6 graphically depicts the various TED "views" that are in place in the current Multi-Level IS-IS test bed.

FIGURE 2.6 Varying TED views



To provide an example of how this condition can affect a Fast Reroute computation, consider the case of r4 attempting to compute a detour around its fe-0/0/3 interface for a LSP that terminates on r7. In this case, r4 will not be able to compute a detour through r5, due to the absence of the 10.0.8.8/30 link in its TED.

To complete this configuration scenario, you must configure the following functionality:

- Establish LSP *r6–r4*, with r6 as the ingress and r4 as the egress.
- Ensure that the LSP uses the 10.0.2.4/30 link, and that it does not transit r5.
- Without using a secondary LSP, ensure that a failure of r3's so-0/2/0.100 interface does not disrupt LSP-based forwarding.

In this example, the use of either Fast Reroute or link protection is implied through the restriction on secondary LSP usage, combined with the need for continued LSP forwarding in the event of the failure of a *specific* interface that lies along the primary LSP's path. Note that Fast Reroute is generally preferred to link bypass when the goal is to protect the entire primary path. The specifics of the topology make either approach workable, so both techniques are demonstrated and verified in the following sections.

Configuring Fast Reroute

You begin with the definition of the new LSP at r6. Note that CSPF is not disabled in this example, and that a primary path is defined to constrain the LSP's routing through r3 in accordance with the scenario's requirements. r6 can use CSPF to calculate the path to r4 in this example, because its TED contains information for both the Level 1 area 0002 and the backbone:

```
[edit protocols mpls]
lab@r6# set label-switched-path r6-r4 to 10.0.3.4

[edit protocols mpls]
lab@r6# set label-switched-path r6-r4 primary r6-r4

[edit protocols mpls]
lab@r6# set path r6-r4 10.0.3.3 loose
```

Support for Fast Reroute is now configured:

```
[edit protocols mpls]
lab@r6# set label-switched-path r6-r4 fast-reroute
```

The resulting LSP configuration is displayed next with added highlights:

```
[edit protocols mpls]
lab@r6# show
label-switched-path r6-r4 {
    to 10.0.3.4;
    fast-reroute;
    primary r6-r4;
}
```

```
path r6-r4 {
    10.0.3.3 loose;
}
interface all;
```

Verifying Fast Reroute

After committing the changes, verification begins at the ingress node with confirmation of an operational LSP that transits r3:

```
[edit protocols mpls]
lab@r6# run show rsvp session ingress detail
Ingress RSVP: 1 sessions

10.0.3.4
  From: 10.0.9.6, LSPstate: Up, ActiveRoute: 0
  LSPname: r6-r4, LSPpath: Primary
  Suggested label received: -, Suggested label sent: -
  Recovery label received: -, Recovery label sent: 100008
  Resv style: 1 FF, Label in: -, Label out: 100008
  Time left:    -,  Since: Wed Feb 19 10:25:29 2003
  Tspec: rate 0bps size 0bps peak Infbps m 20 M 1500
  Port number: sender 1 receiver 43294 protocol 0
  FastReroute desired
  PATH rcvfrom: localclient
  PATH sentto: 10.0.2.14 (fe-0/1/1.0) 38 pkts
  RESV rcvfrom: 10.0.2.14  (fe-0/1/1.0) 37 pkts
  Explct route: 10.0.2.14 10.0.2.6
  Record route: <self>  10.0.2.14  10.0.2.6
    Detour is Up
    Detour PATH sentto: 10.0.8.6 (fe-0/1/0.0) 37 pkts
    Detour RESV rcvfrom: 10.0.8.6  (fe-0/1/0.0) 35 pkts
    Detour Explct route: 10.0.8.6 10.0.2.10
    Detour Record route: <self>  10.0.8.6  10.0.2.10
    Detour Label out: 100012
Total 1 displayed, Up 1, Down 0
```

Worthy of note in this display is the indication that r6 has computed a detour around r3 through r5. To confirm that a detour also exists around r3's so-0/2/0.100 interface, a similar command is issued at r3, this time using the `transit` switch and some additional CLI filtering to reduce clutter:

```
[edit protocols rsvp]
lab@r3# run show rsvp session transit detail name r6-r4
Transit RSVP: 3 sessions
```

```
10.0.3.4
  From: 10.0.9.6, LSPstate: Up, ActiveRoute: 1
  LSPname: r6-r4, LSPpath: Primary
  Suggested label received: -, Suggested label sent: -
  Recovery label received: -, Recovery label sent: 3
  Resv style: 1 FF, Label in: 100008, Label out: 3
  Time left:  154,  Since: Wed Feb 19 02:18:49 2003
  Tspec: rate 0bps size 0bps peak Infbps m 20 M 1500
  Port number: sender 1 receiver 43294 protocol 0
  FastReroute desired
  PATH rcvfrom: 10.0.2.13  (fe-0/0/3.0) 59 pkts
  PATH sentto: 10.0.2.6 (so-0/2/0.100) 55 pkts
  RESV rcvfrom: 10.0.2.6  (so-0/2/0.100) 58 pkts
  Explct route: 10.0.2.6
  Record route: 10.0.2.13  <self>  10.0.2.6
    Detour is Up
    Detour PATH sentto: 10.0.4.2 (fe-0/0/1.0) 54 pkts
    Detour RESV rcvfrom: 10.0.4.2  (fe-0/0/1.0) 52 pkts
    Detour Explct route: 10.0.4.2 10.0.4.9
    Detour Record route: 10.0.2.13  <self>  10.0.4.2  10.0.4.9
    Detour Label out: 100000
```

The presence of a functional Fast Reroute detour around r3's so-0/2.0.100 interface confirms that you have met the requirements of this configuration task using FRR. Note that the CSPF algorithm has chosen the Level 1 path through r2 instead of the L2 path through r5. This is the result of L1 routes having a lower, and therefore more preferred, global preference setting. Note that you have no way to control the routing of a Fast Reroute detour with the JUNOS software version in use in the test bed.

Configuring Link Protection

With the Fast Reroute approach confirmed, you now set out to meet the requirements of this task using link protection. You begin by removing the Fast Reroute configuration at r6:

```
[edit protocols mpls]
lab@r6# delete label-switched-path r6-r4 fast-reroute
```

Before adding the interface-related link protection settings at r3, you first flag the *r6–r4* LSP as being a candidate for link protection with the following command. Note that link protection is interface based, and that only LSPs that are so flagged will be able to make use of the bypass LSP:

```
[edit protocols mpls]
lab@r6# set label-switched-path r6-r4 link-protection
```

The resulting configuration change is displayed with highlights added:

```
[edit protocols mpls]
lab@r6# show
```

```
label-switched-path r6-r4 {
    to 10.0.3.4;
    link-protection;
    primary r6-r4;
}
path r6-r4 {
    10.0.3.3 loose;
}
interface all;
```

With the ingress node correctly configured to request the use of a bypass LSP, you now move to r3 to configure link protection for its so-0/2/0.100 interface. Note that link protection is a per-interface setting under the edit protocols rsvp hierarchy.

```
[edit protocols rsvp]
lab@r3# set interface so-0/2/0.100 link-protection
```

The modified configuration at r3 is displayed next:

```
[edit protocols rsvp]
lab@r3# show
interface fe-0/0/0.0 {
    authentication-key "$9$ME-L7Vji.mT3"; # SECRET-DATA
}
interface fe-0/0/1.0;
interface fe-0/0/3.0;
interface at-0/1/0.0;
interface so-0/2/0.100 {
    link-protection;
}
```

If required, you can control the routing of the bypass LSP by adding EROs under the link-protection stanza. In this example, the mere existence of a bypass LSP is sufficient to get you credit on the exam, so you leave the routing of the bypass to the default CSPF path selection criteria.

Verifying Link Protection

After committing the changes on r3 and r6, you begin your confirmation steps at r6, where an operational ingress LSP is affirmed:

```
[edit]
lab@r6# run show rsvp session ingress detail
Ingress RSVP: 1 sessions

10.0.3.4
  From: 10.0.9.6, LSPstate: Up, ActiveRoute: 0
```

```
  LSPname: r6-r4, LSPpath: Primary
  Suggested label received: -, Suggested label sent: -
  Recovery label received: -, Recovery label sent: 100003
  Resv style: 1 SE, Label in: -, Label out: 100003
  Time left:    -,  Since: Thu Feb 20 00:59:34 2003
  Tspec: rate 0bps size 0bps peak Infbps m 20 M 1500
  Port number: sender 1 receiver 21419 protocol 0
  Link protection desired
  PATH rcvfrom: localclient
  PATH sentto: 10.0.2.14 (fe-0/1/1.0) 7 pkts
  RESV rcvfrom: 10.0.2.14  (fe-0/1/1.0) 9 pkts
  Explct route: 10.0.2.14 10.0.2.6
  Record route: <self>  10.0.2.14  10.0.2.6
Total 1 displayed, Up 1, Down 0
```

The output confirms that the *r6–r4* LSP has been correctly established, and that its routing complies with the requirement that it transits `r3` to make use of its so-0/2/0.100 interface. The added highlights also indicate that the LSP is correctly flagged as a candidate for link protection. You now move to `r3` to confirm the establishment of the bypass LSP:

```
[edit protocols rsvp]
lab@r3# run show rsvp session ingress detail
Ingress RSVP: 1 sessions

10.0.3.4
  From: 10.0.3.3, LSPstate: Up, ActiveRoute: 0
  LSPname: Bypass_to_10.0.2.6
  Suggested label received: -, Suggested label sent: -
  Recovery label received: -, Recovery label sent: 100001
  Resv style: 1 SE, Label in: -, Label out: 100001
  Time left:    -,  Since: Wed Feb 19 16:55:36 2003
  Tspec: rate 0bps size 0bps peak Infbps m 20 M 1500
  Port number: sender 1 receiver 55608 protocol 0
  Type: Bypass LSP
  PATH rcvfrom: localclient
  PATH sentto: 10.0.4.2 (fe-0/0/1.0) 9 pkts
  RESV rcvfrom: 10.0.4.2  (fe-0/0/1.0) 9 pkts
  Explct route: 10.0.4.2 10.0.4.9
  Record route: <self>  10.0.4.2  10.0.4.9
Total 1 displayed, Up 1, Down 0
```

The display indicates that `r3` has successfully established a bypass LSP to `r4` in an effort to protect its so-0/2/0.100 interface. The output also indicates that the bypass LSP has been routed

through Level 1 area 0001 using r2. Note that bypass LSPs are not listed under the show mpls lsp command because link protection is strictly an RSVP thing.

```
[edit protocols rsvp]
lab@r3# run show mpls lsp ingress extensive
Ingress LSP: 0 sessions
Total 0 displayed, Up 0, Down 0
```

The routing path and operational state of the *r6–r4* LSP, combined with the successful establishment of a bypass LSP protecting r3's so-0/2/0.100 interface, confirms that you have met all the criteria for this configuration task.

Preemption

LSP priority settings determine the relative setup priority of an LSP, and also control the likelihood of that LSP being torn down in an effort to establish a new, higher-priority LSP. Preemption is normally used when LSPs reserve bandwidth to ensure that high-priority LSPs can be established in the face of reduced network capacity. Setup priority is also used to influence the order in which LSPs are signaled; higher-priority LSPs are established first in the event of a reboot or manual clearing of RSVP sessions.

The default preemption behavior results in low-priority LSPs being torn down when a *new* higher-priority LSP must be signaled and there is insufficient bandwidth to accommodate all LSPs. Setting preemption to aggressive modifies this behavior by allowing preemption in the event of bandwidth reduction as well as the need to establish a new LSP. Preemption can also be disabled. Preemption operates based on two LSP priority settings, namely the LSP's *setup* and *hold* priority. An LSP's setup priority determines the order in which it is signaled, and the likelihood of it being able to preempt an already established LSP. The LSP's hold priority determines whether the LSP can be preempted by another LSP with a high setup priority. Put simply, setup priority determines the likelihood of this LSP preempting another session, while the hold priority determines whether this LSP can in turn be preempted by another session.

Priority settings range from 0–7, with 0 being the strongest and 7 being the weakest. By default, all LSPs use the weakest setup priority (0) and the strongest hold priority (7), such that preemption is not possible. To enable LSP preemption, you must elevate a new LSP's setup priority, while also lowering the hold priority of existing LSPs.

To complete this configuration scenario, you must configure r5 according to the following criteria:

- Establish LSP *r5–r1* and *r5–r1–prime* with r5 as ingress and r1's 10.0.4.5 fe-0/0/2 interface address as the egress.
- Configure each LSP to reserve 100Mbps of bandwidth.
- Make sure that r5 signals the *r5–r1* LSP first, and ensure that the *r5–r1* LSP is established in favor of the *r5–r1–prime* LSP.
- Use a single ERO that forces the LSPs routing through r3.

Configuring Preemption

The restriction on ERO usage results in r5 attempting to establish both LSPs over a common path. With their 200Mbps aggregate bandwidth requirement, something will have to give. You begin your configuration on the ingress router by defining the new LSPs and the required explicit path through r3. Note that CSPF is turned off, due to previously described TED issues.

```
[edit protocols mpls]
lab@r5# set label-switched-path r5-r1 to 10.0.4.5 no-cspf

[edit protocols mpls]
lab@r5# set label-switched-path r5-r1 bandwidth 100m

[edit protocols mpls]
lab@r5# set label-switched-path r5-r1 primary r5-r1

[edit protocols mpls]
lab@r5# set path r5-r1 10.0.3.3 loose
```

The *r5–r1–prime* LSP is now defined, along with its explicit path:

```
[edit protocols mpls]
lab@r5# set label-switched-path r5-r1-prime to 10.0.4.5 no-cspf

[edit protocols mpls]
lab@r5# set label-switched-path r5-r1-prime bandwidth 100m

[edit protocols mpls]
lab@r5# set label-switched-path r5-r1-prime primary r5-r1-prime

[edit protocols mpls]
lab@r5# set path r5-r1-prime 10.0.3.3 loose
```

The current LSP configuration at r5 is now displayed with the recent additions highlighted:

```
[edit protocols mpls]
lab@r5# show
admin-groups {
    blue 4;
    red 8;
}
label-switched-path r5-r1 {
    to 10.0.4.5;
    bandwidth 100m;
    no-cspf;
```

```
    primary r5-r1;
}
label-switched-path r5-r1-prime {
    to 10.0.4.5;
    bandwidth 100m;
    no-cspf;
    primary r5-r1-prime;
}
path r5-r1 {
    10.0.3.3 loose;
}
path r5-r1-prime {
    10.0.3.3 loose;
}
interface all;
interface at-0/2/1.0 {
    admin-group red;
}
interface so-0/1/0.0 {
    admin-group red;
}
```

Before modifying any LSP setup and hold priorities, you commit the changes made thus far to verify the network's behavior with the default priority settings:

```
[edit protocols mpls]
lab@r5# run show mpls lsp ingress detail
Ingress LSP: 2 sessions

10.0.4.5
  From: 10.0.3.5, State: Up, ActiveRoute: 0, LSPname: r5-r1
  ActivePath: r5-r1 (primary)
  LoadBalance: Random
  Encoding type: Packet, Switching type: Packet, GPID: IPv4
 *Primary   r5-r1            State: Up
    Bandwidth: 100Mbps
    Received RRO:
          10.0.2.2 10.0.4.14

10.0.4.5
  From: 10.0.3.5, State: Dn, ActiveRoute: 0, LSPname: r5-r1-prime
  ActivePath: (none)
```

```
  LoadBalance: Random
  Encoding type: Packet, Switching type: Packet, GPID: IPv4
  Primary   r5-r1-prime      State: Dn
    Bandwidth: 100Mbps
    2 Feb 19 11:42:28  Requested bandwidth unavailable[13 times]
Total 2 displayed, Up 1, Down 1
```

As predicted, r5 is not able to simultaneously establish both LSPs due to insufficient bandwidth at r1's fe-0/0/1 interface. In this example, the *r5–r1* LSP was successfully established, but the default priority settings currently in effect result in this behavior being non-deterministic. To confirm, you temporarily deactivate the *r5–r1* LSP, thereby allowing the establishment of the *r5–r1–prime* LSP:

```
[edit protocols mpls]
lab@r5# deactivate label-switched-path r5-r1

[edit protocols mpls]
lab@r5# commit
commit complete

[edit protocols mpls]
lab@r5# activate label-switched-path r5-r1

[edit protocols mpls]
lab@r5# commit
commit complete
```

As expected, you now find that the *r5–r1–prime* LSP has been established, this time to the detriment of the *r5–r1* LSP:

```
[edit protocols mpls]
lab@r5# run show mpls lsp ingress detail
Ingress LSP: 2 sessions

10.0.4.5
  From: 10.0.3.5, State: Up, ActiveRoute: 0, LSPname: r5-r1-prime
  ActivePath: r5-r1-prime (primary)
  LoadBalance: Random
  Encoding type: Packet, Switching type: Packet, GPID: IPv4
 *Primary   r5-r1-prime      State: Up
    Bandwidth: 100Mbps
    Received RRO:
          10.0.2.2 10.0.4.2 10.0.4.5
```

```
10.0.4.5
  From: 10.0.3.5, State: Dn, ActiveRoute: 0, LSPname: r5-r1
  ActivePath: (none)
  LoadBalance: Random
  Encoding type: Packet, Switching type: Packet, GPID: IPv4
  Primary   r5-r1            State: Dn
    Bandwidth: 100Mbps
    2 Feb 19 11:48:58  Requested bandwidth unavailable[4 times]
Total 2 displayed, Up 1, Down 1
```

You now modify the priority settings to ensure that the *r5-r1* LSP can preempt the *r5-r1-prime* LSP when needed:

```
[edit protocols mpls]
lab@r5# set label-switched-path r5-r1 priority 4 0
```

The previous command modifies the setup priority for the *r5-r1* LSP from the default value of *7* (weak) to a setting of *4* (medium). The hold priority is left at the default setting of *0* (strong) in this case. A similar command is now used to adjust the hold priority for the *r5-r1-prime* LSP:

```
[edit protocols mpls]
lab@r5# set label-switched-path r5-r1-prime priority 7 5
```

The modified configuration at r5 is now displayed:

```
[edit protocols mpls]
lab@r5# show
admin-groups {
    blue 4;
    red 8;
}
label-switched-path r5-r1 {
    to 10.0.4.5;
    bandwidth 100m;
    no-cspf;
    priority 4 0;
    primary r5-r1;
}
label-switched-path r5-r1-prime {
    to 10.0.4.5;
    bandwidth 100m;
    no-cspf;
    priority 7 5;
    primary r5-r1-prime;
}
```

```
path r5-r1 {
    10.0.3.3 loose;
}
path r5-r1-prime {
    10.0.3.3 loose;
}
interface all;
interface at-0/2/1.0 {
    admin-group red;
}
interface so-0/1/0.0 {
    admin-group red;
}
```

With a hold priority of *5*, the *r5-r1-prime* LSP should be torn down to accommodate the higher setup priority (*4*) of the *r5-r1* LSP. With the changes committed, proper operation is easily verified:

```
[edit protocols mpls]
lab@r5# run show mpls lsp ingress extensive
Ingress LSP: 2 sessions

10.0.4.5
  From: 10.0.3.5, State: Up, ActiveRoute: 0, LSPname: r5-r1
  ActivePath: r5-r1 (primary)
  LoadBalance: Random
  Encoding type: Packet, Switching type: Packet, GPID: IPv4
 *Primary   r5-r1             State: Up
    Priorities: 4 0
    Bandwidth: 100Mbps
    Received RRO:
          10.0.2.2 10.0.4.2 10.0.4.5
    4 Feb 19 12:11:24  Selected as active path
    3 Feb 19 12:11:24  Record Route:  10.0.2.2 10.0.4.2 10.0.4.5
    2 Feb 19 12:11:24  Up
    1 Feb 19 12:11:24  Originate Call
  Created: Wed Feb 19 12:11:11 2003

10.0.4.5
  From: 10.0.3.5, State: Dn, ActiveRoute: 0, LSPname: r5-r1-prime
  ActivePath: (none)
  LoadBalance: Random
```

```
  Encoding type: Packet, Switching type: Packet, GPID: IPv4
  Primary   r5-r1-prime      State: Dn
    Priorities: 7 5
    Bandwidth: 100Mbps
    8 Feb 19 12:15:27  Requested bandwidth unavailable[19 times]
    7 Feb 19 12:11:24  ResvTear received
    6 Feb 19 12:11:24  Requested bandwidth unavailable
    5 Feb 19 12:11:24  Session preempted
    4 Feb 19 12:11:24  Down
    3 Feb 19 12:11:24  Record Route:  10.0.2.2 10.0.4.14
    2 Feb 19 12:11:24  Up
    1 Feb 19 12:11:24  Originate Call
  Created: Wed Feb 19 12:11:11 2003
Total 2 displayed, Up 1, Down 1
```

The display confirms that the setup and hold priorities have been modified, and also confirms that the *r5-r1-prime* LSP has been preempted by the *r5-r1* LSP. This behavior is in full accordance with the requirements of the configuration scenario.

Summary of Traffic Protection

JUNOS software supports a wide range of options that are used to protect and prioritize LSPs. This section demonstrated the use of secondary paths, and how the `standby` option results in the pre-establishment of a secondary path to minimize disruption in the event of primary path failure. The section also detailed how Fast Reroute and link protection can be used to protect an LSP's path, or particular interfaces, without the need for secondary paths. The need for CSPF computations to support both Fast Reroute and link protection was described in detail, as were issues that can occur when dealing with a network comprising multiple TE domains.

The section ended with an example of LSP preemption, which demonstrated how LSP setup and hold priorities work in conjunction with RSVP preemption to help ensure that high-priority paths are established first, and remain established, in the face of insufficient network bandwidth. The difference between normal and aggressive preemption was also described.

Miscellaneous MPLS Capabilities and Features

This section provides various MPLS configuration tasks that will demonstrate MPLS features not incorporated into the previous configuration scenarios.

To complete this section, you must modify your network to meet these requirements:

- Make changes only to the ingress router and ensure that the topology of the *r2-r6* LSP is not displayed in traceroute tests.

- Configure the *r5–r1* LSP so that you can validate the data plane without using external prefixes or modifying default routing table integration.
- Ensure that r6 performs a pop operation on all LSPs for which it serves as the egress node.
- Modify the *r4–r3* LSP at r4 so it reserves a minimum of 1Mbps of bandwidth while ensuring that the bandwidth reservation can be automatically adjusted during periods of high traffic volume.

The first configuration task requires that you hide the presence of LSP forwarding for traceroutes conducted over the *r2–r6* LSP. In theory, this goal can be achieved using either the no-propagate-ttl or no-decrement-ttl configuration options. However, because no-propagate-ttl must be configured at all nodes along the LSP's path, you will have to use the no-decrement-ttl approach due to the restriction that you modify only the ingress router's configuration.

Before altering the default traceroute behavior of the *r2–r6* LSP, you perform a quick test to verify that LSP forwarding is still in effect, and to serve as a contrast to the behavior you hope to soon achieve:

```
[edit]
lab@r2# run traceroute 220.220.0.1
traceroute to 220.220.0.1 (220.220.0.1), 30 hops max, 40 byte packets
 1  10.0.4.9 (10.0.4.9)  0.676 ms  9.119 ms  0.433 ms
     MPLS Label=100009 CoS=0 TTL=1 S=1
 2  10.0.2.17 (10.0.2.17)  0.182 ms  0.175 ms  0.155 ms
     MPLS Label=100003 CoS=0 TTL=1 S=1
 3  10.0.8.9 (10.0.8.9)  0.571 ms  0.515 ms  0.492 ms
     MPLS Label=100016 CoS=0 TTL=1 S=1
 4  10.0.8.5 (10.0.8.5)  0.212 ms  0.205 ms  0.186 ms
 5  220.220.0.1 (220.220.0.1)  0.300 ms  0.283 ms  0.265 ms
```

The results confirm LSP forwarding to C2 routes, making the LSP's topology open for all to see. You now modify r2's configuration by flagging the *r2–r6* LSP as having no-decrement-ttl:

```
[edit protocols]
lab@r2# set mpls label-switched-path r2-r6 no-decrement-ttl

[edit protocols]
lab@r2# show mpls label-switched-path r2-r6
to 10.0.9.6;
no-decrement-ttl;
no-cspf;
primary visit-r7-r5;
```

After committing the changes, you verify that the LSP's path is now hidden to traceroutes:

```
[edit]
lab@r2# commit
```

```
commit complete

[edit]
lab@r2# run traceroute 220.220.0.1
traceroute to 220.220.0.1 (220.220.0.1), 30 hops max, 40 byte packets
 1  220.220.0.1 (220.220.0.1)  0.547 ms  0.350 ms  0.290 ms
```

As expected, only a single hop is shown for traceroute testing to C2 prefixes. Having changed the configuration of only the ingress router, you have met all requirements for this task.

The next configuration requirement dictates that you be able to test the data plane (LSP forwarding) of the *r5–r1* LSP without changing the default LSP routing table integration behavior at r5, and without using external destinations as the target of your testing. The only way to accomplish this goal is to use MPLS pings, which are a relatively new JUNOS software feature. However, the results of your first MPLS ping attempt indicate that nothing in the JNCIE lab is as simple as it first appears:

```
[edit]
lab@r5# run ping mpls rsvp r5-r1
.....
--- lsping statistics ---
5 packets transmitted, 0 packets received, 100% packet loss
```

The most frustrating part of the MPLS ping failure is that show commands indicate that the LSP is operational and you are unable to verify the LSP's data plane by tracing the path to P1's 120.120/16 routes, if only for peace of mind, due to the current state of the test bed! Recall that traffic from r5 to 120.120/16 destinations does not use the *r5–r1* LSP because the BGP next hop of 10.0.5.254 cannot be resolved through r5's inet.3 routing table. Having been precluded from altering the default routing table integration behavior on r5, there is little you can do to change this situation.

```
[edit]
lab@r5# run traceroute 120.120.0.1
traceroute to 120.120.0.1 (120.120.0.1), 30 hops max, 40 byte packets
 1  10.0.2.10 (10.0.2.10)  0.823 ms  0.668 ms  0.602 ms
 2  10.0.4.10 (10.0.4.10)  0.540 ms  0.550 ms  0.500 ms
 3  120.120.0.1 (120.120.0.1)  0.618 ms  0.634 ms  0.581 ms
```

Traffic from r5 to 120.120/16 destinations is not using the *r5–r1* LSP because the associated BGP next hop of 10.0.5.254 cannot be resolved in r5's inet.3 table.

```
[edit]
lab@r5# run show route table inet.3

inet.3: 3 destinations, 3 routes (3 active, 0 holddown, 0 hidden)
Restart Complete
+ = Active Route, - = Last Active, * = Both
```

```
10.0.4.5/32        *[RSVP/7] 04:37:00, metric 30
                    > via at-0/2/1.0, label-switched-path r5-r1
10.0.9.6/32        *[LDP/9] 03:42:09, metric 1
                    > to 10.0.8.5 via fe-0/0/0.0, Push 0
10.0.9.7/32        *[LDP/9] 03:42:34, metric 1
                    > to 10.0.8.10 via fe-0/0/1.0
```

Given these particulars, it would seem that an MPLS ping really is the *only* way for you to verify the LSP's data plane after all. The trick to getting MPLS pings to work lies in understanding that the MPLS ping mechanism always targets a destination address of 127.0.0.1. The lack of a 127.0.0.1 loopback address assignment on the egress router's lo0 interface will result in the silent discard of the MPLS ping requests. You rectify this situation with the addition of the 127.0.0.1 loopback address to r1's loopback interface:

```
[edit interfaces]
lab@r1# set lo0 unit 0 family inet address 127.0.0.1

[edit]
lab@r1# commit
commit complete

[edit]
lab@r1# run show interfaces terse lo0
Interface               Admin Link Proto Local                 Remote
lo0                     up    up
lo0.0                   up    up   inet  10.0.6.1            --> 0/0
                                        127.0.0.1            --> 0/0
                                   iso   49.0001.1111.1111.1111.00
```

After assigning a loopback address to r1, the MPLS ping test is repeated at r5:

```
[edit]
lab@r5# run ping mpls rsvp r5-r1
!!!!!
--- lsping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
```

The 100 percent success rate leaves little to be desired! The data plane of the *r5–r1* LSP has been confirmed in full accordance with all restrictions.

Your next assignment is to configure the network so that r6 performs a pop function for all traffic arriving on its egress LSPs. The wording of the task is designed to test a candidate's MPLS knowledge, in that even understanding what is being asked for requires that the candidate be familiar with Penultimate Hop Popping (PHP) and the significance of the explicit and implicit null label values 0 and 3, respectively. Before changing the configuration of r6, you first confirm

that r6 has requested the default PHP behavior for its LDP and RSVP signaled egress LSPs:

```
[edit]
lab@r6# run show ldp database
Input label database, 10.0.9.6:0--10.0.3.5:0
  Label     Prefix
      3     10.0.3.5/32
 100000     10.0.9.6/32
 100003     10.0.9.7/32

Output label database, 10.0.9.6:0--10.0.3.5:0
  Label     Prefix
 100007     10.0.3.5/32
      3     10.0.9.6/32
 100008     10.0.9.7/32

[edit]
lab@r6# run show rsvp session egress
Egress RSVP: 1 sessions
To              From            State Rt Style Labelin Labelout LSPname
10.0.9.6        10.0.6.2        Up     0  1 FF       3        - r2-r6
Total 1 displayed, Up 1, Down 0
```

The presence of the implicit null label (3) in both displays indicates that r6 is currently performing no label pop operations on its egress LSP traffic due to PHP behavior. You now modify r6's configuration so it will signal its desire to receive an explicit null label (0), which it will then pop in full accordance with the requirements of this configuration task:

```
[edit]
lab@r6# set protocols ldp explicit-null

[edit]
lab@r6# set protocols mpls explicit-null
```

To Pop or Not to Pop?

While PHP behavior is a fine default, you may need to configure the use of an explicit null label for compatibility with the MPLS implementations of equipment made by other companies. You may also want to leave a null MPLS label for processing at the egress node when MPLS CoS, based on the use of the EXP bits in the shim header, is in effect. By leaving a MPLS label on the packet, the egress node is able to classify and queue based on the settings of the MPLS EXP bits as opposed to the Diffserv/ToS bits in the IP header.

The changes to r6's configuration are now displayed:

```
[edit]
lab@r6# show protocols ldp
traffic-statistics {
    file ldp-stats;
    interval 90;
}
explicit-null;
keepalive-interval 5;
interface fe-0/1/0.0;
interface fe-0/1/1.0;

[edit]
lab@r6# show protocols mpls
explicit-null;
label-switched-path r6-r4 {
    to 10.0.3.4;
    link-protection;
    primary r6-r4;
}
path r6-r4 {
    10.0.3.3 loose;
}
interface all;
```

After a commit, the results are easily verified:

```
[edit]
lab@r6# run show ldp database
Input label database, 10.0.9.6:0--10.0.3.5:0
  Label     Prefix
      3     10.0.3.5/32
 100000     10.0.9.6/32
 100003     10.0.9.7/32

Output label database, 10.0.9.6:0--10.0.3.5:0
  Label     Prefix
 100009     10.0.3.5/32
      0     10.0.9.6/32
 100010     10.0.9.7/32
```

```
[edit]
lab@r6# run show rsvp session egress
Egress RSVP: 1 sessions
To              From            State Rt Style Labelin Labelout LSPname
10.0.9.6        10.0.6.2        Up     0  1 FF       0        - r2-r6
```

The use of the explicit null label for r6's LDP and RSVP signaled egress LSPs indicates you have successfully completed this task.

Attention to detail is important! Many candidates who are presented with a configuration task such as this one will correctly configure the explicit null label for one signaling protocol or the other, but few will correctly modify the behavior of *both* signaling protocols!

The final task in this section requires that you modify the *r4–r3* LSP so that it will adaptively request bandwidth reservations based on actual LSP usage. To achieve this goal, you need to make use of the JUNOS software `auto-bandwidth` feature. This feature causes a new LSP to be signaled when the actual utilization rate of the LSP no longer matches the existing LSP's current bandwidth reservation. In operation, a new fixed filter (FF) style reservation with a higher (or lower) bandwidth reservation is signaled when MPLS statistics indicate that the actual LSP bandwidth utilization no longer matches the LSP's reserved bandwidth. Once the new LSP is established, traffic is switched over and the original LSP is torn down. Configuration settings allow you to limit the LSP's minimum and maximum bandwidth reservation.

You begin configuration of automatic bandwidth by enabling the gathering of MPLS statistics for use by the `auto-bandwidth` feature:

```
[edit protocols mpls]
lab@r4# set statistics auto-bandwidth

[edit protocols mpls]
lab@r4# set statistics file mpls-stats
```

You now modify the *r4–r3* LSP so that it uses the `auto-bandwidth` feature, being careful to specify a minimum bandwidth of 1Mbps under the *lsp-name* portion of the hierarchy:

```
[edit protocols mpls label-switched-path r4-r3]
lab@r4# set auto-bandwidth minimum-bandwidth 1M
```

The automatic bandwidth related changes are now displayed with highlights:

```
[edit protocols mpls]
lab@r4# show
statistics {
    file mpls-stats;
    auto-bandwidth;
}
```

```
admin-groups {
    blue 4;
    red 8;
}
label-switched-path r4-r3 {
    to 10.0.3.3;
    admin-group {
        include red;
        exclude blue;
    }
    auto-bandwidth {
        minimum-bandwidth 1m;
    }
}
interface so-0/1/0.100 {
    admin-group blue;
}
interface so-0/1/1.0 {
    admin-group red;
}
interface fe-0/0/1.0;
interface fe-0/0/2.0;
interface fe-0/0/3.0;
```

Proper operation is confirmed after the changes are committed:

```
[edit]
lab@r4# run show mpls lsp ingress detail
Ingress LSP: 1 sessions

10.0.3.3
  From: 10.0.3.4, State: Up, ActiveRoute: 117276, LSPname: r4-r3
  ActivePath:  (primary)
  LoadBalance: Random
  Autobandwidth
  MinBW: 1000kbps
  AdjustTimer: 86400 secs
  Max AvgBW util: 0bps, Bandwidth Adjustment in 85290 second(s).
  Encoding type: Packet, Switching type: Packet, GPID: IPv4
 *Primary                    State: Up
    Include: red    Exclude: blue
    Computed ERO (S [L] denotes strict [loose] hops): (CSPF metric: 20)
          10.0.2.9 S 10.0.2.2 S
```

```
    Received RRO:
          10.0.2.9 10.0.2.2
Total 1 displayed, Up 1, Down 0
```

The display indicates that automatic bandwidth adjustments are in effect for the *r4–r3* LSP, and that the minimum bandwidth has been correctly set to 1Mbps. Setting the `adjust-interval` to a value lower than the 86,000-second default allows you to confirm that the minimum bandwidth is *actually* reserved (due to the low amount of traffic in the test bed, an automatic bandwidth increase is unlikely to occur) without having to wait for an inordinate amount of time:

```
[edit protocol mpls label-switched-path r4-r3]
lab@r4# set auto-bandwidth adjust-interval 300
```

After 5 minutes or so, the LSP and RSVP interface status is again displayed at `r4`:

```
[edit protocols mpls]
lab@r4# run show mpls lsp ingress extensive
Ingress LSP: 1 sessions

10.0.3.3
  From: 10.0.3.4, State: Up, ActiveRoute: 0, LSPname: r4-r3
  ActivePath:  (primary)
  LoadBalance: Random
  Autobandwidth
  MinBW: 1000kbps
  AdjustTimer: 300 secs
  Max AvgBW util: 0bps, Bandwidth Adjustment in 171 second(s).
  Encoding type: Packet, Switching type: Packet, GPID: IPv4
 *Primary                    State: Up
    Bandwidth: 1000kbps
    Include: red    Exclude: blue
    Computed ERO (S [L] denotes strict [loose] hops): (CSPF metric: 20)
          10.0.2.9 S 10.0.2.2 S
    Received RRO:
          10.0.2.9 10.0.2.2
   17 Feb 20 01:07:52  Change in active path
   16 Feb 20 01:07:52  Record Route:  10.0.2.9 10.0.2.2
   15 Feb 20 01:07:52  Up
   14 Feb 20 01:07:52  Autobw adjustment succeeded
   13 Feb 20 01:07:51  CSPF: computation result accepted
   12 Feb 20 01:02:56  Selected as active path
   11 Feb 20 01:02:56  Record Route:  10.0.2.9 10.0.2.2
   . . .
  Created: Thu Feb 20 00:36:38 2003
Total 1 displayed, Up 1, Down 0
```

```
[edit protocols mpls]
lab@r4# run show rsvp interface
RSVP interface: 6 active
                    Active                Static      Available  Reserved  Highwater
Interface     State resv   Subscription   BW          BW         BW        mark
fe-0/0/0.0    Up       0   100%           100Mbps     100Mbps    0bps      0bps
fe-0/0/1.0    Up       0   100%           100Mbps     100Mbps    0bps      100Mbps
fe-0/0/2.0    Up       0   100%           100Mbps     100Mbps    0bps      100Mbps
fe-0/0/3.0    Up       1   100%           100Mbps     100Mbps    0bps      0bps
so-0/1/0.100  Up       0   100%           155.52Mbps  155.52Mbps 0bps      0bps
so-0/1/1.0    Up       1   100%           155.52Mbps  154.52Mbps 1000kbps  1000kbps
```

The output shows that a 1Mbps reservation is in place, and that automatic bandwidth adjustments have succeeded. Now we can move to the chapter case study, as you have completed all requirements for this section.

Summary

JNCIE candidates are expected to configure a variety of MPLS-related features in the lab exam. Successful candidates will be fluent with virtually all of the LSP signaling, routing table integration, protection, and general usage options available in the JUNOS software.

This chapter provided configuration scenarios and verification techniques for LDP and RSVP signaled LSPs, and also demonstrated how LSP routing can be controlled with EROs and CSPF-based link coloring. The default rules for MPLS routing table integration was demonstrated, and various configuration scenarios showed how this behavior can be altered through the use of `install`, TE shortcuts, and/or `traffic-engineering bgp-igp`.

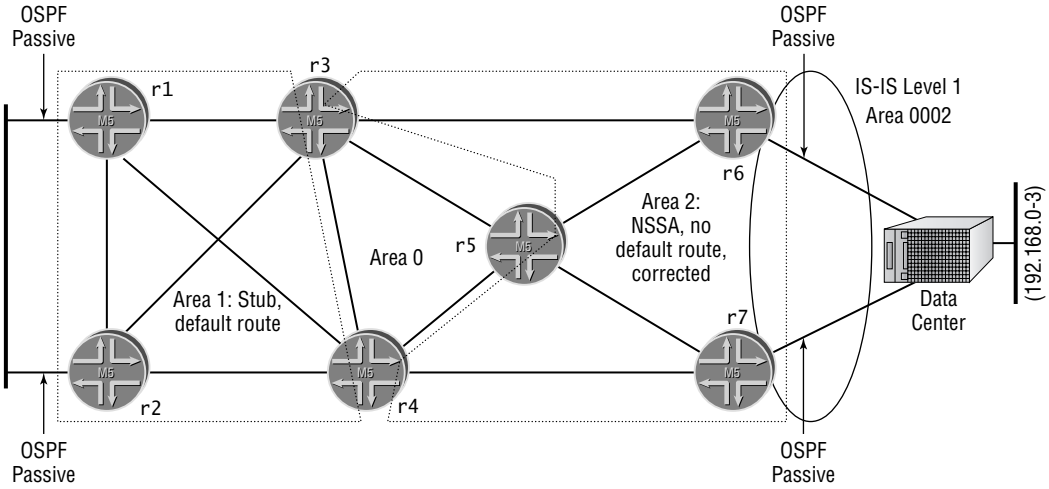
The chapter went on to describe LSP protection options, and demonstrated the configuration and verification of standby paths, Fast Reroute, and link protection. The final configuration section provided examples of miscellaneous MPLS features, which included automatic bandwidth adjustment, hiding an LSP's topology through the use of `no-decrement-ttl`, and how to configure a router to support MPLS pings.

Case Study: MPLS and Traffic Engineering

This chapter case study is designed to simulate a typical JNCIE-level MPLS and traffic engineering configuration scenario. In the interest of "keeping you on your toes," you will be performing your MPLS and traffic engineering case study using the OSPF baseline configuration

that was discovered and documented in the body of Chapter 1. The OSPF baseline topology is shown in Figure 2.7 so you can reacquaint yourself with it.

FIGURE 2.7 OSPF discovery findings



Notes:

Loopback addresses have not been assigned to specific areas (lo0 address advertised in Router LSA in all areas).

Passive OSPF interfaces on P1 and data center segments.

No authentication or route summarization in effect; summaries (LSA type 3) allowed in all areas.

Redistribution of OSPF default route to data center from both `r6` and `r7` was broken. Fixed with default-metric command on `r3`, `r4`, and `r5`.

Data center router running IS-IS, Level 1. `r6` and `r7` compatibly configured and adjacent.

Redistribution of 192.168.0/24 through 192.168.3/24 into OSPF from IS-IS by both `r6` and `r7`.

Adjustment to IS-IS level 1 external preference to ensure `r6` and `r7` always prefer IS-IS level 1 externals over OSPF externals.

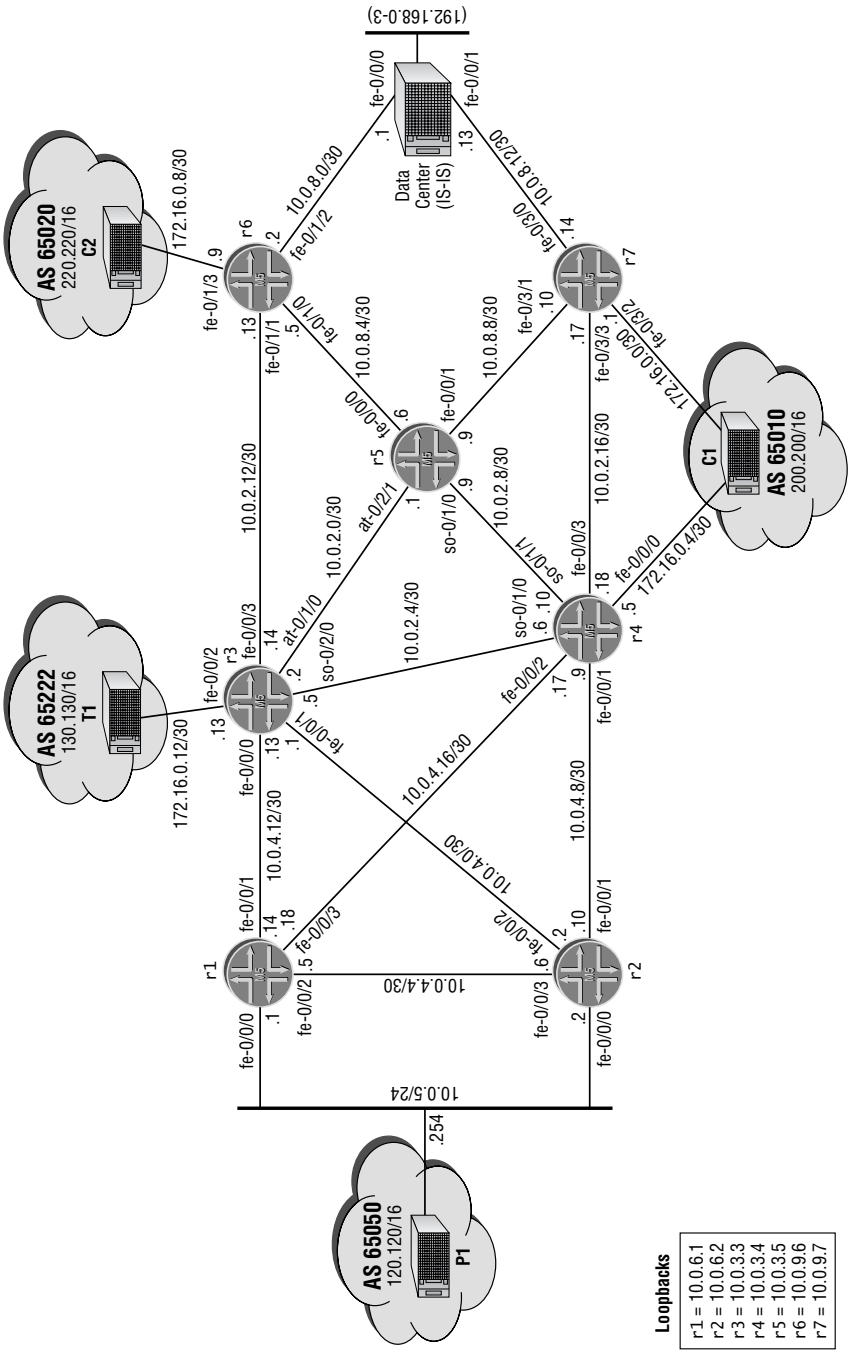
All adjacencies up and full reachability confirmed.

Sub-optimal routing detected at the data center router for some locations. This is the result of random nexthop choice for its default route. Considered to be working as designed; no action taken.

Because you will now be using the OSPF baseline topology, you should load and commit the baseline OSPF configuration to ensure that your routers will look and behave like the examples shown here. Before starting the MPLS case study, you should quickly verify the correct operation of the baseline network's OSPF IGP, IS-IS redistribution, and IBGP/EBGP peerings. Problems are not expected in the baseline network at this stage, but it never hurts to verify that you are, in fact, starting off with a functional network.

You will need to refer to the case study criteria listing and the case study topology, as shown in Figure 2.8, for the information needed to complete the MPLS/TE case study. It is expected that a JNCIE candidate will be able to complete this case study in approximately one hour, with the result being an MPLS network that exhibits no significant operational problems.

FIGURE 2.8 MPLS case study topology



Sample configurations from all seven routers are provided at the end of the case study for comparison with your own configurations. Because multiple solutions may be possible for a given task, differences between the examples provided and your own configurations do not automatically indicate that you have made a mistake. Because you are graded on the overall functionality of your network, and its conformance to the specified criteria, various operational mode commands are included so that you can compare the behavior of your network to that of a known good example.

To complete this case study, your MPLS and traffic engineering configuration must meet the following criteria:

- Your MPLS-related configuration must be added to the OSPF baseline topology from the body of Chapter 1.
- Enable labeled packet support and RSVP signaling for all internal-facing transit interfaces.
- Establish an LDP session between `r1` and `r7` *without* enabling LDP on `r3` and `r4`.
- Establish LSP *`r6–r1`* and *`r7–r1`*. Ensure that traffic to 120.120/16 prefixes are forwarded over these LSPs from `r6` and `r7`, respectively.
- Configure `r3`'s so-0/2/0.100 interface so that no more than 50Mbps of its bandwidth can be reserved by RSVP. Do not alter the default subscription percentage on this interface.
- Establish LSP *`r3–r7`*, and without using a secondary LSP, ensure that you provide protection for the entire LSP path.
- Establish LSP *`r4–r6`* with a 2Mbps reservation. Ensure that a backup path is pre-established, that no transit elements are shared between the two paths, and that the LSP signals an SE style of reservation.
- Establish LSPs *`r4–r3`* and *`r4–r3–prime`*. Ensure that prefixes with a length equal to or less than /20 are mapped to the *`r4–r3`* LSP while all other prefixes are mapped to the *`r4–r3–prime`* LSP. You must not modify the policy stanza at `r4` to achieve this goal.
- Configure `r5` and `r6` to authenticate RSVP signaling with the key *`jni`*.
- Configure `r5` and `r6` so that RSVP state is preserved in the event of a routing restart at `r5` or `r6`.
- Configure RSVP so that the loss of nine consecutive hello messages is required to declare neighbor loss between `r5` and `r6`.
- Configure `r5` and `r6` so that they *bundle* RSVP Path Tear and Error messages.

You can assume that the data center router has been reconfigured to advertise the 192.168.0-3/24 routes to both `r6` and `r7` using the IS-IS protocol. Please refer back to Chapter 1, or to your IGP discovery notes, as needed, for specifics on the OSPF and IS-IS route redistribution in the OSPF baseline network.

MPLS Case Study Analysis

Each configuration requirement for the case study will now be matched to one or more valid router configurations and, where applicable, the commands that are used to confirm whether your network is operating within the specified case study guidelines. We begin with these

criteria, as they serve to establish your baseline network and core MPLS support:

- Your MPLS-related configuration must be added to the OSPF baseline topology from the body of Chapter 1.
- Enable labeled packet support and RSVP signaling for all internal-facing transit interfaces.

The case study analysis begins with a quick operational sanity check of the OSPF baseline network:

```
[edit]
lab@r3# run show ospf neighbor
  Address          Interface              State      ID               Pri  Dead
10.0.2.1         at-0/1/0.0             Full       10.0.3.5         128   37
10.0.2.6         so-0/2/0.100           Full       10.0.3.4         128   37
10.0.4.14        fe-0/0/0.0             Full       10.0.6.1         128   35
10.0.4.2         fe-0/0/1.0             Full       10.0.6.2         128   31
10.0.2.13        fe-0/0/3.0             Full       10.0.9.6         128   35

[edit]
lab@r3# run show route protocol ospf | match /32
10.0.3.4/32        *[OSPF/10] 00:01:34, metric 1
10.0.3.5/32        *[OSPF/10] 00:01:34, metric 1
10.0.6.1/32        *[OSPF/10] 00:01:34, metric 1
10.0.6.2/32        *[OSPF/10] 00:01:34, metric 1
10.0.9.6/32        *[OSPF/10] 00:01:34, metric 1
10.0.9.7/32        *[OSPF/10] 00:01:29, metric 3
192.168.0.1/32     *[OSPF/150] 00:00:35, metric 10, tag 0
224.0.0.5/32       *[OSPF/10] 00:03:53, metric 1
```

All of r3's OSPF adjacencies have been fully established, and it has received an OSPF route for the loopback address of all other routers in the test bed. This output provides a strong indication that the baseline network's OSPF IGP is operational. You now verify BGP session status at r3:

```
[edit]
lab@r3# run show bgp summary
Groups: 4 Peers: 7 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
inet.0            125872     125860          0          0          0          0
Peer           AS  InPkt  OutPkt  OutQ  Flaps Last Up/Dwn State|#Active/
                                                          Received/Damped...
172.16.0.14 65222  29139   30023    0     0   3:42:37     125857/125857/0  0/0/0
10.0.3.4    65412    454   30297    0     0   3:44:46     1/1/0            0/0/0
10.0.3.5    65412     13   21965    0     1      5:37     0/0/0            0/0/0
10.0.6.1    65412    446   84470    0     1      5:02     1/1/0            0/0/0
10.0.6.2    65412    449   30012    0     0   3:43:16     0/1/0            0/0/0
10.0.9.6    65412     10   21875    0     1      3:12     1/6/0            0/0/0
10.0.9.7    65412     11   21878    0     1      3:55     0/6/0            0/
```

All of r3's IBGP and EBGP sessions are established, providing another good sign that the OSPF baseline network is operating normally. You next quickly confirm the presence of the expected IS-IS and BGP routes:

```
[edit]
lab@r3# run show route 120.120/16

inet.0: 125915 destinations, 125929 routes (125915 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

120.120.0.0/16     *[BGP/170] 00:06:35, MED 0, localpref 100, from 10.0.6.1
                      AS path: 65050 I
                    > to 10.0.4.14 via fe-0/0/0.0
                      to 10.0.4.2 via fe-0/0/1.0
                    [BGP/170] 03:44:49, MED 0, localpref 100, from 10.0.6.2
                      AS path: 65050 I
                    > to 10.0.4.14 via fe-0/0/0.0
                      to 10.0.4.2 via fe-0/0/1.0

[edit]
lab@r3# run show route 200.200/16

inet.0: 125915 destinations, 125929 routes (125915 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

200.200.0.0/16     *[BGP/170] 02:22:05, MED 0, localpref 100, from 10.0.3.4
                      AS path: 65010 I
                    > via so-0/2/0.100
                    [BGP/170] 00:03:26, MED 0, localpref 100, from 10.0.9.7
                      AS path: 65010 I
                    > to 10.0.2.13 via fe-0/0/3.0
```

Though not shown here for brevity reasons, you can assume that the 192.168.0/21, 130.130/16 and 220.220/16 routes are also present on r3. The output obtained from r3 indicates that the OSPF baseline network is operational, so you move on to the requirement that all routers in the test bed support MPLS labeled packets and RSVP signaling on their internal-facing transit interfaces. You will need to add the `mpls` family to the correct logical unit of all internal transit interfaces, and you must list these interfaces in both the `protocols mpls` and the `protocols rsvp` stanzas to meet the core MPLS functionality requirement. The following highlights call out the changes made to r5's configuration to provide core MPLS functionality:

```
[edit]
lab@r5# show interfaces
```

```
fe-0/0/0 {
    unit 0 {
        family inet {
            address 10.0.8.6/30;
        }
        family mpls;
    }
}
fe-0/0/1 {
    unit 0 {
        family inet {
            address 10.0.8.9/30;
        }
        family mpls;
    }
}
so-0/1/0 {
    encapsulation ppp;
    unit 0 {
        family inet {
            address 10.0.2.9/30;
        }
        family mpls;
    }
}
at-0/2/1 {
    atm-options {
        vpi 0 {
            maximum-vcs 64;
        }
    }
    unit 0 {
        point-to-point;
        vci 50;
        family inet {
            address 10.0.2.1/30;
        }
        family mpls;
    }
}
fxp0 {
```

```
    unit 0 {
        family inet {
            address 10.0.1.5/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 10.0.3.5/32;
        }
    }
}
```

These highlights confirm that the `mpls` family is correctly configured on `r5`'s internal-facing transit interfaces.

```
[edit]
lab@r5# show protocols
rsvp {
    interface all;
    interface fxp0.0 {
        disable;
    }
}
mpls {
    interface all;
}
bgp {
    group int {
        type internal;
        local-address 10.0.3.5;
        neighbor 10.0.6.1;
        neighbor 10.0.6.2;
        neighbor 10.0.3.3;
        neighbor 10.0.3.4;
        neighbor 10.0.9.6;
        neighbor 10.0.9.7;
    }
}
ospf {
    area 0.0.0.0 {
```

```
        interface at-0/2/1.0;
        interface so-0/1/0.0;
    }
    area 0.0.0.2 {
        nssa {
            default-lsa default-metric 10;
        }
        interface fe-0/0/0.0;
        interface fe-0/0/1.0;
    }
}
```

The highlighted portion of r5's `protocols` stanza shows that the `all` keyword has been used to associate r5's interfaces with the router's MPLS and RSVP processes. While you could have listed each of r5's internal transit interfaces explicitly, the use of `interface all` should not produce problems, especially because RSVP support has been explicitly disabled on the router's fxp0 interface. Core MPLS functionality is now confirmed on r5:

```
[edit]
lab@r5# run show mpls interface
Interface        State       Administrative groups
fe-0/0/0.0       Up          <none>
fe-0/0/1.0       Up          <none>
so-0/1/0.0       Up          <none>
at-0/2/1.0       Up          <none>

lab@r5# run show rsvp interface
RSVP interface: 4 active
                  Active                 Static      Available  Reserved  Highwater
Interface   State resv    Subscription   BW          BW         BW        mark
fe-0/0/0.0  Up         0           100%  100Mbps     100Mbps    0bps      0bps
fe-0/0/1.0  Up         1           100%  100Mbps     100Mbps    0Mbps     0Mbps
so-0/1/0.0  Up         0           100%  155.52Mbps  155.52Mbps 0bps      0bps
at-0/2/1.0  Up         0           100%  155.52Mbps  155.52Mbps 0bps      0bps
```

Before proceeding to the next case study configuration requirement, be sure that you add similar core MPLS functionality to the remaining routers in the test bed, and make sure that all routers display the correct interfaces as being MPLS and RSVP enabled. For those interfaces that are using a non-default logical unit number, in other words, r3's so-0/2/0.100 interface, be careful that you add the `mpls` family to the correct logical unit, and that the correct logical unit is in turn specified under the `mpls` and `rsvp` stanzas when the `all` keyword is not used.

You now address this case study requirement:

- Establish an LDP session between r1 and r7 without enabling LDP on r3 and r4.

Because LDP sessions are normally established between neighboring routers, the requirement that you establish an LDP session between r1 and r7, without running LDP on intervening routers r3 and r4, poses somewhat of a dilemma. You will need to tunnel the LDP session between r1 and r7 through an RSVP signaled LSP to achieve this goal. There are several ways that you can decide to tunnel the LDP session between r1 and r7; for example, you could run LDP between r1 and r2, and between r5 and r7 with LDP tunneling occurring over an RSVP LSP that is established between r2 and r5. In this example, you decide to run LDP only on r1 and r7, with the LDP tunneling occurring over a RSVP signaled LSP between the same pair of routers.

These highlights call out the changes made to r1's configuration to support LDP, and RSVP-based LDP tunneling:

```
[edit]
lab@r1# show protocols mpls
label-switched-path r1-r7 {
    to 10.0.9.7;
    ldp-tunneling;
    no-cspf;
}
interface all;

[edit]
lab@r1# show protocols ldp
interface lo0.0;
```

You must enable LDP on the router's lo0 interface for extended neighbor discovery to succeed. The RSVP-based LSP definition at r1 is pretty straightforward, except for the presence of the ldp-tunneling keyword. This aptly named option is needed to enable LDP session tunneling over a given RSVP signaled LSP. CSPF has been turned off on the LSP, in part because it is not needed, but also due to the lack of a domain-wide TED in the current test bed. Recall that, unlike IS-IS, OSPF does not automatically build a TED without explicit configuration.

Keep in mind that you must establish a similar RSVP tunnel in the r7-to-r1 direction so that bidirectional LDP communications, as needed for successful session establishment, is possible between the remote LDP neighbors. Though not shown, r7's configuration has undergone similar modifications.

After committing the changes, you verify that the RSVP sessions are correctly established between r1 and r7:

```
[edit]
lab@r7# run show rsvp session
Ingress RSVP: 1 sessions
To              From            State Rt Style Labelin Labelout LSPname
10.0.6.1        10.0.9.7        Up     0  1 FF       -   100022 r7-r1
Total 1 displayed, Up 1, Down 0
```

```
Egress RSVP: 1 sessions
To              From            State Rt Style Labelin Labelout LSPname
10.0.9.7        10.0.6.1        Up     0  1 FF       3        - r1-r7
Total 1 displayed, Up 1, Down 0

Transit RSVP: 0 sessions
Total 0 displayed, Up 0, Down 0
```

Note that the display confirms the presence of bidirectional LSPs between `r1` and `r7`. With the RSVP sessions established, you now confirm that LDP tunneling is operational:

```
[edit]
lab@r7# run show ldp neighbor
Address            Interface          Label space ID         Hold time
10.0.6.1           lo0.0              10.0.6.1:0               10

[edit]
lab@r7# run show ldp session
  Address           State        Connection     Hold time
10.0.6.1            Operational  Open             24

[edit]
lab@r7# run show ldp database
Input label database, 10.0.9.7:0--10.0.6.1:0
  Label     Prefix
      3     10.0.6.1/32

Output label database, 10.0.9.7:0--10.0.6.1:0
  Label     Prefix
      3     10.0.9.7/32
```

The output confirms that `r1` and `r7` have successfully established an extended LDP session through the RSVP signaled LSP. Though not shown here, you can assume that `r1` produces similar output, which confirms that you have met the requirements of this task. Note that a LDP LSP is not installed in the `inet.3` routing table in this case. This is because LDP traffic will be forwarded over the RSVP LSP that terminates on the LDP-speaking routers:

```
[edit]
lab@r1# run show route table inet.3

inet.3: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.9.7/32        *[RSVP/7] 00:44:39, metric 2
```

Given this topology, the lack of LDP entries in the `inet.3` table is expected, so you move on to the next case study requirement:

- Establish LSP *r6–r1* and *r7–r1*. Ensure that traffic to 120.120/16 prefixes is forwarded over these LSPs from `r6` and `r7`, respectively.

The only tricky aspect of this configuration task is your need to modify the default LSP routing table integration behavior at `r6` and `r7`, due to `r1` and `r2` not overwriting the BGP next hop for the routes they receive from P1. The Multi-Area OSPF topology now in effect will cause problems for any approach that makes use of TE shortcuts. This is because TE shortcuts are computed based on matching an OSPF router ID (RID) with the egress address of an LSP. However, the fact that OSPF router IDs are known only within a given area means that `r6` and `r7` will be unable to perform this LSP-Egress-to-OSPF-RID match, therefore making TE shortcuts ineffective in this particular scenario.

With TE shortcuts off the table, you will have to use `install` to place P1's next hop into the `inet.3` table on both `r6` and `r7`. The `active` keyword is not needed, as LSP forwarding is only required to the external prefixes advertised by P1. The modifications made to the configuration of `r6` in support of this task are shown next with highlights:

```
[edit]
lab@r6# show protocols mpls
label-switched-path r6-r1 {
    to 10.0.6.1;
    install 10.0.5.254/32;
    no-cspf;
}
interface all;
```

Note the use of `install`, and that CSPF has been disabled due to issues with missing or incomplete TEDs in the current test bed. The presence of P1's BGP next hop is confirmed in `r6`'s `inet.3` routing table:

```
[edit]
lab@r6# run show route table inet.3

inet.3: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.5.254/32      *[RSVP/7] 00:02:05, metric 2
                    > to 10.0.2.14 via fe-0/1/1.0, label-switched-path r6-r1
10.0.6.1/32        *[RSVP/7] 00:02:05, metric 2
                    > to 10.0.2.14 via fe-0/1/1.0, label-switched-path r6-r1
```

Traceroute testing to P1 advertised prefixes from both `r6` and `r7` provides the final confirmation for this task:

```
[edit]
lab@r6# run traceroute 120.120.0.1
traceroute to 120.120.0.1 (120.120.0.1), 30 hops max, 40 byte packets
```

```
 1  10.0.2.14 (10.0.2.14)  0.607 ms  0.486 ms  0.434 ms
     MPLS Label=100004 CoS=0 TTL=1 S=1
 2  10.0.4.14 (10.0.4.14)  0.156 ms  0.150 ms  0.133 ms
 3  120.120.0.1 (120.120.0.1)  0.242 ms  0.228 ms  0.209 ms
```

The presence of LSP forwarding from r6 to the 120.120/16 route, as advertised by the P1 router, confirms that you have met the requirements for this task. Though not shown, similar modifications are made at r7, and the same operational behavior is confirmed.

The next case study requirement to be addressed is as follows:

- Configure r3's so-0/2/0.100 interface so that no more than 50Mbps of its bandwidth can be reserved by RSVP. Do not alter the default RSVP subscription percentage on this interface.

This task requires the modification of the bandwidth associated with the so-0/2/0.100 interface so that, from the perspective of RSVP, the interface is seen as having only 50Mbps of bandwidth available. This setting will accommodate the requirement that you not modify the default RSVP behavior of allowing 100 percent of an interface's bandwidth to be reserved, while still meeting the restriction on limiting the interface's aggregate subscriptions to no more than 50Mbps. The configuration changes needed on r3 to meet this requirement are shown next with highlights:

```
[edit]
lab@r3# show protocols rsvp
interface all;
interface fxp0.0 {
    disable;
}
interface so-0/2/0.100 {
    bandwidth 50m;
}
```

The correct behavior is readily confirmed:

```
[edit]
lab@r3# run show rsvp interface
RSVP interface: 5 active
                    Active               Static      Available   Reserved  Highwater
Interface     State resv   Subscription  BW          BW          BW        mark
fe-0/0/0.0    Up    1              100%  100Mbps     100Mbps     0bps      0bps
fe-0/0/1.0    Up    0              100%  100Mbps     100Mbps     0bps      0bps
fe-0/0/3.0    Up    0              100%  100Mbps     100Mbps     0bps      0bps
at-0/1/0.0    Up    0              100%  155.52Mbps  155.52Mbps  0bps      0bps
so-0/2/0.100  Up    0              100%  50Mbps      50Mbps      0bps      0bps
```

The highlighted sections of the output indicate that you have successfully limited the so-0/2/0.100 interface to 50Mbps of aggregate RSVP bandwidth reservations, without altering RSVP's belief that 100 percent of the interface's bandwidth can be reserved.

The next case study configuration requirement to be tackled is this:

- Establish LSP *r3–r7* and, without using a secondary LSP, ensure that you provide protection for the entire LSP path.

The requirement that you protect the entire path of the primary LSP without using a secondary LSP indicates that you need to use Fast Reroute (or link bypass). Because you must protect the entire LSP path, Fast Reroute makes a lot more sense than having to configure multiple bypass LSPs, but either approach is viable given the current restrictions. You will need to configure OSPF traffic engineering support for all routers attached to Area 2 to enable the CSPF-based Fast Reroute detour calculations. Enabling traffic engineering on `r3`, `r4`, `r5`, `r6`, and `r7` will result in a complete TED for Area 2, which in turn allows the use of CSPF at the ingress node and Fast Reroute computations at transit LSRs, as long as the LSP does not touch any non–Area 2 attached routers.

The changes made to `r3`'s configuration to support this case study task are displayed with highlights:

```
[edit protocols mpls]
lab@r3# show
label-switched-path r3-r7 {
    to 10.0.9.7;
    fast-reroute;
    primary r3-r7;
}
path r3-r7 {
    10.0.3.4 loose;
}
interface all;
```

The modifications to `r3`'s `mpls` stanza define the new LSP, and indicate its desire for Fast Reroute protection. Note that CSPF is not disabled, and that an ERO is used to force the LSP's routing through `r4`. The use of ERO is not strictly necessary, but knowing how the path will be routed makes confirmation of Fast Reroute detours a bit easier.

```
[edit]
lab@r3# show protocols ospf
traffic-engineering;
area 0.0.0.1 {
    stub default-metric 10;
    interface fe-0/0/0.0;
    interface fe-0/0/1.0;
}
area 0.0.0.0 {
    interface so-0/2/0.100;
    interface at-0/1/0.0;
}
```

```
area 0.0.0.2 {
    nssa {
        default-lsa default-metric 10;
    }
    interface fe-0/0/3.0;
}
```

The addition of the `traffic-engineering` keyword in r3's OSPF instance provides support for Type 10 opaque LSAs, which are used by OSPF to build the TED. Although not shown, OSPF traffic engineering support has also been configured on r4, r5, r6, and r7. After committing the changes on all routers, you verify proper LSP establishment at r3:

```
[edit]
lab@r3# run show rsvp session ingress detail
Ingress RSVP: 1 sessions

10.0.9.7
  From: 10.0.3.3, LSPstate: Up, ActiveRoute: 0
  LSPname: r3-r7, LSPpath: Primary
  Suggested label received: -, Suggested label sent: -
  Recovery label received: -, Recovery label sent: 100027
  Resv style: 1 FF, Label in: -, Label out: 100027
  Time left:    -,  Since: Fri Feb 21 00:20:12 2003
  Tspec: rate 0bps size 0bps peak Infbps m 20 M 1500
  Port number: sender 1 receiver 58426 protocol 0
  FastReroute desired
  PATH rcvfrom: localclient
  PATH sentto: 10.0.2.6 (so-0/2/0.100) 11 pkts
  RESV rcvfrom: 10.0.2.6  (so-0/2/0.100) 13 pkts
  Explct route: 10.0.2.6 10.0.2.17
  Record route: <self>  10.0.2.6  10.0.2.17
    Detour is Up
    Detour PATH sentto: 10.0.2.1 (at-0/1/0.0) 10 pkts
    Detour RESV rcvfrom: 10.0.2.1  (at-0/1/0.0) 10 pkts
    Detour Explct route: 10.0.2.1 10.0.8.10
    Detour Record route: <self>  10.0.2.1  10.0.8.10
    Detour Label out: 100050
```

The display confirms that the *r3–r7* LSP has been successfully established, and also indicates that r3 has signaled a Fast Reroute detour to r5 in an effort to bypass r4 and its so-0/2/0.100 interface. Inspection of r5's RSVP sessions confirms the presence of two Fast Reroute detours that provide complete path protection (excepting ingress and egress nodes failures, of course):

```
[edit]
lab@r5# run show rsvp session transit detail
```

```
Transit RSVP: 1 sessions, 1 detours

10.0.9.7
  From: 10.0.3.3, LSPstate: Up, ActiveRoute: 1
  LSPname: r3-r7, LSPpath: Primary
  Suggested label received: -, Suggested label sent: -
  Recovery label received: -, Recovery label sent: 3
  Resv style: 1 FF, Label in: 100050, Label out: 3
  Time left:  134,  Since: Thu Feb 20 17:31:38 2003
  Tspec: rate 0bps size 0bps peak Infbps m 20 M 1500
  Port number: sender 1 receiver 58426 protocol 0
  Detour branch from 10.0.2.2, to skip 10.0.3.4, Up
    PATH rcvfrom: 10.0.2.2  (at-0/2/1.0) 13 pkts
    PATH sentto: 10.0.8.10 (fe-0/0/1.0) 13 pkts
    RESV rcvfrom: 10.0.8.10  (fe-0/0/1.0) 14 pkts
    Explct route: 10.0.8.10
    Record route: 10.0.2.2  <self>  10.0.8.10
    Label in: 100050, Label out: 3
  Detour branch from 10.0.2.10, to skip 10.0.9.7, Up
    PATH rcvfrom: 10.0.2.10  (so-0/1/0.0) 14 pkts
    PATH sentto: 10.0.8.10 (fe-0/0/1.0) 0 pkts
    RESV rcvfrom: 10.0.8.10  (fe-0/0/1.0) 0 pkts
    Explct route: 10.0.8.10
    Record route: 10.0.2.5  10.0.2.10  <self>  10.0.8.10
    Label in: 100051, Label out: 3
Total 1 displayed, Up 1, Down 0
```

The results just shown indicate that all requirements for this aspect of the case study have now been met, and this brings the following case study requirement to the top of your heap:

- Establish LSP *r4–r6* with a 2Mbps reservation. Ensure that a backup path is pre-established, that no transit elements are shared between the two paths, and that the LSP signals an SE style of reservation.

To achieve this goal, you must configure a secondary LSP path that is placed in the `standby` state. Your task is made complex by the specification that your LSP must use an SE style of RSVP reservation. The goal of this wording is to verify that the candidate understands RSVP, and the correlation between an SE style reservation and use of the JUNOS software `adaptive` keyword.

Although CSPF will, in theory, automatically compute a divergent secondary path, the use of EROs is highly recommended (whether CSPF is used or not) to guarantee that you meet the diverse routing requirements posed. The modifications made at `r4` to support the required behavior are shown next with highlights:

```
[edit protocols mpls]
lab@r4# show
```

```
label-switched-path r4-r6 {
    to 10.0.9.6;
    bandwidth 2m;
    adaptive;
    primary r4-r6;
    secondary r4-r6-prime {
        standby;
    }
}
path r4-r6 {
    10.0.3.5 loose;
}
path r4-r6-prime {
    10.0.3.3 loose;
}
interface all;
```

Note that the *r4–r6* LSP definition makes use of the `adaptive` and `standby` keywords, and that CSPF has not been disabled for either LSP path. You can use CSPF for these paths because of previous case study configuration steps that have resulted in a complete TED for Area 2 routers. After the changes are committed, the correct establishment of both the primary and standby LSP paths is confirmed:

```
[edit protocols mpls]
lab@r4# run show mpls lsp ingress detail
Ingress LSP: 1 sessions

10.0.9.6
  From: 10.0.3.4, State: Up, ActiveRoute: 1, LSPname: r4-r6
  ActivePath: r4-r6 (primary)
  LoadBalance: Random
  Encoding type: Packet, Switching type: Packet, GPID: IPv4
 *Primary   r4-r6             State: Up
    Bandwidth: 2Mbps
    Computed ERO (S [L] denotes strict [loose] hops): (CSPF metric: 2)
          10.0.2.9 S 10.0.8.5 S
    Received RRO:
          10.0.2.9 10.0.8.5
  Standby   r4-r6-prime       State: Up
    Bandwidth: 2Mbps
    Computed ERO (S [L] denotes strict [loose] hops): (CSPF metric: 2)
          10.0.2.5 S 10.0.2.13 S
```

```
    Received RRO:
         10.0.2.5 10.0.2.13
Total 1 displayed, Up 1, Down 0
```

Good, both LSP paths are established, and the highlights just shown indicate that the required diverse routing and bandwidth reservations have been correctly configured. To confirm the use of an SE style reservation, a **show rsvp session** command is issued at r4:

```
[edit protocols mpls]
lab@r4# run show rsvp session detail ingress
Ingress RSVP: 2 sessions

10.0.9.6
  From: 10.0.3.4, LSPstate: Up, ActiveRoute: 1
  LSPname: r4-r6, LSPpath: Primary
  Suggested label received: -, Suggested label sent: -
  Recovery label received: -, Recovery label sent: 100052
  Resv style: 1 SE, Label in: -, Label out: 100052
  Time left:    -,  Since: Fri Feb 21 00:40:26 2003
  Tspec: rate 2Mbps size 2Mbps peak Infbps m 20 M 1500
  Port number: sender 1 receiver 51984 protocol 0
  PATH rcvfrom: localclient
  PATH sentto: 10.0.2.9 (so-0/1/1.0) 15 pkts
  RESV rcvfrom: 10.0.2.9  (so-0/1/1.0) 12 pkts
  Explct route: 10.0.2.9 10.0.8.5
  Record route: <self>  10.0.2.9  10.0.8.5

10.0.9.6
  From: 10.0.3.4, LSPstate: Up, ActiveRoute: 0
  LSPname: r4-r6, LSPpath: Secondary
  Suggested label received: -, Suggested label sent: -
  Recovery label received: -, Recovery label sent: 100005
  Resv style: 1 SE, Label in: -, Label out: 100005
  Time left:    -,  Since: Fri Feb 21 00:40:55 2003
  Tspec: rate 2Mbps size 2Mbps peak Infbps m 20 M 1500
  Port number: sender 2 receiver 51984 protocol 0
  PATH rcvfrom: localclient
  PATH sentto: 10.0.2.5 (so-0/1/0.100) 12 pkts
  RESV rcvfrom: 10.0.2.5  (so-0/1/0.100) 13 pkts
  Explct route: 10.0.2.5 10.0.2.13
  Record route: <self>  10.0.2.5  10.0.2.13
Total 2 displayed, Up 2, Down 0
```

The presence of SE style reservations provides the final confirmation that you have met the requirements of this case study task. Your next configuration goal

addresses the following requirement:

- Establish LSPs *r7–r3* and *r7–r3–prime*. Ensure that prefixes with a length equal to or less than /20 are mapped to the *r7–r3* LSP, with all other prefixes mapped to the *r7–r3–prime* LSP. You must not modify the policy stanza at r7 to achieve this requirement, and the failure of a single interface cannot disrupt LSP forwarding.

The chapter body demonstrated a solution to this type of problem that made use of route filters and forwarding table export policy at the LSP ingress node. The restriction on policy usage at r7 means that you must find another way to achieve similar results. JNCIE candidates should be prepared to show that they are able to "shake the JUNOS software tree" in more than one way when the situation calls for it. The previous approach made use of "route receiver" policy to effect the desired ingress LSP mapping behavior; you must achieve similar results through the use of "route advertiser" policy.

The most common solution to this type of problem involves the use of two distinct addresses at the egress node, with each LSP being defined to terminate on one or the other of these addresses. Normally a secondary address is assigned to the router's lo0 interface to provide the extra address, while still offering resiliency to interface failures. The use of a secondary lo0 address is mandated in this case, by virtue of the requirement that your network must have continued LSP forwarding in the face of an interface failure. Note that you will have to find a way to advertise the secondary lo0 address, as OSPF will not do so with the current configuration at r3.

There are numerous approaches that can be taken to get the secondary lo0 address advertised to the routers that need it. For example, you could advertise the direct 10.0.3.33 address that you add to the lo0 interface through IBGP to the set of routers that must handle the routing of the *r7–r3–prime* LSP, or you could redistribute the route into OSPF. Keep in mind that OSPF externals are not sent into the NSSA 2 and stub Area 1, so the latter approach is problematic for r7; given the lack of external route advertisement for the 10.0.3.3 route in Area 2, r7's 10.0/16 local aggregate definition becomes a black hole for all traffic to 10.0.3.33.

Another possible solution involves listing r3's lo0 interface under its area 0 definition, which results in the advertisement of both the primary and secondary lo0 addresses. The potential downside to this method is that r3's lo0 addresses will now be advertised into other areas in the form of summary LSAs. However, the presence of summary LSA filtering at ABRs will create an obvious problem for any lo0-based BGP peerings sessions that will be disrupted as a result of this particular approach.

Be aware that a secondary lo0 address will not be advertised into the TED with the JUNOS software version used in this test bed. This means that you will have to disable CSPF for LSPs that terminate on a secondary lo0 address.

Because the current test bed does not make use of summary LSA filtering, and because listing r3's lo0 interface under the OSPF stanza requires minimum effort, you opt to go with the "lo0 as OSPF interface" approach to make sure that r7, and any intervening routers, have a viable route to r3's secondary lo0 address.

IBGP export policy will be used at the LSP egress node to selectively set the BGP next hop to the desired LSP egress address, based on whatever match criteria are necessary and available.

The solution demonstrated here once again makes use of route filters for the mapping policy's match criteria, due to the need to map based on prefix length. You can readily modify the policy demonstrated here when you must map on other criteria, such as the presence of a specific community. Table 2.2 shows the specifics of your LSP-to-address-to-prefix length plan:

TABLE 2.2 LSP-to-Prefix Mapping Plan

lo0 Address	LSP Name	Prefix to be Mapped
10.0.3.3	*r7-r3*	Prefixes from /0 to /20, inclusive
10.0.3.33	*r7-r3-prime*	All other prefixes

You start by adding the new LSP definitions to r7. When done, the changes are displayed next with highlights added:

```
[edit protocols mpls]
lab@r7# show
label-switched-path r7-r1 {
    to 10.0.6.1;
    ldp-tunneling;
    install 10.0.5.254/32;
    no-cspf;
}
label-switched-path r7-r3 {
    to 10.0.3.3;
}
label-switched-path r7-r3-prime {
    to 10.0.3.33;
    no-cspf;
}
interface all;
```

Note that CSPF has been disabled on the *r7-r3-prime* LSP. This is critical because r3's secondary lo0 address is not present in Area 2's TED, as shown here:

```
[edit protocols mpls]
lab@r7# run show ted database extensive | match 10.0.3.3
NodeID: 10.0.3.3
    To: 10.0.3.3, Local: 0.0.0.0, Remote: 0.0.0.0

[edit protocols mpls]
lab@r7# run show ted database extensive | match 10.0.3.33

[edit protocols mpls]
```

The lack of secondary lo0 address advertisement will condemn CSPF to failure in this case, which makes turning CSPF off a very, very, good idea. After assigning the secondary lo0 address and an appropriate IBGP export policy, you display the changes made to r3:

```
[edit]
lab@r3# show policy-options policy-statement lsp-map
term 1 {
    from {
        protocol bgp;
        neighbor 172.16.0.14;
        route-filter 0.0.0.0/0 upto /20;
    }
    then {
        next-hop self;
        accept;
    }
}
term 2 {
    from {
        protocol bgp;
        neighbor 172.16.0.14;
        route-filter 0.0.0.0/0 upto /32;
    }
    then {
        next-hop 10.0.3.33;
    }
}

[edit]
lab@r3# show interfaces lo0
unit 0 {
    family inet {
        address 10.0.3.3/32 {
            primary;
        }
        address 10.0.3.33/32;
    }
}
```

The *lsp-map* policy functions to set the BGP next hop based on prefix length through the use of route filter–based matching criteria. The `accept` action in the first term is critical, because without it, all routes will fall through to term 2 where their next hops will all be set to 10.0.3.33. When adding the secondary address to r3's lo0 interface, the `primary` keyword is appended to

its "real" lo0 address to ensure that other routers will not detect a change in r3's RID due to the addition of the secondary address. The primary keyword is not technically required in this example because the lowest IP address is considered the interface's primary address by default. Still, it never hurts to play it safe in a lab environment!

In this example, you apply the *lsp-map* policy at the BGP neighbor level, because you have no good reason to mess around with the other IBGP peerings:

```
[edit]
lab@r3# show protocols bgp group int
type internal;
local-address 10.0.3.3;
export nhs;
neighbor 10.0.6.1;
neighbor 10.0.6.2;
neighbor 10.0.3.4;
neighbor 10.0.3.5;
neighbor 10.0.9.6;
neighbor 10.0.9.7 {
    export lsp-map;
}
```

After committing the changes, the results are verified at r7:

```
[edit protocols mpls]
lab@r7# run show rsvp session ingress
Ingress RSVP: 3 sessions
To              From         State Rt   Style  Labelin   Labelout  LSPname
10.0.3.3        10.0.9.7     Up 28520   1 FF         -    100058    r7-r3
10.0.3.33       10.0.9.7     Up 97406   1 FF         -    100030    r7-r3-prime
10.0.6.1        10.0.9.7     Up     1   1 FF         -    100023    r7-r1
Total 3 displayed, Up 3, Down 0
```

The output confirms that both LSPs have been successfully established, and the prefix distribution across these LSPs provides an indication that all is working according to plan. A few more spot checks provide final confirmation:

```
[edit protocols mpls]
lab@r7# run show route receive-protocol bgp 10.0.3.3 | match /16
* 6.1.0.0/16               10.0.3.3                    100        65222 10458
   14203 3561 701 668 7170 1455 I
* 6.4.0.0/16               10.0.3.3                    100        65222 10458
   14203 3561 701 668 7170 1455 I
. . .

[edit protocols mpls]
lab@r7# run show route receive-protocol bgp 10.0.3.3 | match /24
```

```
* 12.1.83.0/24              10.0.3.33                100        65222 10458
  14203 3561 7911 5696 14787 I
* 12.1.96.0/24              10.0.3.33                100        65222 10458
  14203 3561 19024 14359 23306 I
```

The results of your prefix length–to–LSP mapping tests confirm that all requirements for this configuration scenario have been met. The last case study configuration task involves RSVP modification between r5 and r6, according to these criteria:

- Configure r5 and r6 to authenticate RSVP signaling with the key *jni*.
- Configure r5 and r6 so that RSVP state is preserved in the event of a routing restart at r5 or r6.
- Configure RSVP so that the loss of nine consecutive Hello messages is required to declare neighbor loss between r5 and r6.
- Configure r5 and r6 so that they *bundle* RSVP Path Tear and Error messages.

This configuration task is pretty straightforward. The most complex aspect is the need to modify the keep-multiplier, which is used to determine how many consecutive Hellos can be lost before a neighbor is declared down. Using the formula $2 \times keep\text{-}multiplier + 1$, as found in the related JUNOS software documentation, the default keep-multiplier of *3* will result in neighbor loss after seven consecutive Hello messages are lost. To meet the stipulations for this task, you need to set the keep-multiplier to *4* on both r5 and r6. Note that modifying the keep-multiplier will affect all RSVP sessions on r5 and r6. Although the wording implies that your concern is strictly the behavior of r5 and r6, this global modification of their RSVP behavior is not precluded by the case study requirements.

The requirement that r5 and r6 must *bundle* Path Tear and error messages is an indication that you must enable the JUNOS software aggregate feature between r5 and r6. Using the term *bundle*, as used in the Internet Engineering Task Force (IETF) draft on RSVP Refresh Reduction, instead of the JUNOS software keyword aggregate, is designed to test the candidate's understanding of generic RSVP operation and their familiarity with the specifics of the JUNOS software.

You display the changes made at r5 to evoke the desired behavior:

```
[edit]
lab@r5# show routing-options
graceful-restart;
static {
    route 10.0.200.0/24 {
        next-hop 10.0.1.102;
        no-readvertise;
    }
}
autonomous-system 65412;

[edit]
lab@r5# show protocols rsvp
```

```
keep-multiplier 4;
interface all;
interface fxp0.0 {
    disable;
}
interface fe-0/0/0.0 {
    authentication-key "$9$yZJeMXaJDikP"; # SECRET-DATA
    aggregate;
}
```

To enable RSVP graceful restart, you must configure `graceful-restart` under the main routing instance's `routing-options` stanza as shown. `r5`'s `rsvp` stanza correctly shows the modified `keep-multiplier` setting, and the use of authentication on its fe-0/0/0 interface. The support of refresh reduction is also indicated on this interface by virtue of the `aggregate` keyword. Though not shown, similar changes are made and committed on `r6`.

Various `show` commands are now issued to confirm proper operation of your RSVP modifications:

```
lab@r5# run show rsvp neighbor detail
RSVP neighbor: 4 learned
Address: 10.0.2.2   via: at-0/2/1.0    status: Up
  Last changed time: 37:33, Idle: 0 sec, Up cnt: 1, Down cnt: 0
  Message received: 185
  Hello: sent 251, received: 251, interval: 9 sec
  Remote instance: 0x2ae31daf, Local instance: 0x194112e8
  Refresh reduction:  not operational
  Link protection:  disabled
    Bypass LSP: does not exist,  Backup routes: 0,  Backup LSPs: 0

Address: 10.0.8.5   via: fe-0/0/0.0    status: Up
  Last changed time: 12:32, Idle: 5 sec, Up cnt: 2, Down cnt: 1
  Message received: 156
  Hello: sent 213, received: 195, interval: 9 sec
  Remote instance: 0x74b389ff, Local instance: 0x2ae99f9d
  Refresh reduction:  operational
    Remote end: enabled,  Ack-extension: enabled
  Link protection:  disabled
    Bypass LSP: does not exist,  Backup routes: 0,  Backup LSPs: 0
  Restart time: 60000 msec, Recovery time: 0 msec
. . .
```

The highlights call out the differences in RSVP behavior between `r5`'s 10.0.2.2 and 10.0.8.5 neighbors. Note that the neighbor information for `r6` (10.0.8.5) indicates that both refresh reduction and graceful restart are in effect.

```
[edit]
lab@r5# run show rsvp interface detail
RSVP interface: 4 active
fxp0.0  Index 1, State Dis/Up
  NoAuthentication, NoAggregate, NoReliable, NoLinkProtection
  HelloInterval 9(second)
  Address 10.0.1.5
  . . .

fe-0/0/0.0  Index 5, State Ena/Up
  Authentication, Aggregate, NoReliable, NoLinkProtection
  HelloInterval 9(second)
  Address 10.0.8.6
  ActiveResv 1, PreemptionCnt 0, Update threshold 10%
  Subscription 100%, StaticBW 100Mbps, AvailableBW 100Mbps
                          Total                          Last 5   seconds
  PacketType    Sent             Received             Sent          Received
  Path            94                   10                0                 0
  PathErr          2                    0                0                 0

. . .
```

The RSVP interface–related output highlights that RSVP authentication (and aggregation) are in effect on r5's fe-0/0/0 interface. These results confirm that you have met the requirements for modifying the RSVP behavior between r5 and r6 and in so doing you have completed all aspects of the MPLS and traffic engineering case study!

MPLS and Traffic Engineering Case Study Configurations

All changes made to the OSPF baseline network topology, as needed to complete the MPLS and traffic engineering case study, are listed below in Listings 2.1 to 2.7 for all routers in the test bed with highlights added.

Listing 2.1: MPLS Case Study Changes for *r1*

```
[edit]
lab@r1# show interfaces
fe-0/0/0 {
    unit 0 {
        family inet {
            address 10.0.5.1/24;
        }
        family mpls;
    }
```

```
}
fe-0/0/1 {
    unit 0 {
        family inet {
            address 10.0.4.14/30;
        }
        family mpls;
    }
}
fe-0/0/2 {
    unit 0 {
        family inet {
            address 10.0.4.5/30;
        }
        family mpls;
    }
}
fe-0/0/3 {
    unit 0 {
        family inet {
            address 10.0.4.18/30;
        }
        family mpls;
    }
}
fxp0 {
    unit 0 {
        family inet {
            address 10.0.1.1/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 10.0.6.1/32;
        }
    }
}

[edit]
lab@r1# show protocols
```

```
rsvp {
    interface all;
    interface fep0.0 {
        disable;
    }
}
mpls {
    label-switched-path r1-r7 {
        to 10.0.9.7;
        ldp-tunneling;
        no-cspf;
    }
    interface all;
}
bgp {
    group int {
        type internal;
        local-address 10.0.6.1;
        neighbor 10.0.6.2;
        neighbor 10.0.3.3;
        neighbor 10.0.3.4;
        neighbor 10.0.3.5;
        neighbor 10.0.9.6;
        neighbor 10.0.9.7;
    }
    group p1 {
        type external;
        export ebgp-out;
        neighbor 10.0.5.254 {
            peer-as 65050;
        }
    }
}
ospf {
    area 0.0.0.1 {
        stub;
        interface fe-0/0/0.0 {
            passive;
        }
        interface fe-0/0/1.0;
        interface fe-0/0/2.0;
```

```
        interface fe-0/0/3.0;
    }
}
ldp {
    interface lo0.0;
}
```

Listing 2.2: MPLS Case Study Changes for *r2*

```
[edit]
lab@r2# show interfaces
fe-0/0/0 {
    unit 0 {
        family inet {
            address 10.0.5.2/24;
        }
        family mpls;
    }
}
fe-0/0/1 {
    unit 0 {
        family inet {
            address 10.0.4.10/30;
        }
        family mpls;
    }
}
fe-0/0/2 {
    speed 100m;
    unit 0 {
        family inet {
            address 10.0.4.2/30;
        }
        family mpls;
    }
}
fe-0/0/3 {
    unit 0 {
        family inet {
            address 10.0.4.6/30;
        }
        family mpls;
    }
```

```
}
fxp0 {
    unit 0 {
        family inet {
            address 10.0.1.2/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 10.0.6.2/32;
        }
    }
}

[edit]
lab@r2# show protocols
rsvp {
    interface all;
    interface fxp0.0 {
        disable;
    }
}
mpls {
    interface all;
}
bgp {
    group int {
        type internal;
        local-address 10.0.6.2;
        neighbor 10.0.6.1;
        neighbor 10.0.3.3;
        neighbor 10.0.3.4;
        neighbor 10.0.3.5;
        neighbor 10.0.9.6;
        neighbor 10.0.9.7;
    }
    group p1 {
        type external;
        export ebgp-out;
```

```
        neighbor 10.0.5.254 {
            peer-as 65050;
        }
    }
}
ospf {
    area 0.0.0.1 {
        stub;
        interface fe-0/0/0.0 {
            passive;
        }
        interface fe-0/0/1.0;
        interface fe-0/0/2.0;
        interface fe-0/0/3.0;
    }
}
```

Listing 2.3: MPLS Case Study Changes for *r3*

```
[edit]
lab@r3# show interfaces
fe-0/0/0 {
    unit 0 {
        family inet {
            address 10.0.4.13/30;
        }
        family mpls;
    }
}
fe-0/0/1 {
    unit 0 {
        family inet {
            address 10.0.4.1/30;
        }
        family mpls;
    }
}
fe-0/0/2 {
    unit 0 {
        family inet {
            address 172.16.0.13/30;
        }
```

```
    }
}
fe-0/0/3 {
    unit 0 {
        family inet {
            address 10.0.2.14/30;
        }
        family mpls;
    }
}
at-0/1/0 {
    atm-options {
        vpi 0 {
            maximum-vcs 64;
        }
    }
    unit 0 {
        point-to-point;
        vci 50;
        family inet {
            address 10.0.2.2/30;
        }
        family mpls;
    }
}
so-0/2/0 {
    dce;
    encapsulation frame-relay;
    unit 100 {
        dlci 100;
        family inet {
            address 10.0.2.5/30;
        }
        family mpls;
    }
}
fxp0 {
    unit 0 {
        family inet {
            address 10.0.1.3/24;
        }
```

```
    }
}
lo0 {
    unit 0 {
        family inet {
            address 10.0.3.3/32 {
                primary;
            }
            address 10.0.3.33/32;
        }
    }
}

[edit]
lab@r3# show protocols
rsvp {
    interface all;
    interface fxp0.0 {
        disable;
    }
    interface so-0/2/0.100 {
        bandwidth 50m;
    }
}
mpls {
    label-switched-path r3-r7 {
        to 10.0.9.7;
        fast-reroute;
        primary r3-r7;
    }
    path r3-r7 {
        10.0.3.4 loose;
    }
    interface all;
}
bgp {
    advertise-inactive;
    group int {
        type internal;
        local-address 10.0.3.3;
```

```
        export nhs;
        neighbor 10.0.6.1;
        neighbor 10.0.6.2;
        neighbor 10.0.3.4;
        neighbor 10.0.3.5;
        neighbor 10.0.9.6;
        neighbor 10.0.9.7 {
            export lsp-map;
        }
    }
    group ext {
        import ebgp-in;
        export ebgp-out;
        neighbor 172.16.0.14 {
            peer-as 65222;
        }
    }
}
ospf {
    traffic-engineering;
    area 0.0.0.1 {
        stub default-metric 10;
        interface fe-0/0/0.0;
        interface fe-0/0/1.0;
    }
    area 0.0.0.0 {
        interface so-0/2/0.100;
        interface at-0/1/0.0;
        interface lo0.0;
    }
    area 0.0.0.2 {
        nssa {
            default-lsa default-metric 10;
        }
        interface fe-0/0/3.0;
    }
}

[edit]
lab@r3# show policy-options
policy-statement nhs {
```

```
    term 1 {
        from {
            protocol bgp;
            neighbor 172.16.0.14;
        }
        then {
            next-hop self;
        }
    }
}
policy-statement ebgp-out {
    term 1 {
        from {
            protocol aggregate;
            route-filter 10.0.0.0/16 exact;
        }
        then accept;
    }
}
policy-statement ebgp-in {
    term 1 {
        from {
            protocol bgp;
            neighbor 172.16.0.14;
        }
        then {
            community add transit;
        }
    }
}
policy-statement lsp-map {
    term 1 {
        from {
            protocol bgp;
            neighbor 172.16.0.14;
            route-filter 0.0.0.0/0 upto /20;
        }
        then {
            next-hop self;
            accept;
```

```
        }
    }
    term 2 {
        from {
            protocol bgp;
            neighbor 172.16.0.14;
            route-filter 0.0.0.0/0 upto /32;
        }
        then {
            next-hop 10.0.3.33;
        }
    }
}
community transit members 65412:420;
```

Listing 2.4: MPLS Case Study Changes for *r4*

```
[edit]
lab@r4# show interfaces
fe-0/0/0 {
    unit 0 {
        family inet {
            address 172.16.0.5/30;
        }
    }
}
fe-0/0/1 {
    unit 0 {
        family inet {
            address 10.0.4.9/30;
        }
        family mpls;
    }
}
fe-0/0/2 {
    unit 0 {
        family inet {
            address 10.0.4.17/30;
        }
        family mpls;
    }
}
```

```
fe-0/0/3 {
    unit 0 {
        family inet {
            address 10.0.2.18/30;
        }
        family mpls;
    }
}
so-0/1/0 {
    encapsulation frame-relay;
    unit 100 {
        dlci 100;
        family inet {
            address 10.0.2.6/30;
        }
        family mpls;
    }
}
so-0/1/1 {
    encapsulation ppp;
    unit 0 {
        family inet {
            address 10.0.2.10/30;
        }
        family mpls;
    }
}
fxp0 {
    unit 0 {
        family inet {
            address 10.0.1.4/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 10.0.3.4/32;
        }
    }
}
```

```
[edit]
lab@r4# show protocols
rsvp {
    interface all;
    interface fxp0.0 {
        disable;
    }
}
mpls {
    label-switched-path r4-r6 {
        to 10.0.9.6;
        bandwidth 2m;
        adaptive;
        primary r4-r6;
        secondary r4-r6-prime {
            standby;
        }
    }
    path r4-r6 {
        10.0.3.5 loose;
    }
    path r4-r6-prime {
        10.0.3.3 loose;
    }
    interface all;
}
bgp {
    advertise-inactive;
    group int {
        type internal;
        local-address 10.0.3.4;
        export nhs;
        neighbor 10.0.6.1;
        neighbor 10.0.6.2;
        neighbor 10.0.3.3;
        neighbor 10.0.3.5;
        neighbor 10.0.9.6;
        neighbor 10.0.9.7;
    }
    group c1 {
        type external;
```

```
        export ebgp-out;
        neighbor 172.16.0.6 {
            peer-as 65010;
        }
    }
}
ospf {
    traffic-engineering;
    area 0.0.0.1 {
        stub default-metric 10;
        interface fe-0/0/1.0;
        interface fe-0/0/2.0;
    }
    area 0.0.0.0 {
        interface so-0/1/0.100;
        interface so-0/1/1.0;
    }
    area 0.0.0.2 {
        nssa {
            default-lsa default-metric 10;
        }
        interface fe-0/0/3.0;
    }
}
```

Listing 2.5: MPLS Case Study Changes for *r5*

```
[edit]
lab@r5# show interfaces
fe-0/0/0 {
    unit 0 {
        family inet {
            address 10.0.8.6/30;
        }
        family mpls;
    }
}
fe-0/0/1 {
    unit 0 {
        family inet {
            address 10.0.8.9/30;
        }
        family mpls;
    }
```

```
}
so-0/1/0 {
    encapsulation ppp;
    unit 0 {
        family inet {
            address 10.0.2.9/30;
        }
        family mpls;
    }
}
at-0/2/1 {
    atm-options {
        vpi 0 {
            maximum-vcs 64;
        }
    }
    unit 0 {
        point-to-point;
        vci 50;
        family inet {
            address 10.0.2.1/30;
        }
        family mpls;
    }
}
fxp0 {
    unit 0 {
        family inet {
            address 10.0.1.5/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 10.0.3.5/32;
        }
    }
}

[edit]
lab@r5# show protocols
```

```
rsvp {
    keep-multiplier 4;
    interface all;
    interface fxp0.0 {
        disable;
    }
    interface fe-0/0/0.0 {
        authentication-key "$9$yZJeMXaJDikP"; # SECRET-DATA
        aggregate;
    }
}
mpls {
    interface all;
}
bgp {
    group int {
        type internal;
        local-address 10.0.3.5;
        neighbor 10.0.6.1;
        neighbor 10.0.6.2;
        neighbor 10.0.3.3;
        neighbor 10.0.3.4;
        neighbor 10.0.9.6;
        neighbor 10.0.9.7;
    }
}
ospf {
    traffic-engineering;
    area 0.0.0.0 {
        interface at-0/2/1.0;
        interface so-0/1/0.0;
    }
    area 0.0.0.2 {
        nssa {
            default-lsa default-metric 10;
        }
        interface fe-0/0/0.0;
        interface fe-0/0/1.0;
    }
}

[edit]
lab@r5# show routing-options
```

```
graceful-restart;
static {
    route 10.0.200.0/24 {
        next-hop 10.0.1.102;
        no-readvertise;
    }
}
autonomous-system 65412;
```

Listing 2.6: MPLS Case Study Changes for *r6*

```
[edit]
lab@r6# show interfaces
fe-0/1/0 {
    unit 0 {
        family inet {
            address 10.0.8.5/30;
        }
        family mpls;
    }
}
fe-0/1/1 {
    unit 0 {
        family inet {
            address 10.0.2.13/30;
        }
        family mpls;
    }
}
fe-0/1/2 {
    unit 0 {
        family inet {
            address 10.0.8.2/30;
        }
        family iso;
    }
}
fe-0/1/3 {
    unit 0 {
        family inet {
            address 172.16.0.9/30;
        }
    }
}
```

```
fxp0 {
    unit 0 {
        family inet {
            address 10.0.1.6/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 10.0.9.6/32;
        }
        family iso {
            address 49.0002.6666.6666.6666.00;
        }
    }
}

[edit]
lab@r6# show protocols
rsvp {
    keep-multiplier 4;
    interface all;
    interface fxp0.0 {
        disable;
    }
    interface fe-0/1/0.0 {
        authentication-key "$9$dHw2a5T369p"; # SECRET-DATA
        aggregate;
    }
}
mpls {
    label-switched-path r6-r1 {
        to 10.0.6.1;
        install 10.0.5.254/32;
        no-cspf;
    }
    interface all;
}
bgp {
    group int {
        type internal;
```

```
        local-address 10.0.9.6;
        export ibgp;
        neighbor 10.0.6.1;
        neighbor 10.0.6.2;
        neighbor 10.0.3.3;
        neighbor 10.0.3.4;
        neighbor 10.0.3.5;
        neighbor 10.0.9.7;
    }
    group c2 {
        type external;
        export ebgp-out;
        neighbor 172.16.0.10 {
            peer-as 65020;
        }
    }
}
isis {
    export ospf-isis;
    level 2 disable;
    level 1 external-preference 149;
    interface fe-0/1/2.0;
    interface lo0.0;
}
ospf {
    traffic-engineering;
    export isis-ospf;
    area 0.0.0.2 {
        nssa;
        interface fe-0/1/0.0;
        interface fe-0/1/2.0 {
            passive;
        }
        interface fe-0/1/1.0;
    }
}

[edit]
lab@r6# show routing-options
graceful-restart;
static {
    route 10.0.200.0/24 {
```

```
        next-hop 10.0.1.102;
        no-readvertise;
    }
}
aggregate {
    route 10.0.0.0/16;
}
autonomous-system 65412;
```

Listing 2.7: MPLS Case Study Changes for *r7*

```
[edit]
lab@r7# show interfaces
fe-0/3/0 {
    unit 0 {
        family inet {
            address 10.0.8.14/30;
        }
        family iso;
    }
}
fe-0/3/1 {
    unit 0 {
        family inet {
            address 10.0.8.10/30;
        }
        family mpls;
    }
}
fe-0/3/2 {
    unit 0 {
        family inet {
            address 172.16.0.1/30;
        }
    }
}
fe-0/3/3 {
    unit 0 {
        family inet {
            address 10.0.2.17/30;
        }
        family mpls;
    }
}
```

```
fxp0 {
    unit 0 {
        family inet {
            address 10.0.1.7/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 10.0.9.7/32;
        }
        family iso {
            address 49.0002.7777.7777.7777.00;
        }
    }
}

[edit]
lab@r7# show protocols
rsvp {
    interface all;
    interface fxp0.0 {
        disable;
    }
}
mpls {
    label-switched-path r7-r1 {
        to 10.0.6.1;
        ldp-tunneling;
        install 10.0.5.254/32;
        no-cspf;
    }
    label-switched-path r7-r3 {
        to 10.0.3.3;
    }
    label-switched-path r7-r3-prime {
        to 10.0.3.33;
        no-cspf;
    }
    interface all;
}
```

```
bgp {
    group int {
        type internal;
        local-address 10.0.9.7;
        export nhs;
        neighbor 10.0.6.1;
        neighbor 10.0.6.2;
        neighbor 10.0.3.3;
        neighbor 10.0.3.4;
        neighbor 10.0.3.5;
        neighbor 10.0.9.6;
    }
    group c1 {
        type external;
        export ebgp-out;
        neighbor 172.16.0.2 {
            peer-as 65010;
        }
    }
}
isis {
    export ospf-isis;
    level 2 disable;
    level 1 external-preference 149;
    interface fe-0/3/0.0;
    interface lo0.0;
}
ospf {
    traffic-engineering;
    export isis-ospf;
    area 0.0.0.2 {
        nssa;
        interface fe-0/3/1.0;
        interface fe-0/3/0.0 {
            passive;
        }
        interface fe-0/3/3.0;
    }
}
ldp {
    interface lo0.0;
}
```

Spot the Issues: Review Questions

1. You are having problems establishing the *r1–r6* LSP in the IS-IS topology shown earlier in Figure 2.1. Can you spot the problem in r1's configuration?

```
[edit protocols mpls]
lab@r1# show label-switched-path r1-r6
to 10.0.9.6;
bandwidth 1m;
priority 4 4;
primary r1-r6;

[edit protocols mpls]
lab@r1# show path r1-r6
10.0.3.4 loose;
```

2. Is any special configuration required to support MPLS pings?

3. You notice that r5 and r7 do not display each other as RSVP neighbors. Based on the `show RSVP session` command's output, does this indicate a problem?

```
[edit]
lab@r7# run show rsvp neighbor
RSVP neighbor: 0 learned

[edit]
lab@r7# run show rsvp session
Ingress RSVP: 0 sessions
Total 0 displayed, Up 0, Down 0

Egress RSVP: 0 sessions
Total 0 displayed, Up 0, Down 0

Transit RSVP: 0 sessions
Total 0 displayed, Up 0, Down 0
```

4. Your goal is to establish a primary and secondary LSP path that will not have their bandwidth reservations double-counted. Will this configuration meet your needs?

```
[edit protocols mpls]
lab@r4# show label-switched-path r4-r6
to 10.0.9.6;
bandwidth 2m;
primary r4-r6 {
    adaptive;
}
secondary r4-r6-prime {
    standby;
}

[edit protocols mpls]
lab@r4# show path r4-r6
10.0.3.5 loose;

[edit protocols mpls]
lab@r4# show path r4-r6-prime
10.0.3.3 loose;
```

5. Is the use of TE shortcuts possible between `r1` and `r5` in the topology shown earlier in Figure 2.6?

6. Can you spot the problem with this configuration snippet taken from `r3`, based on the topology shown earlier in Figure 2.7?

```
lab@r3# show
label-switched-path no-good {
    to 10.0.9.7;
    no-cspf;
    primary no-good;
}
path no-good {
    10.0.3.5;
    10.0.8.10;
}
interface all;
```

7. Will the configuration shown here support authentication between r5 and r7 in the topology shown earlier in Figure 2.2?

```
[edit protocols ldp]
lab@r5# show
traffic-statistics {
    file ldp-stats;
    interval 90;
}
keepalive-interval 5;
interface fe-0/0/0.0;
interface fe-0/0/1.0;
session 10.0.8.10 {
    authentication-key "$9$ORnI1EydVYgoG"; # SECRET-DATA
}

[edit protocols ldp]
lab@r7# show
traffic-statistics {
    file ldp-stats;
    interval 90;
}
keepalive-interval 5;
interface fe-0/3/2.0;
interface fe-0/3/3.0;
session 10.0.8.9 {
```

Spot the Issues: Answers to Review Questions

1. The *r1–r6* LSP cannot be established because CSPF has not been turned off, and the Multi-Level IS-IS topology has resulted in `r1`'s TED not containing the information it needs to locate a route to `r6`. You need to disable CSPF by adding the `no-cspf` keyword to the LSP's definition to allow IGP-based routing for the RSVP session (according to the ERO restrictions, of course).

2. Yes. You must ensure that the egress node has a 127.0.0.1 address assigned to its loopback interface, because this is the target address of an MPLS ping and you cannot override the default choice of destination address.

3. Not necessarily. Unlike link state routing protocols, RSVP does not perform automatic neighbor discovery. Because RSVP Hellos are unicast, a neighbor can only be discovered, and then maintained, *after* an RSVP signaling exchange has occurred. The lack of RSVP sessions on `r7` indicates that no RSVP LSPs have been signaled, so the lack of neighbor display may well indicate normal operation in the absence of RSVP signaling between `r5` and `r7`.

4. No. The problem in this case is that the `adaptive` keyword has been specified for the primary LSP path, but not for the secondary. This will result in a SE style reservation for the primary LSP path and a FF style reservation for the secondary. To correct the problem, you must add the `adaptive` keyword at the `label-switched-path` *`path-name`* hierarchy, as shown here:

```
[edit]
lab@r4# show protocols mpls label-switched-path r4-r6
to 10.0.9.6;
bandwidth 2m;
adaptive;
primary r4-r6;
secondary r4-r6-prime {
    standby;
```

   Note that adding the `adaptive` keyword under both the primary and secondary portions of the LSP, as shown next, will not work in accordance with your goals:

```
[edit protocols mpls]
lab@r4# show label-switched-path r4-r6
to 10.0.9.6;
bandwidth 2m;
primary r4-r6 {
    adaptive;
}
secondary r4-r6-prime {
    adaptive;
    standby;
}
```

The results of this configuration will be two *independent* SE style sessions that will have their bandwidth requests summed on shared links because the primary and secondary sessions will have unique session IDs.

5. No. When computing TE shortcuts, the ingress node attempts to match the LSP's egress address to an OSPF RID, with the result being the installation of prefixes that are reachable through that RID into the `inet.3` routing table. Because the OSPF RID is contained in router LSAs, and router LSAs are not flooded across area boundaries, `r1` and `r5` will not be able to locate each other's OSPF RID in the link state databases, and therefore no shortcuts will be installed.

6. The problem lies in the *no-good* path definition, in that `r5`'s loopback address has been specified without the `loose` argument. The default behavior results in such an ERO being considered a strict hop, and this means that `r3` cannot honor the first ERO, as shown in the following output:

```
lab@r3# run show rsvp session detail
Ingress RSVP: 1 sessions

10.0.9.7
  From: 10.0.3.3, LSPstate: Dn, ActiveRoute: 0
  LSPname: no-good, LSPpath: Primary
  Suggested label received: -, Suggested label sent: -
  Recovery label received: -, Recovery label sent: -
  Resv style: 0 -, Label in: -, Label out: -
  Time left:    -,  Since: Fri Feb 21 21:55:58 2003
  Tspec: rate 0bps size 0bps peak Infbps m 20 M 1500
  Port number: sender 1 receiver 12029 protocol 0
  PATH rcvfrom: localclient
  PATH sentto: [bad strict route]
  Record route: <self>  ...incomplete
Total 1 displayed, Up 0, Down 1
```

7. No. Although the LDP session will be established, authentication will not be in effect. This is because LDP sessions are established between loopback addresses by default. The omission of the `transport-address interface` statement results in an LDP session between the lo0 addresses of `r5` and `r7`, and this session does not match the session that has been configured to use authentication. Another solution is to redefine the session parameters to use lo0 addresses, assuming that this is permitted by your scenario.

Chapter

3

Firewall Filter and Traffic Sampling

JNCIE LAB SKILLS COVERED IN THIS CHAPTER:

- ✓ **Firewall filters**
  - RE protection
  - Transit filtering
  - Policing
- ✓ **Filter-based forwarding**
- ✓ **Traffic sampling**
  - Cflowd export
  - Port mirroring

This chapter exposes the reader to a variety of JNCIE-level firewall filtering and traffic sampling configuration scenarios. It is assumed that the reader already has a working knowledge of TCP/IP protocol exchanges, port usage, and general firewall concepts to the extent covered in the *JNCIS Study Guide* (Sybex, 2003).

Juniper Networks routing platforms equipped with an Internet Process II ASIC (IP II) are capable of performing a variety of packet filtering, traffic sampling, and accounting functions. These capabilities can be used to secure the local routing engine (RE) and attached devices, and to perform statistical analysis of traffic patterns through a service provider's network. Analyzing traffic patterns can assist in capacity planning and in the detection and tracking of Distributed Denial of Service (DDoS) attacks. Use the `show chassis hardware` command to determine if your router is IP II equipped, as shown next:

```
lab@router> show chassis hardware
Hardware inventory:
Item             Version  Part number  Serial number     Description
Chassis                                20207             M20
Backplane        REV 07   710-001517   AB5912
Power Supply B   Rev 02   740-001465   000255            AC
Display          REV 04   710-001519   AD1899
Routing Engine 0                       9e00000749ab1601  RE-2.0
SSB slot 0       REV 01   710-001951   AP8612            Internet Processor II
SSB slot 1       N/A      N/A          N/A               backup
FPC 0            REV 01   710-001292   AC7598
  PIC 0          REV 04   750-002992   HD7819            4x F/E, 100 BASE-TX
  PIC 2          REV 03   750-000612   AD1438            2x OC-3 ATM, MM
FPC 2            REV 01   710-001292   AD4111
  PIC 0          REV 03   750-000611   AC1506            4x OC-3 SONET, MM
  PIC 2          REV 08   750-001072   AB2552            1x G/E, 1000 BASE-SX
```

Note that as of this writing, Juniper Networks routing platforms do not provide stateful packet inspection or Network Address/Port Address Translation (NAT/PAT) services. However, the hardware-based implementation of firewall filtering in M-series and T-series routing platforms means that service providers can deploy the functionality described in this chapter with little chance of impacting the operation of a production network. This chapter demonstrates firewall filtering, policing, and sampling options that are supported in the 5.6 release of JUNOS software. As this chapter progresses, the reader will be exposed to a variety of operational mode

commands and techniques that will prove useful when you must verify or troubleshoot the operation of JUNOS software firewall filters.

Readers that are familiar with JUNOS software routing policy will immediately find comfort in the syntax used for firewall filters. A key difference between routing policies and firewall filters lies in the fact that only one filter can be applied to a particular logical interface, in a particular direction, at any given time. When this behavior is contrasted to routing policy application, which permits the chaining of multiple policies, one can readily see why multiterm filters are the norm.

The chapter's case study is designed to closely approximate a typical JNCIE firewall and traffic analysis configuration scenario. The results of key operational mode commands are provided in the case study analysis section so you can compare the behavior of your network to a known good example. Example router configurations that meet all case study requirements are provided at the end of the case study for comparison with your own configurations.

The examples demonstrated in the chapter body are based on the MPLS topology left in place at the end of the Chapter 2 case study. If you are unsure as to the state of your test bed, you should take a few moments to load up and confirm the OSPF-based, MPLS chapter case study configuration before proceeding. Figure 3.1 depicts your test bed at the end of the Chapter 2 MPLS case study.

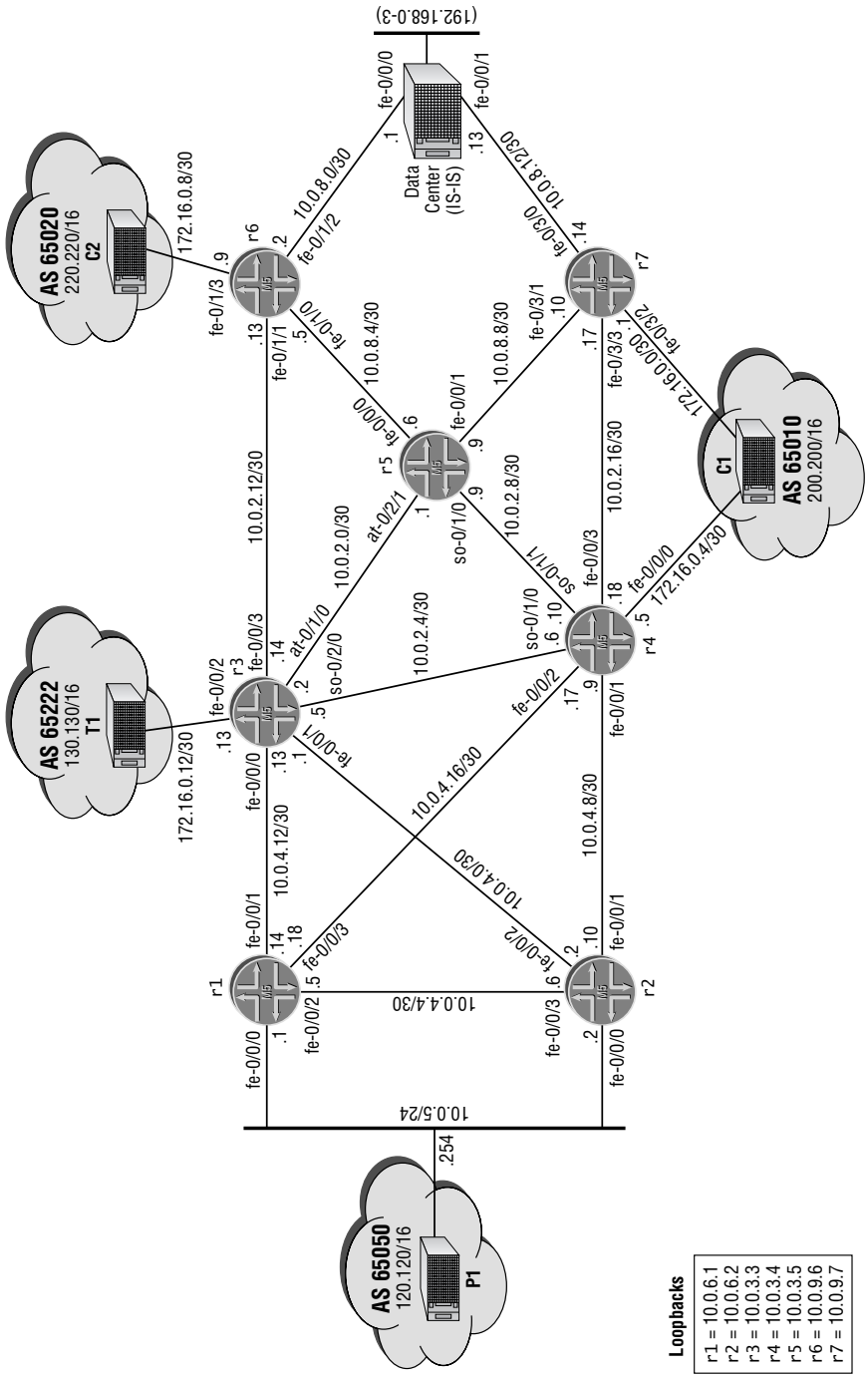
Firewall Filters

The basic function of a JUNOS software firewall filter is to enhance security by filtering packets matching one or more operator-specified criteria. A JUNOS software firewall requires two distinct configuration steps: namely, the definition of the firewall filter itself followed by the application of the firewall to one or more router interfaces. Note that a given firewall filter can be applied to one or more interfaces, and that a particular filter can be applied as an *input* filter, *output* filter, or both simultaneously. Input filters take action on packets being received on the given interface, while output filters act on packets that are transmitted out a given interface.

Generally speaking, you apply a firewall filter for one of three main reasons: to protect the local router itself, to protect another device that is either directly or indirectly attached to the local router, or to perform multi-field classifications for purposes of specialized packets handling, in other words, Class of Service (CoS) or Filter-Based Forwarding (FBF). In the former case, you should apply your firewall filter to the router's lo0 interface, because this ensures that only packets being sent to, or from, the RE (depending on the direction in which the filter is applied) are subjected to the firewall rules. A lo0 firewall filter does not impact transit traffic directly, but such a filter might impact transit traffic indirectly. For example, failing to accommodate your IGP in a firewall filter's rules might result in lost adjacencies and missing routes, which in turn can affect the local router's ability to forward transit traffic. Note that a copy of a lo0 firewall filter is maintained in the PFE where it will act on traffic before it traverses the internal fxp1 link to reach the RE.

Finally, firewall filters are often used to classify and mark packets for QoS-related functions. The use of firewall filters for CoS applications is detailed in Chapter 6.

FIGURE 3.1 MPLS case study topology



JUNOS software firewall filters are based on a "prudent" security philosophy, in that they operate on a "deny all that is not explicitly allowed" model. Put another way, once a filter is applied to an interface, all traffic not explicitly accepted by that filter is silently discarded by default. You can alter the default behavior by including an `accept all` term at the end of your filter. In this case, the previous filter terms will normally be written to reject traffic that is known to be bad. Please note that the resulting "permissive" filter represents significant security concerns in that unanticipated traffic, such as the latest Internet worm, will be allowed by the `accept all` term in your filter. Note that failing to consider the default `deny all` action of a filter can have significant ramifications on the operation of your network. For example, consider the case of a *lo0* filter that fails to adequately accommodate your IGP and BGP protocols; the application of such a filter will have catastrophic effects on your network due to the loss of the router's IGP adjacencies and its BGP sessions. It is always a good idea to make use of the `confirmed` option when committing firewall-related changes, and to include some type of logging action for traffic that is rejected or denied by the filter. The logging information allows a rapid determination of whether valid traffic is being rejected by the new filter, and with this knowledge you can quickly make the necessary adjustments to the firewall filter before the phone starts ringing.

RE Protection

You begin your firewall filter scenario by modifying `r3`'s configuration to meet the following requirements:

- Your filter can not affect transit traffic or the overall packet routing behavior of your network.
- Ensure that `r3` accepts *only* the following traffic:
  - ICMP echo–related exchanges.
  - Incoming telnet sessions only from nodes within your AS. Outgoing telnet sessions to all peers (internal and external).
  - SSH sessions that are initiated by `r3`.
  - FTP sessions that are initiated by `r3`.

Confirming Initial Operation

Before changing anything, you decide to issue some key commands at `r3` to ascertain the overall status of `r3`'s OSPF, BGP, and MPLS-related protocols:

```
[edit]
lab@r3# run show ospf neighbor
  Address          Interface              State     ID               Pri   Dead
10.0.2.1         at-0/1/0.0             Full      10.0.3.5         128   39
10.0.2.6         so-0/2/0.100           Full      10.0.3.4         128   39
10.0.4.14        fe-0/0/0.0             Full      10.0.6.1         128   38
10.0.4.2         fe-0/0/1.0             Full      10.0.6.2         128   33
10.0.2.13        fe-0/0/3.0             Full      10.0.9.6         128   38
```

All of the expected OSPF adjacencies are established at r3, so you move on to verify its BGP session status:

```
[edit]
lab@r3# run show bgp summary
Groups: 3 Peers: 7 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
inet.0            126658     126646          0          0          0          0
Peer            AS  InPkt  OutPkt    OutQ  Flaps  Last Up/Dwn State|#Active/
                                                              Received/Damped...
172.16.0.14   65222 27215   27272     0      0     2:17:55 126643/126643/0   0/0/0
10.0.3.4      65412   289   27432     0      0     2:23:57 1/1/0             0/0/0
10.0.3.5      65412   283   27427     0      0     2:21:14 0/0/0             0/0/0
10.0.6.1      65412   289   27434     0      0     2:24:05 1/1/0             0/0/0
10.0.6.2      65412   277   27421     0      0     2:17:42 0/1/0             0/0/0
10.0.9.6      65412   291   27433     0      0     2:23:52 1/6/0             0/0/0
10.0.9.7      65412   290   31468     0      0     2:23:45 0/6/0             0/0/0
```

Good, all of r3's IBGP and EBGP sessions are established. The next command confirms the state of r3's RSVP signaled LSPs:

```
[edit]
lab@r3# run show rsvp session
Ingress RSVP: 1 sessions
To              From            State Rt Style Labelin Labelout LSPname
10.0.9.7        10.0.3.3        Up     0  1 FF       -   100007 r3-r7
Total 1 displayed, Up 1, Down 0

Egress RSVP: 2 sessions
To              From            State Rt Style Labelin Labelout LSPname
10.0.3.3        10.0.9.7        Up     0  1 FF       3        - r7-r3
10.0.3.33       10.0.9.7        Up     0  1 FF       3        - r7-r3-prime
Total 2 displayed, Up 2, Down 0

Transit RSVP: 2 sessions
To              From            State Rt Style Labelin Labelout LSPname
10.0.6.1        10.0.9.6        Up     0  1 FF  100003        3 r6-r1
10.0.9.6        10.0.3.4        Up     0  1 SE  100004        3 r4
```

The display confirms that r3's ingress, egress, and transit RSVP sessions have been successfully established. The operational displays indicate that r3 and the OSPF-based MPLS case study topology from Chapter 2, are largely operational. With the knowledge that your test bed is operational, you begin your first firewall filter configuration task.

Creating a Firewall Filter

You begin your configuration by defining the *r3–lo0* filter at r3. This filter will be applied as an input filter to r3's lo0 interface to meet the requirement that your filter not affect transit traffic. Note that without an output filter on its lo0 interface, r3 will be able to send any and all traffic that it desires. Writing the input filter to deny response traffic that is not associated with authorized applications ensures that you can meet all requirements posed for this configuration example with a unidirectional *lo0* filter at r3. Also note that you need to deploy a prudent firewall filter to meet the requirements that your filter accept *only* the applications and services listed.

The following commands create the new filter on r3 and define the first filter term, which is designed to accommodate ICMP ping (Echo)–related traffic in this example:

```
[edit]
lab@r3# edit firewall filter r3-lo0

[edit firewall filter r3-lo0]
lab@r3# set term icmp from protocol icmp

[edit firewall filter r3-lo0]
lab@r3# set term icmp from icmp-type echo-reply

[edit firewall filter r3-lo0]
lab@r3# set term icmp from icmp-type echo-request

[edit firewall filter r3-lo0]
lab@r3# set term icmp then accept
```

The resulting configuration is displayed:

```
[edit firewall filter r3-lo0]
lab@r3# show
term icmp {
    from {
        protocol icmp;
        icmp-type [ echo-reply echo-request ];
    }
    then accept;
}
```

Note that the ICMP protocol is specified along with an `icmp-type` match criterion. This is significant because JUNOS software firewall filters do not automatically perform a test for the protocol associated with a given service or application. Failing to correctly test for the ICMP protocol can result in non-ICMP traffic being accepted by the filter's *icmp* term. Also worth pointing out here is the logical OR formed by the specification of both `echo-reply` (ICMP type 0) and `echo-request` (ICMP type 8) message types. This specification allows the use of a single term for

the purpose of accepting echo traffic that is initiated by the local router (incoming ICMP echo replies) or ICMP echo traffic that is initiated by a remote router (incoming ICMP echo request).

With ICMP traffic accommodated, you move on to the requirement that your RE filter limit incoming telnet sessions to nodes within your AS, while allowing r3 to initiate telnet sessions to both external and internal peers:

```
[edit firewall filter r3-lo0]
lab@r3# set term telnet-in from protocol tcp

[edit firewall filter r3-lo0]
lab@r3# set term telnet-in from destination-port 23

[edit firewall filter r3-lo0]
lab@r3# set term telnet-in from address 10.0/16

[edit firewall filter r3-lo0]
lab@r3# set term telnet-in then accept
```

With incoming telnet sessions correctly restricted to source addresses in the 10.0/16 net block, you now define a term that permits outgoing telnet sessions to all nodes:

```
[edit firewall filter r3-lo0]
lab@r3# set term telnet-out from protocol tcp

[edit firewall filter r3-lo0]
lab@r3# set term telnet-out from source-port 23

[edit firewall filter r3-lo0]
lab@r3# set term telnet-out then accept
```

The telnet-related terms are now displayed:

```
[edit firewall filter r3-lo0]
lab@r3# show term telnet-in
from {
    address {
        10.0.0.0/16;
    }
    protocol tcp;
    destination-port 23;
}
then accept;

[edit firewall filter r3-lo0]
lab@r3# show term telnet-out
```

```
from {
    protocol tcp;
    source-port 23;
}
then accept;
```

Note once again that the TCP protocol has been specified along with the telnet port to ensure that non-TCP based traffic is not inadvertently accepted by the filter's telnet-related terms. In this case, the telnet-related terms make use of the `destination-port` and `source-port` keywords to restrict incoming sessions (destination port equals 23) while allowing `r3` to initiate sessions to all hosts (source port will equal *23* in the returning traffic—remember, with an input filter you are filtering only return traffic). The inclusion of an `address`-based match condition in the *`telnet-in`* term is required to meet the condition that incoming telnet sessions are restricted to those nodes that are within your AS.

With telnet support in place, you next define a term that enforces `r3`'s ability to initiate, but not accept, SSH connections:

```
[edit firewall filter r3-lo0]
lab@r3# set term ssh from protocol tcp

lab@r3# set term ssh from source-ports?
Possible completions:
  <range>              Range of values
  smtp                 SMTP
  snmp                 SNMP
  snmptrap             SNMP traps
  snpp                 Simple paging protocol
  socks                Socks
  ssh                  Secure shell (ssh)
  sunrpc               SUN RPC
  syslog               System log (syslog)
[edit firewall filter r3-lo0]
lab@r3# set term ssh from source-port ssh

[edit firewall filter r3-lo0]
lab@r3# set term ssh then accept
```

In this example, the operator has chosen to use a symbolic keyword to identify the port associated with the SSH service (SSH uses port 22). The CLI's Help function is also used to display a list of symbolically named ports, which in this case begin with the letter *s*. The new firewall term is displayed:

```
[edit firewall filter r3-lo0]
lab@r3# show term ssh
```

```
from {
    protocol tcp;
    source-port ssh;
  }
  then accept;
```

The use of a `source-port` based match criteria in this example correctly prevents SSH sessions that are initiated by remote routers, because remotely initiated SSH sessions will have a destination port of 22 and a source port that is in the range of 1024–65,536 inclusive. The next filtering task is to permit outgoing FTP sessions while blocking incoming FTP sessions. In this example, you decide to modify the *`ssh`* term to make it pull double duty by supporting both the SSH and FTP services. You begin by renaming the term to reflect the new SSH and FTP support role:

```
[edit firewall filter r3-lo0]
lab@r3# rename term ssh to term ssh-ftp

[edit firewall filter r3-lo0]
lab@r3# set term ssh-ftp from source-port 20-21
```

The renamed term is displayed with the FTP-related modifications highlighted:

```
[edit firewall filter r3-lo0]
lab@r3# show term 3
from {
    protocol tcp;
    source-port [ ssh 20-21 ];
}
then accept;
```

By adding the FTP data and control ports (20 and 21, respectively) to the *`ssh-ftp`* term, you have created a single term that functions to accept SSH and FTP traffic for those SSH and FTP sessions that are initiated by `r3` only. Of course, the use of an FTP-specific term is also possible in this case.

Although not required by the specifics of this configuration scenario, an explicit *`deny-all-else`* term that logs and counts rejected traffic is not precluded by your rules of engagement either. This author believes that adding such a term is a good idea, because it helps to reinforce the fact that, by default, all traffic not explicitly accepted by your filter will be implicitly denied, and because the logging of rejected traffic can make firewall filter troubleshooting significantly easier. Once a firewall filter is known to operate properly, you can opt to delete (or deactivate) the *`deny-all-else`* term to reduce clutter and minimize processing burden. The next set of commands creates an explicit *`deny-all-else`* term in your *`r3-lo0`* filter:

```
[edit firewall filter r3-lo0]
lab@r3# set term deny-all-else then log

[edit firewall filter r3-lo0]
lab@r3# set term deny-all-else then count r3-lo0-rejection
```

```
[edit firewall filter r3-lo0]
lab@r3# set term deny-all-else then discard
```

The final term is displayed:

```
lab@r3# show term deny-all-else
then {
    count r3-lo0-rejection;
    log;
    discard;
}
```

The presence of the `log` and `count` action modifiers in the *`deny-all-else`* term makes the confirmation and troubleshooting of firewall filter issues far simpler. The `log` action modifier results in entries being written to a temporary kernel cache for traffic that is discarded by the final term. The cache holds about 400 entries before it wraps around to begin overwriting older entries. The `count` action modifier causes the named counter to increment for each packet (and byte) that is rejected by the term.

Note that an explicit `discard` action has also been configured. The specification of a `discard` action is needed in this case because the use of an action modifier, such as `count` and `log`, results in an implicit `accept` action for any traffic that matches that term. Because the *`deny-all-else`* term has no `from` condition specified, all traffic that has not already been accepted by the previous terms in the *`r3-lo0`* firewall filter will match the term. This behavior results in `r3` accepting *all* traffic that it is given! Because such indiscriminate behavior at `r3` falls outside of the configuration example's criteria, the presence of the explicit `discard` action is a very good thing in this case.

Note that the use of `reject` is also possible in this scenario; generally speaking, sending back an indication that packets are being filtered, in the form of ICMP administratively prohibited messages, is considered to be bad security form. The use of the `discard` action, on the other hand, results in silent packet discard. The completed *`r3-lo0`* filter is now displayed:

```
[edit firewall filter r3-lo0]
lab@r3# show
term icmp {
    from {
        protocol icmp;
        icmp-type [ echo-reply echo-request ];
    }
    then accept;
}
term telnet-in {
    from {
        address {
            10.0.0.0/16;
        }
        protocol tcp;
```

```
        destination-port 23;
    }
    then accept;
}
term telnet-out {
    from {
        protocol tcp;
        source-port 23;
    }
    then accept;
}
term ssh-ftp {
    from {
        protocol tcp;
        source-port [ ssh 20-21 ];
    }
    then accept;
}
term deny-all-else {
    then {
         count r3-lo0-rejection;
         log;
         discard;
}
```

Applying and Verifying a RE Firewall Filter

With the firewall filter's definition complete, you now apply the *r3-lo0* filter to r3's lo0 interface in the input direction:

```
[edit interfaces]
lab@r3# set lo0 unit 0 family inet filter input r3-lo0
```

The modified lo0 configuration is displayed:

```
[edit interfaces]
lab@r3# show lo0
unit 0 {
    family inet {
        filter {
            input r3-lo0;
        }
        address 10.0.3.3/32 {
            primary;
        }
```

```
        address 10.0.3.33/32;
    }
}
```

The highlighted portion of the output calls out the application of the *r3-lo0* filter as an input filter. This is a significant point, because true RE protection can only be provided by input filtering. Being aware that packet filtering mistakes can leave you locked out of your router, at least for all access methods except direct console access, you wisely decide to use the `confirmed` argument when you commit your changes to `r3`. You must now issue another `commit` within *5* minutes, or `r3` will roll back and automatically commit the last configuration that was successfully committed.

```
[edit]
lab@r3# commit confirmed 5
commit confirmed will be automatically rolled back in 5 minutes unless confirmed
commit complete

[edit]
lab@r3#
```

With the RE protection firewall changes committed, at least for the next *5* minutes, you quickly confirm that telnet access is working at `r3`. Note that receiving the `commit complete` message shown earlier already indicates that the *lo0* filter has not affected telnet over the OoB network.

```
[edit]
lab@r3# run telnet 10.0.3.3
Trying 10.0.3.3...
Connected to 10.0.3.3.
Escape character is '^]'.

r3 (ttyp1)

login: lab
Password:
Last login: Tue Mar  4 16:15:25 from 10.0.1.100

--- JUNOS 5.6R1.3 built 2003-01-02 20:38:33 UTC

lab@r3>
```

The telnet connection from `r3` back to itself succeeds, which shows that `r3` can both initiate and accept telnet sessions. Further, your existing telnet session is unaffected by the application of the *r3-lo0* firewall filter, which causes you to decide that it is safe to officially commit the `r3`'s firewall configuration:

```
lab@r3> exit

Connection closed by foreign host.
```

```
[edit]
lab@r3# commit
commit complete

[edit]
lab@r3#
```

Before validating the services that are listed as explicitly supported by r3, you decide to view the firewall log to see if any unexpected traffic is being caught (discarded) by the firewall filter:

```
[edit]
lab@r3# run show firewall log
Log :
Time      Filter    Action   Interface Protocol       Src Addr       Dest Addr
21:00:53  pfe       D           fe-0/0/0.0 RSVP      10.0.4.14       10.0.4.13
21:00:52  pfe       D          so-0/2/0.100RSVP       10.0.2.6        10.0.2.5
21:00:52  pfe       D           fe-0/0/0.0 OSPF      10.0.4.14       224.0.0.5
. . .
21:00:36  pfe       D            fe-0/0/2.0 TCP    172.16.0.14     172.16.0.13
. . .
```

The truncated contents of the firewall cache bring the words "uh oh" near the top of the list of expletives that are likely to be uttered at a time like this. The highlights call out that OSPF, RSVP, and BGP-related traffic is being discarded (D) by a copy of the *r3-lo0* filter that is maintained in the router's packet forwarding engine (pfe). The rejection of BGP traffic is indicated by the entry indicating TCP and the 172.16.0.14 address, which is associated with the T1 router.

Note that any traffic originated by r3's RE will display the actual firewall filter name in the Filter column because, unlike the PFE, the RE can store the filter's name. Based on the log entries, it is readily apparent that the *r3-lo0* filter fails to accommodate the various routing and signaling protocols in use at r3! The presence of broken OSPF adjacencies, active BGP sessions, and failed RSVP signaled LSPs indicates that the current filter does not meet the stipulation that your firewall not affect the overall packet forwarding behavior of your network. A few commands quickly display the extent of the network disruption caused by your new firewall filter:

```
[edit]
lab@r3# run show ospf neighbor

[edit]
lab@r3# run show bgp summary
Groups: 3 Peers: 7 Down peers: 7
Table          Tot Paths  Act Paths Suppressed   History  Damp State    Pending
inet.0                 0          0          0         0           0          0
```

```
Peer            AS  InPkt  OutPkt  OutQ  Flaps  Last  Up/Dwn State|#Active/
                                                       Received/Damped...
172.16.0.14  65222  30317   30379     0      1       14:11 Connect
10.0.3.4     65412    587   30631     0      1       14:23 Active
10.0.3.5     65412    581   30652     0      1       14:10 Active
10.0.6.1     65412    587   30633     0      1       14:18 Active
10.0.6.2     65412    575   30645     0      1       14:10 Active
10.0.9.6     65412    589   30632     0      1       14:21 Active
10.0.9.7     65412    588   34751     0      1       14:11 Active

[edit]
lab@r3# run show rsvp session
Ingress RSVP: 0 sessions
Total 0 displayed, Up 0, Down 0

Egress RSVP: 0 sessions
Total 0 displayed, Up 0, Down 0

Transit RSVP: 0 sessions
Total 0 displayed, Up 0, Down 0
```

The various outputs confirm that `r3` has no IGP adjacencies, no established BGP sessions, and no RSVP signaled LSPs. It bears stressing that with the results-based grading approach currently used in the JNCIE examination, a single mistake such as the one shown here could easily spell the difference between a fancy certificate and a long trip home with your tail firmly tucked.

Always be on guard for the inadvertent disruption of valid services when you are deploying a prudent firewall filter. Candidates often miss the fact that some previously confirmed service or application has been broken by the application of a poorly crafted firewall filter. These candidates are often surprised to see points taken off later in the exam for services that they "know were working." The key term here is the word *were*, and the fact that JNCIE examinations are graded on the overall operation of your network. A few mistakes made with a firewall filter can spell doom for your final JNCIE grade! Because it is easy to overlook existing services or applications, it is suggested that you add some type of logging action to all new filters, because examination of the resulting log file will assist you in determining if your filter is taking unanticipated actions. For example, the *r3-lo0* filter shown in this section fails to accommodate the RADIUS protocol and locally initiated traceroute testing. These omissions do not result in "point loss" in this example, because the specifics of the RE filtering task stipulated only that your filter can not affect *packet forwarding behavior*. Because the router's configuration allows authentication against the local password database when RADIUS is unreachable, you can still log in to `r3` (albeit, with a bit of delay), and packet forwarding is unaffected by the inability to use RADIUS or to initiate traceroute testing from `r3`. Note that traceroutes through `r3` still function because expired TTL handling and ICMP error generation occur in the PFE, not the RE.

In a production network, you would be well advised to first deactivate the filter's application to return the network to normal operation as you make changes to the *r3-lo0* filter. In the case of a lab exam, extra steps designed to minimize operational impact are generally not necessary. You therefore move straight to the modification of the *r3-lo0* filter, starting with the addition of a term designed to accept BGP sessions regardless of which router initiates the communications:

```
[edit firewall filter r3-lo0]
lab@r3# set term bgp from protocol tcp

[edit firewall filter r3-lo0]
lab@r3# set term bgp from port bgp

[edit firewall filter r3-lo0]
lab@r3# set term bgp then accept
```

The next set of commands creates a single term that accepts *either* OSPF or RSVP. Note that the key to this approach lies in the knowledge of which applications use ports (BGP) and which applications ride directly within IP (OSPF and RSVP):

```
[edit firewall filter r3-lo0]
lab@r3# set term ospf-rsvp from protocol ospf

[edit firewall filter r3-lo0]
lab@r3# set term ospf-rsvp from protocol rsvp

[edit firewall filter r3-lo0]
lab@r3# set term ospf-rsvp then accept
```

Before committing the modifications, you once again display the complete *r3-lo0* filter:

```
[edit firewall filter r3-lo0]
lab@r3# show
term icmp {
    from {
        protocol icmp;
        icmp-type [ echo-reply echo-request ];
    }
    then accept;
}
term telnet-in {
    from {
        address {
            10.0.0.0/16;
        }
        protocol tcp;
        destination-port 23;
```

```
    }
    then accept;
}
term telnet-out {
    from {
        protocol tcp;
        source-port 23;
    }
    then accept;
}
term ssh-ftp {
    from {
        protocol tcp;
        source-port [ ssh 20-21 ];
    }
    then accept;
}
term deny-all-else {
    then {
        count r3-lo0-rejection;
        log;
        discard;
    }
}
term bgp {
    from {
        protocol tcp;
        port bgp;
    }
    then accept;
}
term ospf-rsvp {
    from {
        protocol [ ospf rsvp ];
    }
    then accept;
}
```

Hmm, something does not seem right; can you spot the problem now? Give yourself a pat on the back if you correctly identified that the last two terms, which are designed to allow BGP, RSVP, and OSPF traffic, are currently having no effect on the filter's operation! This is because the current sequencing of the terms has all traffic that is not accepted by the first four terms being summarily discarded when it matches the *deny-all-else* term.

As with routing policy, the ordering of firewall terms often has a significant impact on the overall operation of your firewall configuration. Use the `insert` command to rearrange terms when sequencing problems are suspected.

To rectify the situation, you could delete the problem terms and rewrite them in the desired sequence, or you can make use of the JUNOS software CLI's `insert` function to save some valuable time as shown here:

```
[edit firewall filter r3-lo0]
lab@r3# insert term bgp before term deny-all-else

[edit firewall filter r3-lo0]
lab@r3# insert term ospf-rsvp before term deny-all-else
```

The modified term sequencing is now displayed and the changes are committed:

```
[edit firewall filter r3-lo0]
lab@r3# show
term icmp {
    from {
        protocol icmp;
        icmp-type [ echo-reply echo-request ];
    }
    then accept;
}
term telnet-in {
    from {
        address {
            10.0.0.0/16;
        }
        protocol tcp;
        destination-port 23;
    }
    then accept;
}
term telnet-out {
    from {
        protocol tcp;
        source-port 23;
    }
    then accept;
}
term ssh-ftp {
    from {
```

```
        protocol tcp;
        source-port [ ssh 20-21 ];
    }
    then accept;
}
term bgp {
    from {
        protocol tcp;
        port bgp;
    }
    then accept;
}
term ospf-rsvp {
    from {
        protocol [ ospf rsvp ];
    }
    then accept;
}
term deny-all-else {
    then {
        count r3-lo0-rejection;
        log;
        discard;
    }
}

[edit firewall filter r3-lo0]
lab@r3# commit
commit complete
```

After a few minutes, r3's OSPF, BGP, and RSVP protocols are once again functional:

```
[edit]
lab@r3# run show ospf neighbor
  Address         Interface             State      ID              Pri  Dead
10.0.2.1          at-0/1/0.0            Full       10.0.3.5        128    33
10.0.2.6          so-0/2/0.100          Full       10.0.3.4        128    34
10.0.4.14         fe-0/0/0.0            Full       10.0.6.1        128    35
10.0.4.2          fe-0/0/1.0            Full       10.0.6.2        128    32
10.0.2.13         fe-0/0/3.0            Full       10.0.9.6        128    37
```

OSPF adjacencies have reformed at r3, making loopback-based IBGP peering once again possible:

```
[edit]
lab@r3# run show bgp summary
```

```
Groups: 3 Peers: 7 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History  Damp State   Pending
inet.0            126842     126830          0          0          0         0
Peer            AS  InPkt  OutPkt  OutQ  Flaps  Last Up/Dwn State|#Active/
                                                          Received/Damped...
172.16.0.14   65222  25859   25869     0      1        5:29  126827/126827/0  0/0/0
10.0.3.4      65412    596   54168     0      1        4:21 1/1/0             0/0/0
10.0.3.5      65412    589   52695     0      1        3:55 0/0/0             0/0/0
10.0.6.1      65412    596   52682     0      1        4:13 1/1/0             0/0/0
10.0.6.2      65412    584   52690     0      1        4:01 0/1/0             0/0/0
10.0.9.6      65412    600   52847     0      1        4:16 1/6/0             0/0/0
10.0.9.7      65412     14   28561     0      1        5:18 0/6/0             0/0/0
```

All BGP sessions have been correctly reestablished at r3. The next command confirms that r3's ingress, egress, and transit LSPs have been re-established:

```
[edit]
lab@r3# run show rsvp session
Ingress RSVP: 1 sessions
To              From            State Rt Style Labelin Labelout LSPname
10.0.9.7        10.0.3.3        Up     0  1 FF       -   100007 r3-r7
Total 1 displayed, Up 1, Down 0

Egress RSVP: 2 sessions
To              From            State Rt Style Labelin Labelout LSPname
10.0.3.3        10.0.9.7        Up     0  1 FF       3        - r7-r3
10.0.3.33       10.0.9.7        Up     0  1 FF       3        - r7-r3-prime
Total 2 displayed, Up 2, Down 0

Transit RSVP: 2 sessions
To              From            State Rt Style Labelin Labelout LSPname
10.0.6.1        10.0.9.6        Up     0  1 FF  100003        3 r6-r1
10.0.9.6        10.0.3.4        Up     0  1 SE  100004        3 r4-r6
Total 2 displayed, Up 2, Down 0
```

With existing services confirmed, you move on to the verification of the services and behavior explicitly stipulated by the RE protection configuration example. You start by verifying that r3 can establish outgoing FTP sessions to the RADIUS/FTP server while not accepting incoming FTP sessions. Note that the FTP service is not enabled on r3 at this time, but this omission does not affect your ability to test the firewall's treatment of incoming FTP connections:

```
[edit]
lab@r3# run ftp 10.0.200.2
Connected to 10.0.200.2.
220 T1-P1 FTP server (Version 6.00LS) ready.
```

```
Name (10.0.200.2:lab): lab
331 Password required for lab.
Password:
230 User lab logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for '/bin/ls'.
total 180998
drwxr-xr-x  2 lab  staff       512 Aug 13  1998 .ssh
-rw-r--r--  1 lab  staff  24001906 Oct 24  1998 jbundle-5.2B3.1-domestic.tgz
-rw-r--r--  1 lab  staff  24217723 Jan 18  1999 jbundle-5.2R2.3-domestic-
   signed.tgz
-rw-r--r--  1 lab  staff  19685721 Nov 30 13:32 jinstall-4.4R1.5-domestic.tgz
-rw-r--r--  1 lab  staff  24614636 Aug 13  1998 jinstall-
. . .
226 Transfer complete.
ftp> exit
221 Goodbye.
```

The display confirms that `r3` can successfully initiate FTP connections. You now verify that incoming FTP connections are blocked by the *r3-lo0* filter in accordance with the example's restrictions:

```
[edit]
lab@r3# run ftp 10.0.3.3
^C
```

The FTP session is aborted by the operator because it does not complete normally. The firewall cache is now examined for evidence of firewall filter activity:

```
[edit]
lab@r3# run show firewall log detail
Time of Log: 2003-03-04 22:12:26 UTC, Filter: r3-lo0, Filter action: discard,
   Name of interface: fxp0.0
Name of protocol: UDP, Packet Length: 78, Source address: 10.0.1.100:137,
   Destination address: 10.0.1.255:137
Time of Log: 2003-03-04 22:12:11 UTC, Filter: r3-lo0, Filter action: discard,
   Name of interface: local
Name of protocol: TCP, Packet Length: 60, Source address: 10.0.3.3:2741,
   Destination address: 10.0.3.3:21
Time of Log: 2003-03-04 22:12:10 UTC, Filter: r3-lo0, Filter action: discard,
   Name of interface: fxp0.0
```

```
Name of protocol: UDP, Packet Length: 229, Source address: 10.0.1.235:138,
   Destination address: 10.0.1.255:138
. . .
```

The highlighted portion of the firewall log confirms that TCP packets destined to port 21, the FTP control port, are being correctly discarded. A similar approach is now used to confirm proper SSH functionality. You begin by verifying r3's ability to initiate SSH connections to another router:

```
[edit]
lab@r3# run ssh 10.0.3.5
The authenticity of host '10.0.3.5 (10.0.3.5)' can't be established.
RSA key fingerprint is 5e:52:6d:e2:83:4f:15:1c:f1:a6:ff:81:f5:df:f7:db.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.0.3.5' (RSA) to the list of known hosts.
lab@10.0.3.5's password:
Last login: Tue Mar  4 09:28:46 2003 from 10.0.1.100
--- JUNOS 5.6R1.3 built 2003-01-02 20:38:33 UTC

lab@r5> quit

Connection to 10.0.3.5 closed.
```

You now verify that incoming SSH sessions are correctly blocked by the *r3-lo0* firewall filter by attempting a "loopback"connection from r3 to itself, because this validates its ability to both generate and receive SSH connections in one fell swoop:

```
[edit]
lab@r3# run ssh 10.0.3.3
^C
```

The SSH session has to be aborted, which is a good indication that the filter is performing as designed. You quickly inspect the log to look for evidence of firewall activity:

```
[edit]
lab@r3# run show firewall log detail
Time of Log: 2003-03-04 22:18:10 UTC, Filter: r3-lo0, Filter action: discard,
   Name of interface: local
Name of protocol: TCP, Packet Length: 60, Source address: 10.0.3.3:3402,
   Destination address: 10.0.3.3:22
. . .
```

As was the case with FTP, the results confirm that your SSH filtering is in full compliance with the configuration example's stipulations. Bidirectional telnet support was confirmed previously when r3 was shown to be able to open a telnet session to itself. You therefore complete your verification steps by confirming that sources outside of your AS's net block can ping r3, while also verifying that telnet attempts from the same external sources are blocked by the *r3-lo0* filter.

The use of the `source` argument is critical in the following commands, which in this case are issued at the data center router. The `source` switch, along with the `192.168.1.1` argument, ensures that the traffic from the DC router is sourced from its 192.168.0.1 external address:

```
lab@dc> ping 10.0.3.3 source 192.168.1.1 count 2
PING 10.0.3.3 (10.0.3.3): 56 data bytes
64 bytes from 10.0.3.3: icmp_seq=0 ttl=254 time=0.736 ms
64 bytes from 10.0.3.3: icmp_seq=1 ttl=254 time=0.637 ms

--- 10.0.3.3 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.637/0.686/0.736/0.049 ms
```

The output confirms that external sources can ping `r3`. Your next command tests whether telnet sessions initiated from source addresses outside of the 10.0/16 net block are correctly blocked:

```
lab@dc> telnet 10.0.3.3 source 192.168.1.1
Trying 10.0.3.3...
Connected to 10.0.3.3.
Escape character is '^]'.

r3 (ttyp1)

login:
telnet> quit
Connection closed.
```

Darn, the telnet session sourced from 192.168.1.1 succeeds, which indicates there is a problem in your firewall filter. Can you spot the problem in term *2* of the *r3-lo0* filter?

```
[edit]
lab@r3# show firewall filter r3-lo0 term 2
from {
    address {
        10.0.0.0/16;
    }
    protocol tcp;
    port 23;
}
then accept;
```

The highlights call attention to the problem area. In this case, the term neglects to identify that the *source* address of telnet request packets must fall within the 10.0/16 net block. As written, the presence of any 10.0/16 address in the IP packet, whether used as a destination or a source address, will satisfy the term's addressing-related match criteria. Because telnet sessions

to r3 will always have a destination address stemming from the 10.0/16 address block, the current filter syntax fails to correctly meet the configuration example's telnet restrictions. To correct the problem, you modify the *telnet-in* term in the *r3-lo0* filter, as shown next:

```
[edit firewall filter r3-lo0 term 2]
lab@r3# show
from {
    source-address {
        10.0.0.0/16;
    }
    protocol tcp;
    port 23;
}
then accept;
```

After the changes are committed, you retest telnet connectivity from the DC router:

```
lab@dc> telnet 10.0.3.3 source 192.168.1.1
Trying 10.0.3.3...
^C
```

The telnet connection from an external source now fails in accordance with the proscribed behavior. A quick examination of the firewall log on r3 confirms that telnet connections from a 192.168.1.1 address are now denied:

```
[edit firewall filter r3-lo0 term 2]
lab@r3# run show firewall log detail
Time of Log: 2003-03-04 23:01:35 UTC, Filter: pfe, Filter action: discard, Name
   of interface: fe-0/0/3.0
Name of protocol: TCP, Packet Length: 60, Source address: 192.168.1.1:1539,
   Destination address: 10.0.3.3:23
. . .
```

The final telnet verification step confirms that r3 can initiate telnet sessions to external peers, which in this case takes the form of the DC router:

```
[edit]
lab@r3# run telnet 192.168.1.1
Trying 192.168.1.1...
Connected to 192.168.1.1.
Escape character is '^]'.

dc (ttyp0)

login: lab
Password:
Last login: Sun Mar 16 01:15:50 on ttyd0
```

```
--- JUNOS 5.6R1.3 built 2003-01-02 20:38:33 UTC

lab@dc>
```

Excellent! The telnet session to a destination address outside of the 10.0/16 range is successful, which means that all criteria for the RE-based firewall filtering configuration scenario have been confirmed operational. With the *r3-lo0* firewall filter's behavior verified, you might want to remove the explicit *deny-all-else* term from the *r3-lo0* firewall filter. In this example, the term is left in place because its presence does not violate any configuration criteria, and because its continued existence will cause no operational impact.

Transit Filtering

This configuration example is designed to simulate a typical transit interface firewall filtering configuration task. To complete this example, you must modify r4's configuration to support the following criteria:

- Count *all* TCP connections initiated by C1, as seen on r4's fe-0/0/0 interface.
- Ensure that r4 never forwards packets received from C1 with spoofed source addresses.
- Ensure that C1 *never receives* the following traffic from r4:
  - ICMP echo request packets with a total length of 1000 bytes (inclusive of Layer 3 headers).
  - Any traffic that claims to be sourced from C1's 200.200/16 net block.
  - Filter UDP and TCP traffic with destination ports in the range of 6000–6010 inclusive. Configure the router to send ICMP port unreachable error messages when filtering these packets.

Creating Firewall Filters

These criteria require that you create, and apply, two independent firewall filters. The first filter, which is called *c1-in*, functions to count all initial TCP SYN segments received on r4's fe-0/0/0 interface while also preventing source address spoofing. The second filter, called *c1-out*, will be applied in the direction of r4 to C1 to meet the remaining requirements of the transit filtering configuration example.

One of the more challenging aspects of this task relates to the candidate's ability to translate "TCP connections initiated by C1" into the use of a tcp-initial based match condition (or a "(syn & !ack)" match criterion if you prefer not to use the keyword designed to spare you this type of syntax). TCP segments with this specific flag pattern identify the initial segment used in the three-way handshake that establishes a TCP connection. By counting the initial TCP SYN segments sent from C1, you are in effect counting the number of TCP sessions that site C1 is attempting to establish through the r4 peering point. You must use care to ensure that no traffic is actually blocked as a result of the *c1-in* filter's application.

The following commands are entered on r4 to create the first term in the *c1-in* firewall filter:

```
[edit firewall c1-in]
lab@r4# set term 1 from tcp-initial
```

```
[edit firewall c1-in]
lab@r4# set term 1 from protocol tcp

[edit firewall filter c1-in]
lab@r4# set term 1 then count c1-syns

[edit firewall filter c1-in]
lab@r4# set term 1 then next term
```

With SYN segment counting addressed by the first term, you next configure a term that enforces source address validity on packets received from C1:

```
[edit firewall filter c1-in]
lab@r4# set term 2 from source-address 200.200/16

[edit firewall filter c1-in]
lab@r4# set term 2 from source-address 172.16.0.6

[edit firewall filter c1-in]
lab@r4# set term 2 then accept
```

The completed *c1-in* filter is now displayed:

```
[edit firewall filter c1-in]
lab@r4# show
term 1 {
    from {
        protocol tcp;
        tcp-initial;
    }
    then {
        count c1-syns;
        next term;
    }
}
term 2 {
    from {
        source-address {
            200.200.0.0/16;
            172.16.0.6/32;
        }
    }
    then accept;
}
```

Note that the first term of the *c1-in* filter correctly specifies a TCP protocol–based match condition to guard against inadvertent packet matches (and erroneous counting in this example). Many candidates incorrectly assume that a keyword such as `tcp-initial` has an embedded TCP protocol test and therefore often fail to include the necessary protocol match criterion required to guarantee accurate packet matching. Recall that the `tcp-initial` keyword is a synonym for any TCP segment with a reset ACK flag and a set SYN flag, and as noted previously, you can use an alternative, TCP flag–based match condition, if you are so inclined.

In this example, a counter named *c1-syns* is defined to tally the number of TCP connection requests received from C1. Term *1* makes use of a `next-term` action modifier to ensure that all TCP SYN segments are counted by term *1*, without actually being accepted, because term 2 is used to validate the source address of all traffic received. The need to count TCP SYN segments, whether they have valid source addresses or not, is indicated by the instruction that you must count *all* connection requests received from C1.

The final term in the firewall filter defines an `accept` action for all traffic received from C1 with source addresses in the 200.200/16 space, and for those packets that are initiated by C1 itself. Note that these packets normally carry a source address of 172.16.0.6. Term *2* provides a critical function, because it negates the default *deny-all* action of a JUNOS software firewall filter for all traffic received from C1 with valid source addressing. Packets with bogus source addresses will not match term *2*, which means they will still fall victim to silent discard at the hands of the implicit *deny-all* action at the end of the firewall processing chain. Including C1's interface address ensures that the application of the *c1-in* filter will not cause any problems with C1's BGP session to `r4`. Candidates often neglect to include the access link's address when implementing a source address verification firewall filter, which can create problems for routing protocols as well as diagnostic tools such as ping and traceroute.

With the *c1-in* filter complete, you begin configuration of the *c1-out* filter. Your first goal is to restrict ICMP packets of a particular length from being sent to C1. The following commands correctly set the filter to match on a packet length of 1000 bytes. Note that in JUNOS software, link-level overhead is not counted in the context of a firewall filter.

```
[edit firewall filter c1-out]
lab@r4# set term lk-icmp from protocol icmp

[edit firewall filter c1-out]
lab@r4# set term lk-icmp from packet-length 1000

[edit firewall filter c1-out]
lab@r4# set term lk-icmp then discard
```

The next term prevents packets with spoofed source addresses from entering C1's network:

```
lab@r4# set term no-spoof from source-address 200.200/16

[edit firewall filter c1-out]
lab@r4# set term no-spoof then discard
```

The final term uses a port range match criteria to block packets with destination ports in the range of 6000 through 6010 inclusive, while sending back ICMP port unreachable error messages when matching packets are filtered:

```
lab@r4# show term ports
from {
    destination-port 6000-6010;
}
then {
    reject port-unreachable;
}
```

Note that a protocol match condition is not specified in the *ports* term. In this case, not specifying the TCP and UDP protocols yields the same result as if you had taken the time to add a `protocol [ tcp udp ]` match condition to the term. A final *accept-all* term is now added, which is in keeping with the filter's intended use on a transit interface:

```
[edit firewall filter c1-out]
lab@r4# set term accept-all then accept
```

You display the completed *c1-out* firewall filter before proceeding to the section on filter application and verification:

```
[edit firewall filter c1-out]
lab@r4# show
term 1k-icmp {
    from {
        packet-length 1000;
        protocol icmp;
    }
    then discard;
}
term no-spoof {
    from {
        source-address {
            200.200.0.0/16;
        }
    }
    then discard;
}
term ports {
    from {
        destination-port 6000-6010;
    }
    then {
        reject port-unreachable;
    }
```

```
}
term accept-all {
    then accept;
}
```

Applying and Verifying Transit Firewall Filters

You now apply the transit firewall filters designed for site C1, being careful that you apply them in the appropriate direction, as demonstrated in the following commands:

```
[edit interfaces fe-0/0/0]
lab@r4# set unit 0 family inet filter input c1-in

[edit interfaces fe-0/0/0]
lab@r4# set unit 0 family inet filter output c1-out
```

After displaying the modified configuration for r4's fe-0/0/0 interface, the changes are committed:

```
[edit interfaces fe-0/0/0]
lab@r4# show
unit 0 {
    family inet {
        filter {
            input c1-in;
            output c1-out;
        }
        address 172.16.0.5/30;
    }
}

[edit interfaces fe-0/0/0]
lab@r4# commit
commit complete
```

Transit filtering verification begins with the determination that initial TCP segments are being counted. You start by examining the current counter value for the *c1-syns* counter:

```
lab@r4# run show firewall
Filter: c1-out
Filter: c1-in
Counters:
Name                                                Bytes              Packets
c1-syns                                                 0                    0
```

The *c1-syns* counter shows that no initial TCP segments have been received on r4's fe-0/0/0 interface. You now establish a telnet session to C1 for reasons of initial TCP segment generation:

```
lab@r4# run telnet 200.200.0.1
Trying 200.200.0.1...
```

```
Connected to 200.200.0.1.
Escape character is '^]'.

c1 (ttyp0)

login: lab
Password:
Last login: Wed Feb 12 13:29:31 from 172.16.0.5

--- JUNOS 5.6R1.3 built 2002-03-23 02:44:36 UTC

lab@c1>
```

Once logged into C1, a telnet session is attempted to r3:

```
lab@c1> telnet 10.0.3.3
Trying 10.0.3.3...
^C
lab@c1> quit

Connection closed by foreign host.
```

Note that the failure of the telnet session is expected, considering that r3 has a filter blocking telnet requests for any packet with a source addresses not contained within the 10.0/16 aggregate, not to mention that r3 has no route back to the 172.16.0.6 address that C1 used to source its telnet request by default. Regardless of the telnet session's success or failure, some TCP initial SYN segments should have been generated by C1, and this knowledge allows you to confirm the proper operation of the *c1-syns* counter as defined in the *c1-in* firewall filter:

```
[edit interfaces fe-0/0/0]
lab@r4# run show firewall
Filter: c1-out
Filter: c1-in
Counters:
Name                                                Bytes              Packets
c1-syns                                               120                    2
```

The count value has incremented by two, providing a strong indication that you have met the requirements for TCP connection request counting. Verifying that spoofed packets sent from C1 are correctly filtered is difficult because you can not generate packets from C1 that do not use a source address from either the 200.200/16 or 172.16.0.4/30 address ranges. For now, you can assume that the anti-spoofing aspects of the *c1-in* filter are functional. You now confirm operation of the *c1-out* filter, starting with the requirement that 1000-byte ICMP packets must be filtered:

```
[edit firewall]
lab@r4# run ping 200.200.0.1 size 1100 count 2
```

```
PING 200.200.0.1 (200.200.0.1): 1100 data bytes
1108 bytes from 200.200.0.1: icmp_seq=0 ttl=255 time=1.248 ms
1108 bytes from 200.200.0.1: icmp_seq=1 ttl=255 time=1.125 ms

--- 200.200.0.1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 1.125/1.187/1.248/0.061 ms
```

The display confirms that packets over 1000 bytes can be sent. In this case, the IP and ICMP headers (28 bytes) yield a total packet size of 1128 bytes.

```
[edit firewall]
lab@r4# run ping 200.200.0.1 size 900 count 2
PING 200.200.0.1 (200.200.0.1): 900 data bytes
908 bytes from 200.200.0.1: icmp_seq=0 ttl=255 time=1.179 ms
908 bytes from 200.200.0.1: icmp_seq=1 ttl=255 time=1.002 ms

--- 200.200.0.1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 1.002/1.091/1.179/0.089 ms
```

Good, ICMP packets smaller than 1000 bytes (928 bytes with Layer 3 encapsulation overhead) can also be sent.

```
[edit firewall]
lab@r4# run ping 200.200.0.1 size 972 count 2
PING 200.200.0.1 (200.200.0.1): 972 data bytes
ping: sendto: Operation not permitted
ping: sendto: Operation not permitted
^C
--- 200.200.0.1 ping statistics ---
2 packets transmitted, 0 packets received, 100% packet loss
```

With the payload set to 972 bytes, the additional 28 bytes of IP and ICMP overhead yield a total packet size of 1000 bytes. The output confirms that these packets are being correctly filtered in accordance with the scenario's requirements. As was the case with the *c1-in* filter, it will be difficult to verify the operation of the *no-spoof* term in the *c1-out* firewall filter. This is because you will not be able to generate a packet with a source address from the 200.200/16 address space at r4 unless you temporarily assign a 200.200/16 address to one of its interfaces. Note that the source switch will not create packets with the specified address unless one of the router's interfaces actually *owns* that address. With these difficulties in mind, you decide to forgo verification of the *no-spoof* term, which brings you to the *c1-out* filter's *ports* term. You test this term with the following command, which generates a TCP packet at r5 with the destination port set to 6005:

```
[edit]
lab@r5# run telnet 200.200.0.1 port 6005
```

```
Trying 200.200.0.1...
^C
[edit]
lab@r5#
```

The output indicates that the telnet session failed, but it is not clear whether this failure is the result of the *c1-out* firewall filter on r4, or whether the failure relates to the fact that C1 does not offer a telnet service on port 6005. To verify that ICMP port unreachable messages are being generated by r4, you open a second telnet session to r5 to facilitate traffic monitoring on its so-0/1/0.0 interface while the telnet connection to port 6005 is retried:

```
[edit]
lab@r5# run monitor traffic interface so-0/1/0
verbose output suppressed, use <detail> or <extensive> for full protocol decode
Listening on so-0/1/0, capture size 96 bytes

12:59:40.625384  In IP 10.0.2.10 > 224.0.0.5:  OSPFv2-hello 48:
   rtrid 10.0.3.4 backbone
12:59:40.845517 Out IP 10.0.2.9 > 10.0.2.10: RSVP Resv Message, length: 128
12:59:41.675217 Out LCP echo request           (type 0x09  id 0x64  len 0x0008)
12:59:41.675787  In LCP echo reply             (type 0x0a  id 0x64  len 0x0008)
12:59:42.911833 Out IP 10.0.3.5.4585 > 200.200.0.1.6005: S 3327432533:
   3327432533(0) win 16384 <mss 4430,nop,wscale 0,nop,nop,timestamp 1114703 0>
   (DF)
12:59:42.912340  In IP 10.0.2.10 > 10.0.3.5: icmp: 200.200.0.1 tcp port 6005
   unreachable
12:59:43.275359 Out IP 10.0.3.5.audio-activmail > 10.0.3.4.bgp: P 128034447:
   128034466(19) ack 2801823782 win 16384 <nop,nop,timestamp 1114740 1446518>:
   BGP, length: 19
12:59:43.375349  In IP 10.0.3.4.bgp > 10.0.3.5.audio-activmail: . ack 19
   win 16384 <nop,nop,timestamp 1449049 1114740>
12:59:43.655368 Out IP 10.0.2.9 > 10.0.2.10: RSVP Hello Message, length: 32
12:59:43.675361  In IP 10.0.2.10 > 10.0.2.9: RSVP Hello Message, length: 32
12:59:44.055338 Out IP 10.0.2.9 > 224.0.0.5:  OSPFv2-hello 48: rtrid 10.0.3.5
   backbone
^C
13 packets received by filter
0 packets dropped by kernel
```

The highlights in the `monitor traffic` command's output confirm that a TCP segment was sent to destination port 6005, and that the required ICMP port unreachable message was returned by r4. With the confirmation of the *ports* term, your transit firewall filter verification steps are complete.

Note that the *r3-lo0* firewall filter currently in place at r3 prevents the detection of ICMP destination unreachable messages because the filter permits only ICMP echo–related messages. In this example, choosing to test the *ports* term in the *c1-out* filter with a telnet session initiated from r3 would have been very tricky, because the *r3-lo0* filter discards ICMP error messages *before* they can be displayed with the monitor traffic command. This situation could easily send a candidate on a wild goose chase based on the misguided belief that r4 is not generating the appropriate ICMP error message, when in fact it is!

Policing

This section demonstrates a variety of policer applications and the techniques used to verify the proper operation of a policer. Policing is used in JUNOS software to rate limit interfaces or protocol flows at either Layer 3 or Layer 4. Although policers are often called from within a firewall filter, a policer can act as a stand-alone entity, such as in the case of an interface policing application where a firewall filter is not used.

To complete this section, you must alter your configurations to meet these criteria:

- Limit the amount of ICMP echo traffic received by r3's RE to 1Mbps with a 10,000-byte burst size.
- Rate limit *all* traffic using the primary r4–r6 LSP to 500Kbps with a 10,000-byte burst size.
- Assume that there are 254 hosts on the 192.168.1/24 data center subnet. Modify the configuration of r3 in accordance with these requirements:
  - Limit TCP flows to 1Mbps/50Kbps for all DC hosts.
  - Count TCP packets sent to *each* host.
  - Ensure that you count and police only the traffic that is received from the T1 peer.

Configuring ICMP Policing

Your first configuration objective requires that you police the ICMP echo traffic received by r3's RE. In this case, you decide to modify the existing *r3-lo0* filter to add the required policing functionality. You begin by defining a policer named icmp, taking care that the specified bandwidth and burst size parameters are in accordance with the provided stipulations:

```
[edit firewall]
lab@r3# set policer icmp if-exceeding bandwidth-limit 1M

[edit firewall]
lab@r3# set policer icmp if-exceeding burst-size-limit 10k

[edit firewall]
lab@r3# set policer icmp then discard
```

The completed policer definition is now displayed:

```
[edit firewall]
lab@r3# show policer icmp
if-exceeding {
    bandwidth-limit 1m;
    burst-size-limit 10k;
}
then discard;
```

The next command modified the existing *r3-lo0* filter to make use of the newly defined policer:

```
[edit firewall filter r3-lo0 term icmp]
lab@r3# set then policer icmp

[edit firewall filter r3-lo0 term icmp]
lab@r3# delete then accept
```

The modified *icmp* term is now displayed:

```
[edit firewall filter r3-lo0 term icmp]
lab@r3# show
from {
    protocol icmp;
    icmp-type [ echo-reply echo-request ];
}
then {
    policer icmp;
}
```

Note that removal of the `accept` action from the *icmp* term was not strictly necessary. The use of a policer is considered an *action modifier*, and as such, the default action for traffic matching the term will become `accept` unless explicitly negated with a `reject` or `discard` action. In this case, traffic that exceeds the *icmp* policer's limits will be discarded, while traffic within the policing profile is handed back to the *icmp* term, where it is accepted by the implicit `accept` action.

Verifying ICMP Policing

Before committing the changes on `r3`, you decide to conduct some flood ping testing between `r5` and `r3` so that you can better judge the effects of the ICMP policing configuration that is soon to be in place at `r3`:

```
[edit firewall]
lab@r5# run ping rapid count 500 size 1200 10.0.3.3
PING 10.0.3.3 (10.0.3.3): 1200 data bytes
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
--- 10.0.3.3 ping statistics ---
500 packets transmitted, 500 packets received, 0% packet loss
round-trip min/avg/max/stddev = 1.927/2.297/10.233/0.659 ms
```

The display indicates that none of the 500 pings are lost between `r5` and `r3` in the absence of ICMP policing. You now commit the policer configuration at `r3`, and repeat the ping test from `r5` to confirm the effects of ICMP echo policing:

```
[edit firewall]
lab@r5# run ping rapid count 500 size 1200 10.0.3.3
PING 10.0.3.3 (10.0.3.3): 1200 data bytes
!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!
   !!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!
   !!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!!.!!!!!!!!!!.
   !!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!
   .!!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!
   !.!!!!!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!
   !!!!!.!!!!!!!!!!.!!!!!!!!!!.!!!!!!!
--- 10.0.3.3 ping statistics ---
500 packets transmitted, 455 packets received, 9% packet loss
round-trip min/avg/max/stddev = 2.065/2.450/9.813/0.942 ms
```

The output confirms that ICMP echo traffic is now being lost between `r5` and `r3`, which provides a good indication that your ICMP policer is working. The final check involves analysis of the *`icmp`* policer discard count back at `r3`:

```
[edit]
lab@r3# run show firewall filter r3-lo0
Filter: r3-lo0
Counters:
Name                                                Bytes              Packets
r3-lo0-rejection                                     8009                  141
Policers:
Name                                              Packets
icmp-icmp                                              45
```

The highlighted entries confirm that 45 packets have been discarded by the *`icmp`* policer, which is called by a term that also happens to be named *`icmp`* in this example. These results conclude the verification steps for the ICMP policing component of this configuration scenario.

Rate Limiting an LSP

The next configuration goal requires that you police the traffic using the primary *`r4-r6`* LSP. To achieve this requirement, you can deploy an interface policer for the `mpls` family to the correct interface on a *transit* LSR, which many candidates find to be somewhat tricky. You can not

police the mpls family at the LSP's ingress node, because the packets are considered family ip *until* they are actually placed into an LSP, which occurs after any interface policing that may be in effect. Note that use of an ip family interface policer at r4 violates the provided criteria, because such a configuration would also rate limit traffic that is *not* actually forwarded through the primary r4–r6 LSP.

Note that it is also possible to achieve the specified rate limiting behavior by using a firewall filter in conjunction with a policer to rate limit the traffic that is mapped to the *r4-r6* LSP at r4. In this case, the only route that currently maps to the *r4-r6* LSP is the 220.220/16 route stemming from C2. Therefore your firewall filter would have to match on destination addresses belonging to the 220.220/16 block for subjection to your policer. This example demonstrates the interface-based policing approach for the mpls family, because it is the most expedient solution.

You begin by defining the policer at r5, which in this case has been named limit-mpls. You have decided to police the LSP at r5 because in this example it is the first transit LSR for the primary *r4-r6* LSP:

```
[edit firewall]
lab@r5# set policer limit-mpls if-exceeding bandwidth-limit 500k

[edit firewall]
lab@r5# set policer limit-mpls if-exceeding burst-size-limit 10k

[edit firewall]
lab@r5# set policer limit-mpls then discard
```

The policer definition is displayed:

```
[edit firewall]
lab@r5# show policer limit-mpls
if-exceeding {
    bandwidth-limit 500k;
    burst-size-limit 10k;
}
then discard;
```

The output confirms that the *limit-mpls* policer is defined in accordance with the provided criteria. You need to apply the *limit-mpls* policer to an appropriate interface on r5 to place the policer into service. Use caution to ensure that the policer is applied to an interface at r5 that actually handles the *r4-r6* LSP, and that the filter is applied in the correct direction, which is based on the choice of LSP ingress vs. LSP egress interface at r5. Note that directionality is critical when dealing with MPLS because LSPs are always unidirectional. In this example, you decide to apply the *limit-mpls* policer to r5's so-0/1/0.0 interface, which is an ingress interface for the LSP. The choice of an LSP ingress interface in turn requires that you apply the policer in the input direction for things to work properly. Note that there is no requirement for limiting traffic traveling over the *r4-r6-prime* secondary LSP.

Before applying the mpls family interface policer to r5, the routing of the *r4-r6* LSP is verified:

```
[edit interfaces so-0/1/0]
lab@r5# run show rsvp session detail transit name r4-r6
```

```
Transit RSVP: 5 sessions, 1 detours

10.0.9.6
  From: 10.0.3.4, LSPstate: Up, ActiveRoute: 1
  LSPname: r4-r6, LSPpath: Secondary
  Suggested label received: -, Suggested label sent: -
  Recovery label received: 100005, Recovery label sent: 100005
  Resv style: 1 SE, Label in: 100005, Label out: 100005
  Time left:  193,  Since: Wed Mar  5 15:19:43 2003
  Tspec: rate 2Mbps size 2Mbps peak Infbps m 20 M 1500
  Port number: sender 4 receiver 39454 protocol 0
  PATH rcvfrom: 10.0.2.10  (so-0/1/0.0) 149 pkts
  PATH sentto: 10.0.2.2 (at-0/2/1.0) 316 pkts
  RESV rcvfrom: 10.0.2.2  (at-0/2/1.0) 149 pkts
  Explct route: 10.0.2.2 10.0.2.13
  Record route: 10.0.2.10  <self>  10.0.2.2  10.0.2.13

10.0.9.6
  From: 10.0.3.4, LSPstate: Up, ActiveRoute: 1
  LSPname: r4-r6, LSPpath: Primary
  Suggested label received: -, Suggested label sent: -
  Recovery label received: 100006, Recovery label sent: 3
  Resv style: 1 SE, Label in: 100006, Label out: 3
  Time left:  193,  Since: Wed Mar  5 15:19:43 2003
  Tspec: rate 2Mbps size 2Mbps peak Infbps m 20 M 1500
  Port number: sender 5 receiver 39454 protocol 0
  PATH rcvfrom: 10.0.2.10  (so-0/1/0.0) 149 pkts
  PATH sentto: 10.0.8.5 (fe-0/0/0.0) 2 pkts
  RESV rcvfrom: 10.0.8.5  (fe-0/0/0.0) 4 pkts
  Explct route: 10.0.8.5
  Record route: 10.0.2.10  <self>  10.0.8.5
Total 2 displayed, Up 2, Down 0
```

The highlights in the display indicate that both the primary and secondary versions of the *r4-r6* LSP ingress at `r5` via its so-0/1/0.0 interface (address 10.0.2.9). Armed with this knowledge, you apply the *limit-mpls* policer to `r5`'s so-0/1/0.0 interface in the `input` direction. Note that the choice of a LSP egress interface at `r5` would require an `output` specification for the same policer:

```
[edit interfaces so-0/1/0]
lab@r5# set unit 0 family mpls policer input limit-mpls
```

The policer's application is now confirmed:

```
[edit interfaces so-0/1/0]
lab@r5# show
```

```
encapsulation ppp;
unit 0 {
    family inet {
        address 10.0.2.9/30;
    }
    family mpls {
        policer {
            input limit-mpls;
        }
    }
}
```

Verifying LSP Policing

Before committing the LSP policing changes on r5, you decide to first test the native forwarding performance of the *r4-r6* LSP so you can better judge the effectiveness of your *limit-mpls* policer:

```
[edit]
lab@r4# run traceroute 220.220.0.1
traceroute to 220.220.0.1 (220.220.0.1), 30 hops max, 40 byte packets
 1  10.0.2.9 (10.0.2.9)  4.394 ms  0.914 ms  0.806 ms
     MPLS Label=100006 CoS=0 TTL=1 S=1
 2  10.0.8.5 (10.0.8.5)  0.523 ms  0.537 ms  0.495 ms
 3  220.220.0.1 (220.220.0.1)  0.605 ms  0.606 ms  0.590 ms
```

The traceroute display confirms that packets to 220.220.0.1 are being forwarded through the *r4-r6* LSP. Flood pings are now generated using the rapid switch:

```
[edit]
lab@r4# run ping rapid count 500 size 1400 220.220.0.1
PING 220.220.0.1 (220.220.0.1): 1400 data bytes
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
--- 220.220.0.1 ping statistics ---
500 packets transmitted, 500 packets received, 0% packet loss
round-trip min/avg/max/stddev = 1.628/1.681/9.728/0.525 ms
```

The output indicates that none of the 500 pings were lost as they transited the *r4-r6* LSP on their way to C2. The test is repeated after committing the policer configuration at r5:

```
[edit interfaces so-0/1/0]
lab@r5# commit
commit complete
```

```
[edit interfaces so-0/1/0]

[edit]
lab@r4# run ping rapid count 500 size 1400 220.220.0.1
PING 220.220.0.1 (220.220.0.1): 1400 data bytes
!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!
   .!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!
   !!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!
   .!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!
   !!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.
   !!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!!
   !!!.!!!!!!!!.!!!!!!!!.!!!!!!!!.!!!!
--- 220.220.0.1 ping statistics ---
500 packets transmitted, 445 packets received, 11% packet loss
round-trip min/avg/max/stddev = 1.629/1.688/9.718/0.418 ms
```

The results confirm that packets are now being lost within the LSP, and this provides some indication that the *limit-mpls* policer is taking effect. As a final confirmation, you display some additional policer-related information at r5:

```
[edit interfaces so-0/1/0]
lab@r5# run show interfaces policers so-0/1/0
Interface       Admin Link Proto Input Policer         Output Policer
so-0/1/0        up    up
so-0/1/0.0      up    up   inet
                           mpls   so-0/1/0.0-in-policer
```

Adding the policers switch to a show interfaces command is useful when you want to quickly confirm which interfaces have policers applied, and in which directions(s). The next command provides statistics for interface-level policers, which prove invaluable when the goal is to confirm policer operation:

```
[edit interfaces so-0/1/0]
lab@r5# run show policer
Policers:
Name                                                Packets
__default_arp_policer__                                   0
so-0/1/0.0-in-policer                                    55
```

The output confirms that the interface policer has made 55 packet discards. This confirms that you have met the specified requirements for LSP policing.

Configuring Prefix-Specific Counting and Policing

The last configuration requirement in this section poses some interesting challenges. To meet the stated requirements, you must configure r3 to police and count the TCP flows associated with *each* of the 254 hosts present on the 192.168.1/24 subnet. One possible solution to this dilemma involves the creation of a single firewall filter with some 255 terms (254 terms to match on each

data center host address, and a final term that accepts all other traffic). Note that because in this example all hosts are to be policed using the same parameters, all of the terms in such a gargantuan firewall filter could point to a common policer definition. However, with time pressures being a critical aspect of any lab-based examination, spending the day on a single firewall filter is almost certainly a less-than-ideal use of one's time in the certification lab. The most direct way to tackle this problem is to deploy Prefix Specific Counters and Policers (PSCP). PSCP is a JUNOS software release 5.6 feature designed to greatly simplify the creation of prefix-specific counters and policers, and this is just what this scenario calls for!

You begin your PSCP configuration at `r3` with the definition of a policer called *`dc`*:

```
[edit firewall]
lab@r3# set policer dc if-exceeding bandwidth-limit 1M

[edit firewall]
lab@r3# set policer dc if-exceeding burst-size-limit 50K

[edit firewall]
lab@r3# set policer dc then discard
```

The completed policer is displayed for confirmation:

```
[edit firewall]
lab@r3# show policer dc
if-exceeding {
    bandwidth-limit 1m;
    burst-size-limit 50k;
}
then discard;
```

You now define the desired PSCP behavior by creating a `prefix-action` stanza that evokes both policing and counting functionality. Note that this `prefix-action` set is flagged as being `filter-specific` in this case; this means that the `prefix-action` set can be evoked only once by a given firewall filter, regardless of how many terms in the filter may call the `prefix-action` set. This setting is chosen here because it makes operational analysis simpler. The default term specific behavior of a `prefix-action` set requires the specification of the calling filter's term name when displaying the count and policer values. The following commands define a `prefix-action` set called *`data-center`*:

```
[edit firewall]
lab@r3# edit family inet prefix-action data-center

[edit firewall family inet prefix-action data-center]
lab@r3# set count

[edit firewall family inet prefix-action data-center]
lab@r3# set policer dc
```

```
[edit firewall family inet prefix-action data-center]
lab@r3# set destination-prefix-length 32

[edit firewall family inet prefix-action data-center]
lab@r3# set subnet-prefix-length 24
```

The completed `prefix-action` set is displayed:

```
[edit firewall]
lab@r3# show family inet
prefix-action data-center {
    policer dc;
    count;
    filter-specific;
    subnet-prefix-length 24;
    destination-prefix-length 32;
}
```

The *`data-center`* set is configured to count and to police traffic flows, based on the definition of the policer called *`dc`*. The setting of the `subnet-prefix-length` to *24* tells the router that it should create as many as 256 counters and *`dc`* policer instances. Equally important is the setting of the `destination-prefix-length` value of 32, because this setting indicates that a policer/counter set should be allocated to each 32-bit destination host address; this setting is necessary to achieve the per-host counting and policing behavior required in this scenario. By way of comparison, setting the `destination-prefix-length` to a value of *30* results in the sharing of a policer/counter set by every 4 consecutive destination host addresses, and the need to create only 64 policer/counter instances.

With the `prefix-action` set and the *`dc`* policer defined, you next create a firewall filter that uses destination address and TCP protocol–based match conditions to evoke the *`data-center`* `prefix-action` set:

```
[edit firewall filter dc-pscp]
lab@r3# set term 1 from protocol tcp

[edit firewall filter dc-pscp]
lab@r3# set term 1 from destination-address 192.168.1/24

[edit firewall filter dc-pscp]
lab@r3# set term 1 from source-address 10.0/16 except

[edit firewall filter dc-pscp]
lab@r3# set term 1 from source-address 0/0

[edit firewall filter dc-pscp]
lab@r3# set term 1 then prefix-action data-center
```

```
[edit firewall filter dc-pscp]
lab@r3# set term 2 then accept
```

The completed firewall filter is displayed:

```
[edit firewall filter dc-pscp]
lab@r3# show
term 1 {
    from {
        source-address {
            10.0.0.0/16 except;
            0.0.0.0/0;
        }
        destination-address {
            192.168.1.0/24;
        }
        protocol tcp;
    }
    then prefix-action data-center;
}
term 2 {
    then accept;
}
```

In this example, you are only to police and count packets sent to 192.168.1/24 destinations when these packets are received from the T1 peer. Note that the example filter is written to match on the TCP protocol and destination addresses associated with the 192.168.1/24 data center subnet. The inclusion of the `source-address` match condition may or may not be necessary, depending on whether the *`dc-pscp`* filter is applied as an input filter in `r3`'s fe-0/0/2 interface, versus an output application on its fe-0/0/3 or so-0/2.0.100 interfaces. Including the `except` argument for the 10.0/16 `source-address` match condition results in the declaration of a nonmatch in term *`1`* for any packet carrying a 10.0/16 source address. Note that all other source addresses are considered to match the 0/0 `source-address` declaration. Using `except` to force a mismatch for selected traffic is a very useful firewall filter construct, as demonstrated in this example. Including the source address match criteria in the term that calls the *`data-center`* prefix action set serves to guarantee that you will not inadvertently evoke PSCP functionality for traffic originating within your own AS. The added safety net makes the inclusion of the source address match criteria highly recommended in this case.

You complete your PSCP configuration by applying the *`dc-pscp`* firewall filter in the input direction on `r3`'s fe-0/0/2 interface. Note that you could also apply the filter as an output to `r3`'s fe-0/0/3 *and* so-0/2/0.100 interfaces with the same results:

```
[edit interfaces fe-0/0/2]
lab@r3# set unit 0 family inet filter input dc-pscp
```

Be sure to commit your PSCP changes before proceeding to the confirmation section.

Verifying Prefix-Specific Counting and Policing

You begin the verification of your PSCP configuration by confirming the presence of 256 unique counters and policers on r3:

```
[edit]
lab@r3# run show firewall prefix-action-stats filter dc-pscp prefix-action
   data-center
Filter: dc-pscp
Counters:
Name                                                Bytes              Packets
data-center-0                                           0                    0
data-center-1                                           0                    0
. . .
data-center-253                                         0                    0
data-center-254                                         0                    0
data-center-255                                         0                    0
     h
Name                                                Packets
data-center-0                                           0
data-center-1                                           0
. . .
data-center-254                                         0
    h
```

The truncated display confirms that the required number of policers and counters (254) are present. In this configuration, two counters and policers will never be used; this is because they index the unassigned *all 0s* or *all 1s* host IDs on the 192.168.1/24 subnet. You now verify policing and counting functionality by generating TCP traffic from the T1 router. Note that traffic generated locally at r3 will not be counted or policed, even if the *dc-pscp* filter is applied as an output filter to its fe-0/0/3 or so-0/2/0.100 interfaces:

```
lab@T1-P1> telnet 192.168.1.1 source 130.130.0.1
Trying 192.168.1.1...
Connected to 192.168.1.1.
Escape character is '^]'.

dc (ttyp0)

login: lab
Password:
Last login: Mon Mar 17 02:31:14 from 10.0.2.14

--- JUNOS 5.6R2.4 built 2003-02-14 23:22:39 UTC

lab@dc> quit
```

```
Connection closed by foreign host.

lab@T1-P1> telnet 192.168.1.10 source 130.130.0.1
Trying 192.168.1.10...
^C
lab@T1-P1>
```

The routers in the test bed are now running the 5.6R2.4 version of JUNOS software, which became available while this section was being written.

In this example, it is critical that you `source` the telnet connection from the T1 router's loopback address. This is necessary because the 172.16.0.12/30 subnet is not being advertised within your AS, which leaves `r3` as the only router capable of routing to this prefix. Note that a connection attempt is made to two different host IDs on the 192.168.1/24 subnet. Because host ID 192.168.1.10 does not actually exist in the current test bed, the connection attempt fails and is aborted by the operator. The rationale for making two telnet attempts is to verify that each host address on the 192.168.1/24 subnet is being tracked by a unique set of counters. You confirm the correct counting behavior at `r3` with the following command:

```
[edit]
lab@r3# run show firewall prefix-action-stats filter dc-pscp prefix-action
   data-center
Filter: dc-pscp
Counters:
Name                                                Bytes              Packets
data-center-0                                           0                    0
data-center-1                                        2410                   43
data-center-2                                           0                    0
data-center-3                                           0                    0
data-center-4                                           0                    0
data-center-5                                           0                    0
data-center-6                                           0                    0
data-center-7                                           0                    0
data-center-8                                           0                    0
data-center-9                                           0                    0
data-center-10                                        120                    2
data-center-11                                          0                    0
data-center-12                                          0                    0
. . .
```

The edited display uses added highlights to call out the correct operation of per-prefix counters. Though not shown, all policer discard counters are still at 0 because the telnet application did

not generate enough traffic to instigate policer discards. To test policing, an FTP session is initiated; note that the `interface` switch is used to source the FTP session from T1's loopback address:

```
lab@T1-P1> ftp interface lo0 192.168.1.1
Connected to 192.168.1.1.
220 dc FTP server (Version 6.00LS) ready.
Name (192.168.1.1:lab):
331 Password required for lab.
Password:
230 User lab logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> mput jbun*
mput jbundle-5.2B3.1-domestic.tgz? y
200 PORT command successful.
150 Opening BINARY mode data connection for 'jbundle-5.2B3.1-domestic.tgz'.
  0% |
   |   166 KB    06:59 ETA
^C
send aborted
waiting for remote to finish abort.
226 Transfer complete.
219136 bytes sent in 4.30 seconds (49.79 KB/s)
Continue with mput? n
ftp> quit
221 Goodbye.
```

This example made use of an FTP *put* operation because a *get* function will not generate enough data to evoke policing in the T1 to the data center router direction. The presence of policer drops is verified at r3:

```
[edit]
lab@r3# ...on-stats filter dc-pscp prefix-action data-center | find policer
Policers:
Name                                                Packets
data-center-0                                             0
data-center-1                                            29
data-center-2                                             0
data-center-3                                             0
data-center-4                                             0
. . .
```

As expected, policer-induced drops are now evident at r3. A careful candidate will take some time to confirm the accuracy of their firewall filter's classification behavior by confirming that

non-TCP traffic is neither counted nor policed:

```
lab@r3> clear firewall all

lab@r3>
```

After clearing all firewall counters on r3, some traceroutes and pings are performed at the T1 router:

```
lab@T1-P1> traceroute 192.168.1.1 source 130.130.0.1
traceroute to 192.168.1.1 (192.168.1.1) from 130.130.0.1, 30 hops max,
   40 byte packets
 1  172.16.0.13 (172.16.0.13)  0.533 ms  0.293 ms  0.276 ms
 2  10.0.2.13 (10.0.2.13)  0.236 ms  0.191 ms  0.190 ms
 3  192.168.1.1 (192.168.1.1)  0.402 ms  0.328 ms  0.324 ms

lab@T1-P1> traceroute 192.168.1.1 source 130.130.0.1
traceroute to 192.168.1.1 (192.168.1.1) from 130.130.0.1, 30 hops max,
   40 byte packets
 1  172.16.0.13 (172.16.0.13)  0.418 ms  0.289 ms  0.276 ms
 2  10.0.2.13 (10.0.2.13)  0.222 ms  0.193 ms  0.189 ms
 3  192.168.1.1 (192.168.1.1)  0.396 ms  0.329 ms  0.326 ms

lab@T1-P1> ping count 3 source 130.130.0.1 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=251 time=0.510 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=251 time=0.375 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=251 time=0.412 ms

--- 192.168.1.1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.375/0.432/0.510/0.057 ms
```

After the generation of ICMP and UDP-based traffic streams, you display the PSCP counters at r3:

```
[edit]
lab@r3# run show firewall prefix-action-stats filter dc-pscp prefix-action
   data-center
Filter: dc-pscp
Counters:
Name                                                Bytes              Packets
data-center-0                                           0                    0
data-center-1                                           0                    0
data-center-2                                           0                    0
. . .
```

As expected, the PSCP have not incremented, proving that the *dc-pscp* filter is selectively acting on the TCP protocol only. These results conclude the verification of PSCP functionality at r3.

Firewall Filter Summary

JUNOS software firewall filters offer a rich set of Layer 3 and Layer 4 packet-matching criteria that can be used to match on specific traffic for the purposes of counting, policing, multi-field classification, or plain old packet filtering. Because you can apply only one firewall filter per direction on a given logical unit, firewall filters require the use of terms, unlike routing policies, which can be applied in a daisy-chain fashion.

JUNOS software firewall filters end with an implicit *deny-all* term. Failing to take this final term into account can spell disaster, especially when the intent is to write a permissive style of filter. You can override this default action with an explicit *accept-all* term as needed. It is a good idea to add logging functionality to new filters so that you can readily determine if the filter is taking unexpected actions on your traffic. Making use of the `commit confirmed` switch when committing firewall-related changes is always a good idea, because mistakes might otherwise leave you locked out of the router!

Note that using an action modifier, such as `count`, changes the corresponding term's action from implicit `discard` to implicit `accept`. If this behavior is not desired, you can add an explicit `reject` or `discard` action to the term, or you can use the `next-term` option to either filter the traffic or cause continued firewall filter evaluation.

You must apply a firewall filter to one or more router interfaces before the filter can take effect. For protection of the local RE, you should apply the filter to the router's lo0 interface. Make sure that you accommodate routing, signaling, and remote access protocols when writing a firewall filter that is intended to protect the router's RE, or be prepared to suffer the consequences! The direction in which the filter is applied can have a significant impact on the overall results, so make sure that you understand any directional constructs in the filter *before* you decide to apply a filter as input, output, or both.

M-series and T-series platforms support leaky bucket–based policing through the power of the IP II processor. Policing is used to rate limit an interface or particular protocol flows. A policer can be called from within a firewall filter, or the policer can be applied directly to an interface; the former approach is normally used when the goal is to police a particular Layer 3 or Layer 4 flow. Starting with JUNOS software release 5.6, you can make use of prefix-specific counters and policers (PSCPs) to simplify the act of gathering usage statistics, and enforcing bandwidth restrictions, on a large number of individual prefixes. Using PSCP is far simpler than trying to configure hundreds, if not thousands, of terms in a firewall filter.

Filter Based Forwarding

This section demonstrates how a JUNOS software firewall filter can be used to facilitate policy-based routing. Policy-based routing refers to the ability to route packets based on factors other than longest match lookups against the destination address. By way of example, consider a policy that requires all traffic using TCP port 80 (HTTP) to be sent over one set of links while packets not using port 80 are routed over a different set of links, even though both sets of packets might carry the same destination address.

JUNOS software refers to policy-based routing as Filter Based Forwarding (FBF). FBF is so named because it makes use of firewall filter–based packet classification to index matching traffic against a particular routing instance. Once matched to a particular routing instance, a conventional longest-match lookup is performed against the packet's destination address and matched to any routes that are present in that *particular* instance's routing table. In most cases, a FBF routing instance will be populated with one or more static routes that are designed to direct the packet out an egress interface that differs from that which would have been selected by the router's main routing instance.

To complete this configuration task, you must modify the configuration at `r4` to meet these criteria:

- Ensure that `r4` forwards all HTTP traffic received from C1 over the 10.0.2.4/30 subnet to `r3`.
- Ensure that `r4` forwards all UDP traffic received from C1 over the 10.0.2.8/30 subnet to `r5`.
- Do not alter forwarding behavior for any other traffic.

Configuring FBF

A functional FBF configuration requires the configuration of routing instances, RIB groups for interface route resolution, and a firewall filter to classify and direct traffic to the desired routing instance.

Configuring Routing Instances and RIB Groups

You begin your FBF configuration task by creating the two routing instances required by the need for specialized treatment of HTTP versus UDP traffic. The first set of commands creates a routing instance called ***`http`***, and defines a static default route identifying `r3`'s 10.0.2.5 address as the next hop:

```
[edit]
lab@r4# edit routing-instances http

[edit routing-instances http]
lab@r4# set instance-type forwarding

[edit routing-instances http]
lab@r4# set routing-options static route 0/0 next-hop 10.0.2.5
```

The completed forwarding instance is now displayed:

```
[edit routing-instances http]
lab@r4# show
instance-type forwarding;
routing-options {
    static {
        route 0.0.0.0/0 next-hop 10.0.2.5;
    }
}
```

Similar commands are now entered to define a routing instance called *udp*. Note that the next hop used for the *udp* instance's default route directs packets over the 10.0.2.8/30 subnet to r5. The completed *udp* routing instance is now shown:

```
lab@r4# show
instance-type forwarding;
routing-options {
    static {
        route 0.0.0.0/0 next-hop 10.0.2.9;
    }
```

Your next configuration step involves the use of rib-groups to install interface routes into each forwarding instance. The presence of interface routes is required in each routing instance to accommodate next hop resolution for the static default route. The following commands create an interface RIB group for the inet family called *interfaces*, and then link the *interfaces* RIB group with the main routing instance and both of the new forwarding instances on r4:

```
[edit routing-options]
lab@r4# set interface-routes rib-group inet interfaces

[edit routing-options]
lab@r4# set rib-groups interfaces import-rib [inet.0 http.inet.0 udp.inet.0]
```

In this example, the use of [] characters allowed the specification of all RIBs that should receive the interface routes using a single command line. You could have also entered each import-rib value separately if desired. Note that you should always list the main routing instance as the first import RIB, as shown in these examples. The modified routing-options stanza is displayed next with highlights added to call out new additions:

```
[edit routing-options]
lab@r4# show
interface-routes {
    rib-group inet interfaces;
}
static {
    route 10.0.200.0/24 {
        next-hop 10.0.1.102;
        no-readvertise;
    }
}
aggregate {
    route 10.0.0.0/16;
}
rib-groups {
    interfaces {
        import-rib [ inet.0 http.inet.0 udp.inet.0 ];
    }
```

```
}
autonomous-system 65412;
```

You decide to commit the changes made thus far to confirm that interface routes have been correctly installed into each routing instance:

```
[edit routing-options]
lab@r4# run show route table http

http.inet.0: 16 destinations, 16 routes (16 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[Static/5] 00:00:05
                    > to 10.0.2.5 via so-0/1/0.100
10.0.1.0/24        *[Direct/0] 00:00:05
                    > via fxp0.0
10.0.1.4/32        *[Local/0] 00:00:05
                      Local via fxp0.0
10.0.2.4/30        *[Direct/0] 00:00:05
                    > via so-0/1/0.100
10.0.2.6/32        *[Local/0] 00:00:05
                      Local via so-0/1/0.100
10.0.2.8/30        *[Direct/0] 00:00:05
                    > via so-0/1/1.0
10.0.2.10/32       *[Local/0] 00:00:05
                      Local via so-0/1/1.0
10.0.2.16/30       *[Direct/0] 00:00:05
                    > via fe-0/0/3.0
10.0.2.18/32       *[Local/0] 00:00:05
                      Local via fe-0/0/3.0
10.0.3.4/32        *[Direct/0] 00:00:05
                    > via lo0.0
10.0.4.8/30        *[Direct/0] 00:00:05
                    > via fe-0/0/1.0
10.0.4.9/32        *[Local/0] 00:00:05
                      Local via fe-0/0/1.0
10.0.4.16/30       *[Direct/0] 00:00:05
                    > via fe-0/0/2.0
10.0.4.17/32       *[Local/0] 00:00:05
                      Local via fe-0/0/2.0
172.16.0.4/30      *[Direct/0] 00:00:05
                    > via fe-0/0/0.0
172.16.0.5/32      *[Local/0] 00:00:05
```

The output relating to the *http* instance confirms the presence of interface routes, and also shows that the static default route is marked as an active route. Note that the so-0/1/0.100 interface must be present in the *http* instance *before* the static default route can be considered usable. The presence of the so-0/1/0.100 interface is highlighted in the capture. Although it is not shown, you may assume that the display associated with the *udp* instance is similar to that shown for the *http* instance, and that the *udp* instance's static default route is also active.

Modifying Firewall Filter for FBF

With the forwarding instances on r4 ready to go, you now modify the *c1-in* firewall filter to direct HTTP and UDP traffic to the corresponding forwarding instance. The modified *c1-in* firewall filter is displayed with the changes highlighted:

```
[edit]
lab@r4# show firewall filter c1-in
term 1 {
    from {
        protocol tcp;
        tcp-initial;
    }
    then {
        count c1-syns;
        next term;
    }
}
term 2 {
    from {
        source-address {
            0.0.0.0/0;
            200.200.0.0/16 except;
            172.16.0.6/32 except;
        }
    }
    then discard;
}
term 3 {
    from {
        protocol tcp;
        port 80;
    }
    then routing-instance http;
}
```

```
term 4 {
    from {
        protocol udp;
    }
    then routing-instance udp;
}
term 5 {
    then accept;
}
```

Besides the obvious changes reflected in terms *3* through *5*, note that the source address verification term has been modified to `deny` *all* source addresses, *except* those associated with site C1. The change in term *2* necessitates the addition of an *`accept-all`* term at the end of the *`c1-in`* filter to ensure that transit traffic is not erroneously filtered. Note that terms *3* and *4* are used to direct matching traffic to a particular forwarding instance. The use of `port http`, as opposed to `destination-port` or `source-port`, is intentional because your requirements state that *all* HTTP traffic received from C1 should be forwarded to `r3`. The use of the `port` keyword ensures that HTTP request and response traffic, as associated with clients and servers, will be forwarded to `r3` as required.

Verifying FBF

Once the changes are committed at `r4`, you can verify that FBF is working by comparing the forwarding path taken by traffic that is associated with the main, *`http`*, or *`udp`* routing instances. Traceroute testing makes verification of the UDP aspects of your FBF scenario easy; this is because JUNOS software uses a UDP-based version of the traceroute utility, and because traceroute exists for the purpose of identifying the forwarding path taken by a packet. You begin FBF verification by determining how `r4` normally routes packets that are addressed to T1:

```
[edit]
lab@r4# run show route 130.130.0.1

inet.0: 126882 destinations, 126897 routes (126882 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

130.130.0.0/16     *[BGP/170] 01:52:10, MED 0, localpref 100, from 10.0.3.3
                      AS path: 65222 I
                    > via so-0/1/0.100

http.inet.0: 16 destinations, 16 routes (16 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[Static/5] 00:27:26
                    > to 10.0.2.5 via so-0/1/0.100
```

```
udp.inet.0: 16 destinations, 16 routes (16 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[Static/5] 00:11:04
                    > to 10.0.2.9 via so-0/1/1.0
```

The output is quite telling. From the display, you can see that when the main routing instance is used, packets addressed to 130.130.0.1 will be sent to r3 over the so-0/1/0.100 interface. The highlights also show that, should the *udp* routing instance be consulted, the same packet will be sent to r5 using the so-0/1/1.0 interface. Assuming for the moment that the *c1-in* firewall filter is written to correctly match on and direct traffic to the correct routing instance, then all should be well for your FBF configuration. A quick traceroute from C1 confirms that you have met the UDP-related requirements for this scenario:

```
lab@c1> traceroute 130.130.0.1 source 200.200.0.1
traceroute to 130.130.0.1 (130.130.0.1) from 200.200.0.1, 30 hops max,
   40 byte packets
 1  172.16.0.5 (172.16.0.5)  0.417 ms  0.293 ms  0.279 ms
 2  10.0.2.9 (10.0.2.9)  0.351 ms  0.300 ms  0.296 ms
 3  10.0.2.2 (10.0.2.2)  0.491 ms  0.648 ms  0.709 ms
 4  130.130.0.1 (130.130.0.1)  0.581 ms  0.689 ms  0.705 ms
```

The traceroute display clearly shows that r5 is in the forwarding path for the traceroute between C1 and T1, despite the main instance on r4 showing that such a packet would normally be sent directly to r3. Verifying that HTTP traffic is being correctly shunted to r3 is possible, assuming you can generate HTTP traffic from C1. In this case, confirmation is provided by monitoring traffic on r5's at-0/2/1 interface while you attempt to load a (nonexistent) configuration file from r5 using a URL that identifies HTTP as the transport protocol. The key point here is that r4 normally forwards to r5's lo0 address using its so-0/1/1.0 interface. Therefore, the presence of HTTP traffic, as generated by C1, on r5's at-0/2/1 interface provides a strong indication that your HTTP-related FBF configuration is working within the provided guidelines. The following captures show the HTTP load request at C1, along with the traffic monitoring output seen at r5 when the command is issued at C1:

```
[edit]
lab@c1# load merge http://10.0.3.5/test
fetch: /var/home/lab/...transferring.file.........JBCVZ2/test: Operation timed out
error: file-fetch failed
load complete (1 errors)
```

The error in the file load is to be expected, considering that r5 does not provide HTTP services, and even if it did, the *test* file does not exist. Regardless of the completion status, the command should have generated HTTP traffic from C1 to 10.0.3.5, which is good enough for the verification of your FBF setup. The next capture shows the traffic monitoring output obtained at r5's at-0/2/0 interface:

```
[edit]
lab@r5# run monitor traffic interface at-0/2/1 matching tcp
```

```
verbose output suppressed, use <detail> or <extensive> for full protocol decode
Listening on at-0/2/1, capture size 96 bytes

13:57:27.239982  In IP 10.0.2.2 > 10.0.2.1: RSVP Hello Message, length: 32
                        0200 0000 45c0 0034 48d8 0000 012e 5802
                        0a00 0202 0a00 0201 1014 dee8 0100 0020
                        000c 1601 f1fa 4d19 5089 e72a 000c 8301
                        0000 0000 0000 0000
13:57:27.259454 Out IP 10.0.2.1 > 10.0.2.2: RSVP Hello Message, length: 32
. . .
13:57:30.068307  In IP 172.16.0.6.3538 > 10.0.3.5.http: S 1013026747:
   1013026747(0) win 16384 <mss 1460,nop,wscale 0,nop,nop,timestamp 10353920 0>
   (DF)
                        0200 0000 4500 003c 98f8 4000 3e06 eaa8
                        ac10 0006 0a00 0305 0dd2 0050 3c61 8fbb
                        0000 0000 a002 4000 7a10 0000 0204 05b4
                        0103 0300 0101 080a 009d fd00 0000 0000
13:57:31.570227  In IP 10.0.3.3 > 10.0.9.7: RSVP Path Message, length: 228
                        0200 0000 46c0 00fc 48f5 0000 ff2e bc10
                        0a00 0303 0a00 0907 9404 0000 1001 . . .
^C
16 packets received by filter
0 packets dropped by kernel
```

The highlighted entry confirms that the HTTP request that was generated by C1 was routed through `r3`, by virtue of its arrival at `r5`'s at-0/2/1 interface. This result, combined with the UDP-related verification steps, confirms that you have met all stipulations for the FBF configuration example.

Note that at the time of this writing, the ability to filter output from the `monitor traffic` command with the `matching` switch was not working. This is a known issue, and a problem report (pr 13559) is already on file. This issue is caused by the fact that Layer 2 headers are striped from the frames as they arrive at the I/O manager ASIC in the FPC. You can work around this issue by escaping to a shell and running tcpdump with the `-w` switch, which causes the traffic to be written to the filename specified. During this process, pseudo Layer 2 headers are added to the frames to permit decode and analysis with external protocol analysis equipment. Once written to a file, tcpdump can also be used to analyze with functional protocol filtering. You must be root to run tcpdump from the shell. The lack of a functional filter mechanism accounts for the presence of RSVP traffic (IP-based) alongside the HTTP (TCP)-based traffic, despite inclusion of the `matching tcp` argument.

Filter Based Forwarding Summary

Filter Based Forwarding accommodates policy-defined routing through the creation of specific forwarding instances that direct packets over paths that differ from the forwarding decision of the main instance. You should consider using FBF whenever you are required to route packets using criteria other than their destination address.

For proper operation, you must configure a forwarding instance under `routing-instances`, and populate this instance with one or more static routes that direct traffic to the desired next hop. You need to use RIB groups to install interface routes into each forwarding instance for next hop resolution purposes. Traffic is then directed to a particular forwarding instance through the use of firewall filter–based classification.

Note that only input firewall filters can be used for FBF; this is because the forwarding decision for a given packet is already been made by the time it encounters any output filters that may exist.

Traffic Sampling

Firewall filters are also used to identify traffic as a candidate for sampling. Sampling allows for the statistical analysis of the type and quantity of packets that are flowing across a given point in a service provider's network. Service providers can use the information gleaned from statistical analysis to derive traffic benchmarks that are later used to provide early detection of aberrant traffic patterns, which may signal a DDoS attack. Statistical analysis of traffic patterns is also used for the purposes of capacity planning and BGP peering negotiations. Sampled traffic can be stored locally on the router, or can be exported to a remote machine running Cisco's NetFlow or CAIDA's (Cooperative Association for Internet Data Analysis) cflowd and/or cfdcollect applications. For more information on the CAIDA tools, visit their website at `www.caida.org/tools/measurement/cflowd/`.

Note that conventional traffic sampling does not store the entire packet. Instead, the goal of sampling is to store enough information from the packet headers to allow the characterization of a given *flow*. A flow is identified by parameters including source and destination addresses, transport protocol, and application ports. A related JUNOS software feature called `port-mirroring` allows complete copies of sampled packets to be sent out a mirroring interface, where the mirrored traffic can be stored (and analyzed) using off-the-shelf protocol analysis equipment. Port mirroring is normally used for law enforcement wiretap applications where the entire contents of the packet must be stored for subsequent analysis.

This section provides typical JNCIE-level traffic sampling and port mirroring configuration scenarios.

Traffic Sampling

You begin this configuration scenario by modifying `r5`'s configuration in accordance with these requirements:

- Sample 1 percent of the ICMP traffic arriving at `r5`'s transit interfaces.
- Store the samples locally in a file called *`icmp-sample`*.
- Configure the sampling file to be 15MB, and allow seven archived copies.

Configuring Sampling

You begin the sampling configuration by defining a `sampling` stanza on `r5` in accordance with the requirements posed in this example:

```
[edit forwarding-options]
lab@r5# set sampling input family inet rate 100

[edit forwarding-options]
lab@r5# set sampling output file filename icmp-sample

[edit forwarding-options]
lab@r5# set sampling output file files 8

[edit forwarding-options]
lab@r5# set sampling output file size 15M
```

The resulting `sampling` stanza is now displayed:

```
[edit forwarding-options]
lab@r5# show
sampling {
    input {
        family inet {
            rate 100;
        }
    }
    output {
        file filename icmp-sample files 8 size 15m;
    }
}
```

In this example, the sampling rate has been set to 1 sample for every 100 candidates, which equates to the specified 1% sampling rate. The default `run-length` parameter of 0 is in effect, which indicates that no additional packets are sampled after a given trigger event. Overriding the default by setting the `run-length` parameter to a non-zero value results in the sampling of the specified number of consecutive packets after each sampling event is triggered. The `output` portion of the `sampling` stanza has been configured to write to a file called *`icmp-sample`*, which is allowed to be up to 15MB in length. In keeping with the provided criteria, a total of 8 files will be maintained: the 7 archived copies and the active sampling file.

Your next command set defines a firewall filter that evokes a sampling action modifier for ICMP traffic:

```
[edit]
lab@r5# edit firewall filter sample-icmp
```

```
[edit firewall filter sample-icmp]
lab@r5# set term 1 from protocol icmp

[edit firewall filter sample-icmp]
lab@r5# set term 1 then sample

[edit firewall filter sample-icmp]
lab@r5# set term 2 then accept
```

The resulting *sample-icmp* filter is displayed:

```
[edit firewall filter sample-icmp]
lab@r5# show
term 1 {
    from {
        protocol icmp;
    }
    then sample;
}
term 2 {
    then accept;
}
```

The *sample-icmp* firewall filter is rather basic. The key difference between this filter and others that have been demonstrated previously in this chapter is the use of the `sample` action modifier. Note that term 2 provides an *accept-all* action that is required to ensure that network disruption will not result from the application of the filter.

Your next command applies the *sample-icmp* filter to *all* of `r5`'s interfaces, both transit and OoB. Failing to apply the sampling filter to the router's fxp0 interface might result in exam point loss, as the instructions state that you must sample all of the traffic arriving at `r5`. Also note that the filter is applied as an input in accordance with the goal of sampling traffic that *arrives* at `r5`. Using the filter as an output results in sampling of the ICMP traffic that *egresses* `r5` and this is not the goal of this configuration exercise.

You begin by quickly confirming that no other input firewall filters exist at `r5`, because you can apply only one filter per direction per logical interface. Note that sampling does not function for traffic flowing on the router's fxp0 OoB management port. The presence of existing input filters would require modifying the existing filter to include the sampling functionality that is now required:

```
[edit]
lab@r5# show interfaces | match filter

[edit]
lab@r5#
```

The lack of output indicates that you are free to apply the *sample-icmp* filter to any of r5's interfaces without concern that you might overwrite an existing filter application. Armed with this knowledge, you apply the *sample-icmp* filter to all interfaces in use at r5:

```
[edit interfaces]
lab@r5# set at-0/2/1 unit 0 family inet filter input sample-icmp

[edit interfaces]
lab@r5# set so-0/1/0 unit 0 family inet filter input sample-icmp

[edit interfaces]
lab@r5# set fe-0/0/0 unit 0 family inet filter input sample-icmp

[edit interfaces]
lab@r5# set fe-0/0/1 unit 0 family inet filter input sample-icmp
```

To save space, the filter's application is confirmed only on the router's fe-0/0/0 interface:

```
[edit interfaces]
lab@r5# show fe-0/0/0
unit 0 {
    family inet {
        filter {
            input sample-icmp;
        }
        address 10.0.8.6/30;
    }
    family mpls;
}
```

You should commit your changes when you are satisfied that the filter has been correctly applied to all of r5's transit and OoB interfaces.

Verifying Sampling

The verification of sampling begins with confirmation of a local log file named *icmp-sample*:

```
lab@r5> show log /var/tmp/icmp-sample
# Mar  7 15:31:18
#          Dest             Src  Dest   Src Proto  TOS   Pkt  Intf    IP   TCP
#          addr            addr  port  port             len   num  frag flags
```

Note that the sampling log is located at /var/tmp, as opposed to /var/log, where most other log files are stored. The location of the sampling file can not be changed, which means you have to specify the full path when viewing its contents. At this time, the sampling file is empty. This could indicate that something is broken, or it might simply mean that no ICMP traffic is arriving at R5 to be sampled. To confirm, you generate a known number of pings that are

known to either transit or terminate at r5. You start with a traceroute that confirms r5 is in the forwarding path for the destination address chosen:

```
[edit]
lab@r4# run traceroute 10.0.9.6
traceroute to 10.0.9.6 (10.0.9.6), 30 hops max, 40 byte packets
 1  10.0.2.17 (10.0.2.17)  0.582 ms  0.457 ms  0.401 ms
 2  10.0.8.9 (10.0.8.9)  0.649 ms  0.626 ms  0.591 ms
 3  10.0.9.6 (10.0.9.6)  0.551 ms  0.528 ms  0.507 ms
```

The highlighted entry confirms r5's presence in the forwarding path between r4 and r6. Your next command generates 1000 rapid pings to r6:

```
[edit]
lab@r4# run ping rapid count 1000 10.0.9.6
PING 10.0.9.6 (10.0.9.6): 56 data bytes
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!. . .
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !!!!!!!!!!!!!!!!!!!!!!!!
--- 10.0.9.6 ping statistics ---
1000 packets transmitted, 1000 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.512/0.607/9.831/0.880 ms
```

With a known quantity of ICMP traffic hitting r5, you expect to see approximately 10 samples (1% of 1000) in the *icmp-sample* log file:

```
lab@r5> show log /var/tmp/icmp-sample
# Mar  7 15:31:18
#          Dest               Src  Dest   Src Proto  TOS   Pkt  Intf   IP   TCP
#          addr              addr  port  port              len   num frag flags
       10.0.9.6       10.0.2.18     0     0     1  0x0    84     6  0x0   0x0
      10.0.2.18        10.0.9.6     0     0     1  0x0    84     5  0x0   0x0
      10.0.2.18        10.0.9.6     0     0     1  0x0    84     5  0x0   0x0
      10.0.2.18        10.0.9.6     0     0     1  0x0    84     5  0x0   0x0
       10.0.9.6       10.0.2.18     0     0     1  0x0    84     6  0x0   0x0
      10.0.2.18        10.0.9.6     0     0     1  0x0    84     5  0x0   0x0
      10.0.2.18        10.0.9.6     0     0     1  0x0    84     5  0x0   0x0
       10.0.9.6       10.0.2.18     0     0     1  0x0    84     6  0x0   0x0
      10.0.2.18        10.0.9.6     0     0     1  0x0    84     5  0x0   0x0
       10.0.9.6       10.0.2.18     0     0     1  0x0    84     6  0x0   0x0
       10.0.9.6       10.0.2.18     0     0     1  0x0    84     6  0x0   0x0
       10.0.9.6       10.0.2.18     0     0     1  0x0    84     6  0x0   0x0
      10.0.2.18        10.0.9.6     0     0     1  0x0    84     5  0x0   0x0
       10.0.9.6       10.0.2.18     0     0     1  0x0    84     6  0x0   0x0
      10.0.2.18        10.0.9.6     0     0     1  0x0    84     5  0x0   0x0
```

```
10.0.2.18       10.0.9.6        0    0    1    0x0    84    5    0x0    0x0
10.0.2.18       10.0.9.6        0    0    1    0x0    84    5    0x0    0x0
10.0.2.18       10.0.9.6        0    0    1    0x0    84    5    0x0    0x0
10.0.2.18       10.0.9.6        0    0    1    0x0    84    5    0x0    0x0
10.0.2.18       10.0.9.6        0    0    1    0x0    84    5    0x0    0x0
10.0.9.6        10.0.2.18       0    0    1    0x0    84    6    0x0    0x0
10.0.2.18       10.0.9.6        0    0    1    0x0    84    5    0x0    0x0
10.0.9.6        10.0.2.18       0    0    1    0x0    84    6    0x0    0x0
10.0.9.6        10.0.2.18       0    0    1    0x0    84    6    0x0    0x0
10.0.9.6        10.0.2.18       0    0    1    0x0    84    6    0x0    0x0
```

The entries confirm that the sampling daemon is writing to the file, but there appears to be a fair bit more than the 10 samples expected. Closer analysis of the entries confirm why there are many more samples than initially anticipated; the ICMP echo reply traffic from `r6` back to `r4` is also hitting `r5`, which brings the total count of ICMP packets arriving at `r5` to 2000. The astute reader will note that there are in fact 25 samples, which is still slightly higher than the theoretical 20 samples that should have occurred with the configured 1% sampling rate:

```
lab@r5> show log /var/tmp/icmp-sample | match 10 | count
Count: 25 lines
```

The disparity in the sample count results from the fact that JUNOS software intentionally dithers the sampling trigger to ensure greater statistical accuracy. Put another way, the sampling rate that is configured represents the nominal sampling rate over a large number of sampling events; in other words, it is an *average* sampling rate. As with any form of statistics, the outcome becomes increasingly accurate as the sample population is increased. By way of example, after another 1000 pings are sent from `r4` to `r6`, the sampling count is now spot-on for a 1% sampling of 4000 packets:

```
lab@r5> show log /var/tmp/icmp-sample | match 10 | count
Count: 40 lines
```

The only real way to verify the archive settings for your sampling file is to fill up the logs so that you can actually confirm the size of each file along with the total file count. In this case, the log file–related portion of the configuration is pretty straightforward, so the assumption is made that you have correctly configured the sampling file size and archive count in accordance with the restrictions posed. You would need to generate a lot of ICMP traffic to fill even one sampling log file with a size of 15MB, at least with the current 1% sampling rate!

Cflowd Export

Sampled traffic can be exported to a remote host running cflowd in the form of flow records. This capability functions in a manner similar to NetFlow, as supported on Cisco routing platforms. A cflowd collector can be used to accumulate flow records from multiple cflowd collection points, with the resulting data files subjected to analysis by a variety of shareware and commercial tools.

While the installation and configuration of cflowd applications on a UNIX machine is beyond the scope of this book, a JNCIE candidate should be prepared to configure flow record export to a host that is preconfigured to support the cflowd application.

Configuring Cflowd Export

To complete this section, you must modify the configuration at r5 to meet these requirements:

- Export cflowd version 8 flow records, aggregated by protocol and port.
- Send the records to 10.0.1.201 using port 5000.

You begin by deleting the existing output portion of the sampling stanza on r5:

```
[edit forwarding-options sampling]
lab@r5# delete output
```

You now issue the commands needed to define cflowd export according to the parameters provided:

```
[edit forwarding-options sampling]
lab@r5# set output cflowd 10.0.1.201 port 5000

[edit forwarding-options sampling]
lab@r5# set output cflowd 10.0.1.201 version 8

[edit forwarding-options sampling]
lab@r5# set output cflowd 10.0.1.201 aggregation protocol-port
```

The modified output stanza is viewed:

```
lab@r5# show output
cflowd 10.0.1.201 {
    port 5000;
    version 8;
    aggregation {
        protocol-port;
    }
}
```

The output stanza on r5 correctly identifies the cflowd host, UDP port number, and the support of version 8 flow records with aggregation based on protocol and port. Satisfied with your work thus far, you commit the changes and proceed to the verification section.

Verifying Cflowd Export

Verifying cflowd export can be tricky when you do not have access to the cflowd host or an external protocol analyzer. The fact that flow records are sent about every 3 minutes also complicates your verification task. Based on the address of the cflowd host, the records should be sent out r5's fxp0 interface. You could monitor the traffic on this interface looking for a

message that is sent about every 60 seconds to host 10.0.1.201 using port 5000 to confirm that flow records are being sent. A far more effective validation technique involves the tracing of flow record export, which is configured with the following commands:

```
[edit forwarding-options sampling]
lab@r5# set traceoptions file test-cflowd

[edit forwarding-options sampling]
lab@r5# set output cflowd 10.0.1.201 local-dump
```

These commands specify the tracefile name, and indicate that cflowd records should be dumped to the tracefile configured. If a tracefile is not specified, cflowd flow records are dumped to `/var/log/sampled` by default. The modified `sampling` stanza is displayed next:

```
[edit forwarding-options sampling]
lab@r5# show
traceoptions {
    file test-cflowd;
}
input {
    family inet {
        rate 100;
    }
}
output {
    cflowd 10.0.1.201 {
        port 5000;
        version 8;
        local-dump;
        aggregation {
            protocol-port;
        }
    }
}
```

After the tracing changes are committed, you begin monitoring the *`test-cflowd`* log file. Note that the tracefile is stored in `/var/log`, which is the default location for log files, making the specification of the path unnecessary.

```
[edit forwarding-options sampling]
lab@r5# run monitor start test-cflowd
```

Note that ICMP traffic is known to be flowing through `r5` at this time due to rapid pings being generated at `r4`. After a few minutes, the following output is observed:

```
Mar  7 17:01:59 Read 6496198 bytes; total 6496198
Mar  7 17:02:03 send_cflowd: v8(PROTO_PORT_AGGREGATION) aggr and export tree empty
```

```
Mar  7 17:02:03 send_cflowd: v8: Skipping export
Mar  7 17:03:03 send_cflowd: v8 switching to tree 1 for export
Mar  7 17:03:03 Send 1 pkts every 5 secs for PROTO_PORT_AGGREGATION
Mar  7 17:03:03 v8(PROTO_PORT_AGGREGATION): starting export...
Mar  7 17:03:03 v8 flow entry
Mar  7 17:03:03    Num of flows: 2
Mar  7 17:03:03    Pkts in flow: 219
Mar  7 17:03:03    Bytes in flow: 18396
Mar  7 17:03:03    Start time of flow: 20646808
Mar  7 17:03:03    End time of flow: 20680815
Mar  7 17:03:03 Proto-port aggregation
Mar  7 17:03:03    Protocol 1
Mar  7 17:03:03    Src port 0; Dst port 0
Mar  7 17:03:03 cflowd header:
Mar  7 17:03:03   Num-records: 1
Mar  7 17:03:03   Version: 8
Mar  7 17:03:03   Flow seq num: 0
Mar  7 17:03:03   Sys Uptime: 20694314 (msecs)
Mar  7 17:03:03   Time-since-epoch: 1047056583 (secs)
Mar  7 17:03:03   Aggr version: 0
Mar  7 17:03:03   Aggr method: 2
Mar  7 17:03:03   Engine id: 0
Mar  7 17:03:03   Engine type: 0
Mar  7 17:03:03   Sample interval: 100
Mar  7 17:03:03 Sent v8(PROTO_PORT_AGGREGATION) flows (0 entries left in tree)
Mar  7 17:03:03 v8(PROTO_PORT_AGGREGATION): exporting done
Mar  7 17:03:03 v8(PROTO_PORT_AGGREGATION): export tree pruned
```

The highlights confirm that version 8 records are being sent, that `protocol-port` aggregation is in effect, and that two flows for protocol type 1 (ICMP) were detected and reported in this interval. This display, combined with the knowledge that you have specified the correct host address and UDP port, confirms the proper export of cflowd records. Before proceeding to the next section, it is suggested that you remove the cflowd tracing configuration because it is no longer needed. Leaving the tracing configuration is place does not violate any restrictions, and should cause no harm, however.

When sampling and export problems are suspected in spite of what appears to be a valid configuration, you might try bouncing the sampling daemon with a `restart sampling` command before resorting to a reboot.

Port Mirroring

Port mirroring allows for the transmission of a complete copy of sampled IPv4 packets out an interface of your choosing. Normally a data collection device is attached to this interface to provide packet storage and analysis functionality. When using an Ethernet interface for mirroring, you might need to add a static ARP entry for the IP address of the data collection device, depending on whether it can answer ARP requests. You also need to specify the data collection device's IP address under the `port-mirror` configuration stanza when using a multipoint interface for mirroring. When mirroring to a point-to-point interface that uses /32 addressing, be sure that you specify the destination address, using the `destination` keyword, when configuring the mirroring interface.

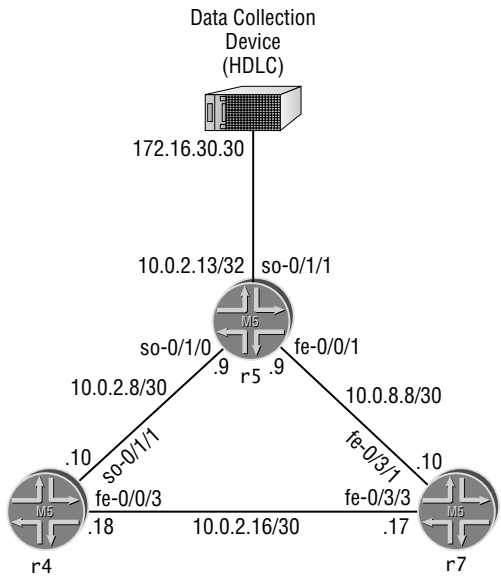
Port mirroring only works for traffic that transits the router's PFE. As a result, you can not use the `monitor traffic` command to verify port mirroring output. Also note that the mirroring interface can not have any firewall filters applied, and that mirroring currently works only for IPv4 traffic.

Configuring Port Mirroring

To complete this section, you must configure `r5` to meet the requirement listed. Figure 3.2 contains additional information that is needed to complete this assignment:

- Send sampled packets out the so-0/1/1 interface, ensuring that the data collection device receives entire packets.

FIGURE 3.2 Port mirroring topology



Although the wording of the requirement does not say "port mirroring," the need to send sampled packets, in their entirety, out a specific interface is a sure clue that you need to configure port mirroring. Figure 3.2 indicates that you need to configure `r5`'s so-0/1/1 interface to support

attachment to the data collection device. You also need to correctly configure the router's `port-mirroring` stanza to get this task behind you.

It is assumed that you will use the existing sampling configuration as a starting point for your port mirroring configuration, although a completely new sampling configuration is not precluded. In this case, you take the path of least typing by adding port mirroring functionality to the existing `sampling` stanza. You begin by deleting the current `output` configuration so the soon-to-be-added port mirroring configuration will commit; note that you can not commit a configuration containing both `cflowd` and `port-mirroring`:

```
[edit forwarding-options sampling]
lab@r5# delete output
```

The next command creates the `port-mirroring` stanza, and enables port mirroring to the so-0/1/1 interface:

```
[edit forwarding-options sampling]
lab@r5# set output port-mirroring interface so-0/1/1
```

Because the SONET interface is shown as running a point-to-point protocol in this example (HDLC), there is no need to specify a `next-hop` address under the mirroring interface's definition. A `next-hop` declaration is needed if the mirroring interface is multipoint, which would also be the case when using an Ethernet interface. The modified `sampling` stanza is displayed next:

```
[edit forwarding-options sampling]
lab@r5# show
traceoptions {
    file test-cflowd;
}
input {
    family inet {
        rate 100;
    }
}
output {
    port-mirroring {
        interface so-0/1/1.0;
    }
}
```

The next step is to configure the mirroring interface in accordance with the addressing and encapsulation information shown earlier in Figure 3.2:

```
[edit interfaces so-0/1/1]
lab@r5# set encapsulation cisco-hdlc

[edit interfaces so-0/1/1]
lab@r5# set unit 0 family inet address 10.0.2.13/32
```

The resulting configuration is displayed for confirmation:

```
[edit interfaces so-0/1/1]
lab@r5# show
encapsulation cisco-hdlc;
unit 0 {
    family inet {
        address 10.0.2.13/32;
    }
}
```

With `port-mirroring` configured under the sampling stanza, and the mirroring interface configured according to the provided diagram, you decide to commit the changes and proceed to the section on verification.

Verifying Port Mirroring

Because you can not use the `monitor traffic` command for transit traffic, and because port mirroring works only for transit traffic, you need to find another way to validate that sampled packets are being correctly sent to the data collection device. Although you could use various counters to try to confirm packet output on the so-0/1/1 interface, most operators prefer the real-time feedback of the `monitor interface` command. In this example, you once again generate rapid pings from `r4` to `r6`, noting that this ICMP traffic will transit `r5` via its fe-0/0/1 and fe-0/0/0 interfaces, while simultaneously monitoring `r5`'s so-0/1/1 interface. Figure 3.3 shows the resulting display.

FIGURE 3.3 Initial port mirroring results

```
r5                          Seconds: 32              Time: 20:38:37
                                                     Delay: 2/0/2
Interface: so-0/1/1, Enabled, Link is Down
Encapsulation: Cisco-HDLC, Keepalives, Speed: OC3
Traffic statistics:                                  Current delta
  Input bytes:                    0 (0 bps)                  [0]
  Output bytes:                 299 (88 bps)                 [69]
  Input packets:                  0 (0 pps)                  [0]
  Output packets:                13 (0 pps)                  [3]
Encapsulation statistics:
  Input keepalives:               0                          [0]
  Output keepalives:             22                          [3]
Error statistics:
  Input errors:                   0                          [0]
  Input drops:                    0                          [0]
  Input framing errors:           0                          [0]
  Input runts:                    0                          [0]
  Input giants:                   0                          [0]
  Policed discards:               0                          [0]
  L3 incompletes:                 0                          [0]
  L2 channel errors:              0                          [0]
  L2 mismatch timeouts:           0                          [0]
  Carrier transitions:            1                          [0]
  Output errors:                  0                          [0]
  Output drops:                   0  Aged packets:  2        [0]

Interface warnings:
  o Received keepalive count is zero


Next='n', Quit='q' or ESC, Freeze='f', Thaw='t', Clear='c', Interface='i'
```

The output from the `monitor interface so-0/1/1` command indicates a relative dearth of output traffic. Considering the 1% sampling rate at `r5`, the use of rapid pings, and the fact that

the ICMP traffic is being sampled in both directions, there should be more traffic coming out of r5's so-0/1/1 interface if the port mirroring configuration is working. Looking back at Figure 3.3, you happen to note that the interface is reported as being down at the link layer. Perhaps this has something to do with the reason samples are not transmitted out the mirroring interface....

Real World Scenario

Troubleshooting an Interface Problem

It is always a good idea to start an interface troubleshooting task with the determination of the lowest layer that is demonstrating symptoms of problems. In this case, it has to be either Layer 1 or Layer 2, and signs are that the interface's physical layer is up because there are no alarm indications shown in the output of the `monitor interface` command, as depicted earlier in Figure 3.3. One of the most telling symptoms in Figure 3.3 is the indication that several keepalive packets have been generated by r5, but no keepalives have been received. Noting this, you `monitor traffic` for a brief time and receive the output shown next:

```
[edit interfaces so-0/1/1]
lab@r5# run monitor traffic interface so-0/1/1
verbose output suppressed, use <detail> or <extensive> for full protocol decode
Listening on so-0/1/1, capture size 96 bytes

21:11:41.793973 Out SLARP (length: 18), keepalive: mineseen=0x00000109 yourseen=
   0x00000000 reliability=0xffff t1=543.25952
21:11:53.194099 Out SLARP (length: 18), keepalive: mineseen=0x0000010a yourseen=
   0x00000000 reliability=0xffff t1=543.36952
21:12:04.494220 Out SLARP (length: 18), keepalive: mineseen=0x0000010b yourseen=
   0x00000000 reliability=0xffff t1=543.48952
21:12:13.294316 Out SLARP (length: 18), keepalive: mineseen=0x0000010c yourseen=
   0x00000000 reliability=0xffff t1=543.56952
21:12:21.994409 Out SLARP (length: 18), keepalive: mineseen=0x0000010d yourseen=
   0x00000000 reliability=0xffff t1=544.416
```

Definitely a one-way situation with regard to the Cisco's Serial Line Address Resolution Protocol (SLARP) protocol traffic; there is no indication of incoming keepalive traffic at all. Considering that the keepalive specifics were not specified in the configuration details, the so-0/1/1 interface it attached to some type of test equipment, and there are no obvious signs of physical layer impairments, it makes sense to turn off keepalives to see if the problems clear.

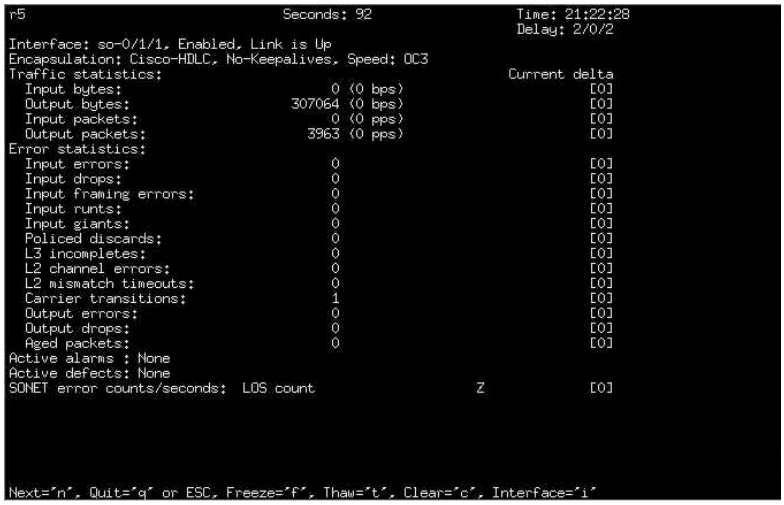
```
[edit interfaces so-0/1/1]
lab@r5# set no-keepalives

[edit interfaces so-0/1/1]
lab@r5# show
no-keepalives;
encapsulation cisco-hdlc;
```

```
unit 0 {
    family inet {
        address 10.0.2.13/32;
    }
}
```

In this case, adding `no-keepalives` causes the interface's Link layer status to display `Up`. However, without the ability to detect problems via keepalive exchanges, the link layer's `Up` indication means very little. You are still in the thick of it until you can verify that packets are being mirrored correctly.

With the so-0/1/1 interface now up at the link level, you repeat the rapid pings at `r4`, while once again monitoring the mirroring interface at `r5`. The results are shown in Figure 3.4.

FIGURE 3.4 Port mirroring results—second pass



Unfortunately, Figure 3.4 shows that there is still no significant output activity on the port mirroring interface, despite it now being declared `Up` at the link level. A quick ping test to the data collection device's IP address sheds some light on the remaining issue:

```
[edit interfaces so-0/1/1]
lab@r5# run ping 172.16.30.30
PING 172.16.30.30 (172.16.30.30): 56 data bytes
ping: sendto: No route to host
ping: sendto: No route to host
^C
```

The error message causes you to think back on the assignment of a /32 host address on the router's so-0/1/1 interface. The use of a /32 host address requires that you use the `destination` keyword to explicitly state the IP address of the neighbor at the remote end of the link before

routing can occur over the interface. Lacking any better ideas, you decide to add the destination address to r5's so-0/1/1 interface to test your hypothesis:

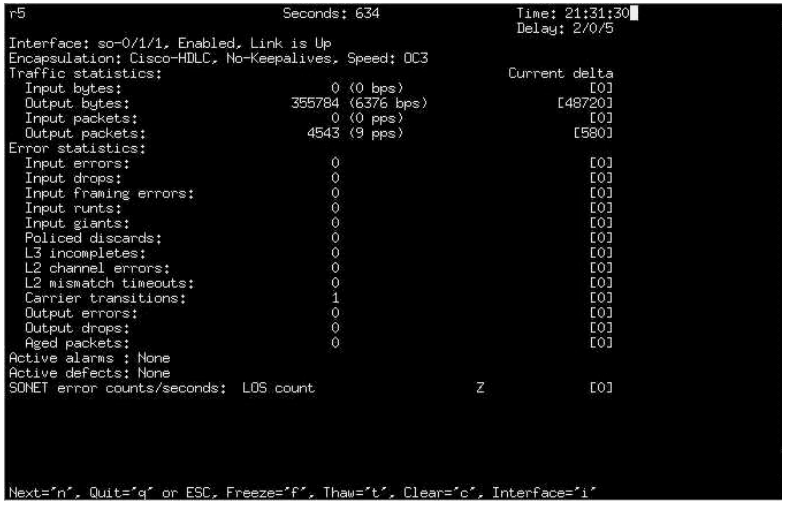
```
[edit interfaces so-0/1/1 unit 0 family inet]
lab@r5# set address 10.0.2.13/32 destination 172.16.30.30
```

The modification is displayed with added highlights:

```
[edit interfaces so-0/1/1 unit 0 family inet]
lab@r5# show
address 10.0.2.13/32 {
    destination 172.16.30.30;
}
```

After a `commit`, you once again generate rapid pings and monitor the so-0/1/1 interface at r5. The results of this run are shown in Figure 3.5.

FIGURE 3.5 Port mirroring confirmed



The results are much better now! Figure 3.5 shows that on average nine packets per second are now being sent out the mirroring interface. While you have no way to prove that the data collection device is actually accepting them, the fact that traffic counters return to zero when the pings are stopped at r4 provides a very good indication that port mirroring is now operational.

Traffic Sampling Summary

This section provided several configuration scenarios that demonstrated M-series and T-series traffic sampling and export functionality. Operational-mode commands that prove useful for verifying sampling and export functionality were also demonstrated.

Firewall filters are used to identify traffic that is a candidate for sampling. The `sampling` configuration under the `[edit forwarding-options]` hierarchy in turn determines how many candidates for sampling are actually sampled, and what is done with the sampled traffic. This

section demonstrated the local storage of sampling statistics, the use of cflowd for export of flow records to a remote host, and port mirroring. Note that you can not configure cflowd and port mirroring at the same time, and that a port mirroring configuration might require next hop identification, static ARP entries, or the use of the `destination` keyword, depending on whether the mirroring interface is point-to-point or multipoint.

Summary

JNCIE candidates should be prepared to deploy JUNOS software firewall filters that will meet any of the configuration scenarios presented in this chapter. Because incorrect application of a firewall filter can have a dramatic impact on the operational status of your network, special care must be taken whenever packet filtering enters the fray.

This chapter provided configuration and validation techniques for RE-based filters, which are designed to protect the local RE, and for transit filtering, which are used to filter traffic from a downstream device. Firewall and interface rate limiting, based on the use of policers, was also demonstrated, as was the use of Prefix Specific Counters and Policers (PSCP), which allow for the simplified configuration of counting and policing actions on individual prefixes. The chapter concluded with examples of traffic sampling, cflowd export, and port mirroring, all of which used a firewall filter to perform multi-field classifications to identify which packets are candidates for sampling.

Special attention is warranted when deciding on the direction in which a given filter will be applied, and to the use of flow-significant keywords such as `destination-port` versus `source-port` when creating your firewall filters. In many cases, the difference between perfection and exam point loss comes down to the specific choice of matching keyword or the direction in which the filter was applied.

The default firewall filter action of denying all traffic not explicitly permitted by previous terms often results in trouble for a less-than-prepared candidate. In many cases, the wording of your task will not explicitly warn of the need to support some routing protocol, or a given remote access method. It is assumed in these cases that the candidate understands that the application of a source address validation filter to prevent spoofed packets should not disrupt existing network operation. Even when the candidate is aware of this assumption, the large number of routing, signaling, utility, and remote access protocols present in a typical JNCIE test bed often result in errors of omission, and a resulting impact to that protocol or service. We all make mistakes; adding a log action to a new firewall can help identify any mistakes, thereby allowing the candidate to take corrective actions before the proctor gets around to grading the overall operation of your test bed.

Although not exhaustive, Table 3.1 identifies typical routing, utility, signaling, and remote access protocols that may need support within your firewall filter. The table lists only the well-known ports and the primary transport protocol associated with the server process. Note that not all of these have human-friendly matching keywords. In some instances, the candidate is expected to reference outside documents, such as user manuals or RFCs, to determine which port/protocol is used to support a given service. Services that ride directly within IP, such as OSPF and VRRP, are identified in the table with a PID (Protocol ID) designation, as they are not

associated with any transport-level protocol ports. Many of the following values are identified in the now-obsolete Assigned Numbers RFC (RFC 1700). Currently, assigned numbers are tracked in an online database located at `www.iana.org/numbers.html`.

TABLE 3.1 Common Protocol and Port Assignments

Service	Protocol	Port(s)/Protocol ID
Telnet	TCP	23
FTP	TCP	20,21
SSH	TCP	22
DNS	UDP/TCP	53
BOOT/DHCP	UDP	67 (server) 68 (client)
HTTP	TCP	80
RADIUS	UDP	1812
TACACS +	TCP	49
NTP	UDP	123
VRRP	IP	PID 112
RSVP	IP	PID 46
LDP	UDP/TCP	646
PING	IP	PID 1
Traceroute	UDP	Varies
OSPF	IP	PID 89
BGP	TCP	179

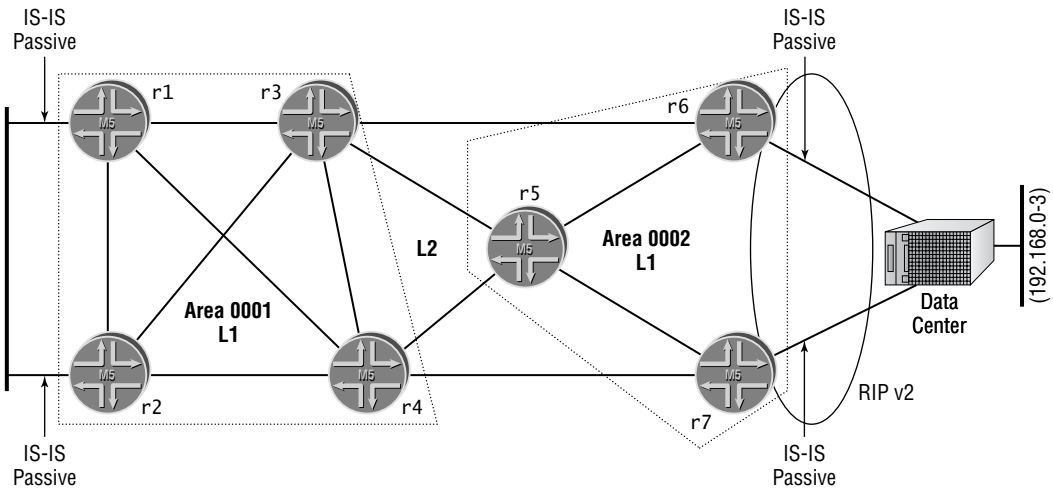
You can also view common protocol/port-to-application mappings using the JUNOS software CLI to view the `/etc/services` file. You might want to evoke the CLI match functionality due to the size of the `services` file:

```
[edit]
lab@r3# run file show /etc/services | match ssh
ssh             22/tcp    #Secure Shell Login
ssh             22/udp    #Secure Shell Login
```

Case Study: Firewall Filter and Traffic Sampling

This chapter case study is designed to simulate a typical JNCIE-level firewall filtering and traffic sampling configuration scenario. To keep things interesting, you will be performing your firewall filtering case study using the IS-IS baseline configuration that was discovered and documented in the Chapter 1 case study. The IS-IS baseline topology is shown in Figure 3.6 so that you may reacquaint yourself with it.

FIGURE 3.6 IS-IS discovery findings



Notes:

Multi-level IS-IS, Areas 0001 and 0002 with ISO NET based on router number.

lo0 address of `r3` and `r4` not injected into Area 0001 to ensure optimal forwarding between 10.0.3.3 and 10.0.3.4.

Passive setting on `r5`'s core interfaces for optimal Area 0002-to-core routing.

No authentication or route summarization. Routing policy at `r5` to leak L1 externals (DC routes) to L2.

Redistribution of static default route to data center from both `r6` and `r7`. Redistribution of 192.168.0/24 through 192.168.3/24 routes from RIP into IS-IS by both `r6` and `r7`.

All adjacencies are up, reachability problem discovered at `r1` and `r2` caused by local aggregate definition. Corrected through IBGP policy to effect 10.0/16 route advertisement from `r3` and `r4` to `r1` and `r2`; removed local aggregate from `r1` and `r2`.

Suboptimal routing detected at the data center and at `r1`/`r2` for some locations. This is the result of random nexthop choice for data center's default, and the result of `r1` and `r2`'s preference for `r3`'s RID over `r4` with regard to the 10.0/16 route. This is considered normal behavior, so no corrective actions are taken.

You should load and commit the IS-IS baseline configuration to make sure that your routers look and behave like the examples shown here. Before starting the firewall filtering case study, it is suggested that you quickly verify the correct operation of the baseline network's IS-IS IGP, RIP redistribution, and IBGP/EBGP peerings. Problems are not expected in the baseline network

at this stage, but it never hurts to make sure that you are starting off with a functional network. Note that MPLS functionality is not present in the IS-IS baseline topology, as was discovered in the Chapter 1 case study. The absence of MPLS means that, in contrast to the examples provided in the chapter body, you do not have to worry about filter support for the RSVP and LDP signaling protocols in the case study.

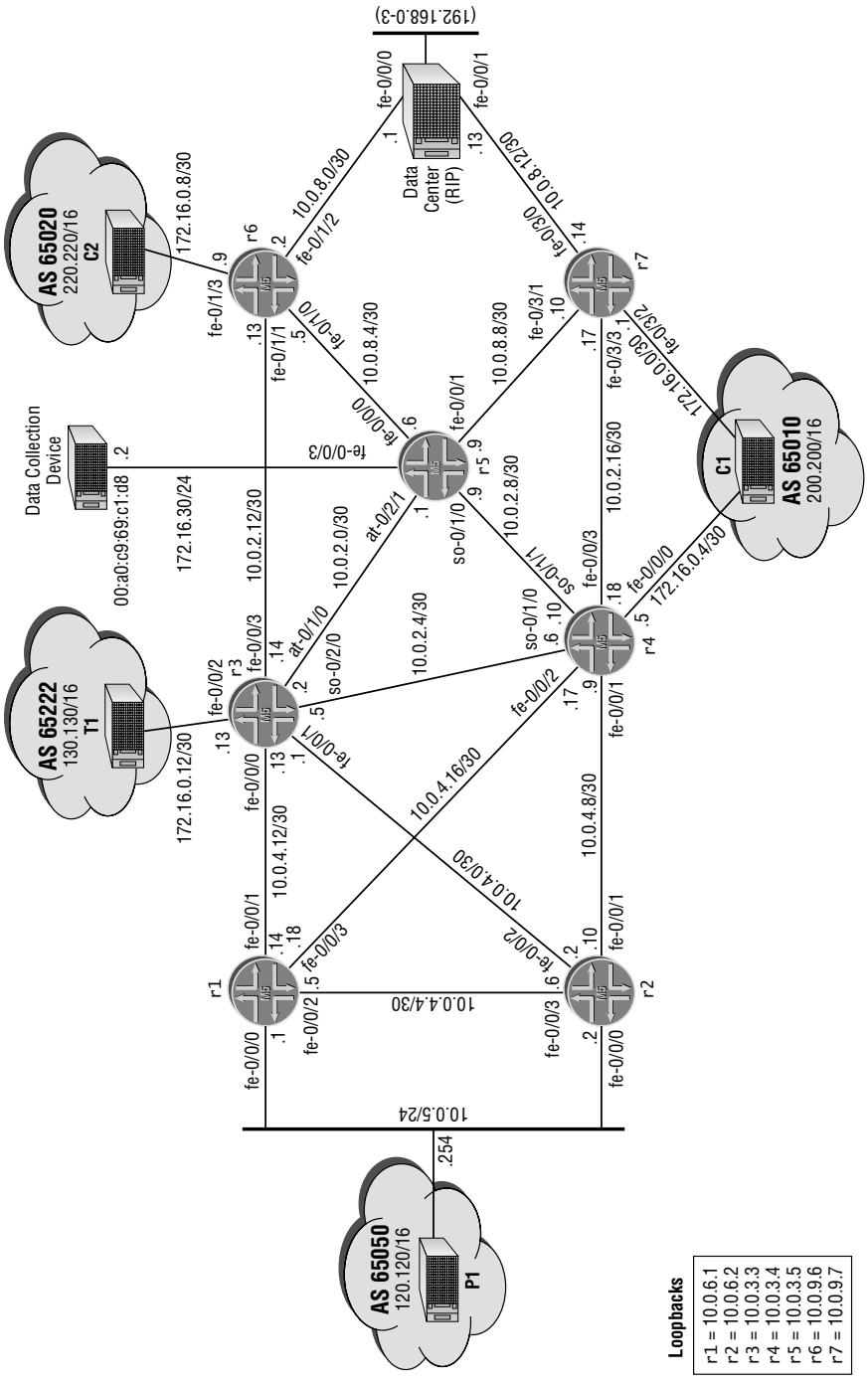
You need to refer to the case study criteria listing and the case study topology, shown in Figure 3.7, for the information needed to complete the firewall filter and traffic sampling case study. It is expected that a JNCIE candidate will be able to complete this case study in approximately one and one-half hours, with the result being a network that meets the majority of the provided criteria with no significant operational problems.

Configuration listings that identify all changes made to the baseline network for all seven routers in the test bed are provided at the end of the case study for comparison with your own configurations. Because multiple solutions may be possible for a given task, differences between the examples provided and your own configurations do not automatically indicate that mistakes have been made. Recall that in the JNCIE lab you are graded on the overall functionality of your network and on its conformity to all specified criteria. To accommodate differing configuration approaches, various operational mode commands are included in the case study analysis so that you can compare the behavior of your network to that of a known good example.

To complete the case study, your network must be configured to meet these criteria:

- Your firewall filtering case study configuration must be added to the IS-IS based Chapter 1 case study topology.
- Configure a filter on `r5` according to these stipulations:
  - Rate limit all ICMP messages to 500Kbps.
  - Limit incoming telnet and SSH sessions to sources within AS 65412, including the data center prefixes. Ensure that incoming connection attempts from external addresses are logged to a file called `access-log`.
  - Allow `r5` to initiate telnet, SSH, and FTP connections to all destinations.
  - Do not disrupt any existing services or applications while preventing unknown traffic from reaching the RE.
- Prevent source address spoofing at all customer peering points. Count and log any spoofed packets.
- Rate limit HTTP *responses* sent from `r4` to C1 in accordance with the following:
  - Accept all traffic at or below 1Mbps.
  - Mark traffic in excess of 1Mbps for local discard in the event of output congestion.
  - Discard traffic in excess of 2Mbps.
- Rate limit all traffic arriving from T1 to 5% of the peering interface's bandwidth.
- Count and police traffic sent to *all* host IDs on the 192.168.0/24 subnet at `r5`, according to these restrictions:
  - Count and police traffic originated from external sources that is directed at FTP and web servers only.

FIGURE 3.7 Firewall filtering case study



  - Police FTP to 500Kbps.
  - Police HTTP to 2Mbps.
  - Configure one policer/counter for every four consecutive host addresses. Ensure that you do not create more policer/counter pairs than are required by the parameters specified.
- Configure `r3` and `r4` to forward all HTTP and FTP request traffic received from external sources that is addressed to the 192.168.0/24 subnet to `r5`.
- Configure `r5` to perform the following:
  - Sample 50% of the TCP connection requests received on its so-0/1/0 interface.
  - Ensure that three additional samples are taken for each trigger.
  - Forward complete packets for all sampled traffic to the data collection device.

Assume that the data center router has been correctly reconfigured to advertise the 192.168.0-3/24 routes to both `r6` and `r7` using RIP. Please refer back to the Chapter 1 case study for specifics on the IS-IS and RIP based route redistribution in the IS-IS baseline network as needed.

Firewall Filtering Case Study Analysis

Each configuration requirement for the case study will now be matched to one or more valid router configurations and, where applicable, the commands that are used to confirm that the network is operating within the specified case study guidelines. We begin with this criterion because it serves to establish your baseline network:

- Your firewall case study configuration must be added to the IS-IS baseline topology from the Chapter 1 case study.

The case study analysis begins with a quick operational sanity check of the IS-IS baseline network. It is assumed that the corresponding baseline configurations have been loaded up and committed at this time:

```
[edit]
lab@r3# run show isis adjacency
Interface             System         L State       Hold (secs) SNPA
at-0/1/0.0            r5             2 Up                   25
fe-0/0/0.0            r1             1 Up                    8  0:a0:c9:6f:7b:3e
fe-0/0/1.0            r2             1 Up                    6  0:a0:c9:6f:7a:ff
fe-0/0/3.0            r6             2 Up                    8  0:d0:b7:3f:af:73
so-0/2/0.100          r4             2 Up                   23

[edit]
lab@r3# run show route protocol isis | match /32
10.0.3.4/32        *[IS-IS/18] 00:03:59, metric 10
10.0.3.5/32        *[IS-IS/18] 00:04:05, metric 10
10.0.6.1/32        *[IS-IS/15] 00:04:13, metric 10
```

```
10.0.6.2/32        *[IS-IS/15] 00:04:10, metric 10
10.0.9.6/32        *[IS-IS/18] 00:04:04, metric 10
10.0.9.7/32        *[IS-IS/18] 00:02:37, metric 20
```

This output confirms that all of r3's IS-IS adjacencies have been successfully established, and that the router has received an IS-IS route for the loopback address of all other routers in the test bed. These findings provide a strong indication that the baseline network's IS-IS IGP is operational. You now verify BGP session status at r3:

```
[edit]
lab@r3# run show bgp summary
Groups: 5 Peers: 7 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
inet.0            126807     126796          0          0          0          0
Peer             AS  InPkt  OutPkt  OutQ  Flaps Last Up/Dwn State|#Active/
                                                    Received/Damped...
172.16.0.14   65222 27858   28201     0      0      2:50:31 126792/126792/0  0/0/0
10.0.3.4      65412    11   24775     0      0         4:18 2/2/0            0/0/0
10.0.3.5      65412    10   22083     0      0         4:04 0/0/0            0/0/0
10.0.6.1      65412   332   82893     0      1         4:28 1/1/0            0/0/0
10.0.6.2      65412    11   26245     0      0         4:34 0/1/0            0/0/0
10.0.9.6      65412    11   22087     0      0         4:07 1/5/0            0/0/0
10.0.9.7      65412     8   24866     0      2         2:46 0/6/0            0/0/0
```

All of r3's IBGP and EBGP sessions are established, providing further indication that the IS-IS baseline network is operating normally. You now quickly confirm the presence of the RIP and EBGP peer routes:

```
[edit]
lab@r3# run show route 120.120/16

inet.0: 126830 destinations, 126841 routes (126830 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

120.120.0.0/16     *[BGP/170] 00:05:27, MED 0, localpref 100, from 10.0.6.1
                      AS path: 65050 I
                    > to 10.0.4.14 via fe-0/0/0.0
                      to 10.0.4.2 via fe-0/0/1.0
                    [BGP/170] 00:05:31, MED 0, localpref 100, from 10.0.6.2
                      AS path: 65050 I
                    > to 10.0.4.14 via fe-0/0/0.0
                      to 10.0.4.2 via fe-0/0/1.0

[edit]
lab@r3# run show route 192.168.0/24
```

```
inet.0: 126851 destinations, 126862 routes (126851 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

192.168.0.0/24     *[IS-IS/165] 00:06:17, metric 12
                    > to 10.0.2.13 via fe-0/0/3.0
                    [BGP/170] 00:05:35, MED 2, localpref 100, from 10.0.9.6
                      AS path: I
                    > to 10.0.2.13 via fe-0/0/3.0
                    [BGP/170] 00:04:15, MED 2, localpref 100, from 10.0.9.7
                      AS path: I
                    > to 10.0.2.1 via at-0/1/0.0
                      to 10.0.2.6 via so-0/2/0.100
```

Though not shown here for reasons of brevity, you may assume that the 200.200/16 and 220.220/16 customer routes and all T1 routes are also confirmed present on `r3`. With knowledge that the IS-IS baseline network is operational, you begin your firewall case study by addressing the configuration changes needed to meet the following case study requirement:

- Configure a filter on `r5` according to these stipulations:
  - Rate limit all ICMP messages to 500Kbps.
  - Limit incoming telnet and SSH sessions to sources within AS 65412, including the data center prefixes. Ensure that incoming connection attempts from external addresses are logged to a file called *`access-log`*.
  - Allow `r5` to initiate telnet, SSH, and FTP connections to all destinations.
  - Do not disrupt any existing services or applications while preventing unknown traffic from reaching the RE.

The changes made to `r5`'s configuration in support of this firewall filter requirement are displayed next:

```
[edit]
lab@r5# show firewall
policer limit-icmp {
    if-exceeding {
        bandwidth-limit 500k;
        burst-size-limit 15k;
    }
    then discard;
}
filter lo0 {
    term police-icmp {
        from {
            protocol icmp;
        }
```

```
        then policer limit-icmp;
    }
    term limit-access {
        from {
            source-address {
                10.0.0.0/16;
                192.168.0.0/21;
            }
            protocol tcp;
            destination-port [ ssh telnet ];
        }
        then accept;
    }
    term access-log {
        from {
            source-address {
                0.0.0.0/0;
            }
            protocol tcp;
            destination-port [ 22 23 ];
        }
        then {
            syslog;
            discard;
        }
    }
    term outgoing-tcp-services {
        from {
            protocol tcp;
            source-port [ 22 23 20 21 ];
        }
        then accept;
    }
    term outgoing-udp-services {
        from {
            protocol udp;
            source-port [ 1024-65535 1812 ];
        }
        then accept;
    }
    term bgp {
```

```
        from {
            protocol tcp;
            port bgp;
        }
        then accept;
    }
    term else {
        then {
            log;
            discard;
        }
    }
}
```

The policer's burst size was not specified in this example; therefore the *limit-icmp* policer has been configured with the recommended burst size for low-speed interfaces of ten times the interface's (lo0) 1500-byte MTU. Note that for a high-speed interface, such as an OC-192, the recommended burst size is a function of interface bandwidth times burst duration, where burst duration is normally in the range of 3–5 milliseconds. Next is a quick breakdown of the functionality supported by each of the terms in the *lo0* filter:

- The *police-icmp* term functions to match on, and then direct all ICMP traffic to, the *limit-icmp* policer.
- The *limit-access* term is written to accept incoming TCP traffic from internal and data center destinations destined for the telnet and SSH services. Note that a `destination-port` match condition is used to support incoming connections to local services, and that symbolic keywords are used in lieu of numeric port numbers.
- The *access-log* term matches all other incoming telnet and SSH connection attempts and associates them with a `syslog` action modifier and a `discard` action. This term works in conjunction with syslog modifications (shown next) to support the requirement that you log unauthorized access attempts.
- The *outgoing-tcp-services* term accepts return traffic associated with TCP connections that are initiated to allowed services. Note that the term uses a `source-port` match condition, which is needed to correctly match traffic associated with a TCP connection initiated by `r5` to a remote service.
- The *outgoing-udp-services* term functions in the same manner as the *outgoing-tcp-services* term, with the exception of its UDP protocol and UDP service–related match criteria.
- The *bgp* term allows TCP connections to and from the port (179) associated with the BGP protocol. The use of the *port* keyword provides bidirectional (incoming and outgoing) BGP session support.
- The *else* term functions to match on all remaining traffic for discard and logging to the kernel cache. This term is not a requirement, but its inclusion will make filter verification and troubleshooting much easier.

Note that all terms in the filter make use of a protocol test condition to avoid the potential for inadvertent matches. To support the logging action of the *access-log* term, you must modify r5's syslog stanza to write firewall entries to the specified log file, as is shown here with added highlights:

```
[edit]
lab@r5# show system syslog
user * {
    any emergency;
}
file messages {
    any notice;
    authorization info;
}
file r5-cli {
    interactive-commands any;
    archive files 5;
}
file access-log {
    firewall info;
}
```

The *lo0* filter is applied to r5's loopback interface, in the input direction, which is in keeping with the goal of protecting the local RE:

```
[edit]
lab@r5# show interfaces lo0
unit 0 {
    family inet {
        filter {
            input lo0;
        }
        address 10.0.3.5/32;
    }
    family iso {
        address 49.0002.5555.5555.5555.00;
    }
}
```

Verification of the RE protection filter begin with analysis of the cache log to quickly determine if valid traffic is being filtered:

```
lab@r5> show firewall log

lab@r5>
```

The lack of entries is a very good sign because it indicates that the filter has been written to correctly accommodate all valid services. Note that firewall filters do not function for the `iso` family, and therefore no actions are necessary to support the IS-IS IGP used by the test bed. To verify that the filter is preventing unauthorized services, you attempt an FTP session from `r5` to `r5` and examine the firewall cache:

```
lab@r5> ftp 10.0.3.5
^C
lab@r5> show firewall log
Log :
Time      Filter    Action Interface  Protocol Src Addr         Dest Addr
13:45:18  lo0       D      local      TCP      10.0.3.5         10.0.3.5
```

The TCP entry in the cache indicates that FTP was blocked, and also confirms the *`else`* term's logging functions are working. The next set of commands verifies incoming telnet connections from internal sources, and also shows that `r5` can initiate telnet sessions to all destinations:

```
[edit]
lab@r6# run telnet 10.0.3.5
Trying 10.0.3.5...
Connected to 10.0.3.5.
Escape character is '^]'.

r5 (ttyp2)

login: lab
Password:
Last login: Sun Mar  9 13:49:08 from 10.0.8.5

--- JUNOS 5.6R2.4 built 2003-02-14 23:22:39 UTC
```

The connection request from an internal host is honored. Now logged into `r5`, you attempt a telnet session to an external destination:

```
lab@r5> telnet 200.200.0.1
Trying 220.220.0.1...
Connected to 220.220.0.1.
Escape character is '^]'.

C1 (ttyp0)

login: lab
Password:
Last login: Fri Mar  7 13:15:15 from 172.16.0.5
```

```
--- JUNOS 5.6R2.4 built 2003-02-14 23:22:39 UTC

lab@c1>
```

The outgoing connection works as required. Now logged into C1, you attempt to telnet back to r5. This connection should fail with an entry being written to the syslog at r5:

```
lab@c1> telnet source 200.200.0.1 10.0.3.5
Trying 10.0.3.5...
^C
lab@c1>
```

The telnet connection initiated by an external source fails as required, and back at r5, the syslog entry is confirmed:

```
lab@r5> show log access-log
Mar 9 13:53:38 r5 clear-log[1348]: logfile cleared
Mar  9 13:53:46 r5 feb FW: so-0/1/0.0   D  tcp 200.200.0.1 10.0.3.5  2918    23
   (1 packets)
```

Although it is not shown here, you may assume that SSH and FTP functionality is also confirmed with the approach shown for telnet verification. Next, you verify the UDP-based traceroute and RADIUS applications:

```
lab@r5> traceroute 130.130.0.1
traceroute to 130.130.0.1 (130.130.0.1), 30 hops max, 40 byte packets
 1  10.0.2.2 (10.0.2.2)  1.371 ms  1.030 ms  1.139 ms
 2  130.130.0.1 (130.130.0.1)  1.091 ms  1.018 ms  1.170 ms
```

The output confirms that traceroutes are working. To test RADIUS, you could verify RADIUS transactions by monitoring the traffic on the OoB network while authenticating a user at r3, or you can take the approach shown next. This technique disables automatic consultation of the local password database for authentication when the RADIUS server becomes unreachable. Note that `confirmed` is used when the changes are committed; it never hurts to play things safe when dealing with your access to a router:

```
lab@r5> configure
Entering configuration mode

[edit]
lab@r5# delete system authentication-order password

[edit]
lab@r5# commit confirmed 5
commit confirmed will be automatically rolled back in 5 minutes unless confirmed
commit complete

[edit]
```

```
lab@r5# quit
Exiting configuration mode

lab@r5> quit


r5 (ttyp0)


login: lab
Password:
Last login: Sun Mar  9 13:49:12 from 10.0.8.5

--- JUNOS 5.6R2.4 built 2003-02-14 23:22:39 UTC

lab@r5>
```

The ability to log in to `r5`, without being prompted for your local password, confirms that the *lo0* filter adequately supports the RADIUS protocol. The following capture shows the effects of a filter that does not support RADIUS in the context of the current `authentication-order` settings. The highlight calls out the local password prompt, which results from the inability to authenticate through RADIUS; note this prompt does not occur in the previous capture, which proves that RADIUS is supported by the *lo0* filter:

```
lab@r5> quit


r5 (ttyp0)


login: lab
Password:
Local password:
Last login: Thu Mar 13 14:59:33 from 10.0.1.100

--- JUNOS 5.6R2.4 built 2003-02-14 23:22:39 UTC

lab@r5>
```

Make sure that you return the baseline configuration to its original state now that RADIUS functionality is confirmed. The use of `confirmed` when issuing the `commit` helps to ensure that temporary changes such as these will in fact be temporary! The final *lo0* filter confirmation involves verification of BGP protocol support. Because no BGP traffic is showing up in the cache

file, the chances that BGP has been adversely affected by your filter are virtually nil, but it never hurts to be sure.

```
[edit]
lab@r5# run show bgp summary
Groups: 1 Peers: 6 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
inet.0            126865     126854          0          0          0          0
Peer         AS  InPkt  OutPkt  OutQ  Flaps  Last Up/Dwn State|#Active/Received/
                                                                Damped...
10.0.3.3  65412  24144     223     0      0     1:50:19 126850/126850/0    0/0/0
10.0.3.4  65412    222     223     0      0     1:50:29 2/2/0              0/0/0
10.0.6.1  65412    221     223     0      0     1:50:27 1/1/0              0/0/0
10.0.6.2  65412    221     223     0      0     1:50:23 0/1/0              0/0/0
10.0.9.6  65412    223     223     0      0     1:50:14 1/5/0              0/0/0
10.0.9.7  65412    220     220     0      0     1:48:50 0/6/0              0/0/0
```

The quick assessment that all BGP sessions are still in the established state at `r5` quickly belies any remaining concerns for BGP, and serves to complete the verification of the RE protection case study task. The next case study requirement to address is as follows:

- Prevent source address spoofing at all customer peering points. Count and log any spoofed packets.

It might be argued that, because you are already on `r5`, you should attempt to group together all case study tasks involving configuration changes at `r5` in an effort to minimize the number of times that you must visit the router. While this is a valid philosophy, this author finds that with telnet sessions always open to all routers, the amount of time lost by having to revisit a given router numerous times in the same scenario is minimal. Time issues aside, the remaining requirements for `r5` are somewhat complex, so moving on to some of the more straightforward requirements, in an effort to get as many points as possible in the event that you can not complete a difficult task, is a testing strategy that also has merit.

This configuration task requires that you apply an input filter on all customer-facing interfaces to discard any traffic not using a source address that is assigned to that customer's site. The filter shown next functions correctly for the `r4`–C1 and `r7`–C1 peering points by virtue of the /29 netmask, which is designed to match the source address of packets originated from either of C1's EBGP peering interfaces:

```
[edit firewall filter no-spoof]
lab@r4# show
term 1 {
    from {
        source-address {
```

```
            200.200.0.0/16;
            172.16.0.0/29;
        }
    }
    then accept;
}
term 2 {
    then {
        count spoofs;
        discard;
    }
}
```

Because the `count` action modifier changes the default action of the term to accept, an explicit `discard` action is needed for proper operation. Using a `reject` action is a mistake when dealing with packets that have bogus addresses, because the generation of reject messages can result in a *smurf*-like attack when they are sent to the legitimate (and innocent) owner of the spoofed address. Term *2* in this example has no match criteria, and therefore matches everything not accepted by the first term. The *no-spoof* filter is applied as an input filter for both C1 peering interfaces. The example shown here displays the correct filter application for the context of `r4`:

```
[edit]
lab@r4# show interfaces fe-0/0/0
unit 0 {
    family inet {
        filter {
            input no-spoof;
        }
        address 172.16.0.5/30;
    }
}
```

Note that similar firewall filter functionality is needed at `r6` for the C2 peering. You need to modify the filter's match criteria to reflect the 172.16.0.8/30 address used on the `r6`-C2 peering interface, however.

Unless you have the ability to reconfigure the C1 and C2 routers, it will be difficult to confirm the behavior of the *no-spoof* filter in the presence of spoofed packets. This is because you need to assign a non-customer-owned address to one of their interfaces before you can use the `source` switch to generate packets with "spoofed" address. In this example, the match conditions are pretty straightforward, so confirmation is limited to visual inspection of the configuration and verification that the *no-spoof* filter is not causing any operational problems on customer peering links.

```
[edit]
lab@r4# run traceroute 200.200.0.1 source 10.0.3.4
```

```
traceroute to 200.200.0.1 (200.200.0.1) from 10.0.3.4, 30 hops max,
   40 byte packets
 1  200.200.0.1 (200.200.0.1)  0.794 ms  0.495 ms  0.430 ms

[edit]
lab@r4# run ping 200.200.0.1
PING 200.200.0.1 (200.200.0.1): 56 data bytes
64 bytes from 200.200.0.1: icmp_seq=0 ttl=255 time=0.682 ms
64 bytes from 200.200.0.1: icmp_seq=1 ttl=255 time=0.506 ms
^C
--- 200.200.0.1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.506/0.594/0.682/0.088 ms
```

The output confirms that traceroutes and pings are unaffected by the firewall filter. BGP session status is now confirmed:

```
[edit]
lab@r4# run show bgp neighbor 172.16.0.6
Peer: 172.16.0.6+2023 AS 65010 Local: 172.16.0.5+179  AS 65412
  Type: External    State: Established    Flags: <>
  Last State: OpenConfirm   Last Event: RecvKeepAlive
  Last Error: None
  Export: [ ebgp-out ]
  Options: <Preference HoldTime AdvertiseInactive PeerAS Refresh>
  Holdtime: 90 Preference: 170
  Number of flaps: 0
  Peer ID: 200.200.0.1      Local ID: 10.0.3.4         Active Holdtime: 90
  Keepalive Interval: 30
  Local Interface: fe-0/0/0.0
  NLRI advertised by peer: inet-unicast
  NLRI for this session: inet-unicast
  Peer supports Refresh capability (2)
  Table inet.0 Bit: 10000
    RIB State: BGP restart is complete
    Send state: in sync
    Active prefixes:            2
    Received prefixes:          2
    Suppressed due to damping:  0
  Last traffic (seconds): Received 3    Sent 3    Checked 13
  Input messages:  Total 82792  Updates 82785   Refreshes 0     Octets 7508078
  Output messages: Total 52959  Updates 52950   Refreshes 0     Octets 4903028
  Output Queue[0]: 0
```

The final check is to display the spoofed address counter, which should be *0* in the absence of spoofed packets:

```
[edit]
lab@r7# run show firewall
Filter: no-spoof
Counters:
Name                                                Bytes              Packets
spoofs                                                  0                    0
```

Though not shown, similar commands and techniques are used to quickly verify BGP session status and traffic forwarding at the r6–C2 and r7–C1 peering points. Assuming that you obtain results similar to those shown here for the r4–C1 peering, you can consider the address spoofing aspect of the case study confirmed. This brings the following requirement to the top of your case study processing heap:

- Rate limit HTTP *responses* sent from r4 to C1 according to these requirements:
  - Accept all traffic at or below 1Mbps.
  - Mark traffic in excess of 1Mbps for local discard in the event of output congestion.
  - Discard traffic in excess of 2Mbps.

The policing behavior described here requires that you evoke the services of two distinct policers for the *same* traffic. To accomplish this task, you need to define the two policers and craft a firewall filter that uses the `next term` action modifier to allow the same traffic to be processed by consecutive filter terms. The completed policers are displayed first:

```
[edit]
lab@r4# show firewall policer 1m-http
if-exceeding {
    bandwidth-limit 1m;
    burst-size-limit 15k;
}
then loss-priority high;

[edit]
lab@r4# show firewall policer 2m-http
if-exceeding {
    bandwidth-limit 2m;
    burst-size-limit 15k;
}
then discard;
```

Followed by the firewall filter that will evoke them:

```
[edit]
lab@r4# show firewall filter limit-http
term 1 {
```

```
    from {
        protocol tcp;
        source-port 80;
    }
    then {
        policer 1m-http;
        next term;
    }
}
term 2 {
    from {
        protocol tcp;
        source-port 80;
    }
    then policer 2m-http;
}
term 3 {
    then accept;
}
```

The highlights call out the differing actions associated with the two new policers, and the key aspects of the *limit-http* firewall filter that calls them. The use of `source-port` based matching correctly identifies only HTTP response traffic that is being sent back to clients within C1's network. Term *1* evokes the *1m-http* policer for matching traffic, which returns *all* traffic back to term *1* with a loss priority bit setting of either *0* (low priority) for in-profile traffic or *1* for traffic in excess of the policer's limits. This makes the presence of the `next term` action modifier critical, because without it, all matching will be accepted by term *1*, defeating the second level of policing and causing exam point loss. Note that a final term has been added to ensure that all other traffic is passed out the fe-0/0/0 interface unimpeded.

The *limit-http* filter is applied as output to r4's fe-0/0/0 interface:

```
[edit]
lab@r4# show interfaces fe-0/0/0
unit 0 {
    family inet {
        filter {
            input no-spoof;
            output limit-http;
        }
        address 172.16.0.5/30;
    }
}
```

Verifying the correct operation of the *limit-http* filter and its related policers will be difficult without access to a web server and/or traffic generation equipment. Note that the

confirmation of accurate policing is almost impossible when relying on user applications, due to the presence of implicit back-off and other congestion control mechanisms. Assuming that you lack the ability to surf the Web from site C1, and that you do not have access to traffic generation equipment, validation is limited to the simple confirmation that the *limit-http* filter has not broken anything between r4 and C1:

```
[edit]
lab@r4# run traceroute 200.200.0.1
traceroute to 200.200.0.1 (200.200.0.1), 30 hops max, 40 byte packets
 1  200.200.0.1 (200.200.0.1)  0.647 ms  0.501 ms  0.431 ms

[edit]
lab@r4# run show bgp summary | match 172.16.0.6
172.16.0.6    65010    34496    26501    0    2     1:44 2/2/0       0/0/0
```

When you are really desperate to prove that a filter, policer, or counter functions correctly for an application that is not available in the JNCIE test bed, you might consider modifying the filter to match on traffic that can be empirically tested from within the lab. By way of example, you could change the *limit-http* filter's match criteria to also include ICMP, which is a protocol that can easily be generated with a rich variety of options while in the test bed. Although confirming the setting of the loss priority will still be difficult (because this bit never leaves the chassis), the counters associated with each policer should indicate whether two-level policing is working. Make sure that you do not forget to remove any additional match criteria when you are finished to avoid point loss, however.

The output confirms that application of the *limit-http* filter has not impacted traceroute and BGP functionality between r4 and C1, and this finding concludes the verification steps for the two-level rate-limiting configuration task. This brings you to the next case study requirement:

- Rate limit all traffic arriving from T1 to 5% of the peering interface's bandwidth.

Because this task involves rate limiting *all* of the traffic received on a given interface, you can meet the requirements with a policer applied directly to the r3's fe-0/0/2 interface. If desired, you can also call the policer from within a firewall filter employing a *match-all* term. Note that you do not need to worry about various protocol families because only IPv4 traffic is present on the peering interface. The choice of a Fast Ethernet interface for this facet of the JNCIE test bed prevents the deployment of interface-based rate-limiting approaches such as leaky bucket rate limiting or subrate configuration. The approach shown here adopts the policer applied directly to interface approach, which is chosen over a firewall filter solution due to its simplicity:

```
[edit]
lab@r3# show firewall policer t1
if-exceeding {
    bandwidth-percent 5;
    burst-size-limit 15k;
```

```
}
then discard;

[edit]
lab@r3# show interfaces fe-0/0/2
unit 0 {
    family inet {
        policer {
            input t1;
        }
        address 172.16.0.13/30;
    }
}
```

Note that the *t1* policer is configured to use `bandwidth-percent`, as opposed to the previously demonstrated `bandwidth-limit`, which is in keeping with the need to limit received traffic to 5% of the peering interface's speed. Setting the policer to a `bandwidth-limit` of 5Mbps might result in exam point loss, even though 5% of 100Mbps equates to 5Mbps. While this may seem draconian, consider the case of an OC-3 interface that is later upgraded to an OC-12. While this may seem a bit of a stretch, consider that a PIC swap-out would allow the interface configuration stanza to be used for all SONET PICS from OC-3 to OC-192! If the PIC is swapped, it will be hard to claim that the static `bandwidth-limit` setting adequately complies with the need to operate at 5% of the interface's speed.

The burst parameter has again been set to the recommended default, owing to the fact that no burst tolerance parameters were specified. After committing the changes, the effect of the *t1* policer can be verified with some rapid pings as shown here. You begin by clearing the firewall counters at `r3`:

```
[edit]
lab@r3# run clear firewall all
```

The ping test is now performed at the T1 router:

```
[edit]
lab@T1-P1# run ping rapid count 100 size 1400 10.0.3.4 source 130.130.0.1
PING 10.0.3.4 (10.0.3.4): 1400 data bytes
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!.!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!.!!!!!!!!!!!!!!
   !!!!!!!!!!!!!!!!!!.!
--- 10.0.3.4 ping statistics ---
100 packets transmitted, 97 packets received, 3% packet loss
round-trip min/avg/max/stddev = 1.248/1.259/1.442/0.024 ms
```

The display indicates approximately 3% packet loss for traffic originating at the T1 router. Back at `r3`, the loss is confirmed to be the result of policer discards:

```
[edit]
lab@r3# run show policer
```

```
Policers:
Name                                                Packets
__default_arp_policer__                                   0
fe-0/0/2.0-in-policer                                     3
```

Also note that you can confirm application of an interface policer by including the `detail` switch when displaying the interface's status:

```
[edit]
lab@r3# run show interfaces fe-0/0/2 detail
Physical interface: fe-0/0/2, Enabled, Physical link is Up
  Interface index: 14, SNMP ifIndex: 17, Generation: 13
. . .
Logical interface fe-0/0/2.0 (Index 7) (SNMP ifIndex 18) (Generation 6)
    Flags: SNMP-Traps Encapsulation: ENET2
    Protocol inet, MTU: 1500, Generation: 14, Route table: 0
      Flags: None
      Policer: Input: fe-0/0/2.0-in-policer
      Addresses, Flags: Is-Preferred Is-Primary
        Destination: 172.16.0.12/30, Local: 172.16.0.13, Broadcast: 172.16.0.15,
           Generation: 12
```

With interface policing now confirmed, you move on to the next case study configuration task:

- Count and police traffic sent to *all* host IDs on the 192.168.0/24 subnet at `r5`, according to these restrictions:
  - Count and police traffic originated from external sources that is directed at FTP and web servers only.
  - Police FTP to 500Kbps.
  - Police HTTP to 2Mbps.
  - Configure one policer/counter for every four consecutive host addresses. Ensure that you do not create more policer/counter pairs than are required by the parameters specified.

To meet the specified behavior, you need to configure PSCP at `r5` with two different `prefix-action` sets. The following highlights call out changes made to existing configuration stanzas on `r5` in support of this configuration challenge:

```
[edit firewall]
lab@r5# show family inet
prefix-action ftp {
    policer ftp;
    count;
    subnet-prefix-length 24;
    destination-prefix-length 30;
```

```
}
prefix-action http {
    policer http;
    count;
    subnet-prefix-length 24;
    destination-prefix-length 30;
}
```

The two `prefix-action` definitions correctly link associated traffic to a counting action and a policer definition. The lack of a `filter-specific` statement means that a set of counters and policers will be created for each term in the firewall that calls the related `prefix-action` set. Besides giving you per term granularity for counter/policer statistics, this fact becomes significant later when you decide to display the resulting policers and counters.

The use of a `destination-prefix-length` setting of *30* correctly maps each grouping of four consecutive host IDs to a counter/policer pair, in that there are four combinations to bits 30 and 31, which lie outside of the specified prefix length. Setting the prefix length to *32* results in a one-to-one mapping of prefix to policer/counter instances. The `subnet-prefix-length` parameter is set to *24*, which permits the creation of 256 policer/counter pairs. The actual number of policer/counter instances is determined by the formula 2 ^ (prefix-length – subnet-length), which in this case yields 2 ^ (30 – 24), or 2 ^ 6, which yields a total of 64 policer/counter pairs. With each policer/counter pair being shared by four consecutive host IDs, the `prefix-action` configuration shown correctly accommodates the need to count and police traffic to all 254 hosts assumed present on the 192.168.0/24 subnet. Note that the admonition against creating more policer/counter pairs than are actually required can easily lead a candidate to the incorrect setting of the `subnet-prefix-length` parameter if the formula shown earlier is not well understood. The bottom line is that you need a total of 64 policer/counter instances, each policing four consecutive host addresses, to get these exam points. The configuration shown earlier meets these requirements, even though the subnet length setting shown often causes the uninitiated to believe that you will end up with 256 policer/counter pairs, in clear violation of the "no extra policer counter pairs" clause.

You now examine the new policer definitions at `r5`:

```
[edit firewall]
lab@r5# show policer ftp
if-exceeding {
    bandwidth-limit 500k;
    burst-size-limit 15k;
}
then discard;

[edit firewall]
lab@r5# show policer http
if-exceeding {
    bandwidth-limit 2m;
```

```
    burst-size-limit 15k;
}
then discard;
```

There is nothing very fancy about the *ftp* and *http* policer definitions. Each policer makes use of the recommended burst size and correctly limits traffic rates to 500Kbps and 2Mbps, respectively. You now display the firewall filter that is used to direct matching traffic to the associated `prefix-action` set:

```
[edit firewall]
lab@r5# show filter dc-pscp
term ftp-traffic {
    from {
        source-address {
            0.0.0.0/0;
            10.0.0.0/16 except;
        }
        destination-address {
            192.168.0.0/24;
        }
        protocol tcp;
        destination-port [ ftp ftp-data ];
    }
    then prefix-action ftp;
}
term http-traffic {
    from {
        source-address {
            0.0.0.0/0;
            10.0.0.0/16 except;
        }
        destination-address {
            192.168.0.0/24;
        }
        protocol tcp;
        destination-port http;
    }
    then prefix-action http;
}
term all-else {
    then accept;
}
```

The first two terms in the *dc-pscp* filter match on the TCP protocol and the destination ports associated with either FTP or HTTP traffic, in accordance with the requirements of the scenario. Note that the terms also make use of destination and source address match criteria that limit matches to packets with source addresses that *do not* equal 10.0/16 and destination addresses that *are* equal to 192.168.0/24. Traffic matching the *ftp-traffic* or *http-traffic* terms is directed to the appropriate *prefix-action* set. The final term in the *dc-pscp* filter functions to accept all other traffic without counting or policing, which accommodates the requirement that you count only FTP and HTTP traffic streams. The *dc-pscp* firewall filter is applied as an output to both of r5's Fast Ethernet interfaces. This ensures that all traffic hitting r5, which is destined for the 192.168.0/24 subnet, will be screened against the filter for possible counting and policing:

```
[edit]
lab@r5# show interfaces fe-0/0/0
unit 0 {
    family inet {
        filter {
            output dc-pscp;
        }
        address 10.0.8.6/30;
    }
    family iso;
}

[edit]
lab@r5# show interfaces fe-0/0/1
unit 0 {
    family inet {
        filter {
            output dc-pscp;
        }
        address 10.0.8.9/30;
    }
    family iso;
}
```

To verify the correct operation of r5's PSCP configuration, you need to complete the next case study task. This is because you need to be sure that *all* FTP and HTTP request traffic destined for the 192.168.0/24 data center subnet that is received at r3 and r4 from the T1 and C1 peerings is routed through r5 if you hope to get accurate measurements. Note that, currently, traffic arriving from T1 that is addressed to the 192.168.0/24 subnet is normally sent from r3 directly to r6, while similar traffic arriving from C1 is normally sent from r4 directly to r7. This forwarding behavior is the result of metrics, and stems from the fact that both r6 and r7 advertise

the data center prefixes with a metric of 2 (the RIP metric) while r5 advertises the same routes with a metric of 12.

Note that simply downing the r3–r6 and r4–r7 links in an attempt to force traffic through r5 will result in the loss of the IBGP sessions between r3/r4 and r6/r7. The IBGP session interruption stems from the presence of a local 10.0/16 aggregate definition on both r6 and r7, coupled with the lack of r3/r4 loopback address advertisement in their Level 1 IS-IS area.

The choice of r5 as the location for PSCP functionality serves to complicate your confirmation of PSCP behavior owing to the fact that qualifying traffic normally never passes through r5! The default forwarding behavior described previously is confirmed here with a traceroute from C1:

```
lab@c1> traceroute 192.168.0.1 source 200.200.0.1
traceroute to 192.168.0.1 (192.168.0.1) from 200.200.0.1, 30 hops max,
   40 byte packets
 1  172.16.0.5 (172.16.0.5)  0.505 ms  0.336 ms  0.280 ms
 2  10.0.2.17 (10.0.2.17)  0.206 ms  0.175 ms  0.173 ms
 3  192.168.0.1 (192.168.0.1)  0.343 ms  0.288 ms  0.283 ms
```

For now, confirmation of PSCP at r5 is confined to the verification that no services or applications have been adversely affected by the application of the *dc-pscp* filter. You can also display the PSCP counters and policers to verify that the required number of policer/counter instances exist for both the HTTP and FTP protocols:

```
lab@r5> show firewall prefix-action-stats filter dc-pscp prefix-action
   http-http-traffic
Filter: dc-pscp
Counters:
Name                                                Bytes              Packets
http-http-traffic-0                                     0                    0
http-http-traffic-1                                     0                    0
. . .
http-http-traffic-62                                    0                    0
http-http-traffic-63                                    0                    0
Policers:
Name                                              Packets
http-http-traffic-0                                     0
http-http-traffic-1                                     0
. . .
http-http-traffic-62                                    0
http-http-traffic-63                                    0
```

As expected, there are 64 policer/counter instances present for the *http* prefix-action. Note that you must specify the term name of the calling firewall filter to display the counter and policer values because the prefix-actions sets in this example are not identified as being filter specific. In this example, the *http* prefix-action set is called from the *http-traffic* term

in the *dc-pscp* filter; the *http-traffic* term name can be seen concatenated to the end of the command in the display. You can assume that a similar display is provided for the *ftp* prefix-action set. With the required number of counters and policers confirmed for both the HTTP and FTP protocols, you move on to the next case study requirement:

- Configure r3 and r4 to forward all HTTP and FTP request traffic received from external sources and addressed to the 192.168.0/24 subnet to r5.

The previous task confirmed that r3 and r4 *do not* forward through r5 when routing to the 192.168.0/24 subnet. You therefore need to use FBF on r3 and r4 to direct matching HTTP and FTP traffic into a routing instance that forwards the selected traffic through r5 to complete this configuration task. Note that you need to modify the existing *no-spoof* input filter in place at r4 to add FBF functionality, because FBF can only be called from an input filter and only one filter is allowed per direction on a given logical unit. The following highlights call out the changes made to r4's configuration to support the FBF task. Note that the terms in the original *no-spoof* filter have been renamed, and rearranged, using the rename and insert commands. Also note that by default the new *dc* term was added to the end of the *no-spoof* filter, where it had absolutely no effect:

```
[edit]
lab@r4# show firewall filter no-spoof
term dc {
    from {
        source-address {
            200.200.0.0/16;
            172.16.0.4/30;
        }
        protocol tcp;
        destination-port [ http ftp ftp-data ];
    }
    then routing-instance dc;
}
term valid-sa {
    from {
        source-address {
            200.200.0.0/16;
            172.16.0.0/29;
        }
    }
    then accept;
}
term spoofed-sa {
    then {
        count spoofs;
```

```
        discard;
    }
}
```

The new *dc* term, which functions to direct matching FTP and HTTP traffic to the *dc* routing instance, also includes source address spoofing prevention functionality. Candidates often forget to back-port existing firewall functionality to a new term that is added after the fact. Remember, failing to prevent spoofed packets from C1 will result in exam point loss! The 172.16.0.4/30 route filter in the *dc* term is designed to accommodate the r4–C1 peering link only because the *dc* term is not used in the *no-spoof* filter on r7. The new routing instance is displayed next:

```
[edit]
lab@r4# show routing-instances
dc {
    instance-type forwarding;
    routing-options {
        static {
            route 0.0.0.0/0 next-hop 10.0.2.9;
        }
    }
}
```

The default route will direct traffic in this instance to r5, which is the desired behavior in this case. To resolve the 10.0.2.9 next hop, you must add interface routes to the *dc* instance with the (much dreaded) RIB group configuration.

```
[edit]
lab@r4# show routing-options
interface-routes {
    rib-group inet interface-rib;
}
static {
    route 10.0.200.0/24 {
        next-hop 10.0.1.102;
        no-readvertise;
    }
}
aggregate {
    route 10.0.0.0/16;
}
rib-groups {
    interface-rib {
        import-rib [ inet.0 dc.inet.0 ];
```

```
    }
}
autonomous-system 65412;
```

Although not shown here, similar changes are also made at r3. You now generate FTP traffic from the C1 site to confirm FBF functionality at r4, while also validating the PSCP configuration at r5. You start by clearing all counters at r5:

```
[edit]
lab@r5# run clear firewall all
```

With the counters cleared, you verify that forwarding paths for non-HTTP and non-FTP traffic do not involve r5:

```
lab@c1> traceroute 192.168.0.1 source 200.200.0.1
traceroute to 192.168.0.1 (192.168.0.1) from 200.200.0.1, 30 hops max,
   40 byte packets
 1  172.16.0.5 (172.16.0.5)  0.409 ms  0.288 ms  0.278 ms
 2  10.0.2.17 (10.0.2.17)  0.202 ms  0.186 ms  0.173 ms
 3  192.168.0.1 (192.168.0.1)  0.319 ms  0.292 ms  0.286 ms
```

As required, the FBF configuration at r4 has no effect on the forwarding of UDP-based traceroute traffic. If FBF is working at r4, then FTP and HTTP traffic should be forwarded through r5, where the PSCP settings will count and police the traffic. With fingers crossed, you initiate the test:

```
lab@c1> ftp 192.168.0.1 interface lo0
Connected to 192.168.0.1.
220 dc FTP server (Version 6.00LS) ready.
Name (192.168.0.1:lab): lab
331 Password required for lab.
Password:
230 User lab logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

The FTP session succeeds. Note that the interface switch is used to source the FTP session from C1's loopback interface. The PSCP counters are now displayed on r5:

```
lab@r5> show firewall prefix-action-stats filter dc-pscp prefix-action
   ftp-ftp-traffic from 0 to 3
Filter: dc-pscp
Counters:
Name                                                Bytes              Packets
ftp-ftp-traffic-0                                     560                   10
ftp-ftp-traffic-1                                       0                    0
ftp-ftp-traffic-2                                       0                    0
ftp-ftp-traffic-3                                       0                    0
```

```
Policers:
Name                                                Packets
ftp-ftp-traffic-0                                         0
ftp-ftp-traffic-1                                         0
ftp-ftp-traffic-2                                         0
ftp-ftp-traffic-3                                         0
```

The display confirms that the FTP traffic was counted and that no policer discards have occurred. To verify that the number of prefixes per counter/policer pair has been correctly configured, FTP connections are now attempted to (nonexistent) hosts 192.168.0.2 and 192.168.0.3:

```
ftp> quit
221 Goodbye.

lab@c1> ftp 192.168.0.2 interface lo0
^C
lab@c1>

lab@c1> ftp 192.168.0.3 interface lo0
^C
lab@c1>
```

The PSCP counters are once again analyzed at `r5`, with the expectation that traffic to hosts 192.168.0-3 on the 192.168.0/24 subnet will be counted by the first of the 64 policer/counter pairs:

```
lab@r5> ... firewall prefix-action-stats filter dc-pscp prefix-action
   ftp-ftp-traffic from 0 to 3
Filter: dc-pscp
Counters:
Name                                                Bytes              Packets
ftp-ftp-traffic-0                                     902                   16
ftp-ftp-traffic-1                                       0                    0
ftp-ftp-traffic-2                                       0                    0
ftp-ftp-traffic-3                                       0                    0
Policers:
Name                                                Packets
ftp-ftp-traffic-0                                         0
ftp-ftp-traffic-1                                         0
ftp-ftp-traffic-2                                         0
ftp-ftp-traffic-3                                         0
```

As expected, only the first counter has incremented. This provides a good indication that the PSCP settings at `r5` are in accordance with the provided restrictions. The final proof is obtained when traffic to 192.168.0.4 causes the second counter to begin incrementing:

```
lab@c1> ftp 192.168.0.4 interface lo0
^C
```

```
lab@c1>

lab@r5> show firewall prefix-action-stats filter dc-pscp prefix-action
   ftp-ftp-traffic from 1 to 1
Filter: dc-pscp
Counters:
Name                                                Bytes              Packets
ftp-ftp-traffic-1                                      60                    1
Policers:
Name                                                Packets
ftp-ftp-traffic-1                                         0
```

These results confirm FBF functionality at `r4` and the PSCP settings at `r5`. You should perform similar tests at `r3` to validate the FBF configuration, which is not shown here, before proceeding. Note that HTTP validation is difficult because there is no way to source HTTP connection request from the loopback address of an M-series or T-series platform. Besides, the knowledge that FTP traffic is behaving properly greatly lessens the need for explicit validation of HTTP traffic. With FBF confirmed, you arrive at the final case study configuration requirement:

- Configure `r5` to perform the following:
  - Sample 50% of TCP connection requests received on the so-0/1/0 interface.
  - Ensure that three additional samples are taken for each sample trigger.
  - Forward samples as complete packets to the data collection device.

The final configuration requirement involves a firewall filter that selectively matches on initial TCP segments with a `sample` action. You also need to configure sampling and the mirroring interface parameters needed to communicate with the data collection device. The following highlights call out the changes made to `r5`'s configuration:

```
[edit]
lab@r5# show firewall filter sample
term 1 {
    from {
        protocol tcp;
        tcp-initial;
    }
    then sample;
}
term 2 {
    then accept;
}

[edit]
lab@r5# show forwarding-options
sampling {
```

```
    input {
        family inet {
            rate 2;
            run-length 3;
        }
    }
    output {
        port-mirroring {
            interface fe-0/0/3.0 {
                next-hop 172.16.30.2;
            }
        }
    }
}

[edit]
lab@r5# show interfaces so-0/1/0
encapsulation ppp;
unit 0 {
    family inet {
        filter {
            input sample;
        }
        address 10.0.2.9/30;
    }
    family iso;
}

[edit]
lab@r5# show interfaces fe-0/0/3
unit 0 {
    family inet {
        address 172.16.30.1/24 {
            arp 172.16.30.2 mac 00:a0:c9:69:c1:d8;
        }
    }
}
```

The changes to `r5`'s configuration illustrate a correct sampling ratio of 1:2 (50%) and a properly written firewall filter that passes all traffic while evoking the `sample` action modifier for initial TCP SYN segments only. The correct identification of the mirroring interface and the presence of a `next-hop` entry, which is required when using a multipoint interface for mirroring output, are evident in the `sampling output` stanza. The fe-0/0/3 mirroring interface has been

appropriately configured with an IP address from the data collection subnet and the MAC address of the data collection device. A static ARP entry is required when the attached test equipment does not respond to ARP requests; the inclusion of the MAC address on the case study topology is a strong indication that a static ARP entry is required in this case.

As we discussed in the section on port mirroring, trying to validate a port mirroring configuration without access to the data collection device can be difficult. In this example, you could monitor the fe-0/0/3 interface on `r5` while you generate TCP connection requests over the 10.0.2.8/30 subnet, all the while hoping that you will see a definitive correlation in the packet counts. Although you can easily generate this type of traffic at site C1 with the commands shown in the PSCP and FBF confirmation steps, note that generating a large or even a fixed number of initial TCP SYN segments is all but impossible when relying on the user interface to a TCP protocol stack. Another approach involves temporary changes to the *`sample`* filter so that it also marks ICMP traffic as a candidate for sampling. With these modifications in place, you can generate flood pings over `r4`'s so-0/1/1 interface while expecting to see a significant reflection of this traffic in the counter-related output of a `monitor interface fe-0/0/3` command at `r5`. Because this approach was demonstrated in the section on port mirroring, it will not be repeated here. Assume that the changes shown in `r5`'s configuration have been verified as working, at least to the extent made possible by the restrictions of the test bed. If you decide to modify the *`sample`* filter as suggested previously, make sure that you return the firewall filter to its original form when you are done with your confirmation.

It is recommended that you spend a few more moments quickly reassessing the operational state of your network before you consider your firewall filter and traffic sampling case study complete. Your confirmations should include the overall health of your test bed's IGP and BGP protocols, as well as internal and external connectivity. While everyone makes mistakes in a lab environment, it is often the undetected mistake that tends to have the most significant impact on your final grade.

Firewall Filter and Traffic Sampling Case Study Configurations

The changes made to the IS-IS baseline network topology to support the firewall filter and traffic sampling case study are listed here for all routers in the test bed; highlights have been added to Listings 3.1 to 3.5 as needed to call out changes to existing configuration stanzas.

`r1`'s configuration required no modifications to complete this case study.

`r2`'s configuration required no modifications to complete this case study.

Listing 3.1: Firewall Filter Case Study Configuration for *r3*

```
[edit]
lab@r3# show interfaces
fe-0/0/0 {
    unit 0 {
        family inet {
            address 10.0.4.13/30;
        }
```

```
        family iso;
    }
}
fe-0/0/1 {
    unit 0 {
        family inet {
            address 10.0.4.1/30;
        }
        family iso;
    }
}
fe-0/0/2 {
    unit 0 {
        family inet {
            filter {
                input dc;
            }
            policer {
                input t1;
            }
            address 172.16.0.13/30;
        }
    }
}
fe-0/0/3 {
    unit 0 {
        family inet {
            address 10.0.2.14/30;
        }
        family iso;
    }
}
at-0/1/0 {
    atm-options {
        vpi 0 {
            maximum-vcs 64;
        }
    }
    unit 0 {
        point-to-point;
        vci 50;
```

```
        family inet {
            address 10.0.2.2/30;
        }
        family iso;
    }
}
so-0/2/0 {
    dce;
    encapsulation frame-relay;
    unit 100 {
        dlci 100;
        family inet {
            address 10.0.2.5/30;
        }
        family iso;
    }
}
fxp0 {
    unit 0 {
        family inet {
            address 10.0.1.3/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 10.0.3.3/32;
        }
        family iso {
            address 49.0001.3333.3333.3333.00;
        }
    }
}

[edit]
lab@r3# show firewall
policer t1 {
    if-exceeding {
        bandwidth-percent 5;
        burst-size-limit 15k;
```

```
    }
    then discard;
}
filter dc {
    term 1 {
        from {
            protocol tcp;
            destination-port [ ftp ftp-data http ];
        }
        then routing-instance dc;
    }
    term 2 {
        then accept;
    }
}

[edit]
lab@r3# show routing-options
interface-routes {
    rib-group inet interface-rib;
}
static {
    route 10.0.200.0/24 {
        next-hop 10.0.1.102;
        no-readvertise;
    }
}
aggregate {
    route 10.0.0.0/16;
}
rib-groups {
    interface-rib {
        import-rib [ inet.0 dc.inet.0 ];
    }
}
autonomous-system 65412;

[edit]
lab@r3# show routing-instances
dc {
    instance-type forwarding;
```

```
    routing-options {
        static {
            route 0.0.0.0/0 next-hop 10.0.2.1;
        }
    }
}
```

Listing 3.2: Firewall Filter Case Study Configuration for *r4*

```
[edit]
lab@r4# show interfaces
fe-0/0/0 {
    unit 0 {
        family inet {
            filter {
                input no-spoof;
                output limit-http;
            }
            address 172.16.0.5/30;
        }
    }
}
fe-0/0/1 {
    unit 0 {
        family inet {
            address 10.0.4.9/30;
        }
        family iso;
    }
}
fe-0/0/2 {
    unit 0 {
        family inet {
            address 10.0.4.17/30;
        }
        family iso;
    }
}
fe-0/0/3 {
    unit 0 {
        family inet {
            address 10.0.2.18/30;
```

```
        }
        family iso;
    }
}
so-0/1/0 {
    encapsulation frame-relay;
    unit 100 {
        dlci 100;
        family inet {
            address 10.0.2.6/30;
        }
        family iso;
    }
}
so-0/1/1 {
    encapsulation ppp;
    unit 0 {
        family inet {
            address 10.0.2.10/30;
        }
        family iso;
    }
}
fxp0 {
    unit 0 {
        family inet {
            address 10.0.1.4/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 10.0.3.4/32;
        }
        family iso {
            address 49.0001.4444.4444.4444.00;
        }
    }
}
```

```
[edit]
lab@r4# show firewall
policer 1m-http {
    if-exceeding {
        bandwidth-limit 1m;
        burst-size-limit 15k;
    }
    then loss-priority high;
}
policer 2m-http {
    if-exceeding {
        bandwidth-limit 2m;
        burst-size-limit 15k;
    }
    then discard;
}
filter no-spoof {
    term dc {
        from {
            source-address {
                200.200.0.0/16;
                172.16.0.4/30;
            }
            protocol tcp;
            destination-port [ http ftp ftp-data ];
        }
        then routing-instance dc;
    }
    term valid-sa {
        from {
            source-address {
                200.200.0.0/16;
                172.16.0.0/29;
            }
        }
        then accept;
    }
    term spoofed-sa {
        then {
            count spoofs;
            discard;
        }
```

```
    }
}
filter limit-http {
    term 1 {
        from {
            protocol tcp;
            source-port 80;
        }
        then {
            policer 1m-http;
            next term;
        }
    }
    term 2 {
        from {
            protocol tcp;
            source-port 80;
        }
        then policer 2m-http;
    }
    term 3 {
        then accept;
    }
}

[edit]
lab@r4# show routing-options
interface-routes {
    rib-group inet interface-rib;
}
static {
    route 10.0.200.0/24 {
        next-hop 10.0.1.102;
        no-readvertise;
    }
}
aggregate {
    route 10.0.0.0/16;
}
rib-groups {
    interface-rib {
```

```
        import-rib [ inet.0 dc.inet.0 ];
    }
}
autonomous-system 65412;

[edit]
lab@r4# show routing-instances
dc {
    instance-type forwarding;
    routing-options {
        static {
            route 0.0.0.0/0 next-hop 10.0.2.9;
        }
    }
}
```

Listing 3.3: Firewall Filter Case Study Configuration for *r5*

```
[edit]
lab@r5# show interfaces
fe-0/0/0 {
    unit 0 {
        family inet {
            filter {
                output dc-pscp;
            }
            address 10.0.8.6/30;
        }
        family iso;
    }
}
fe-0/0/1 {
    unit 0 {
        family inet {
            filter {
                output dc-pscp;
            }
            address 10.0.8.9/30;
        }
        family iso;
    }
}
fe-0/0/3 {
```

```
    unit 0 {
        family inet {
            address 172.16.30.1/24 {
                arp 172.16.30.2 mac 00:a0:c9:69:c1:d8;
            }
        }
    }
}
so-0/1/0 {
    encapsulation ppp;
    unit 0 {
        family inet {
            filter {
                input sample;
            }
            address 10.0.2.9/30;
        }
        family iso;
    }
}
at-0/2/1 {
    atm-options {
        vpi 0 {
            maximum-vcs 64;
        }
    }
    unit 0 {
        point-to-point;
        vci 50;
        family inet {
            address 10.0.2.1/30;
        }
        family iso;
    }
}
fxp0 {
    unit 0 {
        family inet {
            address 10.0.1.5/24;
        }
    }
```

```
}
lo0 {
    unit 0 {
        family inet {
            filter {
                input lo0;
            }
            address 10.0.3.5/32;
        }
        family iso {
            address 49.0002.5555.5555.5555.00;
        }
    }
}

[edit]
lab@r5# show firewall
policer limit-icmp {
    if-exceeding {
        bandwidth-limit 500k;
        burst-size-limit 15k;
    }
    then discard;
}
policer ftp {
    if-exceeding {
        bandwidth-limit 500k;
        burst-size-limit 15k;
    }
    then discard;
}
policer http {
    if-exceeding {
        bandwidth-limit 2m;
        burst-size-limit 15k;
    }
    then discard;
}
family inet {
    prefix-action ftp {
        policer ftp;
```

```
        count;
        subnet-prefix-length 24;
        destination-prefix-length 30;
    }
    prefix-action http {
        policer http;
        count;
        subnet-prefix-length 24;
        destination-prefix-length 30;
    }
}
filter lo0 {
    term police-icmp {
        from {
            protocol icmp;
        }
        then policer limit-icmp;
    }
    term limit-access {
        from {
            source-address {
                10.0.0.0/16;
                192.168.0.0/21;
            }
            protocol tcp;
            destination-port [ ssh telnet ];
        }
        then accept;
    }
    term access-log {
        from {
            source-address {
                0.0.0.0/0;
            }
            protocol tcp;
            destination-port [ 22 23 ];
        }
        then {
            syslog;
            discard;
        }
```

```
    }
    term outgoing-tcp-services {
        from {
            protocol tcp;
            source-port [ 22 23 20 21 ];
        }
        then accept;
    }
    term outgoing-udp-services {
        from {
            protocol udp;
            source-port [ 1024-65535 1812 ];
        }
        then accept;
    }
    term bgp {
        from {
            protocol tcp;
            port bgp;
        }
        then accept;
    }
    term else {
        then {
            log;
            discard;
        }
    }
}
filter dc-pscp {
    term ftp-traffic {
        from {
            source-address {
                0.0.0.0/0;
                10.0.0.0/16 except;
            }
            destination-address {
                192.168.0.0/24;
            }
            protocol tcp;
            destination-port [ ftp ftp-data ];
```

```
        }
        then prefix-action ftp;
    }
    term http-traffic {
        from {
            source-address {
                0.0.0.0/0;
                10.0.0.0/16 except;
            }
            destination-address {
                192.168.0.0/24;
            }
            protocol tcp;
            destination-port http;
        }
        then prefix-action http;
    }
    term all-else {
        then accept;
    }
}
filter sample {
    term 1 {
        from {
            protocol tcp;
            tcp-initial;
        }
        then sample;
    }
    term 2 {
        then accept;
    }
}
[edit]
lab@r5# show forwarding-options
sampling {
    input {
        family inet {
            rate 2;
            run-length 3;
        }
```

```
    }
    output {
        port-mirroring {
            interface fe-0/0/3.0 {
                next-hop 172.16.30.2;
            }
        }
    }
}
```

Listing 3.4: Firewall Filter Case Study Configuration for *r6*

```
[edit]
lab@r6# show interfaces
fe-0/1/0 {
    unit 0 {
        family inet {
            address 10.0.8.5/30;
        }
        family iso;
    }
}
fe-0/1/1 {
    unit 0 {
        family inet {
            address 10.0.2.13/30;
        }
        family iso;
    }
}
fe-0/1/2 {
    unit 0 {
        family inet {
            address 10.0.8.2/30;
        }
        family iso;
    }
}
fe-0/1/3 {
    unit 0 {
        family inet {
            filter {
                input no-spoof;
```

```
            }
            address 172.16.0.9/30;
        }
    }
}
fxp0 {
    unit 0 {
        family inet {
            address 10.0.1.6/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 10.0.9.6/32;
        }
        family iso {
            address 49.0002.6666.6666.6666.00;
        }
    }
}

[edit]
lab@r6# show firewall
filter no-spoof {
    term 1 {
        from {
            source-address {
                200.200.0.0/16;
                172.16.0.8/30;
            }
        }
        then accept;
    }
    term 2 {
        then {
            count spoofs;
            discard;
        }
    }
}
```

Listing 3.5: Firewall Filter Case Study Configuration for *r7*

```
[edit]
lab@r7# show interfaces
fe-0/3/0 {
    unit 0 {
        family inet {
            address 10.0.8.14/30;
        }
        family iso;
    }
}
fe-0/3/1 {
    unit 0 {
        family inet {
            address 10.0.8.10/30;
        }
        family iso;
    }
}
fe-0/3/2 {
    unit 0 {
        family inet {
            filter {
                input no-spoof;
            }
            address 172.16.0.1/30;
        }
    }
}
fe-0/3/3 {
    unit 0 {
        family inet {
            address 10.0.2.17/30;
        }
        family iso;
    }
}
fxp0 {
    unit 0 {
        family inet {
```

```
            address 10.0.1.7/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 10.0.9.7/32;
        }
        family iso {
            address 49.0002.7777.7777.7777.00;
        }
    }
}

[edit]
lab@r7# show firewall
filter no-spoof {
    term 1 {
        from {
            source-address {
                200.200.0.0/16;
                172.16.0.0/29;
            }
        }
        then accept;
    }
    term 2 {
        then {
            count spoofs;
            discard;
        }
    }
}
```

Spot the Issues: Review Questions

1. You must deploy a firewall filter at r4 that permits all outgoing TCP sessions from C1 while preventing all incoming TCP connections. Will the configuration shown next achieve the desired behavior?

```
[edit]
lab@r4# show interfaces fe-0/0/0
unit 0 {
    family inet {
        filter {
            input c2;
        }
        address 172.16.0.5/30;
    }
}

[edit]
lab@r4# show firewall filter c1
term 1 {
    from {
        protocol tcp;
        tcp-established;
    }
    then {
        count estab;
        accept;
    }
}
term 2 {
    from {
        protocol tcp;
    }
    then {
        count initial;
        reject;
    }
}
term 3 {
    then accept;
}
```

2. This filter is not protecting the RE from unauthorized traffic. Can you spot the reason?

```
[edit]
lab@r5# show firewall filter lo0
filter lo0 {
    term police-icmp {
        from {
            protocol icmp;
        }
        then policer limit-icmp;
    }
    term limit-access {
        from {
            source-address {
                10.0.0.0/16;
                192.168.0.0/21;
            }
            protocol tcp;
            destination-port [ ssh telnet ];
        }
        then accept;
    }
    term access-log {
        from {
            source-address {
                0.0.0.0/0;
            }
            protocol tcp;
            destination-port [ 22 23 ];
        }
        then {
            syslog;
            discard;
        }
    }
    term outgoing-tcp-services {
        from {
            protocol tcp;
            source-port [ 22 23 20 21 ];
        }
        then accept;
    }
```

```
    term outgoing-udp-services {
        from {
            protocol udp;
            source-port [ 1024-65535 1812 ];
        }
        then accept;
    }
    term bgp {
        from {
            protocol tcp;
            port bgp;
        }
        then accept;
    }
    term else {
        then {
            log;
        }
    }
}

[edit]
lab@r5# show interfaces lo0
unit 0 {
    family inet {
        filter {
            input lo0;
        }
        address 10.0.3.5/32;
    }
    family iso {
        address 49.0002.5555.5555.5555.00;
    }
}
```

3. Based on the `prefix-action` set shown here, how many policer/counter pairs will be created?

```
[edit firewall]
lab@r5# show family inet prefix-action ftp
policer ftp;
count;
subnet-prefix-length 29;
destination-prefix-length 30;
```

4. Can you spot the problem in r3's FBF configuration from the case study topology?

```
[edit]
lab@r3# show routing-options
interface-routes {
    rib-group inet interface-rib;
}
static {
    route 10.0.200.0/24 {
        next-hop 10.0.1.102;
        no-readvertise;
    }
}
aggregate {
    route 10.0.0.0/16;
}
rib-groups {
    interface-rib {
        import-rib [ inet.0 dc.inet.0 ];
        import-policy fbf;
    }
}
autonomous-system 65412;

[edit]
lab@r3# show routing-instances
dc {
    instance-type forwarding;
    routing-options {
        static {
            route 0.0.0.0/0 next-hop 10.0.2.1;
        }
    }
}

[edit]
lab@r3# show policy-options policy-statement fbf
term 1 {
    from {
        protocol [ static local ];
        route-filter 10.0.2.0/30 orlonger;
    }
```

```
    then accept;
}
term 2 {
    then reject;
}

[edit]
```

5. The port mirroring component of the case study is not working. Is anything amiss in r5's configuration?

```
[edit]
lab@r5# show forwarding-options
sampling {
    input {
        family inet {
            rate 2;
            run-length 3;
        }
    }
    output {
        port-mirroring {
            interface fe-0/0/3.0 {
            }
        }
    }
}

[edit]
lab@r5# show firewall filter sample
term 1 {
    from {
        protocol tcp;
        tcp-initial;
    }
    then sample;
}
term 2 {
    then accept;
}
```

```
[edit]
lab@r5# show interfaces so-0/1/0
encapsulation ppp;
unit 0 {
    family inet {
        filter {
            input sample;
        }
        address 10.0.2.9/30;
    }
    family iso;
}
```

Spot the Issues: Answers to Review Questions

1. No. The problem with this firewall filter lies in the direction in which it has been applied to r4's fe-0/0/0 interface. The *c1* filter will prevent site C1 from initiating TCP connections when applied as an input filter.

2. The problem with this firewall filter lies in the final term called *else*. Using a log action modifier without the inclusion of an explicit discard or reject action results in the acceptance of all traffic that matches the *else* term. Note that the default term action of discard is changed to accept when an action modifier is used.

3. According to the formula provided in the chapter body, there should be two policer/counter instances. Each instance will be shared by four consecutive IP addresses with the prefix length of *30* that is configured.

```
lab@r5> show firewall prefix-action-stats filter dc-pscp prefix-action
   ftp-ftp-traffic
Filter: dc-pscp
Counters:
Name                                                Bytes              Packets
ftp-ftp-traffic-0                                       0                    0
ftp-ftp-traffic-1                                       0                    0
Policers:
Name                                               Packets
ftp-ftp-traffic-0                                       0
ftp-ftp-traffic-1                                       0
```

4. The problem lies in the import policy being applied to the *interface-rib* RIB group. The *fbf* policy fails to include direct routes, and r3 needs the 10.0.2.0/30 direct route in the *dc*.inet.0 routing instance to activate the static default route. Note that with this configuration the 10.0.2.2/32 route is installed in the *dc*.inet.0 instance, but the static default route is not, because of the inability to resolve the static route's 10.0.2.1/32 next hop:

```
[edit]
lab@r3# run show route table dc

dc.inet.0: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.2.2/32        *[Local/0] 00:02:33
                      Local via at-0/1/0.0
```

To correct this situation, add the `direct` protocol to term *1* of the *fbf* policy. Note that you do not need to include the `static` protocol:

```
[edit policy-options policy-statement fbf]
lab@r3# show
term 1 {
    from {
        protocol direct;
        route-filter 10.0.2.0/30 orlonger;
    }
    then accept;
}
term 2 {
    then reject;
}

[edit policy-options policy-statement test]
lab@r3# run show route table dc

dc.inet.0: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[Static/5] 00:00:07
                    > to 10.0.2.1 via at-0/1/0.0
10.0.2.0/30        *[Direct/0] 00:00:07
                    > via at-0/1/0.0
```

5. `r5`'s port-mirroring configuration is broken because the `next-hop` parameter is missing from the `sampling output` stanza. The specification of a next hop is required when mirroring over a multipoint interface. A functional sampling `output stanza` for the case study topology contains the `next-hop` parameter, as shown next:

```
[edit]
lab@r5# show forwarding-options sampling output
port-mirroring {
    interface fe-0/0/3.0 {
        next-hop 172.16.30.2;
    }
}
```

Chapter

4

Multicast

JNCIE LAB SKILLS COVERED IN THIS CHAPTER:

- ✓ **IGMP**
  - Static joins
- ✓ **DVMRP**
- ✓ **PIM**
  - Dense mode
  - Sparse mode (Bootstrap and Auto-RP)
- ✓ **MSDP**
  - Any-Cast RP
  - Interdomain

This chapter exposes the reader to a variety of JNCIE-level configuration scenarios that validate a candidate's theoretical and practical understanding of multicast technology on M-series and T-series routing platforms. It is assumed that the reader already has a working knowledge of multicast protocols and their operation, to the extent covered in the *JNCIS Study Guide* (Sybex, 2003).

Juniper Networks routing platforms support a variety of multicast protocols and standards, including Internet Group Management Protocol (IGMP) versions 1, 2, and 3, Distance Vector Multicast Routing Protocol (DVMRP), Protocol Independent Multicast (PIM) versions 1 and 2 (in both dense and sparse mode), Session Announcement Protocol (SAP), Session Discovery Protocol (SDP), and Multicast Source Discovery Protocol (MSDP). Note that a tunnel services (TS) Physical Interface Card (PIC) is required when supporting DVMRP tunnels across non-multicast enabled network elements. A TS is also needed at the Rendezvous Point (RP) and in first hop routers (those routers with directly attached multicast sources), when deploying a PIM sparse mode topology.

Use the `show chassis fpc pic-status` command to determine if your router is equipped with a TS PIC:

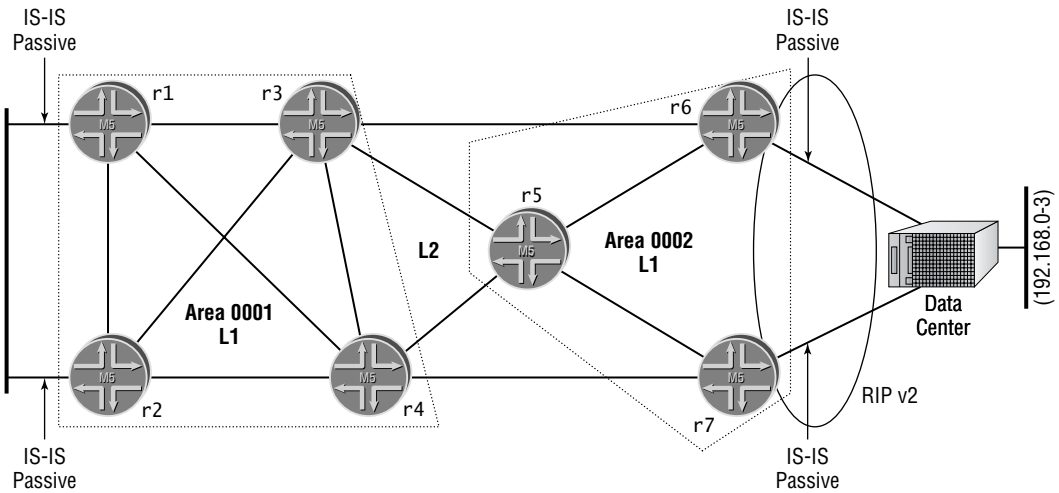
```
lab@montreal> show chassis fpc pic-status
Slot 0 Online
  PIC 0    4x OC-3 SONET, SMIR
  PIC 2    2x OC-3 ATM, MM
Slot 3 Online
  PIC 1    4x OC-3 SONET, MM
Slot 6 Online
  PIC 0    1x G/E, 1000 BASE-SX
  PIC 1    1x G/E, 1000 BASE-SX
  PIC 2    1x Tunnel
Slot 7 Online
```

Note that the absence of a TS PIC can have a dramatic impact on the overall success of a given multicast design. A prepared JNCIE candidate will know when (and where) TS PICs are needed, and will be ready to perform inventory steps on their test bed to ensure that their network design makes appropriate use of routers that are outfitted with TS PICs.

The chapter's case study is designed to closely approximate a typical JNCIE multicast configuration scenario. Examples of key operational mode command output are provided to allow for an operational comparison of your network to that of a known good example. Examples of baseline configuration modifications that are known to meet all case study requirements are provided at the end of the case study for all routers in the multicast test bed.

The examples demonstrated in the chapter body are based on the IS-IS baseline configuration as discovered in the case study for Chapter 1, "Network Discovery and Verification." If you are unsure as to the state of your test bed, you should take a few moments to load up and confirm the operation of the IS-IS discovery configuration before proceeding. Figure 4.1 reviews the IS-IS IGP discovery findings from the case study in Chapter 1.

FIGURE 4.1 Summary of IS-IS IGP discovery



Notes:

Multi-level IS-IS, Areas 0001 and 0002 with ISO NET based on router number.

lo0 address of `r3` and `r4` not injected into Area 0001 to ensure optimal forwarding between 10.0.3.3 and 10.0.3.4.

Passive setting on `r5`'s core interfaces for optimal Area 0002-to-core routing.

No authentication or route summarization. Routing policy at `r5` to leak L1 externals (DC routes) to L2.

Redistribution of static default route to data center from both `r6` and `r7`. Redistribution of 192.168.0/24 through 192.168.3/24 routes from RIP into IS-IS by both `r6` and `r7`.

All adjacencies are up, reachability problem discovered at `r1` and `r2` caused by local aggregate definition. Corrected through IBGP policy to effect 10.0/16 route advertisement from `r3` and `r4` to `r1` and `r2`; removed local aggregate from `r1` and `r2`.

Suboptimal routing detected at the data center and at `r1`/`r2` for some locations. This is the result of random nexthop choice for data center's default, and the result of `r1` and `r2`'s preference for `r3`'s RID over `r4` with regard to the 10.0/16 route. This is considered normal behavior, so no corrective actions are taken.

IGMP

The Internet Group Management Protocol (IGMP) is used by hosts and routers to determine whether there are any attached listeners for a given multicast group address (G). Routers will stop forwarding traffic addressed to G when the IGMP protocol indicates that no attached hosts are interested in the corresponding traffic. IGMP uses a variety of timers, and the concept of a querier router, to facilitate and control the exchange of IGMP messages on a given subnet.

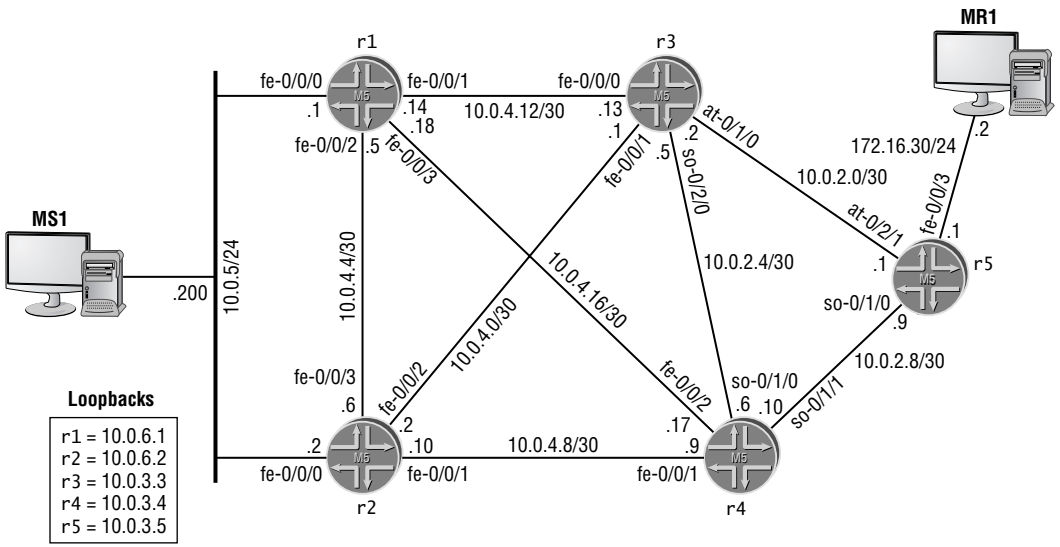
By default, IGMP version 2 is enabled on all broadcast interfaces when you configure PIM or DVMRP on that interface. You must explicitly enable IGMP on point-to-point interfaces when needed; the default JUNOS software behavior reflects the fact that multicast hosts are normally attached to routers using some form of Ethernet topology while point-to-point links are normally used to interconnect routers themselves.

After verifying the operation of the baseline network using the steps outlined in previous chapters, your multicast configuration scenario begins with these IGMP configuration requirements:

- Enable IGMP version 3 to MR1. You must not configure PIM or DVMRP at this time.
- Configure `r5` to wait no more than 5 seconds for IGMP responses.
- Without interacting with MR1, ensure that `r5` displays a *,G join for group 225.0.0.1.

Figure 4.2 focuses on the subset of routers that make up the multicast test bed and provides additional details about the multicast receiver, MR1.

FIGURE 4.2 Multicast test bed topology



Configuring IGMP

You begin at r5 by configuring its fe-0/0/3 interface for proper connectivity to the MR1 device:

```
[edit interfaces]
lab@r5# set fe-0/0/3 unit 0 family inet address 172.16.30.1/24
```

The modified configuration is displayed for confirmation:

```
[edit interfaces]
lab@r5# show fe-0/0/3
unit 0 {
    family inet {
        address 172.16.30.1/24;
    }
}
```

The following command sequence correctly configures IGMP for the criteria posed in this example:

```
[edit protocols igmp]
lab@r5# set query-response-interval 5

[edit protocols igmp]
lab@r5# set interface fe-0/0/3 version 3

[edit protocols igmp]
lab@r5# set interface fe-0/0/3 static group 225.0.0.1
```

The configuration changes are displayed:

```
[edit protocols igmp]
lab@r5# show
query-response-interval 5;
interface fe-0/0/3.0 {
    version 3;
    static {
        group 225.0.0.1;
    }
}
```

Note that the query-response-interval has been set to the required 5 seconds, and that IGMP version 3 has been specified. The static join does not include a source address, which correctly meets the stipulations that you create a *,G join state. Note that including a source address results in a S,G state for that particular source. Satisfied with your changes, you commit your candidate configuration at r5:

```
[edit protocols igmp]
lab@r5# commit and-quit
```

```
commit complete
Exiting configuration mode

lab@r5>
```

Verifying IGMP

Multicast protocols can take a rather long time to converge. Candidates that are not experienced with multicast protocols sometimes postpone (or prevent) proper operation by continuously changing their configurations, or by constantly restarting rpd or rebooting the router. By way of example, a PIM bootstrap router will remain in the candidate state for at least 60 seconds, during which time it monitors bootstrap messages in an attempt to determine whether it will remain a BSR candidate or be elected the BSR. During this time, other routers will not learn of this router's candidate RP status. In some cases, a restart of the routing daemon may expedite multicast protocol convergence, because restarting rpd may eliminate state information that would otherwise have to age out. The bottom line with multicast convergence is that when you think you have waited long enough for some multicast related protocol to "do its thing," you are well advised to wait "some more" before making changes to your configuration.

The verification of your IGMP configuration begins with the determination that r5's fe-0/0/3 interface is operational:

```
lab@r5> show interfaces fe-0/0/3 terse
Interface               Admin Link Proto Local                 Remote
fe-0/0/3                up    up
fe-0/0/3.0              up    up   inet  172.16.30.1/24
```

The display confirms that the fe-0/0/3 interface is operational and provides you with an ideal opportunity to reconfirm that the correct address was assigned. A ping test is now performed to MR1 to verify interface operation:

```
lab@r5> ping 172.16.30.2 count 2
PING 172.16.30.2 (172.16.30.2): 56 data bytes

--- 172.16.30.2 ping statistics ---
2 packets transmitted, 0 packets received, 100% packet loss
```

Noting that the ping test fails, you decide to examine the Address Resolution Protocol (ARP) cache on r5:

```
lab@r5> show arp
MAC Address             Address         Name            Interface
00:b0:d0:10:73:2f       10.0.1.100      10.0.1.100      fxp0.0
00:03:47:42:37:7c       10.0.1.102      10.0.1.102      fxp0.0
```

```
Total entries: 2

lab@r5>
```

The absence of an ARP entry for the MR1 device helps to explain why the ping test between r5 and MR1 fails. These symptoms could indicate interface configuration mistakes, hardware malfunctions, or that the MR1 device does not really exist. Lacking access to the MR1 device, there is little you can do to fault isolate this situation further without assistance from the proctor. In this case, assume that the proctor has confirmed that the lack of ARP entry and IGMP echo failures are "normal." This feedback provides a strong clue that the MR1 device does not really exist, or that it is some type of test equipment such as a packet sniffer. The lack of a multicast receiver should not impair your ability to configure and verify multicast forwarding as long as you are able to create static IGMP joins that direct traffic out any interfaces associated with identified multicast receivers.

Having gone as far as you can with regard to verification of the link between r5 and MR1, you decide to move on to IGMP verification:

```
lab@r5> show igmp interface
Interface: fe-0/0/3.0
    Querier:  172.16.30.1
    State: Up         Timeout:    None    Version:  3    Groups:       1

Configured Parameters:
IGMP Query Interval: 125.0
IGMP Query Response Interval: 5.0
IGMP Last Member Query Interval: 1.0
IGMP Robustness Count: 2

Derived Parameters:
IGMP Membership Timeout: 255.0
IGMP Other Querier Present Timeout: 252.500000
```

The highlighted entries confirm that the fe-0/0/3 interface is considered "up" from the perspective of IGMP, that IGMP version 3 is configured, that r5 is the querier for the 172.16.30/24 subnet, and that the response interval has been correctly set to *5* seconds. The indication that a single multicast group is active on this interface is also highlighted. Your next command displays information about active multicast groups:

```
lab@r5> show igmp group
Interface: fe-0/0/3.0
    Group: 225.0.0.1
        Source: 0.0.0.0   Last Reported by: 0.0.0.0
        Timeout:       0          Type: Static
```

The output confirms the presence of a static join to group 225.0.0.1 for traffic generated by any source address (0.0.0.0). The display confirms that you have correctly configured IGMP

and the static *,G join state, as outlined by the restrictions in this configuration example. Displaying general IGMP statistics often helps to draw attention to problem areas:

```
lab@r5> show igmp statistics
IGMP packet statistics for all interfaces
IGMP Message type      Received       Sent  Rx errors
Membership Query              0         10          0
V1 Membership Report          0          0          0
DVMRP                         0          0          0
PIM V1                        0          0          0
Cisco Trace                   0          0          0
V2 Membership Report          0          0          0
Group Leave                   0          0          0
Mtrace Response               0          0          0
Mtrace Request                0          0          0
Domain Wide Report            0          0          0
V3 Membership Report          0          0          0
Other Unknown types                              0
IGMP v3 unsupported type                         0
IGMP v3 source required for SSM                  0
IGMP v3 mode not applicable for SSM              0

IGMP Global Statistics
Bad Length                    0
Bad Checksum                  0
Bad Receive If                0
Rx non-local                  0
```

The statistics output indicates that no protocol errors are occurring and that `r5` has sent 10 membership queries since the statistics were cleared (or started in this case). Considering that there are no active multicast receives attached to `r5` at this time, the lack of membership reports and group leave messages is normal in this case. IGMP statistics can be cleared with the `clear igmp statistics` operational mode command. Although not called for in this example, IGMP tracing may prove invaluable when troubleshooting group join and leave issues. The highlights in the next capture call out a typical IGMP tracing configuration:

```
[edit protocols igmp]
lab@r5# show
traceoptions {
    file igmp;
    flag leave detail;
    flag report detail;
    flag query detail;
}
```

```
query-response-interval 5;
interface fe-0/0/3.0 {
    version 3;
    static {
        group 225.0.0.1;
    }
}
```

The trace output garnered from this configuration indicates that the only IGMP traffic present on the 172.16.30/24 subnet is in the form of version 3 membership queries. This is expected, considering the lack of multicast receivers on this subnet:

```
[edit protocols igmp]
lab@r5# run monitor start igmp

[edit protocols igmp]
lab@r5#
*** igmp ***
Mar 26 14:27:37 IGMP SENT 172.16.30.1 -> 224.0.0.1 Membership Query(v3):
   length 12 intvl 5

lab@r5# Mar 26 14:29:39 IGMP SENT 172.16.30.1 -> 224.0.0.1 Membership Query(v3):
   length 12 intvl 5

Mar 26 14:31:41 IGMP SENT 172.16.30.1 -> 224.0.0.1 Membership Query(v3):
   length 12 intvl 5
```

The IGMP verification steps shown in this section have all indicated that IGMP is operational and configured in accordance with the provided criteria.

IGMP Summary

IGMP is used by routers and multicast receivers to communicate group membership status. IGMP is used by multicast *receivers* only. Note that multicast senders are not required to join any multicast groups; therefore, a multicast sender does not have to support the IGMP protocol.

The operation of IGMP ensures that routers will stop forwarding multicast traffic to a given subnet shortly after the last interested host indicates that it no longer wishes to receive traffic for that group. By default, JUNOS software operates according to IGMP version 2; you can explicitly configure IGMP version 1 or 3 as needed. A router configured for IGMP version 3 will fall back to IGMP version 2 (or 1) when needed, but this behavior will prevent version 3 features, such as Source Specific Multicast (SSM), from functioning.

IGMP is automatically enabled on broadcast interfaces when PIM or DVMRP is enabled on that interface. You can explicitly configure IGMP when needed, such as in the case of a point-to-point interface, or when interface-specific behavior is desired. You can configure static IGMP join state to help test and verify multicast forwarding in the absence of multicast

receivers. The static join can take the form of a *,G or S,G entry, based on the inclusion of a source address in the latter case. You need to include a source address when configuring a static SSM join.

The operation of IGMP can be verified with a variety of operational mode commands that were demonstrated in this section. You can also trace IGMP protocol exchanges to assist in fault isolation.

DVMRP

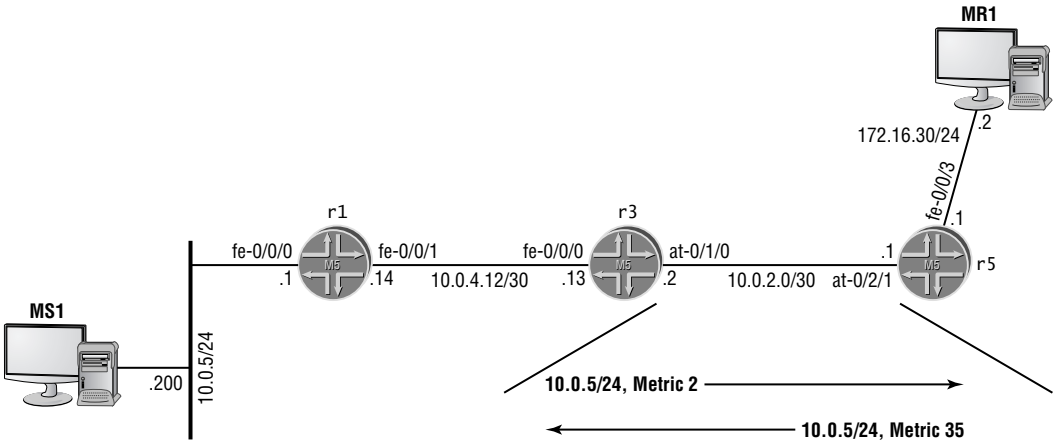
This section details a typical Distance Vector Multicast Routing Protocol (DVMRP) configuration scenario. Various verification and confirmation techniques for general multicast routing, and for DVMRP in particular, are provided in the confirmation section.

DVMRP operates much like the RIP protocol, in that it is based on the periodic announcement (every 60 seconds) of multicast reachability (a vector) along with an associated metric (distance) to all neighboring DVMRP routers. DVMRP is a dense protocol that operates according to a flood and prune model. By default, all multicast traffic is flooded away from the source (according to Reverse Path Forwarding [RPF] procedures) to all downstream nodes using what is called a truncated broadcast tree. Leaf nodes with no interested listeners send prune messages to their upstream neighbors to cease the flow of multicast traffic for that particular *,G entry. DVMRP supports a graft mechanism that allows the rapid removal of prune state in the event that a multicast receiver expresses interest in a previously pruned group.

The DVMRP protocol represents infinity with a metric of 32; however, the protocol uses poison reverse in a unique way. Specifically, a router that considers itself to be downstream for a particular source prefix adds 32 to the received route metric, and then proceeds to advertise the "poisoned" route back to the neighbor it was learned from. Upon seeing the poison reverse metric setting, the upstream neighbor adds the interface associated with the downstream router into its outgoing interface list (OIL). Figure 4.3 illustrates this behavior in the context of `r3` and `r5` for the 10.0.5.200/24 prefix associated with MS1. In this case, the 10.0.5/24 route is advertised by `r3` to `r5` with a metric of 2. `r5` adds 1 to the revived metric upon receipt. After determining that it wishes to be added to `r3`'s OIL for the 10.0.5/24 route, `r5` adds 32 to the route metric and sends the route back to `r3`, now with a metric of 35.

The hardest part about configuring DVMRP relates to the need for a multicast specific routing table, which is used by DVMRP to store and advertise multicast routes. To facilitate Reverse Path Forwarding (RPF) checks, you must ensure that interface routes are also placed into this routing table. These requirements necessitate the use of RIB groups, which are a common source of confusion for the uninitiated. Juniper Networks recommends that you use the `inet.2` routing table to house your interface and DVMRP routes, and the examples in this section are in line with this recommendation.

FIGURE 4.3 DVMRP poison reverse




```
[DVMRP Report Tracing at r5:]
Mar 27 13:32:58  DVMRP SENT 10.0.2.1 → 224.0.0.4 len 70 Report:  Vers:  3.255
       mask 255.255.255.252:
             10.0.2.4            34
             10.0.4.16           35
Mar 27  13:32:58
       mask 255.255.255.255:
             10.0.6.1            35
Mar 27  13:32:58
       mask 255.255.255.0:
             10.0.5.0            35
```

To complete this section, you must alter your configurations to meet these criteria:

- Configure DVMRP on `r1`, `r2`, `r3`, `r4`, and `r5` so that multicast traffic sent from MS1 to group 225.0.0.1 is made available to MR1.
- Configure DVMRP to wait 40 seconds before it declares the loss of a neighbor.
- Adjust DVMRP metrics to ensure that traffic from MS1 to MR1 transits the 10.0.2.8/30 subnet between `r4` and `r5`.

Configuring DVMRP

You start DVMRP configuration on `r5` by creating an interface RIB group called *`interface-rib`*, which is used to populate the `inet.2` routing table with interface routes. Interface routes are needed to perform RPF checks:

```
[edit routing-options]
lab@r5# set interface-routes rib-group inet interface-rib
```

The next set of commands instructs `r5` to copy the contents of the *`interface-rib`* into both the `inet.0` and `inet.2` routing tables:

```
[edit routing-options]
```

```
lab@r5# set rib-groups interface-rib import-rib inet.0

[edit routing-options]
lab@r5# set rib-groups interface-rib import-rib inet.2
```

The final RIB-related configuration steps relate to the creation of a RIB group for DVMRP, which is called *dvmrp-rg* in this example. The *dvmrp-rg* is associated with the DVMRP protocol in a subsequent configuration step. The `import` and `export` RIB statements instruct the *dvmrp-rg* group to import and export routes from the `inet.2` routing table:

```
[edit routing-options]
lab@r5# set rib-groups dvmrp-rg import-rib inet.2

[edit routing-options]
lab@r5# set rib-groups dvmrp-rg export-rib inet.2
```

The RIB group configuration in support of DVMRP is shown at `r5`, with recent changes highlighted:

```
[edit routing-options]
lab@r5# show
interface-routes {
    rib-group inet interface-rib;
}
static {
    route 10.0.200.0/24 {
        next-hop 10.0.1.102;
        no-readvertise;
    }
}
rib-groups {
    interface-rib {
        import-rib [ inet.0 inet.2 ];
    }
    dvmrp-rg {
        export-rib inet.2;
        import-rib inet.2;
    }
}
autonomous-system 65412;
```

The DVMRP protocol is now configured at `r5`. In this example, the `all` keyword is used to save some typing by enabling DVMRP on all of the router's interfaces. Because the `all` shortcut catches the OoB interface, the router's fxp0 interface is explicitly disabled to ensure that DVMRP does not attempt to operate over the OoB network:

```
[edit protocols dvmrp]
lab@r5# set interface all hold-time 40
```

```
[edit protocols dvmrp]
lab@r5# set interface fxp0 disable

[edit protocols dvmrp]
lab@r5# set rib-group dvmrp-rg
```

The completed DVMRP stanza at r5 is displayed:

```
[edit protocols dvmrp]
lab@r5# show
rib-group dvmrp-rg;
interface all {
    hold-time 40;
}
interface fxp0.0 {
    disable;
}
```

The specification of the RIB group that is to be used by DVMRP is critical. In this example, you can see the correct specification of the *dvmrp-rg* RIB group. Recall that previous RIB group configuration steps in this section function to link the *dvmrp-rg* to the `inet.2` routing table for route import and export purposes.

Although not shown, a similar DVMRP configuration is needed on `r1`, `r2`, `r3`, and `r4`. You might consider the use of `load merge terminal` to replicate the basic DVMRP configuration in these routers. A sample configuration that highlights the DVMRP changes at `r3` is shown next:

```
lab@r3# show routing-options
interface-routes {
    rib-group inet interface-rib;
}
static {
    route 10.0.200.0/24 {
        next-hop 10.0.1.102;
        no-readvertise;
    }
}
aggregate {
    route 10.0.0.0/16;
}
rib-groups {
    interface-rib {
        import-rib [ inet.0 inet.2 ];
    }
    dvmrp-rg {
        export-rib inet.2;
        import-rib inet.2;
```

```
    }
}
autonomous-system 65412;

[edit]
lab@r3# show protocols dvmrp
rib-group dvmrp-rg;
interface all {
    hold-time 40;
}
interface fxp0.0 {
    disable;
}
```

With similar changes in `r1` through `r5`, you commit your configurations and proceed to the verification section. Note that additional configuration changes may be required to ensure that multicast traffic from MS1 to MR1 traverses the POS link between `r4` and `r5`.

Verifying DVMRP

When testing DVMRP with the 5.6R2.4 code installed in the test bed, this author encountered a known bug that prevented proper operation. The bug is tracked under PR 32590. At the time of this writing, the fix for the problem was available only in the 5.*x* release train through a daily JUNOS software build, which was loaded to allow confirmation of DVMRP operation. Candidates should expect to find only production releases of JUNOS software in the actual JNCIE test bed. The 5.6R2.4 production code will be reloaded onto the routers at the conclusion of DVMRP testing.

You begin verification of DVMRP by confirming the presence of interface (and DVMRP) routes in the `inet.2` table:

```
[edit]
lab@r3# run show route table inet.2

inet.2: 27 destinations, 27 routes (27 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.1.0/24        *[Direct/0] 03:21:36
                    > via fxp0.0
10.0.1.3/32        *[Local/0] 03:10:05
                      Local via fxp0.0
10.0.2.0/30        *[Direct/0] 03:21:36
                    > via at-0/1/0.0
```

```
10.0.2.2/32        *[Local/0] 03:10:05
                      Local via at-0/1/0.0
10.0.2.4/30        *[Direct/0] 02:52:57
                    > via so-0/2/0.100
10.0.2.5/32        *[Local/0] 02:52:57
                      Local via so-0/2/0.100
10.0.2.8/30        *[DVMRP/110] 00:00:46, metric 2
                    > to 10.0.2.1 via at-0/1/0.0
10.0.2.12/30       *[Direct/0] 03:12:54
                    > via fe-0/0/3.0
10.0.2.14/32       *[Local/0] 03:10:05
                      Local via fe-0/0/3.0
10.0.2.16/30       *[DVMRP/110] 00:00:46, metric 2
                    > to 10.0.2.6 via so-0/2/0.100
10.0.3.3/32        *[Direct/0] 03:21:36
                    > via lo0.0
10.0.3.4/32        *[DVMRP/110] 00:00:46, metric 2
                    > to 10.0.2.6 via so-0/2/0.100
10.0.3.5/32        *[DVMRP/110] 00:00:46, metric 2
                    > to 10.0.2.1 via at-0/1/0.0
10.0.4.0/30        *[Direct/0] 00:54:36
                    > via fe-0/0/1.0
10.0.4.1/32        *[Local/0] 00:54:36
                      Local via fe-0/0/1.0
10.0.4.4/30        *[DVMRP/110] 00:00:46, metric 2
                    > to 10.0.4.2 via fe-0/0/1.0
10.0.4.8/30        *[DVMRP/110] 00:00:46, metric 2
                    > to 10.0.4.2 via fe-0/0/1.0
10.0.4.12/30       *[Direct/0] 02:59:03
                    > via fe-0/0/0.0
10.0.4.13/32       *[Local/0] 02:59:03
                      Local via fe-0/0/0.0
10.0.4.16/30       *[DVMRP/110] 00:00:46, metric 2
                    > to 10.0.2.6 via so-0/2/0.100
10.0.5.0/24        *[DVMRP/110] 00:00:46, metric 2
                    > to 10.0.4.2 via fe-0/0/1.0
10.0.6.1/32        *[DVMRP/110] 00:00:46, metric 2
                    > to 10.0.4.14 via fe-0/0/0.0
10.0.6.2/32        *[DVMRP/110] 00:00:46, metric 2
                    > to 10.0.4.2 via fe-0/0/1.0
10.0.8.4/30        *[DVMRP/110] 00:00:46, metric 2
                    > to 10.0.2.1 via at-0/1/0.0
```

```
10.0.8.8/30        *[DVMRP/110] 00:00:46, metric 2
                    > to 10.0.2.1 via at-0/1/0.0
172.16.0.13/32     *[Local/0] 03:10:05
                       Reject
172.16.30.0/24     *[DVMRP/110] 00:00:46, metric 2
                    > to 10.0.2.1 via at-0/1/0.0
```

The `inet.2` routing table on `r3` contains the expected interface routes, as well as entries learned through DVMRP. The highlighted entry calls out the 172.16.30/24 prefix associated with MR1. You may assume that `r1` through `r4` have similar entries in their `inet.2` routing tables. Displays such as this one indicate that your interface and *`dvmrp-rg`* RIB groups are correctly configured. The next confirmation step involves the verification of DVMRP neighbor status at `r1` through `r5`:

```
[edit]
lab@r4# run show dvmrp neighbors
Neighbor      Interface      Version    Flags    Routes    Timeout   Transitions
10.0.4.10     fe-0/0/1.0       3.255     PGM         4         31             2
10.0.4.18     fe-0/0/2.0       3.255     PGM         1         32             1
10.0.2.5      so-0/1/0.100     3.255     PGM         4         33             1
10.0.2.9      so-0/1/1.0       3.255     PGM         4         33             1
```

The sample display, which was obtained at `r5`, confirms that all expected DVMRP neighbors are present and that two-way communications has been successfully established. Note that two-way communications is indicated by the absence of a `1` in the `Flags` column. The indication that routes have been received from each neighbor provides additional confirmation that DVMRP is working properly. Though not shown, you can assume that all DVMRP neighbors are correctly listed for all routers in the multicast test bed. The next command verifies that the correct routes are being advertised and received thorough the DVMRP protocol:

```
[edit protocols igmp]
lab@r5# run show route protocol dvmrp

inet.0: 36 destinations, 44 routes (36 active, 0 holddown, 4 hidden)
+ = Active Route, - = Last Active, * = Both

224.0.0.4/32       *[DVMRP/0] 01:06:23
                       MultiRecv

inet.1: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

225.0.0.1,10.0.5.200/32*[DVMRP/110] 00:32:29
                       Multicast (IPv4)
```

```
inet.2: 26 destinations, 26 routes (26 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.2.4/30        *[DVMRP/110] 00:00:27, metric 2
                    > to 10.0.2.2 via at-0/2/1.0
10.0.2.12/30       *[DVMRP/110] 00:00:27, metric 2
                    > to 10.0.2.2 via at-0/2/1.0
10.0.2.16/30       *[DVMRP/110] 00:00:29, metric 2
                    > to 10.0.2.10 via so-0/1/0.0
10.0.3.3/32        *[DVMRP/110] 00:00:27, metric 2
                    > to 10.0.2.2 via at-0/2/1.0
10.0.3.4/32        *[DVMRP/110] 00:00:29, metric 2
                    > to 10.0.2.10 via so-0/1/0.0
10.0.4.0/30        *[DVMRP/110] 00:00:29, metric 3
                    > to 10.0.2.10 via so-0/1/0.0
10.0.4.4/30        *[DVMRP/110] 00:00:27, metric 3
                    > to 10.0.2.2 via at-0/2/1.0
10.0.4.8/30        *[DVMRP/110] 00:00:29, metric 2
                    > to 10.0.2.10 via so-0/1/0.0
10.0.4.12/30       *[DVMRP/110] 00:00:27, metric 2
                    > to 10.0.2.2 via at-0/2/1.0
10.0.4.16/30       *[DVMRP/110] 00:00:29, metric 2
                    > to 10.0.2.10 via so-0/1/0.0
10.0.5.0/24        *[DVMRP/110] 00:00:27, metric 3
                    > to 10.0.2.2 via at-0/2/1.0
10.0.6.1/32        *[DVMRP/110] 00:00:27, metric 3
                    > to 10.0.2.2 via at-0/2/1.0
10.0.6.2/32        *[DVMRP/110] 00:00:27, metric 3
                    > to 10.0.2.2 via at-0/2/1.0

iso.0: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)
```

All the expected DVMRP routes are present. The highlights call out the fact that the DVMRP routes are installed into the `inet.2` routing table, and the presence of the 10.0.5/24 route associated with the multicast sender, MS1. The metric for this route is set to 3, which correctly reflects the hop count (+1) between `r5` and the 10.0.5/24 subnet. Note that the `inet.1` routing table functions as a cache that houses transit multicast state. In this example, the `inet.1` entry indicates that source 10.0.5.200 is currently sending to group 225.0.0.1. Noting that IS-IS entries also exist in the main routing table for many of the routes displayed helps to drive home the fact that DVMRP maintains a separate routing table:

```
[edit protocols igmp]
lab@r5# run show route 10.0.5.0/24
```

```
inet.0: 36 destinations, 44 routes (36 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.5.0/24         *[IS-IS/18] 01:41:27, metric 30
                      to 10.0.2.10 via so-0/1/0.0
                    > to 10.0.2.2 via at-0/2/1.0

inet.2: 26 destinations, 26 routes (26 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.5.0/24         *[DVMRP/110] 00:00:10, metric 3
                    > to 10.0.2.2 via at-0/2/1.0
```

Issuing a `show route receiving-protocol dvmrp` *`neighbor`* command also proves helpful when monitoring and troubleshooting DVMRP operation, as shown here in the context of `r3` and the routes it received from `r5`:

```
[edit]
lab@r3# run show route receive-protocol dvmrp 10.0.2.1

inet.0: 37 destinations, 45 routes (37 active, 0 holddown, 0 hidden)

inet.2: 27 destinations, 27 routes (27 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.2.8/30         *[DVMRP/110] 00:00:07, metric 2
                    > to 10.0.2.1 via at-0/1/0.0
10.0.3.5/32         *[DVMRP/110] 00:00:07, metric 2
                    > to 10.0.2.1 via at-0/1/0.0
10.0.8.4/30         *[DVMRP/110] 00:00:07, metric 2
                    > to 10.0.2.1 via at-0/1/0.0
10.0.8.8/30         *[DVMRP/110] 00:00:07, metric 2
                    > to 10.0.2.1 via at-0/1/0.0
172.16.30.0/24      *[DVMRP/110] 00:00:07, metric 2
                    > to 10.0.2.1 via at-0/1/0.0

iso.0: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)
```

Note that the `show route advertising-protocol dvmrp` *`neighbor`* command is currently not functional and is tracked under PR 6307.

With the gross operation of DVMRP verified, it is now time to begin analysis of actual multicast traffic flow between MS1 and MR1. You begin by verifying that the multicast traffic is actually being delivered to the MR1 subnet. You can confirm the delivery of multicast traffic by monitoring interface counters or with the contents of a `show multicast route extensive`

command. The former approach is demonstrated here while the latter command is demonstrated in subsequent sections; note that you must use the `monitor interface` command for this test because `monitor traffic` displays only packets that are moving to or from the local Routing Engine (RE). Figure 4.4 displays the results of this test.

FIGURE 4.4 Monitor interface output

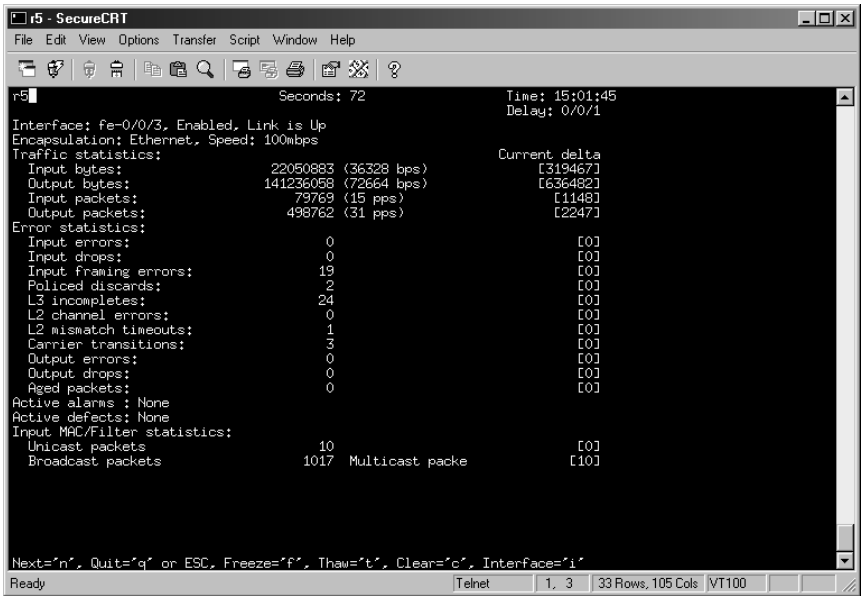


Figure 4.4 shows that some 31 packets per second are being output on `r5`'s fe-0/0/3 interface. While this in itself does not prove that the traffic is being generated by MS1, the steady state of the PPS count, coupled with the knowledge that the 172.16.30/24 subnet is not advertised in your IGP or BGP protocols, provides good indication that the counter is most likely reflecting multicast traffic. Put another way, besides `r5`, only DVMRP-enabled routers are aware of the 172.16.30/24 subnet, making the routing of unicast traffic to this segment all but impossible for all other routers in the test bed. You can assume that an identical PPS count is seen at the ingress routers, `r1` and `r2` in this case. Subsequent tests will provide further proof that the packets being counted do, in fact, represent the multicast traffic that is generated by MS1.

To be more exact as to the types of packets that are being counted, you can write and apply an *input* firewall filter that counts packets with a destination address of 225.0.0.1 and a source address of 10.0.5.200—time permitting, of course.

Output filters do not currently function for multicast traffic.

An example of such an input firewall filter is shown here:

```
[edit]
lab@r5# show firewall filter count-mcast
term 1 {
    from {
        source-address {
            10.0.5.200/32;
        }
        destination-address {
            225.0.0.1/32;
        }
    }
    then {
        count mcast;
        accept;
    }
}
term 2 {
    then accept;
}

[edit]
lab@r5# show interfaces at-0/2/1
unit 0 {
    family inet {
        filter {
            input count-mcast;
        }
        address 10.0.2.1/30;
    }
}
[edit]
lab@r5# show interfaces so-0/1/0
unit 0 {
    family inet {
        filter {
            input count-mcast;
        }
        address 10.0.2.9/30;
    }
}
```

Multicast Applications

While developing this chapter, this author used a combination of UNIX- and Windows-based multicast applications. The mgen and drec applications were used when testing with a Linux platform while the Wsend and Wlisten programs were used when testing with a Windows 2000 platform. As of this writing, mgen can be obtained from `http://manimac.itd.nrl.navy.mil/MGEN/`. The Wsend and Wlisten applications are produced by Microsoft, but an Internet source could not be located as this section was being developed. This author also found the StreamPump and StreamPlayer applications to be quite useful for generating and receiving multicast traffic on a Windows platform. As of this writing, you can download these applications from `www.vbrick.com/freesoftware.asp`.

With delivery of the multicast traffic confirmed at `r5`, your attention now shifts to the ingress routers, `r1` and `r2`, so that you can confirm the forwarding path for the 225.0.0.1 multicast traffic:

```
lab@r1# run show multicast route

Family: INET
Group           Source prefix     Act Pru InIf  NHid  Session Name
225.0.0.1       10.0.5.200     /32 A   P  2     0     MALLOC
```

The presence of a multicast cache entry for the MS1 source and the 225.0.0.1 group is a good sign. A key item to note in this display is that `r1` is not forwarding this traffic, as indicated by the `P` value in the `Pru` column. In this case, the `P` indicates that the prefix has been pruned, in other words, that there are no outgoing interfaces. Adding the `extensive` switch allows you to determine the incoming rate of the multicast traffic from MS1:

```
[edit]
lab@r1# run show multicast route extensive

Family: INET
Group           Source prefix     Act Pru NHid  Packets    IfMismatch Timeout
225.0.0.1       10.0.5.200     /32 A   P  0     293155     0          360
    Upstream interface: fe-0/0/0.0
    Session name: MALLOC
    Forwarding rate: 13 kBps (32 pps)
        Source: 10.0.5.200

Family: INET6
Group           Source prefix     Act Pru NHid  Packets    IfMismatch Timeout
```

The indication that some 32 packets are received each second from the 10.0.5.200 source serves to further validate that the packet count seen on r5's fe-0/0/3 interface is in fact the multicast traffic generated by MS1. The same command is now issued on r2:

```
[edit]
lab@r2# run show multicast route extensive

Family: INET
Group           Source prefix     Act Pru NHid  Packets    IfMismatch Timeout
225.0.0.1       10.0.5.200     /32 A   F  97   219820     0          360
    Upstream interface: fe-0/0/0.0
    Session name: MALLOC
    Forwarding rate: 14 kBps (34 pps)
        Source: 10.0.5.200

Family: INET6
Group           Source prefix     Act Pru NHid  Packets    IfMismatch Timeout
```

The display from r2 confirms that it is forwarding the traffic from 10.0.5.200, as indicated by the F in the Pru column. The display further indicates that the traffic is being sent out the interface associated with a NHid of 97. The show multicast next-hops command displays the interface to NHid mapping in a most convenient fashion:

```
[edit]
lab@r2# run show multicast next-hops
Family: INET
ID      Refcount  KRefcount Downstream interface
97              2          1 fe-0/0/2.0


Family: INET6
```

The downstream interface listing of fe-0/0/2 tells you that traffic from MS1 is being sent from r2 to r3 over the 10.0.4.0/30 subnet. Although not shown, you can assume that a monitor interface at-0/2/1 command issued at r5 confirms that some 32 packets per second are being received from r3. The commands shown next also indicate that multicast traffic from MS1 to MR1 is being sent over the ATM link between r3 and r5, which is not in accordance with the criteria posed in this scenario:

```
[edit]
lab@r3# run show multicast route

Family: INET
Group           Source prefix     Act Pru InIf  NHid  Session Name
225.0.0.1       10.0.5.200     /32 A   F  6     117   MALLOC
```

```
Family: INET6
Group           Source prefix     Act Pru InIf  NHid  Session Name

[edit]
lab@r3# run show multicast next-hops
Family: INET
ID      Refcount  KRefcount Downstream interface
117            2          1 at-0/1/0.0


Family: INET6
```

Well, the good news is that multicast traffic is flowing between the source and receiver subnet; the bad news is that the forwarding path does not comply with the restrictions imposed by this configuration scenario. A JNCIE candidate will have to be familiar with the operation of DVMRP to understand why this forwarding path was chosen, and more importantly, how to influence DVMRP's forwarding paths.

In this case, r3 is receiving DVMRP advertisements for the 10.0.5/24 subnet from both r1 and r2. Because the route metrics are the same, r3 chooses the upstream router based on the lower of the two IP addresses. This results in the generation of a poison reverse advertisement to r2, but not to r1, which in turn leads to the prune/forwarding state observed at r1 and r2, respectively. Note that r2 is not forwarding to r4 at this time because r4 has generated a DVMRP prune for the S,G pair in question:

```
[edit]
lab@r4# run show dvmrp prunes
Group           Source Prefix      Timeout Neighbor
225.0.0.1       10.0.5.200     /32    1565 10.0.4.10
```

The prune was generated by r4 because it has no downstream interfaces associated with this S,G pair.

Modifying DVMRP Metrics

With two equal-cost paths between r5 and the source network, r5 is installing itself as a downstream node for r3, but not r4. As previously noted, this is due to r3's lower IP address. To achieve the desired forwarding path, you need to make r5 prefer the route advertisements for the 10.0.5/24 source network from r4 over the same route that it receives from r3. Because DVMRP interface-based metric settings affect only the cost of directly connected networks, as opposed to adjusting the metrics for all updates sent out that interface, a policy-based approach for DVMRP metric setting is used in this example. Note that setting a DVMRP interface metric on r1 and/or r2 will not result in the desired forwarding path in this example because r5 still receives two equal-cost routes for the 10.0.5/24 source network, given the current test bed topology. The policy-related changes made to r3 are shown here with highlights:

```
[edit]
lab@r3# show policy-options policy-statement dvmrp-metric
```

```
from {
    route-filter 10.0.5.0/24 exact;
}
then {
    metric 3;
    accept;
}

[edit]
lab@r3# show protocols dvmrp
rib-group dvmrp-rg;
export dvmrp-metric;
interface all {
    hold-time 40;
}
interface fxp0.0 {
    disable;
}
```

After committing the *dvmrp-metric* related export policy at r3, the results are easy to confirm at r5:

```
[edit protocols]
lab@r5# run show route 10.0.5/24

inet.0: 36 destinations, 44 routes (36 active, 0 holddown, 4 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.5.0/24        *[IS-IS/18] 00:08:15, metric 30
                    > to 10.0.2.10 via so-0/1/0.0
                      to 10.0.2.2 via at-0/2/1.0

inet.2: 26 destinations, 26 routes (26 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.5.0/24        *[DVMRP/110] 00:00:45, metric 3
                    > to 10.0.2.10 via so-0/1/0.0
```

With r5 now preferring the route from r4, it generates a poison reverse update back to r4, thereby producing the desired forwarding state at r4:

```
lab@r4# run show multicast route

Family: INET
Group           Source prefix     Act Pru InIf  NHid  Session Name
225.0.0.1       10.0.5.200     /32 A   F  6     119   MALLOC
```

```
Family: INET6
Group           Source prefix     Act Pru InIf  NHid  Session Name


[edit]
lab@r4# run show multicast next-hops
Family: INET
ID      Refcount  KRefcount Downstream interface
119            2          1 so-0/1/1.0


Family: INET6
```

The output from r4 confirms that traffic associated with the 10.0.5.200, 225.0.0.1 S,G pairing is being forwarded to r5 over its so-0/1/1 POS interface, in accordance with your restrictions. With r4 now forwarding the multicast traffic, you expect to see that r3 has pruned this S,G entry and that it has issued a prune back to r2:

```
[edit]
lab@r3# run show multicast route

Family: INET
Group           Source prefix     Act Pru InIf  NHid  Session Name
225.0.0.1       10.0.5.200     /32 I   P  6     0     MALLOC

Family: INET6
Group           Source prefix     Act Pru InIf  NHid  Session Name


[edit]
lab@r3# run show dvmrp prunes
Group           Source Prefix      Timeout Neighbor
225.0.0.1       10.0.5.200     /32    6698 10.0.4.2
```

The results match your expectations and provide confirmation that DVMRP has been correctly configured and is operating in full accordance with all provided guidelines and restrictions. Before moving on, you should confirm that multicast traffic from group 225.0.0.1 is still being delivered to the MR1 subnet. Although not shown, you can assume that the multicast traffic is still being correctly delivered to the MR1 subnet.

DVMRP Tracing

Although DVMRP appears to be working fine in the current test bed, protocol tracing is an excellent way to gain additional familiarity with the normal operation of any protocol. The tracing

configuration shown here provides a fair dose of meat without overwhelming the operator with meaningless details:

```
[edit protocols dvmrp]
lab@r3# show traceoptions
file dvmrp;
flag neighbor detail;
```

The trace output capture shown next was taken after the DVMRP protocol is deactivated and reactivated on r3:

```
[edit]
lab@r3# activate protocols dvmrp

[edit]
lab@r3# commit
commit complete

[edit]
lab@r3#
Mar 28 18:01:03 DVMRP RECV 10.0.4.14 -> 224.0.0.4 len 16 Probe: Vers: 3.255
   flags: PGM genid: 0x3ef3843e nbrs: 10.0.4.13
Mar 28 18:01:03 DVMRP SENT 10.0.4.13 -> 10.0.4.14 len 42 Report: Vers: 3.255
        mask 255.255.255.252:
                10.0.4.0          1
Mar 28 18:01:03
        mask 255.255.255.255:
                10.0.3.3          1
Mar 28 18:01:03
        mask 255.255.255.252:
                10.0.2.12         1
                10.0.2.4          1
                10.0.2.0          1
Mar 28 18:01:03 DVMRP RECV 10.0.4.2 -> 224.0.0.4 len 16 Probe: Vers: 3.255
   flags: PGM genid: 0x45f3843e nbrs: 10.0.4.1
Mar 28 18:01:03 DVMRP SENT 10.0.4.1 -> 10.0.4.2 len 42 Report: Vers: 3.255
        mask 255.255.255.252:
                10.0.4.12         1
. . .
Mar 28 18:01:14 DVMRP SENT 10.0.2.2 -> 224.0.0.4 len 44 Report: Vers: 3.255
        mask 255.255.255.252:
                10.0.2.8          34
Mar 28 18:01:14
        mask 255.255.255.0  :
                172.16.30.0       34
```

```
Mar 28 18:01:14
        mask 255.255.255.252:
                10.0.8.8          34
                10.0.8.4          34
Mar 28 18:01:14
        mask 255.255.255.255:
                10.0.3.5          34
. . .
```

The trace output shows DVMRP neighbor probes, and route reports being sent and received. The truncated capture also shows some instances of DVMRP poison reverse, as indicated by metrics higher than 32.

DVMRP Summary

DVRP builds a multicast routing table that is used for RPF checks in support of multicast forwarding. Because the `inet.2` routing table is designed to hold multicast routing information, it is recommended that you configure your DVMRP RIB groups to populate the `inet.2` table. Successful RPF checks require the presence of directly connected interface routes in the routing table used by DVMRP. You configure an interface RIB group and link it to the DVMRP routing table to install copies of interface routes for RPF checking.

DVMRP operates neighbor to neighbor using periodic updates, in a behavior that is similar to RIP. DVMRP builds truncated multicast trees based on downstream nodes using poison reverse to indicate that they consider the upstream node to be closer to the multicast source, thereby showing the node's desire to be added to the particular source tree. When metrics are equal, a DVMRP router breaks a tie by selecting the advertisement associated with the lowest IP address. You can adjust metrics (using policy or with direct interface settings for directly connected networks) to influence the multicast topology. You can trace the operation of DVMRP to observe neighbor establishment and routing exchanges.

This section demonstrated a number of multicast monitoring and troubleshooting techniques that can be applied to virtually any multicast network. These commands and techniques include displaying multicast routes, multicast next hops, statistics; examining RPF results; and monitoring interface traffic loads to detect the presence of multicast.

Although DVMP has lost favor to various flavors of PIM, the protocol is still in use and is supported by JUNOS software. A well-prepared JNCIE candidate will take time to hone their DVMRP configuration and troubleshooting skills to ensure that they are not caught off guard by a DVMRP configuration scenario.

Protocol Independent Multicast

This section covers various Protocol Independent Multicast (PIM) configuration scenarios. PIM gets its name from the fact that it makes use of the main routing table (`inet.0`) for RPF purposes, regardless of what routing protocols are used to populate the main routing table. JUNOS software supports PIM versions 1 and 2 in dense, sparse, or sparse-dense modes of operation.

In dense mode, PIM uses a flood and prune model that is similar to DVMRP, the primary exception being that initial flooding is based on interfaces with detected neighbors yielding a broadcast-like behavior that is referred to as a *broadcast tree*. The broadcast tree morphs into a minimal spanning tree once all prune actions have occurred, however.

In most cases, PIM is deployed in a sparse mode (PIM-SM). PIM-SM makes use of a Rendezvous Point (RP) that acts as the root of a shared tree for all senders and some particular group range. The shared tree is denoted as *,G.

When a receiver first joins the shared tree for a given group, it receives traffic that is sent to this group (regardless of source address) using the shared tree. After receiving traffic from a particular sender, S, the receiver establishes a source rooted tree, denoted as a S,G state, and then proceeds to prune itself from the RP-based shared tree for that particular S,G pair. The net result is that receivers make initial contact with active senders through the RP and its shared tree, with the intent of signaling source-specific trees (also known as shortest path trees) for active senders to achieve optimal forwarding paths between each sender and receiver.

PIM sparse mode makes use of *register* messages to tunnel multicast traffic from the first hop router to the RP using unicast packets. Once received by the RP, the register encapsulation is stripped off and the native multicast traffic is flooded down the shared RP tree. A tunnel services (TS) PIC is required in nodes with active multicast sources to support the register encapsulation process, and at the RP to permit de-encapsulation of the register messages. A TS PIC is not required for PIM dense mode, or in a PIM-SM network when the only active sources are directly attached to the RP.

Although RPs can be statically identified, most PIM-SM networks are not configured this way because a statically defined RP represents a single point of failure; various mechanisms are defined within PIM to provide RP redundancy and automatic RP discovery. The *bootstrap* protocol functions to identify candidate RPs, which are then advertised in a hop-by-hop manner to all PIM routers. A hashing function is used by each router to map particular multicast groups to a given RP. The bootstrap mapping function automatically distributes the load among multiple RPs, and allows for the failover of multicast groups to another candidate RP in the event of RP failure. The bootstrap protocol is supported in PIM version 2 only, however.

In contrast, the non-RFC based *Auto-RP* mechanism makes use of dense mode flooding on reserved group addresses to inform mapping agents of candidate RPs, and to allow the mapping agents to inform other routers of the RP that has been chosen. Unlike bootstrap-based RP election, Auto-RP procedures normally result in the election of a single RP that handles all traffic for a given group range.

Any-Cast refers to a technique that uses Auto-RP procedures in conjunction with duplicate RP addresses and MSDP to facilitate load sharing among multiple RPs elected through Auto-RP. Any-Cast is discussed in detail in the "MSDP" section later in this chapter.

PIM Dense Mode

This section covers PIM dense mode (PIM-DM) configuration and testing. To complete this scenario, you must configure your network according to these requirements:

- Remove DVMRP-related configuration from `r1` through `r5`.
- Configure PIM-DM on `r1`, `r2`, `r3`, `r4`, and `r5` so that multicast traffic sent from MS1 to group 225.0.0.1 is made available to MR1.

- Make sure that r1 is not elected a DR on any segments with neighbors.
- Ensure that traffic from MS1 to MR1 transits the 10.0.2.8/30 subnet between r4 and r5.
- Do not alter the IGMP or interface configuration currently in place at r5.

The requirement listing for the PIM dense mode scenario is virtually identical to the goals specified for DVMRP. In effect, your goal is to replace DVMRP with PIM dense mode while maintaining existing multicast functionality. Refer back to Figure 4.2 for topology details as needed.

Configuring PIM Dense Mode

You begin your PIM-DM configuration task by removing all DVMRP-related configuration from r1 through r5. Do not forget to remove the metric-related policy at r3, as well as the DVMRP RIB group configuration. The commands shown here remove all traces of DVMRP configuration from r4:

```
[edit]
lab@r4# delete protocols dvmrp

[edit]
lab@r4# delete routing-options interface-routes

[edit]
lab@r4# delete routing-options rib-groups
```

To confirm the complete removal of all DVMRP-related configuration, the candidate configuration is compared to the IS-IS baseline configuration from Chapter 1:

```
[edit]
lab@r4# show | compare r4-baseline-isis
[edit]
- version 5.6R1.3;
+ version 5.6R2.4;
```

Note that the only difference between the two configurations is the JUNOS software version, which was upgraded to R2.4 when the newer 5.6 release was made publicly available as this book was being written. You should issue similar commands on all routers in the multicast topology before proceeding. After deleting the DVMRP-related settings at r5, the following capture confirms that the existing IGMP and MR1 interface configuration has been correctly left in place:

```
[edit]
lab@r5# show | compare r5-baseline-isis
[edit]
- version 5.6R1.3;
+ version 5.6R2.4;
[edit interfaces]
+    fe-0/0/3 {
```

```
+       unit 0 {
+           family inet {
+               address 172.16.30.1/24;
+           }
+       }
+   }
[edit protocols]
+   igmp {
+       query-response-interval 5;
+       interface fe-0/0/3.0 {
+           version 3;
+           static {
+               group 225.0.0.1;
+           }
+       }
+   }
```

You begin PIM-DM configuration at `r3` by creating the PIM instance as shown next:

```
[edit protocols]
lab@r3# set pim interface all mode dense

[edit protocols]
lab@r3# set pim interface fxp0 disable
```

Note that you have enabled PIM-DM by virtue of placing all PIM interfaces into dense mode. The use of the `interface all` shortcut results in the need to explicitly disable PIM on the router's OoB interface. The initial PIM stanza on `r3` is now displayed and committed:

```
[edit protocols]
lab@r3# show pim
interface all {
    mode dense;
}
interface fxp0.0 {
    disable;
}

[edit protocols]
lab@r3# commit
commit complete
```

The basic PIM configuration shown should be enough to get PIM-DM up and running. Note that the forwarding path requirements of this scenario may force some modifications to your initial PIM configuration down the road. Before proceeding to the verification section, you should ensure that all routers in the multicast test bed have a PIM-DM configuration that is similar to

that shown for r3. Note that r1's configuration must take into account the fact that it cannot be elected as a PIM DR on segments with attached neighbors. Setting r1's priority to 0 on all of its interfaces configures the prescribed behavior:

```
[edit]
lab@r1# show protocols pim
interface all {
    mode dense;
    priority 0;
}
interface fxp0.0 {
    disable;
}
```

Verifying PIM Dense Mode

The verification of the PIM-DM control plane follows the general approach demonstrated for DVMRP; specifically, you need to verify that PIM is enabled on the correct interfaces, that PIM neighbor discovery is functioning correctly, and that RPF checks through the inet.0 routing table are working. You begin with the determination of PIM interface status at r5:

```
[edit]
lab@r5# run show pim interfaces
Instance: PIM.master

Name                 Stat Mode       IP V State Count DR address
at-0/2/1.0           Up   Dense       4 2 P2P       1
fe-0/0/0.0           Up   Dense       4 2 DR        0 10.0.8.6
fe-0/0/1.0           Up   Dense       4 2 DR        0 10.0.8.9
fe-0/0/3.0           Up   Dense       4 2 DR        0 172.16.30.1
lo0.0                Up   Dense       4 2 DR        0 10.0.3.5
so-0/1/0.0           Up   Dense       4 2 P2P       1
```

The PIM interface display confirms that r5 is running PIM in dense mode on all transit interfaces, including the links to r6 and r7, which are not used in the current multicast test bed. The added highlights call out that r5's at-0/2/1 interface is Up, that it is considered a point-to-point interface, and that a single neighbor has been detected on this interface. Note that point-to-point interfaces do not elect a PIM designated router (DR), which can be seen in r5's output. The display also confirms that PIM version 2 is the default in the 5.6 version of JUNOS software currently deployed in the test bed. Note that no PIM neighbors have been detected on r5's Fast Ethernet links. This is because r6 and r7 are not configured to run PIM (or multicast in general) in the current multicast topology. PIM neighbor discovery is confirmed, once again at r5:

```
[edit]
lab@r5# run show pim neighbors
Instance: PIM.master
```

```
Interface            IP V Mode        Option      Uptime Neighbor addr
at-0/2/1.0            4 2             HPL       00:07:40 10.0.2.2
so-0/1/0.0            4 2             HPL       00:07:32 10.0.2.10
```

The output confirms that `r5` has correctly detected both `r3` and `r4` as PIM neighbors. The *`Option`* flags indicate that the neighbor supports the Hello Option Holdtime, the Hello Option DR Priority, and the LAN Option LAN Prune Delay, respectively. The `Mode` column is used to indicate the PIM mode when it is known. The mode can be `Sparse`, `Dense`, `SparseDense`, or as shown in this example, blank (unknown). Note that the mode is always unknown when running PIM version 2.

The `PIM.master` indication simply means that you are viewing the main routing instance's PIM-related information; it does not, unfortunately, indicate that you have reached multicast nirvana by being designated a "PIM Master." The PIM interface display shown previously confirmed that all of `r5`'s PIM interfaces are correctly set for dense mode operation. To obtain details on negotiated values and timer settings, add the `detail` switch:

```
[edit]
lab@r5# run show pim neighbors detail
Instance: PIM.master

Interface: at-0/2/1.0
    Address: 10.0.2.1,  IPv4, PIM v2, Mode: Dense
        Hello Option Holdtime: 65535 seconds
        Hello Option DR Priority: 1
        Hello Option LAN Prune Delay: delay 500 ms override 2000 ms
                                      Join Suppression supported
. . .
```

The next command verifies that the PIM RPF check is working through the main routing table:

```
lab@r5# run show multicast rpf 10.0.5/24
Multicast RPF table: inet.0, 24 entries

10.0.5.0/24
    Protocol: IS-IS
    Interface: at-0/2/1.0
    Neighbor: 10.0.2.2
```

The output confirms that the RPF check to the 10.0.5/24 source network has succeeded, and that the `inet.0` routing table is used for this purpose via a route that was learned through IS-IS. Your next command draws an interesting contrast between DVMRP and PIM, in that in the former case both an IS-IS and a DVMRP version of the 10.0.5/24 route were present (with the DVMRP copy being placed in the `inet.2` table):

```
[edit]
lab@r5# run show route 10.0.5/24

inet.0: 24 destinations, 24 routes (24 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

```
10.0.5.0/24        *[IS-IS/18] 02:20:37, metric 30
                      to 10.0.2.10 via so-0/1/0.0
                    > to 10.0.2.2 via at-0/2/1.0
```

The single IS-IS-based route entries should hammer home the fact that PIM does not require a special routing table for RPF checking. Recall that PIM-DM does not make use of an RP; the absence of a PIM RP is quickly confirmed at r3, and the display (or lack of) serves to draw an interesting contrast with PIM-SM, which is deployed in a later section:

```
[edit protocols]
lab@r3# run show pim rps
Instance: PIM.master

Family: INET
RP address      Type      Holdtime Timeout Active groups Group prefixes
Family: INET6
RP address      Type      Holdtime Timeout Active groups Group prefixes
```

Before moving on to the PIM forwarding plane, you quickly confirm that r1's priority setting correctly prevents its election as a PIM DR when another PIM router is present:

```
[edit]
lab@r1# run show pim interfaces
Instance: PIM.master

Name                  Stat Mode       IP V State Count DR address
fe-0/0/0.0            Up   Dense       4 2 NotDR     2 10.0.5.2
fe-0/0/1.0            Up   Dense       4 2 NotDR     1 10.0.4.13
fe-0/0/2.0            Up   Dense       4 2 NotDR     1 10.0.4.6
fe-0/0/3.0            Up   Dense       4 2 NotDR     1 10.0.4.17
lo0.0                 Up   Dense       4 2 DR        0 10.0.6.1
```

The output shows that r1 is not considered the DR on any interface with at least one neighbor detected. This display confirms the desired DR election behavior at r1.

Verification of the PIM-DM forwarding plan begins with an examination of the multicast routes at r1 and r2, which are the first hop routers connected to the MS1 source network:

```
[edit]
lab@r1# run show multicast route

Family: INET
Group           Source prefix     Act Pru InIf  NHid  Session Name
225.0.0.1       10.0.5.200      /32 A   F  2     87    MALLOC

Family: INET6
Group           Source prefix     Act Pru InIf  NHid  Session Name
```

The output from r1 confirms that the S,G entry for MS1 exists, and that r1 is forwarding this traffic. The multicast next hop mapping to NHid 87 on r1 is displayed next:

```
[edit]
lab@r1# run show multicast next-hops
Family: INET
ID      Refcount  KRefcount Downstream interface
87             2          1 fe-0/0/1.0
                            fe-0/0/3.0


Family: INET6
```

The output indicates that r1 is forwarding the multicast traffic to r3 over its fe-0/0/1 interface and to r4 over the fe-0/0/3 interface. Before leaving r1, you display its PIM join state:

```
[edit]
lab@r1# run show pim join
Instance: PIM.master Family: INET
Group: 225.0.0.1
    Source: 10.0.5.200
    Flags: dense
    Upstream interface: fe-0/0/0.0
Instance: PIM.master Family: INET6
```

The PIM join display shows that r1 has joined the S,G groups associated with the MS1 source, and that the protocol is operating in dense mode. Given that r1 is forwarding for this S,G pair, you find yourself starting to wonder what r2 is doing with the same traffic. The following command rapidly dispatches your curiosity on this front:

```
[edit]
lab@r2# run show multicast route

Family: INET
Group           Source prefix     Act Pru InIf  NHid  Session Name
225.0.0.1       10.0.5.200      /32 A   F  2     87    MALLOC

Family: INET6
Group           Source prefix     Act Pru InIf  NHid  Session Name
```

Interesting; it would seem that r2 is also forwarding the 10.0.5.200, 225.0.0.1 multicast traffic. You note that this situation differs from the default behavior observed with DVMRP in the previous section, and move on to display the multicast next hop(s) at r2. In this case, you opt to use the extensive switch with the show pim join command:

```
[edit]
lab@r2# run show pim join extensive
```

```
Instance: PIM.master Family: INET
Group: 225.0.0.1
    Source: 10.0.5.200
    Flags: dense
    Upstream interface: fe-0/0/0.0
    Downstream interfaces:
        fe-0/0/1.0 (Pruned timeout 252)
        fe-0/0/2.0 (Pruned timeout 115)
        fe-0/0/3.0 (Pruned timeout 119)

Instance: PIM.master Family: INET6
```

The additional information provided by the `extensive` switch proves quite useful in this case. The highlights call out that `r2` is really not forwarding the S,G traffic at this moment by virtue of its having received PIM prunes on all of its downstream interfaces. Note that the nature of PIM-DM flood and prune behavior will result in `r2` periodically timing out the prunes, which in turn results in `r2` forwarding the S,G traffic for a few moments before it receives another prune. The fact that a multicast network is always in some state of flux can complicate your verification and confirmation tasks, especially if you are the type of person who expects a steady state of affairs!

Knowing that `r1` was last seen forwarding the S,G traffic to both `r3` and `r4`, you issue similar commands at `r3` to further discover the PIM forwarding topology:

```
[edit protocols]
lab@r3# run show multicast route

Family: INET
Group           Source prefix     Act Pru InIf  NHid  Session Name
225.0.0.1       10.0.5.200     /32 A   F  5     96    MALLOC

Family: INET6
Group           Source prefix     Act Pru InIf  NHid  Session Name
```

The multicast route display confirms that `r3` is forwarding the S,G traffic, but to where? Analyzing the PIM join state at `r3` provides your answer:

```
[edit protocols]
lab@r3# run show pim join extensive
Instance: PIM.master Family: INET
Group: 225.0.0.1
    Source: 10.0.5.200
    Flags: dense
    Upstream interface: fe-0/0/0.0
    Downstream interfaces:
        fe-0/0/1.0 (Assert Lost)
```

```
        at-0/1/0.0
        so-0/2/0.100 (Pruned timeout 110)

Instance: PIM.master Family: INET6
```

The highlights in this output are quite telling. Here you can see that r3 has received a prune from r4, which makes a great deal of sense when you consider that r1 was seen to be forwarding the S,G traffic directly to r4. You can also see that r3 has lost the *assert* exchange with r2, which means that r3 will not forward the S,G traffic to the 10.0.4.0/30 subnet. The assert process is triggered by the receipt of S,G traffic on an interface that is in the Outgoing Interface List (OIL). When this occurs, a PIM router sends an assert message to determine which router will be the forwarder for this source. The router with the lowest metric back to the source, which is r2 in this case, wins the assert competition. In the case of equal metrics, the router with the highest IP address "wins." Note that r2 is not forwarding to this subnet due to the receipt of a prune from r3. The final highlight in the display confirms that r3 is forwarding the S,G traffic to r5 over its at-0/1/0 interface. A quick look at r5's PIM join state confirms that it is forwarding the S,G traffic to MR1 over its fe-0/0/3 interface:

```
[edit]
lab@r5# run show pim join extensive
Instance: PIM.master Family: INET
Group: 225.0.0.1
    Source: 10.0.5.200
    Flags: dense
    Upstream interface: at-0/2/1.0
    Downstream interfaces:
        fe-0/0/3.0
        so-0/1/0.0 (Assert Lost)

Instance: PIM.master Family: INET6
```

At this time, r4 is observed not to be forwarding the S,G traffic to r5:

```
[edit]
lab@r4# run show pim join extensive
Instance: PIM.master Family: INET
Group: 225.0.0.1
    Source: 10.0.5.200
    Flags: dense
    Upstream interface: fe-0/0/1.0
    Downstream interfaces:
        fe-0/0/2.0 (Assert Lost)
        so-0/1/0.100
        so-0/1/1.0 (Pruned timeout 20)

Instance: PIM.master Family: INET6
```

The results displayed thus far indicate that PIM-DM is operational and confirm that multicast traffic is being delivered to MR1. However, the displays also show that the traffic is transiting the ATM link between r3 and r5. Your rules of engagement state that this traffic should flow from r4 to r5 using the POS interface. The tricky part of this task relates to the fact that PIM itself does not advertise routes, and therefore has no metric settings that can be used to influence the PIM forwarding behavior. Recall that r5 is currently receiving two equal-cost IS-IS routes for the 10.0.5/24 source network:

```
[edit]
lab@r5# run show route 10.0.5/24

inet.0: 25 destinations, 25 routes (25 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.5.0/24        *[IS-IS/18] 00:07:08, metric 30
                      to 10.0.2.10 via so-0/1/0.0
                    > to 10.0.2.2 via at-0/2/1.0
```

Because a PIM router can select only one interface for performing RPF checks to each source, the presence of equal-cost IS-IS routes for 10.0.5/24 at r5 results in the RPF checks that use one interface, or the other, based on the IGP route that is currently active—in other words, the one with the > next to it. The policy changes shown here eliminate the equal-cost route condition by making r3 advertise the 10.0.5/24 route with a metric that is higher than that being advertised by r4.

```
[edit]
lab@r3# show | compare rollback 1
[edit protocols isis]
+   export ms1;
[edit policy-options]
    policy-statement r2 { ... }
+   policy-statement ms1 {
+       term 1 {
+           from {
+               protocol isis;
+               level 1;
+               route-filter 10.0.5.0/24 exact;
+           }
+           to level 2;
+           then {
+               metric 40;
+               accept;
+           }
+       }
+   }
```

Note that the changes also include the application of the new *ms1* policy as an `export` to the IS-IS instance. The result of these changes is that `r5` now installs the so-0/1/0 interface into its Incoming Interface List (IIL) for the 10.0.5/24 source in a deterministic manner, thereby yielding the desired forwarding behavior:

```
[edit]
lab@r5# run show route 10.0.5/24

inet.0: 25 destinations, 25 routes (25 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.5.0/24        *[IS-IS/18] 00:02:15, metric 30
                    > to 10.0.2.10 via so-0/1/0.0
```

With only one viable route to the source network now present at `r5`, its RPF table indicates that the so-0/1/0 interface is now considered an upstream interface, which means that `r5` correctly expects to receive multicast traffic from source 10.0.5/24 on its so-0/1/0 interface:

```
[edit]
lab@r5# run show multicast rpf 10.0.5.24
Multicast RPF table: inet.0, 25 entries

10.0.5.0/24
    Protocol: IS-IS
    Interface: so-0/1/0.0
    Neighbor: 10.0.2.10
```

The PIM join display at `r5` further confirms that the S,G traffic between MS1 and MR1 is now transiting the POS link between `r4` and `r5`:

```
[edit]
lab@r5# run show pim join extensive
Instance: PIM.master Family: INET
Group: 225.0.0.1
    Source: 10.0.5.200
    Flags: dense
    Upstream interface: so-0/1/0.0
    Downstream interfaces:
        fe-0/0/3.0
        at-0/2/1.0 (Assert Lost)

Instance: PIM.master Family: INET6
```

The results shown in this section verify that you have now met all specified criteria for the PIM-DM configuration scenario. Note that the policy changes made to `r3` do not actually affect `r3`'s routing to the source network, but the changes do eliminate `r5`'s ability to load balance to the 10.0.5/24 subnet.

Real World Scenario

Backwards Routing

Multicast routing has been called "routing in reverse" and "upside-down routing," in an attempt to capture the fact that multicast packets are, in essence, routed away from their source based on the packet's *source* address. The fact that the multicast traffic from MS1 is successfully delivered to the receiver, MR1, is a testament to this point. This "backwards routing" is really brought home when you consider that `r5` is the only router in the test bed with a route to MR1!

```
[edit]
lab@r3# run show route 172.16.30/24

[edit]
lab@r3#
```

The remaining routers cannot route to the 172.16.30/24 subnet because `r5`'s fe-0/0/3 interface is not running an IGP and the interface route is not being advertised into any of its routing protocols. The fact that packets sent from MS1 are being delivered to MR1 really helps to make the point that multicast routing is, well, routing in reverse!

PIM Sparse Mode

This section covers PIM sparse mode (PIM-SM) configuration and testing. Recall that sparse mode operation makes use of one or more Rendezvous Points (RPs) that form a *,G shared tree by which receivers make initial contact with multicast senders. Once a receiver begins to receive traffic from a given source, S, over the shared tree, it generates a S,G join back towards this source in an effort to establish a SPT for that source. Once the SPT is established, the last hop router can prune the S,G entry from the RP-based tree (RPT) to prevent the receipt of duplicate traffic. You will need a TS PIC installed in M-series and T-series routers that function as an RP or have directly connected multicast sources, in other words first hop routers. Recall that the TS PIC is needed to perform PIM register message encapsulation functions that were discussed previously.

Lastly, recall that several mechanisms exist to elect one or more RPs for a given group of multicast destinations. This section will demonstrate the use and testing of the Bootstrap and Auto-RP election mechanisms. The configuration of static RPs is not demonstrated here due to it relative simplicity and the fact that reliability concerns generally prevent the widescale deployment of static RPs.

Configuring PIM-SM Using Bootstrap

To complete this scenario, you must configure the multicast test bed according to these requirements:

- Remove all PIM-DM related configuration from `r1` through `r5`.

- Configure PIM-SM on r1, r2, r3, r4, and r5 so that multicast traffic sent from MS1 to group 225.0.0.1 is made available to MR1.
- You must use sparse mode only on all transit interfaces.
- Ensure that there is no single point of RP failure.
- Do not alter the IGMP or interface configuration currently in place at r5.

Refer back to Figure 4.2 as needed for the details of the multicast topology deployed in the current test bed.

Your PIM-SM configuration starts with the removal of the leftover PIM-DM configuration from r1 through r5. Do not forget to remove the metric-related IS-IS policy at r3 also:

```
[edit]
lab@r3# delete protocols pim

[edit]
lab@r3# delete protocols isis export ms1

[edit]
lab@r3# delete policy-options policy-statement ms1
```

If time permits, you should confirm the complete removal of previous configuration changes by comparing the current configuration to the baseline configuration as demonstrated in the previous section.

You begin PIM-SM configuration by deciding on a bootstrap router and RP plan. Because you are dealing with PIM-SM, it is a good idea to start your planning with a determination of what routers are equipped with TS PICs, as shown here for r5 and r3:

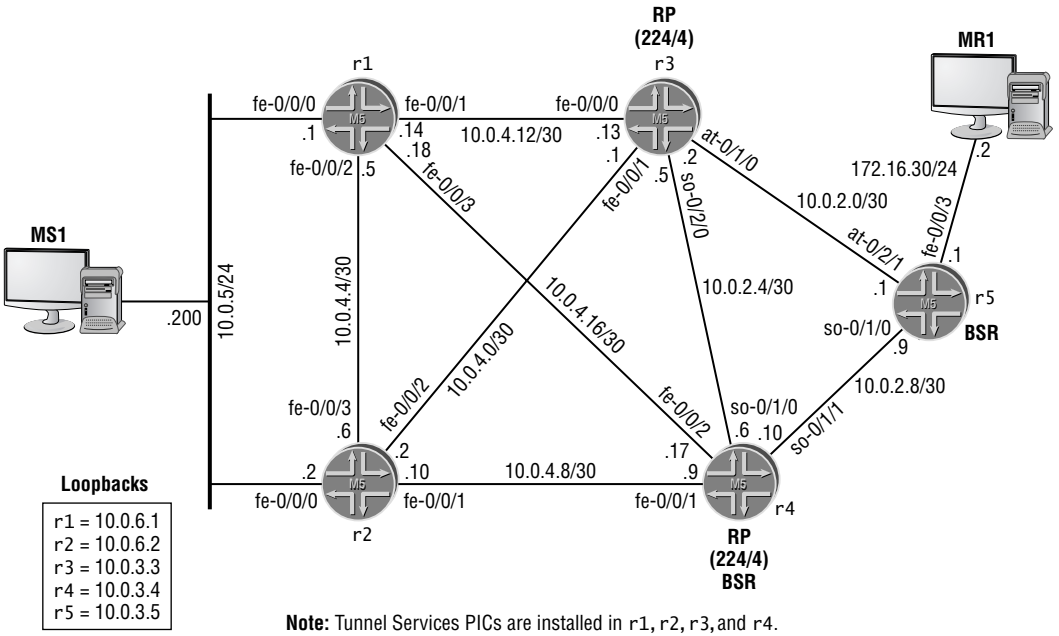
```
[edit]
lab@r5# run show chassis fpc pic-status | match tunnel

[edit]
lab@r5#

[edit]
lab@r3# run show chassis fpc pic-status | match tunnel
  PIC 3    1x Tunnel

[edit]
lab@r3#
```

The output confirms that no TS PICs are installed in r5. This newfound knowledge should have a definite impact on how you design your RP topology, as you now know that r5 *cannot* be an RP! In contrast, the display from r3 confirms the presence of a TS PIC in PIC slot 3.

The wording of the requirements in this PIM-SM scenario is intentionally vague with respect to the placement and number of RPs in the resulting design. The open-ended nature of your task amounts to a long piece of rope, in that a candidate can get into trouble if they fail to think out their PIM-SM design, and specifically in this case, how that design may impose hardware requirements on certain routers. Note that a router with a local RP configuration will allow a commit even though that router does not have the required TS PIC installed. This is by design, and in keeping with the general behavior of JUNOS software with regard to being able to configure hardware that is not installed. However, the TS PIC's absence will likely cause operational problems in your PIM-SM network, and these problems often prove very difficult to troubleshoot. The results-based grading approach of the JNCIE examination will likely exact a harsh penalty on any PIM-SM design that does not have a functional RP.

After your inventory of all routers in the multicast test bed, you conclude that `r1`, `r2`, `r3`, and `r4` are equipped with TS PICs that allow them to operate as RPs or first hop routers. Given these findings, you decide on the PIM-SM topology shown in Figure 4.5.

FIGURE 4.5 PIM-SM topology



Note: Tunnel Services PICs are installed in `r1`, `r2`, `r3`, and `r4`.

Figure 4.5 shows that `r3` and `r4` will be configured as RPs for the entire 224.0.0.0/4 address range. Note that no address range is given in your configuration criteria; therefore both RPs must be able to handle the entire class D address space to meet the redundancy requirements posed in this example. Further, your design indicates the use of bootstrap protocol to elect, and

then communicate, RP information to the remaining nodes; note that Auto-RP is prohibited in this case by the stipulation that you run sparse mode only on all interfaces. You must configure at least two bootstrap capable routers to avoid a single point of failure with regard to RP election. In this example, r4 and r5 will both function as bootstrap routers (BSRs) to provide the required redundancy. Note that there is no need for a TS PIC in a BSR, and that a BSR can also be a candidate RP as is the case with r4 in this design.

You begin configuration on r3 with the following commands:

```
[edit protocols]
lab@r3# set pim interface all mode sparse

[edit protocols]
lab@r3# set pim interface fxp0 disable
```

Note that you have configured PIM sparse mode only and that the OoB interface is once again excluded from running PIM. The next command configures r3's local RP properties:

```
[edit protocols]
lab@r3# set pim rp local address 10.0.3.3
```

It is customary to use the router's lo0 address as the RP address for added robustness in the face of interface failures. The completed PIM stanza is displayed at r3:

```
[edit protocols]
lab@r3# show pim
rp {
    local {
        address 10.0.3.3;
    }
}
interface all {
    mode sparse;
}
interface fxp0.0 {
    disable;
}
```

The PIM configuration for r4 is similar, and is shown here:

```
[edit protocols pim]
lab@r4# show
rp {
    bootstrap-priority 10;
    local {
        address 10.0.3.4;
    }
}
```

```
interface all {
    mode sparse;
}
interface fxp0.0 {
    disable;
}
```

A key difference in r4's configuration is the presence of a bootstrap priority. The non-zero setting makes r4 a BSR candidate. Recall that the BSR functions to build, and then disseminate, a list of candidate RPs for advertised groups ranges. Each router then runs a hashing algorithm that results in all routers selecting the same candidate RP for a given range of group addresses. r4 is also configured to function as an RP for the 224/4 group range. The PIM-SM configuration of r5 is displayed:

```
[edit]
lab@r5# show protocols pim
rp {
    bootstrap-priority 10;
}
interface all {
    mode sparse;
}
interface fxp0.0 {
    disable;
}
```

Note that r5 is not configured with local RP parameters, which is in keeping with the PIM-SM design shown earlier in Figure 4.5. r5 is configured with a non-zero bootstrap priority, thus allowing it to function as a BSR as needed. The PIM-SM configuration of r1 and r2 are virtually identical. r1's configuration is shown next:

```
[edit]
lab@r1# show protocols pim
interface all {
    mode sparse;
}
interface fxp0.0 {
    disable;
}
```

Note that r1 and r2 are not configured to function as a BSR or RP. Be sure that you commit all your PIM-related changes before proceeding to the verification section.

Verifying PIM Sparse Mode

The verification of the PIM-SM control plane follows the general approach demonstrated for PIM-DM; specifically, you need to verify that PIM is enabled on the correct interfaces in the

correct mode, that PIM neighbor discovery is functioning correctly, and that RPF checks through the `inet.0` routing table are working. In addition, you need to confirm that bootstrap-based BSR election and RP selection are working properly. You begin with the determination of PIM interface status at `r3`:

```
[edit]
lab@r3# run show pim interfaces
Instance: PIM.master

Name                 Stat Mode       IP V State Count DR address
at-0/1/0.0           Up   Sparse      4 2 P2P       1
fe-0/0/0.0           Up   Sparse      4 2 DR        1 10.0.4.13
fe-0/0/1.0           Up   Sparse      4 2 NotDR     1 10.0.4.2
fe-0/0/2.0           Up   Sparse      4 2 DR        0 172.16.0.13
fe-0/0/3.0           Up   Sparse      4 2 DR        0 10.0.2.14
lo0.0                Up   Sparse      4 2 DR        0 10.0.3.3
pd-0/3/0.32768       Up   Sparse      4 2 P2P       0
pe-0/3/0.32770       Up   Sparse      4 2 P2P       0
so-0/2/0.100         Up   Sparse      4 2 P2P       1
```

The PIM interface display confirms that `r3` is running PIM in sparse mode only on all expected interfaces. The added highlights call out the effects of the TS PIC, in that PIM Encapsulation (`pe`) and PIM De-capsulation (`pd`) interfaces have automatically been created by the PIM entity on `r3`. Note that explicit configuration is not necessary to support PIM register encapsulation. The `pe` and `pd` interfaces are created automatically based on the presence of a TS PIC and related PIM configuration. You move on to the verification of PIM neighbor status, again on `r3`:

```
[edit]
lab@r3# run show pim neighbors
Instance: PIM.master

Interface           IP V Mode        Option      Uptime Neighbor addr
at-0/1/0.0           4 2             HPL       00:55:37 10.0.2.1
fe-0/0/0.0           4 2             HPL       01:10:26 10.0.4.14
fe-0/0/1.0           4 2             HPL       01:10:26 10.0.4.2
so-0/2/0.100         4 2             HPL       01:10:26 10.0.2.6
```

The display confirms that `r3` has detected four PIM neighbors, which is in keeping with the current multicast test bed topology. A quick RPF check at `r5` proves that PIM-SM is using the `inet.0` routing table for RPF checks:

```
[edit]
lab@r5# run show multicast rpf 10.0.5/24
```

```
Multicast RPF table: inet.0, 25 entries

10.0.5.0/24
    Protocol: IS-IS
    Interface: so-0/1/0.0
    Neighbor: 10.0.2.10
```

Your next verification task is to confirm proper BSR election. You begin at r5:

```
[edit]
lab@r5# run show pim bootstrap
Instance: PIM.master

BSR             Pri Local address   Pri State      Timeout
10.0.3.5         10 10.0.3.5         10 Elected         22
```

The output shows that r5 has elected itself as the PIM domain's BSR, as denoted by the BSR and local addresses being the same. The router's modified priority setting is also shown. The display at r5 provides a nice contrast when compared to that obtained at r3:

```
[edit]
lab@r3# run show pim bootstrap
Instance: PIM.master

BSR             Pri Local address   Pri State      Timeout
10.0.3.5         10 10.0.3.3          0 InEligible     118
```

This output confirms that r3 is not BSR eligible due to its default priority setting. A similar bootstrap status is also seen on r1 and r2 (not shown). The BSR status of r4 is now checked:

```
[edit]
lab@r4# run show pim bootstrap
Instance: PIM.master

BSR             Pri Local address   Pri State      Timeout
10.0.3.5         10 10.0.3.4         10 Candidate      122
```

The output from r4 shows that r5 has won the BSR contention process while also confirming that r4 is a candidate BSR due to its non-zero priority setting. Because r4 and r5 have the same BSR priority, the election's outcome was determined by an IP address comparison that has the highest address taking all. With BSR election confirmed, you move on to the verification of RP election, this time starting at r4:

```
[edit]
lab@r4# run show pim rps
Instance: PIM.master

Family: INET
```

```
RP address      Type      Holdtime Timeout Active groups Group prefixes
10.0.3.3        bootstrap      150     103             0 224.0.0.0/4

10.0.3.4        bootstrap      150     103             0 224.0.0.0/4

10.0.3.4        static           0    None             0 224.0.0.0/4

Family: INET6
RP address      Type      Holdtime Timeout Active groups Group prefixes
```

The highlights call out that r4 has learned its own RP address through local (static) assignment, as well as from the bootstrap protocol. Also of note is that r4 has learned of r3's RP capability through the bootstrap protocol. Though not shown, a similar display is seen on r5. Because proper PIM-SM operation is contingent on all routers electing the same RP for a given group range, it is well worth the time to confirm that all routers in the multicast test bed are displaying both candidate RPs. r1 has the expected RP display, given the current configuration in the multicast test bed:

```
[edit]
lab@r1# run show pim rps
Instance: PIM.master

Family: INET
RP address      Type      Holdtime Timeout Active groups Group prefixes
10.0.3.3        bootstrap      150     149             0 224.0.0.0/4

10.0.3.4        bootstrap      150     149             0 224.0.0.0/4

Family: INET6
RP address      Type      Holdtime Timeout Active groups Group prefixes
```

In stark contrast, r2 does not return the expected RP election results:

```
[edit protocols pim]
lab@r2# run show pim rps
Instance: PIM.master

Family: INET
RP address      Type      Holdtime Timeout Active groups Group prefixes
Family: INET6
RP address      Type      Holdtime Timeout Active groups Group prefixes

[edit protocols pim]
lab@r2#
```

The output from r2 indicates that there is a problem in your PIM-SM network as it has failed to detect either of the two RP candidates. You should resolve this issue before proceeding.

Real World Scenario

Troubleshooting a PIM-SM RP Election Problem

The displays shown in this section indicate that r2 is the only router in the test bed that has not successfully detected the presence of one or more RPs. While various approaches might be taken to troubleshoot this problem, this author believes that PIM tracing is often a good way to start. The tracing configuration shown here is committed on r2:

```
[edit]
lab@r2# show protocols pim
traceoptions {
    file pim;
    flag general detail;
    flag hello detail;
    flag rp detail;
}
interface all {
    mode dense;
}
interface fxp0.0 {
    disable;
}
```

Note that the rp flag is invaluable when you are concerned about either Auto-RP or bootstrap-related problems. After the tracing changes are committed, you observe the output shown next:

```
[edit]
lab@r2# run monitor start pim

lab@r2#
Apr  4 02:19:15 CHANGE   224.0.1.24.10.0.5.200/64  PIM      pref 105/0 metric
   <Delete Int>
Apr  4 02:19:15 rt_close: 1 route proto PIM.master
Apr  4 02:19:15
Apr  4 02:19:16 PIM fe-0/0/0.0 RECV 172.16.30.1 -> 224.0.0.13 V2 Bootstrap sum
   0x55a0 len 46
Apr  4 02:19:16 PIM Bootstrap not enabled on fe-0/0/0.0
Apr  4 02:19:16 PIM fe-0/0/1.0 RECV 10.0.4.9 -> 224.0.0.13 V2 Bootstrap sum
   0x2341 len 46
Apr  4 02:19:16 PIM Bootstrap not enabled on fe-0/0/1.0
Apr  4 02:19:16 PIM fe-0/0/2.0 RECV 10.0.4.1 -> 224.0.0.13 V2 Bootstrap sum
   0xcc3f len 46
Apr  4 02:19:16 PIM Bootstrap not enabled on fe-0/0/2.0
Apr  4 02:19:16 PIM fe-0/0/0.0 RECV 10.0.5.1 -> 224.0.0.13 V2 Bootstrap sum
   0x66cd len 46
Apr  4 02:19:16 PIM Bootstrap not enabled on fe-0/0/0.0
Apr  4 02:19:16 PIM fe-0/0/3.0 RECV 10.0.4.5 -> 224.0.0.13 V2 Bootstrap sum
   0x66cd len 46
Apr  4 02:19:16 PIM Bootstrap not enabled on fe-0/0/3.0
Apr  4 02:19:17 PIM fe-0/0/3.0 SENT 10.0.4.6 -> 224.0.0.13 V2 Hello hold 105
   T-bit LAN prune 500 ms override 2000 ms pri 1 sum 0x55b1 len 26
```

```
Apr  4 02:19:19 PIM fe-0/0/0.0 RECV 172.16.30.1 -> 224.0.0.13 V2 Hello hold 105
   T-bit LAN prune 500 ms override 2000 ms pri 1 sum 0x55b1 len 26
Apr  4 02:19:19 PIM fe-0/0/0.0 RECV 10.0.5.1 -> 224.0.0.13 V2 Hello hold 105
   T-bit LAN prune 500 ms override 2000 ms pri 0 sum 0x55b2 len 26

*** monitor and syslog output disabled, press ESC-Q to enable ***
```

Can you identify the problem in r2's configuration based on the trace output shown? If you focused in on the errors regarding bootstrap not being enabled on any interfaces, then you are certainly getting close. The question now becomes "What is needed to enable bootstrap on an M-series router?" Note that r1 appears to be functioning correctly and that it has no explicit bootstrap-related configuration. The key to this problem lies in the fact that bootstrap is only enabled on a sparse mode interface, and all of r2's interfaces have been incorrectly left in dense mode from the last scenario. The setting of sparse vs. dense does not prevent PIM neighbor detection, but the configured mode clearly has an impact on the bootstrap protocol's operation. After correctly setting r2's interfaces to operate in sparse mode, RP election is confirmed to be working on all routers in the multicast test bed.

With all routers now listing both of the candidate RPs, your confirmation actions transition into the PIM-SM forwarding plane. As before, you start with an examination of the multicast routes at the first hop routers, r1 and r2:

```
[edit]
lab@r1# run show multicast route

Family: INET
Group           Source prefix     Act Pru InIf  NHid  Session Name
225.0.0.1       10.0.5.200     /32 A   F  2     76    MALLOC

Family: INET6
Group           Source prefix     Act Pru InIf  NHid  Session Name
```

This output shows that r1 is forwarding for the S,G pair in question. You now examine the state of PIM joins, again on r1:

```
[edit]
lab@r1# run show pim join extensive
Instance: PIM.master Family: INET
Group: 225.0.0.1
    Source: 10.0.5.200
    Flags: sparse
    Upstream interface: fe-0/0/0.0
    Upstream State: Local Source
    Keepalive timeout: 192
    Downstream Neighbors:
```

```
    Interface: fe-0/0/3.0
        10.0.4.17 State: Join   Flags: S    Timeout: 159


Instance: PIM.master Family: INET6
```

The various highlights in the capture confirm that r1 has established a SPT for 10.0.5.200, 225.0.0.1, and that it has pruned itself from the RPT-based tree. This is indicated by the lack of a *,G entry and the presence of a S,G entry. Recall that forwarding over the RPT tree is short-lived, in that once a receiver detects multicast traffic from source "S" via the shared tree it initiates a S,G join to effect a shortest path tree between that source and its local receivers. It is noted that at this time r2 has pruned this multicast route:

```
[edit]
lab@r2# run show multicast route

Family: INET
Group           Source prefix     Act Pru InIf  NHid  Session Name
225.0.0.1       10.0.5.200     /32 A   P  2     0     MALLOC

Family: INET6
Group           Source prefix     Act Pru InIf  NHid  Session Name
```

The pruned state is reflected in the downstream interfaces listed in the output of a PIM join at r2:

```
[edit]
lab@r2# run show pim join extensive
Instance: PIM.master Family: INET
Group: 225.0.0.1
    Source: 10.0.5.200
    Flags: sparse
    Upstream interface: fe-0/0/0.0
    Upstream State: Local Source
    Keepalive timeout: 153
    Downstream Neighbors:


Instance: PIM.master Family: INET6
```

The next step in the verification of the PIM-SM data plane takes you to r4 because previous output from r1 indicated that its fe-0/0/3 interface is downstream for the S,G entry being tracked:

```
[edit]
lab@r4# run show pim join extensive
Instance: PIM.master Family: INET
Group: 225.0.0.1
    Source: 10.0.5.200
    Flags: sparse
```

```
    Upstream interface: fe-0/0/2.0
    Upstream State: Join to Source
    Keepalive timeout: 158
    Downstream Neighbors:
        Interface: so-0/1/1.0
            10.0.2.9 State: Join   Flags: S    Timeout: 160


Instance: PIM.master Family: INET6
```

As with r1, r4's display confirms that S,G state has been correctly established. The indication that r4's so-0/1/1 interface is downstream tells you that the S,G multicast traffic should be flowing from r4 to r5 over the POS link. Note that r4 does not show a *,G join. This is because it has no local group membership to trigger a join to the shared tree. In contrast, the static joins on r5 have resulted in joins to both the shared and shortest path trees:

```
[edit]
lab@r5# run show pim join extensive
Instance: PIM.master Family: INET
Group: 225.0.0.1
    Source: *
    RP: 10.0.3.3
    Flags: sparse,rptree,wildcard
    Upstream interface: at-0/2/1.0
    Upstream State: Join to RP
    Downstream Neighbors:
        Interface: fe-0/0/3.0
            172.16.30.1 State: Join   Flags: SRW  Timeout: Infinity


Group: 225.0.0.1
    Source: 10.0.5.200
    Flags: sparse,spt
    Upstream interface: so-0/1/0.0
    Upstream State: Join to Source, Prune to RP
    Keepalive timeout: 209
    Downstream Neighbors:
        Interface: fe-0/0/3.0
            172.16.30.1 State: Join   Flags: S    Timeout: Infinity
```

The static group membership definitions on r5 result in a rather long timeout on the PIM Joins (Infinity). Also of note is the presence of both a *,G and S,G join at r5. The capture shows that the bootstrap hashing function has resulted in the selection of 10.0.3.3 as the RP for the 225.0.0.1 group. It bears stressing that the bootstrap mechanism provides inherent load balancing

among a set of candidate RPs. The result is that you are likely to find that r5 has selected 10.0.3.4 as the RP for a different group address.

The presence of a SPT for the S,G pair associated with MS1 and the 225.0.0.1 group proves that the PIM RP and register functionality must have worked, because without these functions there would have been no way for MR1 to discover MS1, and therefore the establishment of the SPT shown would not have been possible. The following capture shows the PIM join state for r3:

```
[edit]
lab@r3# run show pim join extensive
Instance: PIM.master Family: INET
Group: 225.0.0.1
    Source: *
    RP: 10.0.3.3
    Flags: sparse,rptree,wildcard
    Upstream interface: local
    Upstream State: Local RP
    Downstream Neighbors:
        Interface: at-0/1/0.0
            10.0.2.1 State: Join   Flags: SRW  Timeout: 193


Group: 225.0.0.1
    Source: 10.0.5.200
    RP: 10.0.3.3
    Flags: sparse,rptree
    Upstream interface: local
    Upstream State: Local RP
    Keepalive timeout:
    Downstream Neighbors:
        Interface: at-0/1/0.0             (pruned)
            10.0.2.1 State: Prune  Flags: SR   Timeout: 193
```

It is interesting to note that r3 has pruned itself from the S,G SPT, and that it confirms it is the RP for *, 225.0.0.1 by virtue of the Local RP designation for the *,G entry's upstream state. The results seen thus far confirm that PIM SM is operating in accordance with all provided criteria. Before leaving this section, you decide to test RP failover by deactivating PIM functionality on r3, which is the current RP for the *,G being discussed:

```
[edit]
lab@r3# deactivate protocols pim

[edit]
lab@r3# commit
commit complete
```

Because it can take a while for multicast protocols and state to reform, you decide to wait a few moments before analyzing the results. A few minutes later, proper RP failover, and continued PIM-SM data forwarding, are confirmed at r5:

```
[edit]
lab@r5# run show pim rps
Instance: PIM.master

Family: INET
RP address      Type      Holdtime Timeout Active groups Group prefixes
10.0.3.4        bootstrap      150     119             2 224.0.0.0/4

Family: INET6
RP address      Type      Holdtime Timeout Active groups Group prefixes
```

The loss of r3 has not left the PIM domain without an RP, which is in keeping with the requirements that there be no single point of RP failure in your design. Data forwarding is again confirmed in the face of r3's recently deactivated PIM stanza:

```
[edit]
lab@r5# run show pim join extensive
Instance: PIM.master Family: INET
Group: 225.0.0.1
    Source: *
    RP: 10.0.3.4
    Flags: sparse,rptree,wildcard
    Upstream interface: so-0/1/0.0
    Upstream State: Join to RP
    Downstream Neighbors:
        Interface: fe-0/0/3.0
            172.16.30.1 State: Join   Flags: SRW  Timeout: Infinity


Group: 225.0.0.1
    Source: 10.0.5.200
    Flags: sparse
    Upstream interface: so-0/1/0.0
    Upstream State: Join to Source
    Keepalive timeout: 209
    Downstream Neighbors:
        Interface: fe-0/0/3.0
            172.16.30.1 State: Join   Flags: S    Timeout: Infinity
```

The PIM join display confirms that r5 has now mapped the 225.0.0.1 group address to the domain's remaining RP, and that a S,G join is still in effect. This behavior validates the

redundancy aspects of your PIM design. Before proceeding to the next section, it is suggested that you reactivate r3's PIM configuration:

```
[edit]
lab@r3# rollback 1
load complete

[edit]
lab@r3# commit
commit complete
```

Configuring PIM-SM Using Auto-RP

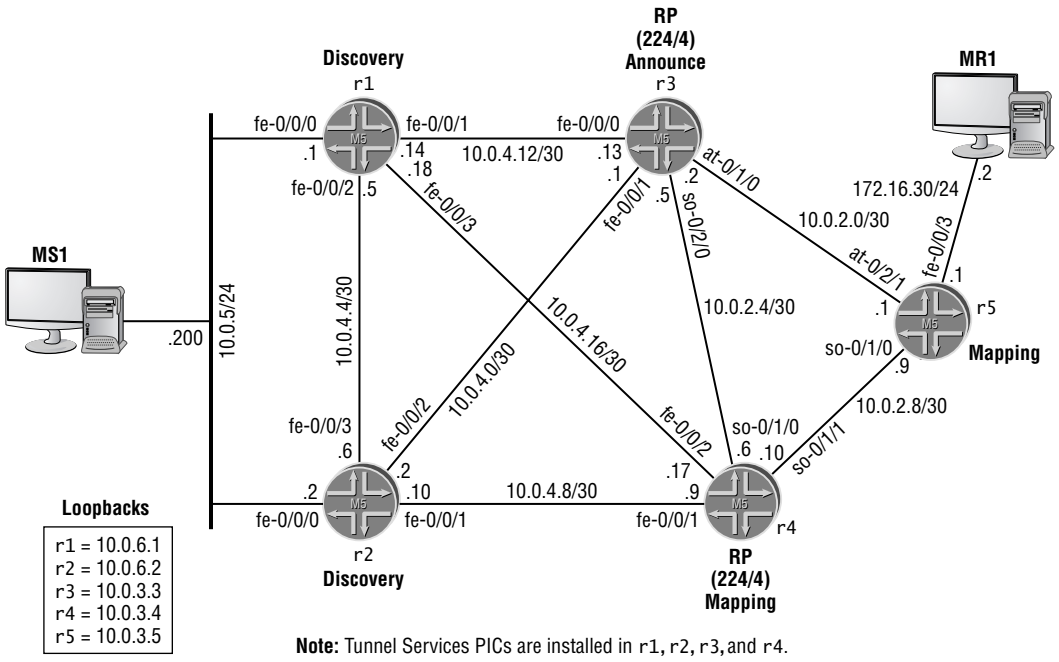
To complete this scenario, you must reconfigure the multicast test bed according to the following criteria:

- Replace the current bootstrap based election with Auto-RP.
- Ensure that there is no single point of RP failure.
- Multicast traffic from MS1 to MR1 must be delivered to MR1.

Refer back to Figure 4.2 as needed for the details of the multicast topology that you are working with.

The goal of this configuration exercise is to convert the existing network from the use of bootstrap to Auto-RP based RP election. Note that your network must continue to provide RP redundancy, which leads you to the Auto-RP design shown in Figure 4.6.

FIGURE 4.6 Auto-RP topology



Note: Tunnel Services PICs are installed in r1, r2, r3, and r4.

Your PIM-SM Auto-RP design once again presses r3 and r4 into the role of RP for the entire 224/4 group range. To provide the required redundancy, you must ensure that your network has at least two candidate RPs, and at least two mapping agents. In this example, r4 and r5 are configured as Auto-RP *mapping* agents. Once again your design has r4 functioning in a dual capacity; this time as a candidate RP and as an Auto-RP mapping agent. Note that an Auto-RP mapping agent router also generates *announce* messages when it is RP capable, and that it also listens to *discovery* messages to learn RP mapping results. When multiple mapping agents are present, each mapping agent operates independently; coordination between mapping agents is not required because all mapping agents compute the same mapping results, which is in keeping with their ability to operate independently.

r3 is configured with the announce option to ensure that it announces its local RP capabilities while also listening to Auto-RP mapping messages generated by the mapping agent. Because r1 and r2 are not configured as RPs, they will be configured to operate in discovery mode, which is the most basic Auto-RP option. In this mode, the routers will only listen for mapping messages to learn the results of RP-to-group range mappings.

The Auto-RP options of discovery, announce, and mapping each build on the previous option's functionality. In other words, discovery listens to mapping messages only while the announce options listens to mapping messages while also generating announce messages. It is common practice in the "real world" to just set all routers to the highest level of functionality (mapping) and then let their individual configurations determine if this level of functionality is actually used. For example, the lack of local RP settings will prevent a mapping router from generating announce messages, although the capability is inherent in its mapping setting. Because an examination may be written to test the candidate's understanding of each option, you may be expected to configure only the functionality that is actually required, which is the approach taken here. The moral here is that over-configuring a network is not necessarily bad, unless, of course, your rules of engagement restrict such over-configuration.

Note that Auto-RP requires the use of dense mode flooding to support *announce* messages (224.0.1.39) and *discovery* messages (224.0.1.40). Further, proper Auto-RP operation also requires that the router's lo0 interface be assigned a routable IP address and that you configure the lo0 interface as a sparse-dense PIM interface.

Your Auto-RP configuration scenario starts with the removal of any bootstrap-related configuration from r5 and r4:

```
[edit protocols]
lab@r3# delete pim rp bootstrap-priority
```

The Auto-RP configuration starts at r4 with the setting of sparse-dense mode on all PIM interfaces. This mode is needed to support dense-mode flooding of Auto-RP discovery and mapping messages while still operating in sparse mode for all other groups:

```
[edit protocols pim]
lab@r4# set interface all mode sparse-dense
```

The required dense mode groups are now defined.

```
[edit protocols pim]
lab@r4# set dense-groups 224.0.1.39

[edit protocols pim]
lab@r4# set dense-groups 224.0.1.40
```

In case you are one of those types that has not yet committed Auto-RP's dense mode groups to memory, do not despair! When taking the JNCIE examination, you will have access to the JUNOS software documentation for the software release used in the JNCIE lab, and happily, the dense mode groups needed for Auto-RP operation are clearly identified in the documentation set. Even better, the answer is also found with the JUNOS software Online Documentation:

```
[edit protocols pim]
lab@r4# run help topic pim auto-rp | match dense
   dense mode PIM to advertise control traffic, it provides an important
   If PIM is operating in sparse or sparse-dense mode, configure how the
   you must enable PIM sparse-dense mode on the lo0.0 interface.
   address 127.0.0.1. Also, you must enable PIM sparse-dense mode on the
           1.     Use the mode statement and specify the option sparse-dense
     dense mode for others. The default is to operate in sparse mode unless
     the router is specifically informed of a dense mode group.
           2.     Configure two multicast dense groups (224.0.1.39 and
           224.0.1.40) using the dense-groups statement at the [edit
     occurs through a PIM dense mode model where group 224.0.1.39 is used for
   To configure auto-RP, include the mode, dense-groups, and auto-rp
             dense-groups {
                 mode sparse-dense;
```

Although the prose is a bit messy after being munged by the CLI's `match` function, the highlights call out the exact information being sought, most likely faster than this information could have been retrieved in any other way! The final command at `r4` enables Auto-RP mapping functionality:

```
[edit protocols pim]
lab@r4# set rp auto-rp mapping
```

The modified PIM stanza is shown next with added highlights:

```
[edit protocols pim]
lab@r4# show
dense-groups {
    224.0.1.39/32;
    224.0.1.40/32;
}
rp {
```

```
    local {
        address 10.0.3.4;
    }
    auto-rp mapping;
}
interface all {
    mode sparse-dense;
}
interface fxp0.0 {
    disable;
}
```

The configuration of r3 is similar, with the primary exception being the use of the announce option:

```
[edit protocols pim]
lab@r3# show
dense-groups {
    224.0.1.39/32;
    224.0.1.40/32;
}
rp {
    local {
        address 10.0.3.3;
    }
    auto-rp announce;
}
interface all {
    mode sparse-dense;
}
interface fxp0.0 {
    disable;
}
```

The modified PIM stanza for r5 enables Auto-RP mapping functions; note the absence of local RP configuration on r5:

```
[edit]
lab@r5# show protocols pim
dense-groups {
    224.0.1.39/32;
    224.0.1.40/32;
}
rp {
    auto-rp mapping;
}
```

```
interface all {
    mode sparse-dense;
}
interface fxp0.0 {
    disable;
}
```

Although only shown here for r2, the PIM configuration of r1 is virtually identical. Note that these routers are set to perform Auto-RP discovery functions only:

```
[edit protocols pim]
lab@r2# show
dense-groups {
    224.0.1.39/32;
    224.0.1.40/32;
}
rp {
    auto-rp discovery;
}
interface all {
    mode sparse-dense;
}
interface fxp0.0 {
    disable;
}
```

Make sure that all modifications are committed before you proceed to the confirmation section.

Verifying PIM Sparse Mode Using Auto-RP

Because the only changes made to the test bed relate to the use of Auto-RP instead of the bootstrap protocol, the confirmation steps in this section will be confined to the verification of RP election. You begin with the confirmation of sparse-dense mode operation on all PIM interfaces:

```
[edit protocols pim]
lab@r3# run show pim interfaces
Instance: PIM.master

Name                 Stat Mode       IP V State Count DR address
at-0/1/0.0           Up   SparseDense 4 2 P2P       1
fe-0/0/0.0           Up   SparseDense 4 2 NotDR     1 10.0.4.14
fe-0/0/1.0           Up   SparseDense 4 2 NotDR     1 10.0.4.2
fe-0/0/2.0           Up   SparseDense 4 2 DR        0 172.16.0.13
fe-0/0/3.0           Up   SparseDense 4 2 DR        0 10.0.2.14
lo0.0                Up   SparseDense 4 2 DR        0 10.0.3.3
pd-0/3/0.32768       Up   Sparse      4 2 P2P       0
pe-0/3/0.32770       Up   Sparse      4 2 P2P       0
```

The PIM interface display confirms that r3 is correctly configured to run sparse-dense mode. Next, you verify that an RP has been successfully elected via the Auto-RP mechanism:

```
[edit protocols pim]
lab@r3# run show pim rps
Instance: PIM.master

Family: INET
RP address      Type      Holdtime Timeout Active groups Group prefixes
10.0.3.4        auto-rp        150     101             0 224.0.0.0/4

10.0.3.3        static           0    None             0 224.0.0.0/4

Family: INET6
RP address      Type      Holdtime Timeout Active groups Group prefixes
```

The output confirms that r3 is locally configured as an RP (static) and that it has learned of the 10.0.3.4 RP through the Auto-RP protocol. Similar confirmation of proper Auto-RP operation is also seen in the display obtained from r1:

```
[edit]
lab@r1# run show pim rps
Instance: PIM.master

Family: INET
RP address      Type      Holdtime Timeout Active groups Group prefixes
10.0.3.4        auto-rp        150     124             0 224.0.0.0/4

Family: INET6
RP address      Type      Holdtime Timeout Active groups Group prefixes
```

Note that both r1 and r3 display the same RP as learned through Auto-RP; it is critical that the results of the Auto-RP mapping function yield a consistent choice of RP-to-group mappings for all routers in the PIM domain. Though not shown, you may assume that all five routers are correctly displaying r4 (10.0.3.4) as the RP. It is interesting to note that all routers have issued dense mode joins to the Auto-RP dense mode groups so that they will receive mapping and announce messages:

```
[edit]
lab@r5# run show pim join
Instance: PIM.master Family: INET
Group: 224.0.1.39
    Source: 10.0.3.3
    Flags: dense
    Upstream interface: at-0/2/1.0
```

```
Group: 224.0.1.39
    Source: 10.0.3.4
    Flags: dense
    Upstream interface: so-0/1/0.0
Group: 224.0.1.40
    Source: 10.0.3.5
    Flags: dense
    Upstream interface: local
Group: 225.0.0.1
    Source: *
    RP: 10.0.3.4
    Flags: sparse,rptree,wildcard
    Upstream interface: so-0/1/0.0

Group: 225.0.0.1
    Source: 10.0.5.200
    Flags: sparse,spt
    Upstream interface: at-0/2/1.0

Instance: PIM.master Family: INET6
```

With Auto-RP based RP election confirmed, you should again confirm the PIM-SM forwarding plane. The steps are not shown here because they are identical to those already demonstrated for both PIM-DM and PIM-SM. You may assume that multicast traffic is confirmed to be flowing from MS1 to MR1 over a SPT, as indicated by this capture from r3:

```
[edit protocols pim]
lab@r3# run show pim join extensive
Instance: PIM.master Family: INET
Group: 224.0.1.39
    Source: 10.0.3.3
    Flags: dense
    Upstream interface: local
    Downstream interfaces:
        local
        lo0.0
        fe-0/0/0.0
        fe-0/0/1.0
        at-0/1/0.0
        so-0/2/0.100

Group: 224.0.1.39
    Source: 10.0.3.4
```

```
    Flags: dense
    Upstream interface: so-0/2/0.100
    Downstream interfaces:
        local
        lo0.0
        fe-0/0/0.0
        fe-0/0/1.0
        at-0/1/0.0 (Pruned timeout 139)

Group: 224.0.1.40
    Source: 10.0.3.5
    Flags: dense
    Upstream interface: at-0/1/0.0
    Downstream interfaces:
        local
        lo0.0
        fe-0/0/0.0
        fe-0/0/1.0
        so-0/2/0.100 (Pruned timeout 124)

Group: 225.0.0.1
    Source: 10.0.5.200
    Flags: sparse
    Upstream interface: fe-0/0/1.0
    Upstream State: Join to Source
    Keepalive timeout: 208
    Downstream Neighbors:
        Interface: at-0/1/0.0
            10.0.2.1 State: Join   Flags: S    Timeout: 159


Instance: PIM.master Family: INET6
```

Before leaving this section, you decide to test RP failover by deactivating PIM functionality on r4, which has been selected as the RP for the 224/4 range by the Auto-RP mapping function:

```
[edit]
lab@r4# deactivate protocols pim

[edit]
lab@r4# commit
commit complete
```

After waiting a few minutes, RP failover is confirmed at r2:

```
[edit protocols pim]
lab@r2# run show pim rps
Instance: PIM.master

Family: INET
RP address      Type      Holdtime Timeout Active groups Group prefixes
10.0.3.3        auto-rp        150     146             0 224.0.0.0/4

Family: INET6
RP address      Type      Holdtime Timeout Active groups Group prefixes
```

Make sure that you restore r4's configuration after you are satisfied that all PIM redundancy requirements have been met:

```
[edit]
lab@r4# rollback 1
load complete

[edit]
lab@r4# commit
commit complete
```

PIM Tracing

As with all protocols, there may be times when you need to closely monitor the operation of PIM to diagnose problems or to better understand its operation. This section provides an example of PIM tracing in the context of the PIM-SM Auto-RP based topology that is currently in place. A typical PIM tracing configuration has been added to r3's configuration:

```
[edit protocols pim]
lab@r3# show traceoptions
file pim;
flag rp detail;
flag hello detail;
flag register detail;
flag join detail;
```

A sample of the tracing output obtained with this configuration is shown next:

```
[edit protocols pim]
lab@r3# run monitor start pim

*** pim ***
Apr  4 20:52:39 PIM at-0/1/0.0 RECV 10.0.2.1 -> 224.0.0.13 V2 Hello hold 105
   T-bit LAN prune 500 ms override 2000 ms pri 1 sum 0x55b1 len 26
lab@r3#
```

```
Apr  4 20:52:41 PIM fe-0/0/1.0 SENT 10.0.4.1 -> 224.0.0.13 V2 JoinPrune to
   10.0.4.2 holdtime 210 groups 1 sum 0xd61f len 34
Apr  4 20:52:41 group 225.0.0.1 joins 1 prunes 0
Apr  4 20:52:41   join list:
Apr  4 20:52:41     source 10.0.5.200 flags sparse
Apr  4 20:52:45 PIM fe-0/0/1.0 SENT 10.0.4.1 -> 224.0.0.13 V2 Hello hold 105
   T-bit LAN prune 500 ms override 2000 ms pri 1 sum 0x55b1 len 26
Apr  4 20:52:55 PIM so-0/2/0.100 SENT 10.0.2.5 -> 224.0.0.13 V2 Hello hold 105
   T-bit LAN prune 500 ms override 2000 ms pri 1 sum 0x55b1 len 26
. . .
Apr  4 20:53:09 PIM at-0/1/0.0 RECV 10.0.2.1 -> 224.0.0.13 V2 Hello hold 105
   T-bit LAN prune 500 ms override 2000 ms pri 1 sum 0x55b1 len 26
Apr  4 20:53:12 PIM at-0/1/0.0 RECV 10.0.2.1 -> 224.0.0.13 V2 JoinPrune to
   10.0.2.2 holdtime 210 groups 1 sum 0xd81f len 34
Apr  4 20:53:12 group 225.0.0.1 joins 1 prunes 0
Apr  4 20:53:12   join list:
Apr  4 20:53:12     source 10.0.5.200 flags sparse
Apr  4 20:53:12 PIM fe-0/0/1.0 SENT 10.0.4.1 -> 224.0.0.13 V2 Hello hold 105
   T-bit LAN prune 500 ms override 2000 ms pri 1 sum 0x55b1 len 26
Apr  4 20:53:19 PIM SENT 10.0.3.3 -> 224.0.1.39+496 AutoRP v1 announce hold 150
   rpcount 1 len 20 rp 10.0.3.3 version 2 groups 1 prefixes 224.0.0.0/4
Apr  4 20:53:19 PIM at-0/1/0.0 RECV 10.0.3.5+496 -> 224.0.1.40 AutoRP v1
   mapping hold 150 rpcount 1 len 20 rp 10.0.3.4 version 2 groups 1 prefixes
   224.0.0.0/4
. . .
Apr  4 20:53:37 PIM fe-0/0/1.0 SENT 10.0.4.1 -> 224.0.0.13 V2 JoinPrune to
   10.0.4.2 holdtime 210 groups 1 sum 0xd61f len 34
Apr  4 20:53:37 group 225.0.0.1 joins 1 prunes 0
Apr  4 20:53:37   join list:
Apr  4 20:53:37     source 10.0.5.200 flags sparse
Apr  4 20:53:38 PIM at-0/1/0.0 RECV 10.0.2.1 -> 224.0.0.13 V2 Hello hold 105
   T-bit LAN prune 500 ms override 2000 ms pri 1 sum 0x55b1 len 26
Apr  4 20:53:42 PIM fe-0/0/1.0 SENT 10.0.4.1 -> 224.0.0.13 V2 Hello hold 105
   T-bit LAN prune 500 ms override 2000 ms pri 1 sum 0x55b1 len 26
```

The edited capture shows quite a few PIM hellos, a join message or two (recall that joins and prunes are sent periodically to maintain state), and the presence of Auto-RP announce and mapping messages.

PIM Summary

This section presented several JNCIE-level multicast configuration scenarios based on PIM. Like DVMRP, enabling PIM on a multi-access interfaces automatically enables the IGMP

protocol on that interface. Unlike DVMRP, PIM is not a routing protocol, and PIM does not require a special routing table for RPF checks. Many find that the absence of RIB group configuration makes PIM far easier to deal with when compared to DVMRP.

When operating in dense mode, PIM uses a flood-and-prune approach based on broadcast trees. PIM-DM does not require RPs, and therefore has no need for Bootstrap or Auto-RP functionality and no need for TS PIC hardware. In sparse mode, PIM requires that sources and receivers for a given multicast group agree upon a Rendezvous Point (RP) where initial contact is made. The RP can be statically defined, or can be dynamically learned using the bootstrap protocol (PIM version 2) or the Auto-RP mechanism (PIM versions 1 or 2). You can configure multiple RPs with either approach, but load balancing among a set of RPs is possible only with the bootstrap protocol (Any-Cast is covered in a later section). For BSR redundancy, ensure that you have at least two BSR candidates by setting a non-zero priority in these routers. For Auto-RP redundancy, you must ensure that at least two routers are capable of providing the mapping function. Recall that Auto-RP requires sparse-dense mode operation and the definition of the two dense groups associated with Auto-RP for proper operation.

It can take a few minutes for all routers to learn the domain's RP, whether you use Auto-RP or bootstrap. Be sure that you give your network time to converge before making configuration changes or deciding to reboot, as these actions may only serve to prolong RP convergence.

MSDP

The Multicast Source Discovery Protocol (MSDP) is used to convey information about active multicast sources between autonomous systems in support of interdomain multicast. MSDP is also used within a single multicast domain to support Any-Cast RPs.

MSDP can work with or without MP-BGP. MP-BGP advertises multicast routes using Sub-AFI 2. These routes are automatically placed into the `inet.2` routing table by JUNOS software. Placing multicast routes into a special routing table allows you to have a multicast forwarding topology that differs from the unicast forwarding topology for the same prefix. When MP-BGP is not configured, all routes are placed into the common `inet.0` routing table, which forces a common forwarding topology for unicast and multicast traffic. Note that to actually achieve different forwarding topologies between autonomous domains, you must have multiple EBGP peering interfaces and you must configure your unicast and multicast EBGP peering sessions to use one, or the other, of these links.

The purpose of MSDP is to provide a mechanism by which active sources in one PIM domain are communicated to RPs in other domains. The knowledge of an active source for group G results in RPs with *,G join state generating S,G joins back toward the active source to establish a source-specific tree that spans domain boundaries.

Configure Any-Cast and MSDP

Any-Cast is a technique that allows load balancing among a set of RPs that are learned through the Auto-RP mechanism. Recall that normally Auto-RP selects a single RP from a set of

candidate RPs that will handle all *,G traffic for the group range advertised. With Any-Cast, a set of RPs are deployed using a duplicated address, such that stations "see" a single RP and simply direct their joins according to the metrically closest route to the "single" RP address. From the perspective of a multicast router, the failure of an Any-Cast RP will either go completely undetected, or manifest itself as a metric increase due to the need to begin directing joins and register messages to the next metrically closest RP.

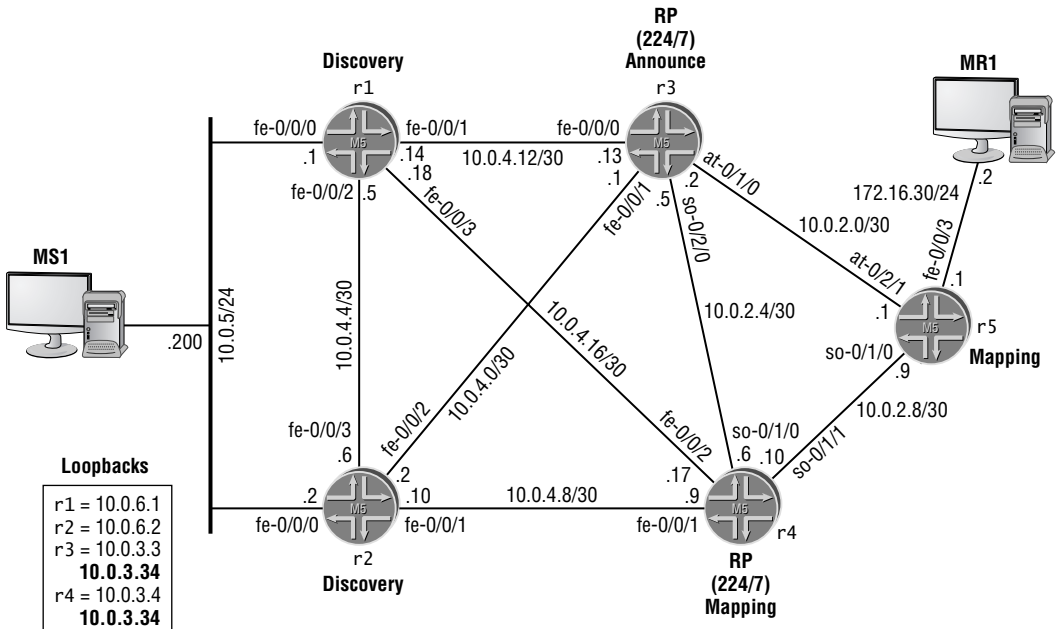
Any-Cast makes use of MSDP to tie together the RPs that make up the Any-Cast RP, just as the MSDP protocol ties together two or more RPs that exist within independent routing domains.

To complete this section, you must alter your configurations to meet these criteria:

- Use Auto-RP to elect two RPs for the group range 224.0.0.0 through 225.255.255.255.
- Ensure that traffic for these group ranges can be balanced over both RPs.
- Ensure that multicast traffic sent from MS1 is delivered to MR1.

The wording of your objectives does not specifically mention Any-Cast by design. The goal here is to verify that the candidate understands the limitations of Auto-RP, and that the Any-Cast approach is designed to overcome Auto-RP's load-balancing shortcomings. Also note that MSDP is not mentioned anywhere in your list of objectives. Without MSDP, you will likely find that traffic will not flow correctly from MS1 to MR1, because the source may send its register messages to one RP while the receiver sends its join to the other. Figure 4.7 documents the Any-Cast design that you have decided on, given the current test bed's configuration and your new configuration objectives.

FIGURE 4.7 Any-Cast topology



Note: Tunnel Services PICs are installed in `r1`, `r2`, `r3`, and `r4`.

Note that the Any-Cast topology tries to leave as much functionality and configuration in place from the previous PIM-SM Auto-RP scenario as practical. A key addition to Figure 4.7 is your choice of the non-unique lo0 address that will be used by your RPs, and the new group assignment that reflects the group range restrictions posed in this scenario.

You start your Any-Cast configuration on r3 with the assignment of the non-unique lo0 address that will be shared by all RPs:

```
[edit interfaces]
lab@r3# set lo0 unit 0 family inet address 10.0.3.3/32 primary

[edit interfaces]
lab@r3# set lo0 unit 0 family inet address 10.0.3.34
```

Note that the existing (and unique) lo0 address is tagged with the primary keyword to ensure that the new, non-unique address cannot become the router's primary address (because this will change the router's ID and could cause a host of other problems, considering its non-unique nature). The primary tag is not strictly needed here because the default behavior makes the numerically lowest value the primary address for the interface, but being safe is never a bad idea. The added safety net makes inclusion of the primary keyword highly recommended. The following commands redefine the router's local RP properties in accordance with the requirements of this scenario:

```
[edit protocols pim]
lab@r3# set rp local address 10.0.3.34 group-ranges 224/7
```

Note that the local RP is defined to use the non-unique address that was assigned to r3's lo0 interface in the previous step. The last set of commands configures the MSDP peering session between r3 and r4:

```
[edit protocols msdp]
lab@r3# set group as-65412 peer 10.0.3.4

[edit protocols msdp]
lab@r3# set group as-65412 local-address 10.0.3.3
```

Note that the MSDP peering session must be established to the unique lo0 address at the remote router. The source-address statement is mandatory for MSDP; specifying the unique lo0 address as the source of MSDP packets is important because the remote router expects to peer with this address. The changes to r3's configuration in support of Any-Cast RP are displayed next with added highlights:

```
[edit]
lab@r3# show protocols msdp
group as-65412 {
    local-address 10.0.3.3;
    peer 10.0.3.4;
}
```

```
[edit]
lab@r3# show protocols pim
traceoptions {
    file pim;
    flag rp detail;
    flag hello detail;
    flag register detail;
    flag join detail;
}
dense-groups {
    224.0.1.39/32;
    224.0.1.40/32;
}
rp {
    local {
        address 10.0.3.34;
        group-ranges {
            224.0.0.0/7;
        }
    }
    auto-rp announce;
}
interface all {
    mode sparse-dense;
}
interface fxp0.0 {
    disable;
}

[edit]
lab@r3# show interfaces lo0
unit 0 {
    family inet {
        address 10.0.3.3/32 {
            primary;
        }
        address 10.0.3.34/32;
    }
    family iso {
        address 49.0001.3333.3333.3333.00;
    }
}
```

Note that the majority of r3's PIM configuration remains in effect from the previous section covering Auto-RP. Similar changes are now made to r4. These changes are displayed here with highlights:

```
[edit]
lab@r4# show protocols msdp
group as-65412 {
    local-address 10.0.3.4;
    peer 10.0.3.3;
}

[edit]
lab@r4# show protocols pim
dense-groups {
    224.0.1.39/32;
    224.0.1.40/32;
}
rp {
    local {
        address 10.0.3.34;
        group-ranges {
            224.0.0.0/7;
        }
    }
    auto-rp mapping;
}
interface all {
    mode sparse-dense;
}
interface fxp0.0 {
    disable;
}

[edit]
lab@r4# show interfaces lo0
unit 0 {
    family inet {
        address 10.0.3.4/32 {
            primary;
        }
        address 10.0.3.34/32;
    }
```

```
    family iso {
        address 49.0001.4444.4444.4444.00;
    }
}
```

Be sure to commit all your changes before proceeding to the verification section.

Verifying Any-Cast and MSDP

Verification of Any-Cast operation begins with confirmation that all routers now display a single RP that is associated with the non-unique address chosen for your Auto-RP:

```
[edit]
lab@r3# run show pim rps
Instance: PIM.master

Family: INET
RP address      Type      Holdtime Timeout Active groups Group prefixes
10.0.3.34       auto-rp        150     138             1 224.0.0.0/7

10.0.3.34       static           0    None             1 224.0.0.0/7

Family: INET6
RP address      Type      Holdtime Timeout Active groups Group prefixes

[edit]
lab@r2# run show pim rps
Instance: PIM.master

Family: INET
RP address      Type      Holdtime Timeout Active groups Group prefixes
10.0.3.34       auto-rp        150     101             0 224.0.0.0/7

Family: INET6
RP address      Type      Holdtime Timeout Active groups Group prefixes
```

r2 and r3 display a single RP using the non-unique 10.0.3.34 address that has been learned through Auto-RP. You can assume that all routers in the multicast test bed are correctly displaying the same RP information. The MSDP session status is now confirmed at r4:

```
[edit]
lab@r4# run show msdp peer 10.0.3.3
Peer address    Local address   State       Last up/down Peer-Group
10.0.3.3        10.0.3.4        Established     00:23:14 as-65412
```

The output confirms that the TCP-based MSDP session between r3 and r4 has been successfully established. The presence of a source active message for the 10.0.1.200 source (MS1) is confirmed next:

```
[edit]
lab@r4# run show msdp source-active
Group address   Source address  Peer address    Originator      Flags
224.0.1.24      10.0.5.200      10.0.3.3        10.0.3.3        Accept
225.0.0.1       10.0.5.200      10.0.3.3        10.0.3.3        Accept
```

Excellent! The display confirms that r4 has received an MSDP source active message for source 10.0.5.200, which is sending to group 225.0.0.1, and that this traffic is being accepted. The Accept action indicates that no policy-related filtering is in effect that would cause this SA message to be filtered. Note that a 224.0.1.24 address is also listed. This address is associated with microsoft-ds (Active Directory services), which evidently makes use of multicast and is running on MS1. Source active messages are now analyzed at r3:

```
[edit]
lab@r3# run show msdp source-active
Group address   Source address  Peer address    Originator      Flags
225.0.0.1       10.0.5.200      local           10.0.3.34       Accept
```

The display at r3 confirms that the multicast source sent register packets to its local RP using the non-unique address 10.0.3.34, which in this case was r3, as indicated by the local indication in the peer column. The presence of an active source resulted in r3 sending a source active message to r4, as observed in the previous step.

So far, things are looking very good for the overall operation of Any-Cast in the current test bed. The final proof comes with confirmation of successful multicast traffic flow between MS1 and MR1. The S,G join at r3 confirms that a SPT has been established between MS1 and MR1, and indicates that multicast traffic is flowing correctly through the test bed:

```
[edit]
lab@r3# run show pim join 225.0.0.1 detail
Instance: PIM.master Family: INET
Group: 225.0.0.1
    Source: 10.0.5.200
    Flags: sparse
    Upstream interface: fe-0/0/1.0
    Downstream interfaces:
        at-0/1/0.0
```

It is interesting to note the lack of a *,G entry on r3 for the 225.0.0.1 group. This is because r5 sent its *,G join to the Any-Cast RP address using its so-0/1/0 interface (it has two equal-cost paths to the Any-Cast address), making r4 the RP for the 225.0.0.1 shared tree.

```
[edit]
lab@r4# run show pim join 225.0.0.1 detail
Instance: PIM.master Family: INET
```

```
Group: 225.0.0.1
    Source: *
    RP: 10.0.3.34
    Flags: sparse,rptree,wildcard
    Upstream interface: local
    Downstream interfaces:
        so-0/1/1.0

Group: 225.0.0.1
    Source: 10.0.5.200
    Flags: sparse,spt-pending
    Upstream interface: fe-0/0/1.0
    Downstream interfaces:
        so-0/1/1.0              (pruned)
```

The pruned state for the S,G entry is in keeping with the SPT that happened to form through r3 in this case, as shown in the previous capture. The results shown in this section confirm that you have configured Any-Cast (and MSDP) according to the provided criteria.

Configuring Interdomain Multicast

As mentioned previously, MSDP's primary purpose is to support interdomain multicast. In this configuration scenario, you configure interdomain multicast between your AS and the T1 router in AS 65222.

To complete this section, you must meet these criteria:

- Ensure that traffic sent from MS2 in AS 65222 to 225.0.0.1 is delivered to MR1.
- Prevent routers in AS 65222 from discovering the RPs in AS 65412. You may only block RP discovery and announce traffic in support of this goal.
- Filter SAs received from T1 for all groups in the 225.8/16 block.

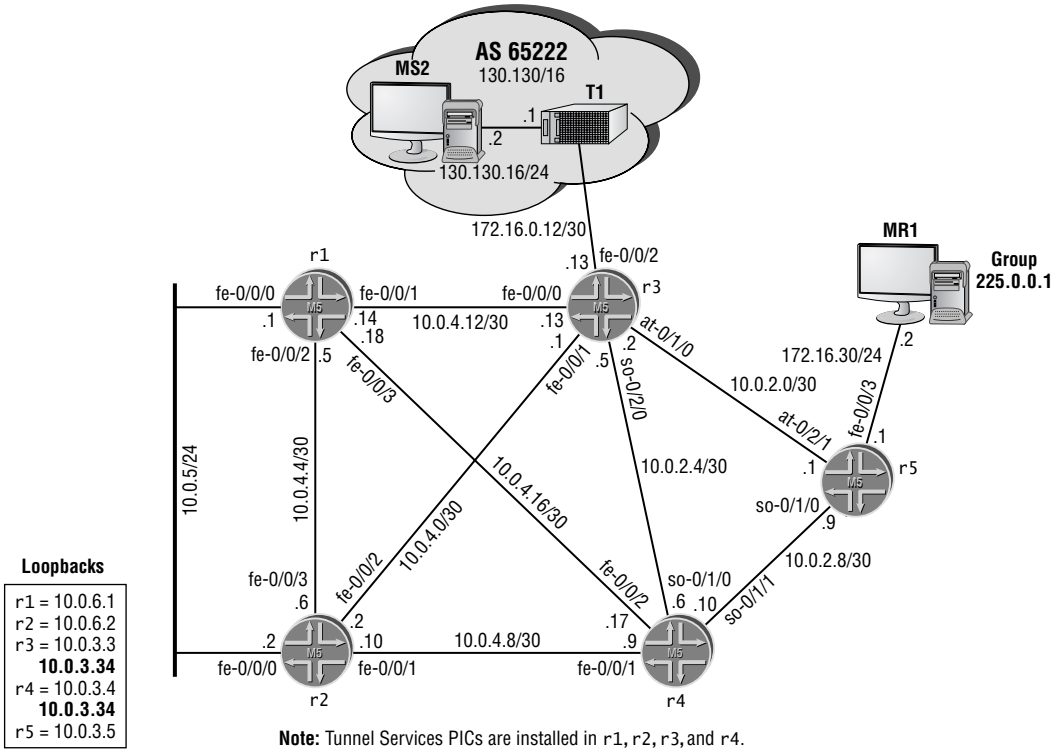
Figure 4.8 provides the details you need to complete this task.

Note that your IS-IS baseline network already contains a working EBGP-related configuration for the T1 peer at r3. You can assume that the T1 router has the necessary multicast configuration in place, and that the MS2 device is sending to group 225.0.0.1. To complete this task, you therefore need to modify r3's MSDP stanza and add multicast scoping.

You begin your task at r3 with the goal of defining a new MSDP peer group that contains the T1 router. Because the configuration details do not specify how the remote MSDP peer is configured with regard to the MSDP peering address (interface vs. lo0), you start with an interface-based peering configuration because the lack of redundant connections between T1 and r3 negates the reliability motivations associated with lo0 peering:

```
[edit]
lab@r3# edit protocols msdp group ext

[edit protocols msdp group ext]
lab@r3# set local-address 172.16.0.13 peer 172.16.0.14
```

FIGURE 4.8 Interdomain multicast topology



If the MSDP session does not become established, you can always switch the peering over to use lo0 addressing, or monitor the MSDP traffic to get a better understanding of exactly what MSDP exchanges are occurring.

MSDP peers wait for the end with the lowest address to initiate the TCP connection. Therefore, attempting to monitor MSDP traffic before `r3` is configured might result in no MSDP activity, depending on the addressing specifics in T1's configuration.

The modified MSDP stanza is shown next at `r3` with highlights:

```
[edit protocols msdp]
lab@r3# show
group as-65412 {
    local-address 10.0.3.3;
    peer 10.0.3.4;
}
group ext {
    local-address 172.16.0.13;
```

```
    peer 172.16.0.14;
}
```

There is no need to MSDP-peer r4 with the T1 router because it will receive the source active messages sent by T1 through its MSDP peering session to r3. You must ensure that PIM is enabled on r3's external interface to T1 (its fe-0/0/2 interface) in order for RPF checks to succeed for 130.130/16 sources, however. With the *ext* MSDP peering group defined, you now address the need to filter source active messages received from T1 for groups in the 225.8/16 range. To this end, you configure the *msdp* policy as shown here:

```
[edit policy-options policy-statement msdp]
lab@r3# show
term 1 {
    from {
        route-filter 225.8.0.0/16 orlonger;
    }
    then reject;
}
```

You apply this policy as import to the *ext* MSDP peer group. Note that a global application of the *msdp* policy could result in point loss, because your instructions do not indicate that you should also filter these groups from any advertisements you receive from r4.

```
[edit protocols msdp]
lab@r3# set group ext import msdp
```

The modified MSDP stanza is displayed next at r3:

```
[edit protocols msdp]
lab@r3# show
group as-65412 {
    local-address 10.0.3.3;
    peer 10.0.3.4;
}
group ext {
    import msdp;
    local-address 172.16.0.13;
    peer 172.16.0.14;
}
```

The final requirement in the interdomain multicast scenario requires that you filter Auto-RP messages from r3 to T1 to prevent T1 from learning your Any-Cast RP. You need to configure multicast scoping to block the output of Auto-RP announce and mapping messages on r3's fe-0/0/2 interface to achieve this goal:

```
[edit routing-options multicast]
lab@r3# set scope announce interface fe-0/0/2 prefix 224.0.1.39

[edit routing-options multicast]
lab@r3# set scope discover interface fe-0/0/2 prefix 224.0.1.40
```

Output filters are not possible for multicast traffic with M-series and T-series hardware (PR 24404). Multicast-based firewall matches are possible with ingress filters, however. Candidates have been observed trying to simulate the effects of multicast scoping with firewall filters applied in the output direction; these same candidates are normally observed deploying scoping when they come back for another go at the examination.

Note that two separate multicast scopes are needed to meet the criteria of blocking *only* Auto-RP related traffic on r3's fe-0/0/2 interface. The modified `routing-options` stanza is displayed next with the additions highlighted:

```
[edit]
lab@r3# show routing-options
static {
    route 10.0.200.0/24 {
        next-hop 10.0.1.102;
        no-readvertise;
    }
}
aggregate {
    route 10.0.0.0/16;
}
autonomous-system 65412;
multicast {
    scope announce {
        prefix 224.0.1.39/32;
        interface fe-0/0/2.0;
    }
    scope discover {
        prefix 224.0.1.40/32;
        interface fe-0/0/2.0;
    }
}
```

When satisfied with your MSDP and scoping changes, you should proceed to the next session. Note that you should not yet commit your changes for reasons that will soon be evident!

Verifying Interdomain Multicast

To help confirm (and demonstrate) the operation of MSDP, the tracing changes shown next are added to r3's configuration:

```
[edit]
lab@r3# show protocols msdp traceoptions
file msdp;
flag source-active detail;
```

```
flag normal detail;
flag packets detail;
flag policy detail;
```

You quickly begin monitoring the *msdp* log file after committing the changes:

```
[edit]
lab@r3# commit
commit complete

[edit]
lab@r3# run monitor start msdp

[edit]
lab@r3#
*** msdp ***
Apr  5 23:13:41 MSDP SENT Keepalive peer 10.0.3.4
Apr  5 23:13:49 RPD_MSDP_PEER_UP: MSDP peer 172.16.0.14 peer-group ext into
   Established state
Apr  5 23:13:49 MSDP RECV SA from 172.16.0.14 RP 130.130.0.1 count 2 len 32
Apr  5 23:13:49 MSDP Accept group 225.8.8.8 source 130.130.16.2 originator
   130.130.0.1 from peer 172.16.0.14
Apr  5 23:13:49 MSDP Accept group 225.0.0.1 source 130.130.16.2 originator
   130.130.0.1 from peer 172.16.0.14
Apr  5 23:13:49 MSDP HASH peer 10.0.3.4 adding 225.0.0.1.130.130.16.2 rp
   130.130.0.1 to bucket 8 count 1
Apr  5 23:13:49 MSDP HASH removing 225.0.0.1.130.130.16.2 from bucket 8 count 1
Apr  5 23:13:49 MSDP SENT SA to 10.0.3.4 RP 130.130.0.1 len 20
. . .
```

The highlights call out the successful establishment of the MSDP session between T1 and r3, and also show that source active messages were received from T1 (130.130.0.1) for the 225.0.0.1 and 225.8.8.8 groups. Note that the trace output indicates that only one of the SA messages was forwarded on to MSDP peer 10.0.3.4; a behavior that is in keeping with your *msdp* policy, which serves to block SAs for groups in the 225.8/16 range. The MSDP tracing configuration is removed from r3, because it is no longer needed; leaving it in place should cause no harm, however:

```
[edit]
lab@r3# delete protocols msdp traceoptions
```

You now analyze the SA messages received by r3:

```
[edit]
lab@r3# run show msdp source-active
Group address   Source address  Peer address    Originator      Flags
225.0.0.1       130.130.16.2    172.16.0.14     130.130.0.1     Accept
225.8.8.8       130.130.16.2    172.16.0.14     130.130.0.1     Accept,Filtered
```

The highlights confirm the receipt of two SA messages, and also confirm that the *msdp* policy is correctly filtering SAs in the 225.8/16 range. You next verify the proper conveyance of the SA for source 225.0.0.1 to r4. Recall that r3 and r4 are also MSDP peers in support of Any-Cast RP:

```
[edit]
lab@r4# run show msdp source-active
Group address   Source address  Peer address    Originator      Flags
225.0.0.1       130.130.16.2    10.0.3.3        130.130.0.1     Accept
```

With MSDP peering and policy confirmed, you move on to verify the correct flow of multicast traffic from MS2 to MR1. You start by analyzing the join state at r5:

```
[edit]
lab@r5# run show pim join 225.0.0.1 detail
Instance: PIM.master Family: INET
Group: 225.0.0.1
    Source: *
    RP: 10.0.3.34
    Flags: sparse,rptree,wildcard
    Upstream interface: at-0/2/1.0
    Downstream interfaces:
        fe-0/0/3.0

Group: 225.0.0.1
    Source: 130.130.16.2
    Flags: sparse,spt-pending
    Upstream interface: at-0/2/1.0
    Downstream interfaces:
        fe-0/0/3.0
```

The S,G join state confirms that r5 has received traffic from MS2 via the shared RP tree, and that it has successfully established a SPT as a result. Another positive indication that traffic is being delivered to MR1 is obtained by the determination that both r3 and r5 are displaying the same traffic rate for the 130.130.16.2, 225.0.0.1 route:

```
[edit]
lab@r3# run show multicast route source-prefix 130.130.16.2 extensive

Family: INET
Group           Source prefix     Act Pru NHid  Packets    IfMismatch Timeout
225.0.0.1       130.130.16.2   /32 A   F  79    4605       0          360
    Upstream interface: fe-0/0/2.0
    Session name: MALLOC
    Forwarding rate: 3 kBps (5 pps)

[edit]
```

```
lab@r5# run show multicast route source-prefix 130.130.16.2 extensive

Family: INET
Group           Source prefix     Act Pru NHid  Packets    IfMismatch Timeout
225.0.0.1       130.130.16.2   /32 A   F  105   4876       0          360
    Upstream interface: at-0/2/1.0
    Session name: MALLOC
    Forwarding rate: 3 kBps (5 pps)
```

r3 shows the correct upstream interface, and a packet rate of five packets per second, which matches the data rate shown at r5 nicely.

It may be difficult to confirm the operation of your multicast scoping, especially if you do not have access to the T1 device. In this example, you are provided with login information, which allows you to confirm that T1 has not learned of your Any-Cast RP:

```
[edit]
lab@r3# run telnet 130.130.0.1
Trying 130.130.0.1...
Connected to 130.130.0.1.
Escape character is '^]'.

T1-P1 (ttyp0)

login: lab
Password:
Last login: Sat Apr  5 14:10:42 on ttyd0

--- JUNOS 5.2R1.4 built 2002-01-30 17:03:27 UTC

lab@T1-P1> show pim rps
RP address      Type      Holdtime Timeout Active groups Group prefixes
130.130.0.1     static           0    None             1 224.0.0.0/4
```

Good, T1 has not learned your 10.0.3.34 Any-Cast RP. A quick glance at the PIM stanza confirms that T1 is configured to run Auto-RP and is listening to mapping messages, which means that it is configured to learn your RP in the event that scoping is not correctly configured:

```
[edit]
lab@T1-P1# show protocols pim
traceoptions {
    file pim;
    flag rp detail;
}
dense-groups {
    224.0.1.39/32;
```

```
    224.0.1.40/32;
}
rp {
    local {
        address 130.130.0.1;
    }
    auto-rp discovery;
}
interface all {
    mode sparse-dense;
}
```

The lack of Auto-RP entries at T1 confirms that the multicast scoping at `r3` is having the desired effect. You can also use the `show multicast scope` command to confirm scoping settings:

```
[edit]
lab@r3# run show multicast scope
                                                  Resolve
Scope name           Group Prefix       Interface              Rejects
announce             224.0.1.39/32      fe-0/0/2.0                   2
discover             224.0.1.40/32      fe-0/0/2.0                   0
local                239.255.0.0/16     fe-0/0/2.0                   0
```

The output confirms that the addition of a custom scope range, in this case to control Auto-RP messages, automatically adds scoping for administratively scoped addresses in the range of 239.255.0.0 through 239.255.255.255 (239.255/16). The confirmation techniques demonstrated in this section indicate that your network meets all specified criteria for the interdomain multicast scenario.

MSDP Summary

MSDP is used to support interdomain multicast and Any-Cast RPs. Normally routers configured to run MSDP are also configured to function as a PIM RP. MSDP operates by sending source active messages to remote peers (using RPF forwarding to prevent loops), which causes the remote RP/MSDP peer to join a SPT for the active source when there are local listeners. Once the remote receiver begins receiving traffic from the active source via the shared tree, it too signals a SPT to the source.

Multicast scoping is used to block certain class D addresses from being sent out the specified interface. Scoping is configured under the routing-options stanza; you need to define multiple scopes when your goal is to block multiple /32 class D addresses. Note that firewall filters do not operate on class D addresses.

You can filter MSDP SAs and PIM joins using routing policy. This section provided an example of how import policy can be used to filter source active messages received from an MSDP peer. A similar approach is used when the goal is to filter joins to a RP-based shared tree or when filtering MSDP SA advertisements.

Summary

This chapter provided various examples of JNCIE-level multicast configuration scenarios ranging from IGMP to interdomain multicast using MSDP.

Multicast configuration is not really that difficult to configure. What most candidates do find difficult is making sense of the dynamic state and ever-changing landscape that is a multicast network. The fact that multicast technologies are not deployed in a widespread manner also adds to the mystique and the general lack of understanding when it comes to how multicast protocols operate. The need for TS PICs in certain multicast environments can also hamper your ability to practice multicast configuration and troubleshooting in certain test beds and can lead to configuration mistakes in the JNCIE examination if the candidate does not design around the presence (or absence) of TS PICs.

A prepared JNCIE candidate will be ready to deploy multicast technology in scenarios similar to those demonstrated in this chapter.

Case Study: Multicast

The chapter case study is designed to simulate a JNCIE-level multicast configuration scenario. To keep things interesting, you will be adding your multicast configuration to the OSPF baseline configuration that was discovered and documented in the Chapter 1 case study. The OSPF baseline topology is shown in Figure 4.9 so you can reacquaint yourself with it.

Before starting the multicast case study, quickly verify the correct operation of the baseline network's OSPF IGP, IS-IS route redistribution, and IBGP/EBGP peerings. Refer to previous case studies for suggestions on how to quickly verify the baseline network's operation if needed. It is expected that a prepared JNCIE candidate will be able to complete this case study in approximately one hour with the resulting network meeting the majority of the specified behaviors and operational characteristics.

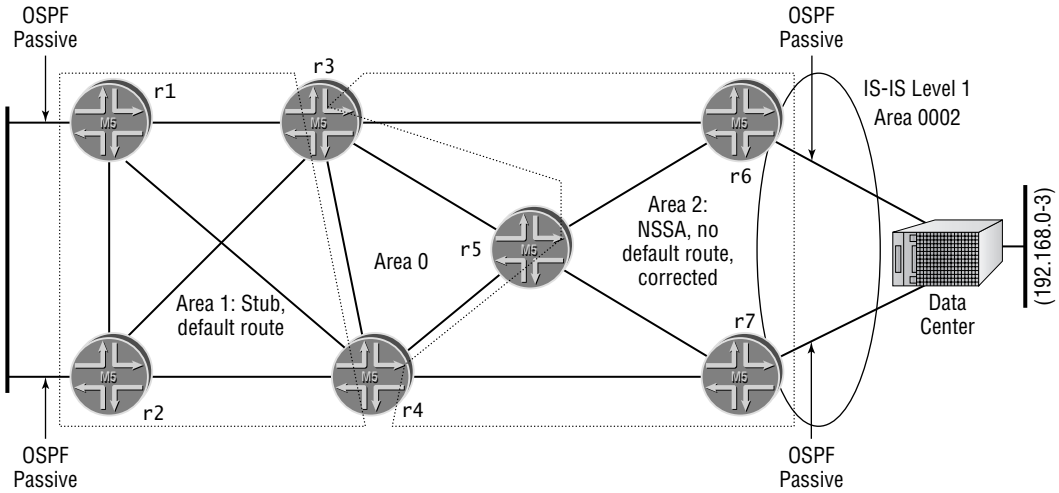
Configuration listings that identify the changes made to the baseline configurations of all five routers in the multicast test bed are provided at the end of the case study for comparison with your own configurations. To accommodate differing configuration approaches, various operational mode commands are included in the case study analysis to permit the comparison of your network to that of a known good example.

To complete the case study, you must configure your network to meet these criteria:

- Add your multicast configuration to the OSPF baseline network.
- Ensure that there is no single point of RP failure in your network.
- Deploy PIM version 1 with at least two candidate RPs for groups in the range of 224.0.0.0 through 224.255.255.255.
- Prevent external sites from discovering your RPs.
- Ensure that traffic sent to group 224.1.1.1 is never sent to C1. This traffic must be delivered to MR1.

- Configure r4 to deliver traffic sent to 224.2.2.2 to C1 without having it join a shared tree. Note that C1 is not configured to run MSDP.
- Filter all joins and source active messages for the 224.8.8.8 group.

FIGURE 4.9 OSPF discovery findings



Notes:

Loopback addresses have not been assigned to specific areas (lo0 address advertised in Router LSA in all areas).

Passive OSPF interfaces on P1 and data center segments.

No authentication or route summarization in effect; summaries (LSA type 3) allowed in all areas.

Redistribution of OSPF default route to data center from both r6 and r7 was broken. Fixed with default-metric command on r3, r4, and r5.

Data center router running IS-IS, Level 1. r6 and r7 compatibly configured and adjacent.

Redistribution of 192.168.0/24 through 192.168.3/24 into OSPF from IS-IS by both r6 and r7.

Adjustment to IS-IS level 1 external preference to ensure r6 and r7 always prefer IS-IS level 1 externals over OSPF externals.

All adjacencies up and full reachability confirmed.

Sub-optimal routing detected at the data center router for some locations. This is the result of random nexthop choice for its default route. Considered to be working as designed; no action taken.

You can assume that the T1 and C1 routers are correctly configured, and that you are not permitted to modify their configurations or log into them remotely.

Multicast Case Study Analysis

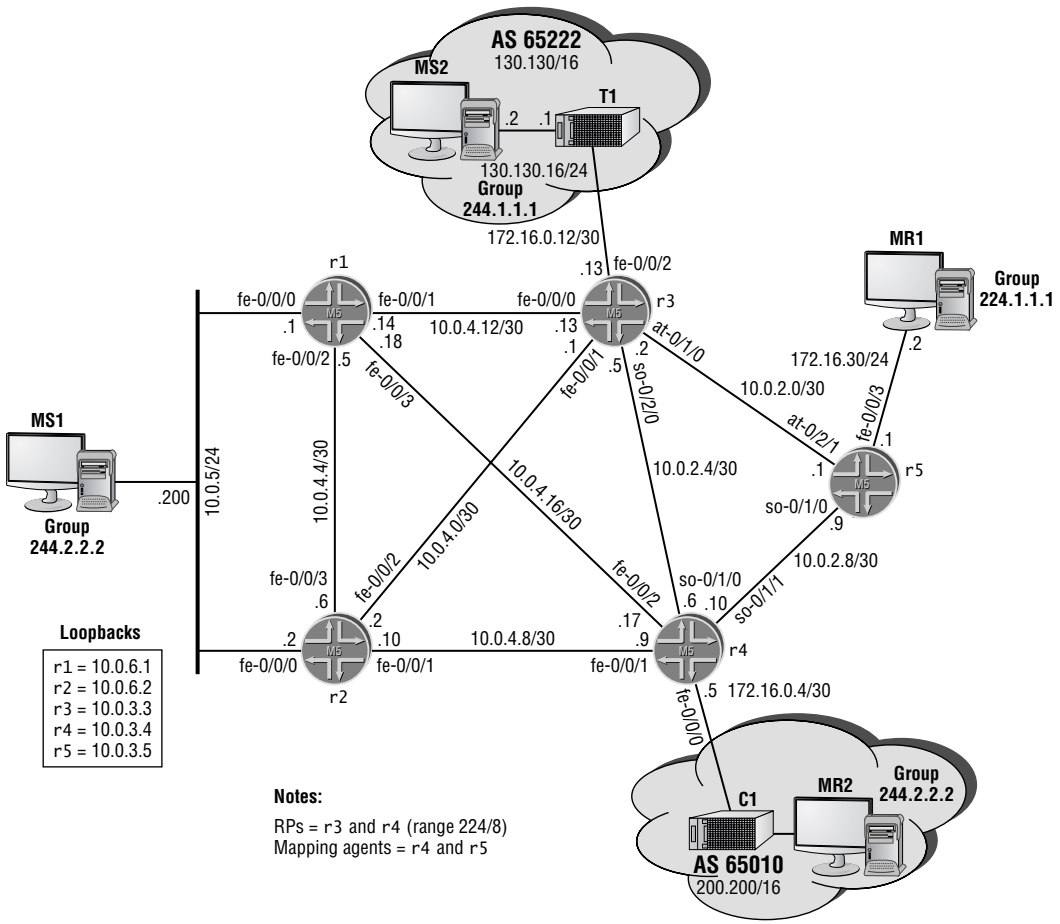
To save space, initial verification of the baseline network is not performed here. Refer to previous chapters for suggested baseline network verification techniques and expected displays; the OSPF baseline network is assumed to be operational. Each configuration requirement for the

case study is matched to one or more valid router configurations and, where applicable, the commands used to confirm that the network's operation adheres to the case study's restrictions and operational requirements. The case study analysis begins with the following multicast criteria:

- Ensure that there is no single point of RP failure in your network.
- Deploy PIM version 1 with at least two candidate RPs for groups in the range of 224.0.0.0 through 224.255.255.255.

Based on the need to use PIM version 1, you know that the bootstrap protocol cannot be used and a static RP approach does not meet the stated redundancy requirements. Being one to take a hint, you decide to deploy an Auto-RP based topology as shown in Figure 4.10.

FIGURE 4.10 Multicast case study design



The design shown in Figure 4.10 reflects the fact that TS PICs are installed in `r1` through `r4` only, which makes `r5` unable to function as an RP. Note that load balancing among your RPs is not a stated requirement. This means that you do not have to deploy Any-Cast (and the related MSDP session between your RPs) to meet the stipulations provided. Having two RP candidates,

and at least two mapping agents, provides the RP redundancy required in this example. The changes made to r3 are shown next:

```
[edit protocols pim]
lab@r3# show
dense-groups {
    224.0.1.39/32;
    224.0.1.40/32;
}
rp {
    local {
        address 10.0.3.3;
        group-ranges {
            224.0.0.0/8;
        }
    }
    auto-rp announce;
}
interface all {
    mode sparse-dense;
    version 1;
}
interface fxp0.0 {
    disable;
```

The configuration of r4 is similar to that shown for r3, with the exception of its local RP address and the use of the mapping keyword. The modifications made to r5 in support of the first two case study criteria are displayed next:

```
[edit protocols pim]
lab@r5# show
dense-groups {
    224.0.1.39/32;
    224.0.1.40/32;
}
rp {
    auto-rp mapping;
}
interface all {
    mode sparse-dense;
    version 1;
}
interface fxp0.0 {
    disable;
}
```

Proper PIM version and Auto-RP election are now verified at r5:

```
[edit protocols pim]
lab@r5# run show pim interfaces
Instance: PIM.master

Name                 Stat Mode       IP V State Count DR address
at-0/2/1.0           Up   SparseDense 4 1 P2P       1
fe-0/0/0.0           Up   SparseDense 4 1 DR        0 10.0.8.6
fe-0/0/1.0           Up   SparseDense 4 1 DR        0 10.0.8.9
lo0.0                Up   SparseDense 4 1 DR        0 10.0.3.5
so-0/1/0.0           Up   SparseDense 4 1 P2P       1

[edit protocols pim]
lab@r5# run show pim neighbors
Instance: PIM.master

Interface          IP V Mode        Option      Uptime Neighbor addr
at-0/2/1.0          4 1 SparseDense           00:06:22 10.0.2.2
so-0/1/0.0          4 1 SparseDense           00:05:27 10.0.2.10
```

These displays confirm the correct PIM protocol version, the required sparse-dense mode of operation (as needed for Auto-RP), and that PIM neighbors have been detected. You move on to verify RP election results, again at r5:

```
[edit protocols pim]
lab@r5# run show pim rps
Instance: PIM.master

Family: INET
RP address      Type      Holdtime Timeout Active groups Group prefixes
10.0.3.4        auto-rp        150     104             0 224.0.0.0/8

Family: INET6
RP address      Type      Holdtime Timeout Active groups Group prefixes
```

Proper RP mapping is confirmed when r3 displays the same Auto-RP elected RP:

```
[edit protocols pim]
lab@r3# run show pim rps
Instance: PIM.master

Family: INET
RP address      Type      Holdtime Timeout Active groups Group prefixes
10.0.3.4        auto-rp        150     147             0 224.0.0.0/8

10.0.3.3        static           0    None             0 224.0.0.0/8
```

```
Family: INET6
RP address      Type      Holdtime Timeout Active groups Group prefixes
```

The correct group range of 224/8 (224.0.0.0 through 224.255.255.255) is also confirmed by the 224/8 entry in this display. To support the initial case study requirements, r1 has its OSPF baseline configuration modified as shown next; r2 has similar changes (not shown):

```
[edit]
lab@r1# show protocols pim
dense-groups {
    224.0.1.39/32;
    224.0.1.40/32;
}
rp {
    auto-rp discovery;
}
interface all {
    mode sparse-dense;
    version 1;
}
interface fxp0.0 {
    disable;
}
```

Proper RP election is now verified at r1:

```
[edit]
lab@r1# run show pim rps
Instance: PIM.master

Family: INET
RP address      Type      Holdtime Timeout Active groups Group prefixes
10.0.3.4        auto-rp        150      95             0 224.0.0.0/8

Family: INET6
RP address      Type      Holdtime Timeout Active groups Group prefixes
```

Though not shown, you can assume that r2 provides a similar display, and that it too displays 10.0.3.4 as an Auto-RP learned RP. With a basic PIM version 1 configuration in all routers, and initial Auto-RP operation confirmed, you move on to these configuration criteria:

- Prevent external sites from discovering your RPs.
- Ensure that traffic sent to group 224.1.1.1 is never sent to C1. This traffic must be delivered to MR1.

Because you need to configure multicast scoping to prevent other networks from learning about your RPs, you decide to also use scoping at r4 in support of the requirement that traffic for group 224.1.1.1 is not to be sent to customer C1. Note that trying to meet the stated behavior using a PIM join policy such as this one could result in point loss:

```
[edit policy-options]
lab@r4# show policy-statement pim-join
term 1 {
    from {
        route-filter 224.1.1.1/32 exact reject;
    }
}
```

This is because the *pim-join* policy will cause problems when it prevents internal and external *,G joins for the 224.1.1.1 group. In this example, you need the added granularity of scoping, which can be applied to individual interfaces, instead of the global effects that a PIM join policy brings to the table. The changes made to r4's configuration are shown here with highlights added:

```
[edit]
lab@r4# show routing-options
static {
    route 10.0.200.0/24 {
        next-hop 10.0.1.102;
        no-readvertise;
    }
}
aggregate {
    route 10.0.0.0/16;
}
autonomous-system 65412;
multicast {
    scope auto-rp-announce {
        prefix 224.0.1.39/32;
        interface fe-0/0/0.0;
    }
    scope auto-rp-discover {
        prefix 224.0.1.40/32;
        interface fe-0/0/0.0;
    }
    scope block-224.1.1.1 {
        prefix 224.1.1.1/32;
        interface fe-0/0/0.0;
    }
}
```

In contrast, at r3 you need to scope only the Auto-RP related dense mode groups:

```
[edit routing-options multicast]
lab@r3# show
scope auto-rp-announce {
    prefix 224.0.1.39/32;
    interface fe-0/0/2.0;
}
scope auto-rp-discover {
    prefix 224.0.1.40/32;
    interface fe-0/0/2.0;
}
```

You have to limit your scope verification to the output of the `show multicast scope` command, given that you cannot access the T1 and C1 devices:

```
[edit routing-options multicast]
lab@r4# run show multicast scope
                                          Resolve
Scope name            Group Prefix        Interface       Rejects
auto-rp-announce      224.0.1.39/32       fe-0/0/0.0           10
auto-rp-discover      224.0.1.40/32       fe-0/0/0.0           11
block-224.1.1.1       224.1.1.1/32        fe-0/0/0.0            0
local                 239.255.0.0/16      fe-0/0/0.0            0
```

Note that both of the dense mode groups associated with Auto-RP related messages and the 224.1.1.1 group address are shown to be correctly blocked by the scoping configuration at r4. The non-zero entries in the `Resolve Rejects` column indicate that packets have been received, for which a resolve request was sent to the multicast routing protocol in the RE. The multicast routing protocol handles the resolve request by computing the packet's incoming and outgoing interface lists (IIL and OIL), and by analyzing the resulting OIL for scope settings that will prevent the transmission of the packet out that interface. The `Resolve Request` counter is incremented for the affected interface when the multicast routing protocol determines that scoping should block the packet's transmission on that interface. Note that this counter does not increment with the receipt of each such packet because the results of the resolve request are cached in the Packet Forwarding Engine (PFE) to cut back on resolution request traffic.

You can test the scope for 224.1.1.1 traffic by adding a static join on r4's fe-0/0/0 interface when traffic is sent to this prefix by MS1:

```
[edit protocols igmp]
lab@r4# set interface fe-0/0/0 static group 224.1.1.1

[edit protocols igmp]
lab@r4# commit
commit complete
```

```
[edit protocols igmp]
lab@r4# run show multicast scope
                                                  Resolve
Scope name           Group Prefix       Interface       Rejects
auto-rp-announce     224.0.1.39/32      fe-0/0/0.0           58
auto-rp-discover     224.0.1.40/32      fe-0/0/0.0           58
block-224.1.1.1      224.1.1.1/32       fe-0/0/0.0            9
local                239.255.0.0/16     fe-0/0/0.0            0
```

The non-zero count value confirms that packets addressed to group 224.1.1.1 are correctly blocked at r4's fe-0/0/0 interface. Also note how the fe-0/0/0 interface is pruned as a result of scoping; this behavior means that the *block-224.1.1.1* counter increments for only a brief period after each clearing of PIM joins at r4:

```
lab@r4# run show pim join detail 224.1.1.1
Instance: PIM.master Family: INET
Group: 224.1.1.1
    Source: *
    RP: 10.0.3.4
    Flags: sparse,rptree,wildcard
    Upstream interface: local
    Downstream interfaces:
        fe-0/0/0.0              (administratively scoped)

Group: 224.1.1.1
    Source: 10.0.5.200
    Flags: sparse,spt
    Upstream interface: fe-0/0/2.0
    Downstream interfaces:
```

Note the absence of downstream interfaces in the S,G join for the 224.1.1.1 group due to your administrative scoping. Be sure to remove the static join for 224.1.1.1 from r4 when satisfied with the operation of your scope settings. With scoping confirmed, you move on to the following criteria; note that some aspects of the first requirements have already been accommodated:

- Ensure that traffic sent to group 224.1.1.1 is never sent to C1. This traffic must be delivered to MR1.
- Configure r4 to deliver traffic sent from 10.0.5.200 to group 224.2.2.2 to C1 *without* having it join a shared tree. Note that C1 is not configured to run MSDP.

The first criterion requires a static *,G join on r5's fe-0/0/3 interface, and that you correctly establish a MSDP peering session to T1 from all candidate RPs. The wording of the second requirement might confuse candidates who are not familiar with the concept of source specific multicast (SSM). You do not need to configure IGMP version 3 on r4's fe-0/0/0 interface to support a static S,G join, but version 3 is needed to support SSM joins from attached hosts through IGMP. Note that configuring version 3 on r4's fe-0/0/0 interface causes no harm in this scenario.

The changes made to r5 are shown next:

```
[edit protocols igmp]
lab@r5# show
interface fe-0/0/3.0 {
    static {
        group 224.1.1.1;
    }
}

[edit]
lab@r5# show interfaces fe-0/0/3
unit 0 {
    family inet {
        address 172.16.30.2/24;
    }
}
```

Make sure to configure IP parameters on r5's fe-0/0/3 interface, which was not used in the OSPF baseline topology. The changes made to r4 in support of static IGMP joins is displayed next; note the presence of a source address in the static join declaration for group 224.2.2.2:

```
[edit protocols igmp]
lab@r4# show
interface fe-0/0/0.0 {
    static {
        group 224.2.2.2 {
            source 10.0.5.200;
        }
    }
}
```

To confirm your static join configuration, view the PIM join at r4 and r5. You expect to see a *,G, and ultimately a S,G join at r5 for the 224.2.2.2 group, but you should see only a S,G join at r4 for the 224.2.2.2 group:

```
[edit protocols igmp]
lab@r4# run show pim join 224.2.2.2 detail
Instance: PIM.master Family: INET
Group: 224.2.2.2
    Source: 10.0.5.200
    Flags: sparse
    Upstream interface: fe-0/0/2.0
    Downstream interfaces:
        fe-0/0/0.0
```

The S,G join is present at r4, as is the *,G entry at r5:

```
[edit]
lab@r5# run show pim join 224.1.1.1
Instance: PIM.master Family: INET
Group: 224.1.1.1
    Source: *
    RP: 10.0.3.4
    Flags: sparse,rptree,wildcard
    Upstream interface: so-0/1/0.0
```

With the static joins configured and confirmed, you need to configure MSDP peering at r3 and r4 so that source active messages are received from T1 when MS1 is sending to group 224.1.1.1. Note that you need to MSDP peer from r3 and r4 to T1, because either r3 or r4 can function as the RP for your multicast domain. You do not need to peer the two RPs with MSDP. Because you have not configured Any-Cast in this example, only one of the two RP candidates will be active for all groups in the 224/8 range at any given time. The changes made to r3 are displayed:

```
[edit]
lab@r3# show protocols msdp
group t1 {
    local-address 172.16.0.13;
    peer 172.16.0.14;
}
```

The MSDP configuration at r4 is similar to that shown for r3, excepting the use of loopback-based peering addresses. The use of lo0 addressing is necessary at r4 in this case, because it does not have a route to the 172.16.0.12/30 peering subnet between r3 and the T1 router:

```
[edit]
lab@r4# show protocols msdp
group t1 {
    local-address 10.0.3.4;
    peer 130.130.0.1;
}
```

Proper MSDP session establishment is now verified at both r3 and r4, as is the receipt of source active messages for 130.130.16.2:

```
[edit]
lab@r3# run show msdp
Peer address     Local address   State       Last up/down Peer-Group
172.16.0.14      172.16.0.13     Established     00:00:25 t1

[edit]
lab@r3# run show msdp source-active
```

```
Group address   Source address  Peer address    Originator      Flags
224.1.1.1       130.130.16.2    172.16.0.14     130.130.0.1     Accept
```

The same commands are now issued at r4:

```
[edit]
lab@r4# run show msdp
Peer address    Local address   State       Last up/down Peer-Group
130.130.0.1     10.0.3.4        Established     00:00:05 t1

[edit]
lab@r4# run show msdp source-active
Group address   Source address  Peer address    Originator      Flags
224.1.1.1       130.130.16.2    130.130.0.1     130.130.0.1     Accept
224.2.2.2       10.0.5.200      local           10.0.3.4        Accept
```

With MSDP operational, and assuming that both multicast sources are actively sending, you expect to see S,G join state at r4 and r5, as well as indications that the multicast traffic is being forwarded out the correct interfaces.

```
[edit protocols igmp]
lab@r4# run show pim join 224.2.2.2 detail
Instance: PIM.master Family: INET
Group: 224.2.2.2
    Source: 10.0.5.200
    Flags: sparse
    Upstream interface: fe-0/0/2.0
    Downstream interfaces:
        fe-0/0/0.0

[edit protocols igmp]
lab@r4# run show multicast route source-prefix 10.0.5.200 extensive

Family: INET
Group           Source prefix     Act Pru NHid  Packets    IfMismatch Timeout
224.2.2.2       10.0.5.200     /32 A   F  79    11323      1          360
    Upstream interface: fe-0/0/2.0
    Session name: Multimedia Conference Calls
    Forwarding rate: 5 kBps (16 pps)

[edit protocols igmp]
lab@r4# run show multicast next-hops 79
Family: INET
ID      Refcount  KRefcount Downstream interface
79              2          1 fe-0/0/0.0
```

```
Family: INET6


[edit protocols igmp]
```

r4 has the expected S,G state, and the interface linking it to C1 is correctly listed as a downstream interface for the 224.2.2.2 group. Also of note is the 16-packet-per-second data rate and forwarding indication in the `show multicast route` display, because this provides good indication that traffic is correctly flowing from MS1 to C1. With forwarding confirmed at r4, the same commands are issued at r5 with attention focused on the 224.1.1.1 join and forwarding state:

```
[edit]
lab@r5# run show pim join 224.1.1.1 detail
Instance: PIM.master Family: INET
Group: 224.1.1.1
    Source: *
    RP: 10.0.3.4
    Flags: sparse,rptree,wildcard
    Upstream interface: so-0/1/0.0
    Downstream interfaces:
        fe-0/0/3.0

Group: 224.1.1.1
    Source: 130.130.16.2
    Flags: sparse,spt
    Upstream interface: at-0/2/1.0
    Downstream interfaces:
        fe-0/0/3.0
```

The join state shows both a shared tree and SPT join for the 224.1.1.1 group, as expected. Note that the S,G join list at-0/2/1 is the upstream interface, which is in keeping with its lying on the shortest path back to source MS2. At this time, r4 is the route of the shared tree, and this accounts for the *,G entry listing the so-0/1/0 interface as upstream.

```
[edit]
lab@r5# run show multicast route source-prefix 130.130.16.2 extensive

Family: INET
Group           Source prefix     Act Pru NHid  Packets    IfMismatch Timeout
224.1.1.1       130.130.16.2   /32 A   F  71    3191       1          360
    Upstream interface: at-0/2/1.0
    Session name: ST Multicast Groups
    Forwarding rate: 4 kBps (8 pps)
```

```
[edit]
lab@r5# run show multicast next-hops 71
Family: INET
ID      Refcount  KRefcount Downstream interface
71             2          1 fe-0/0/3.0


Family: INET6
```

The displays at r5 confirm that packets associated with the 224.1.1.1 group are being correctly forwarded to MR1 using r5's fe-0/0/3 interface. With multicast forwarding confirmed, the last criterion to be addressed relates to your need to filter PIM join and MSDP SA messages:

- Filter all joins and SAs for the 224.8.8.8 group.

This task is designed to verify that you can control the source active and PIM joins that your RPs will accept. The following policy and configuration changes correctly configure r3 to block source active messages and PIM joins to the 224.8.8.8 group, regardless of their source:

```
[edit]
lab@r3# show policy-options policy-statement block-224.8.8.8
term 1 {
    from {
        route-filter 224.8.8.8/32 exact reject;
    }
}

[edit]
lab@r3# show protocols pim import
import block-224.8.8.8;

[edit]
lab@r3# show protocols msdp import
import block-224.8.8.8;
```

Note that the *block-224.8.8.8* policy is applied globally to the MSDP instance in keeping with the requirement that *all* SA messages to 224.8.8.8 be blocked. It will be difficult to confirm the correct operation of MSDP unless you can access and configure the MS2 device so that it begins sending to the 224.8.8.8 group. To test local RP behavior, just add a static join to one of the routers:

```
[edit protocols igmp]
lab@r2# show
interface fe-0/0/0.0 {
    static {
        group 224.8.8.8;
```

```
    }
}

[edit protocols igmp]
lab@r2# run show pim join 224.8.8.8 detail
Instance: PIM.master Family: INET
Group: 224.8.8.8
    Source: *
    RP: 10.0.3.4
    Flags: sparse,rptree,wildcard
    Upstream interface: fe-0/0/1.0
    Downstream interfaces:
        fe-0/0/0.0
```

Note that r2 has generated a *,G join toward the RP using its fe-0/0/1 interface. Correct join filtering is confirmed by the absence of a *,G join for the 224.8.8.8 group at the current RP, which is r4 in this case:

```
[edit]
lab@r4# run show pim join 224.8.8.8
Instance: PIM.master Family: INET


[edit]
lab@r4#
```

The lack of a *,G join at the RP provides a good indication that your *block-224.8.8.8* policy is correctly filtering joins. These results, combined with the previous confirmation steps, indicate that your network is operating within the confines of all case study criteria provided. Congratulations!

Multicast Case Study Configurations

The changes made to the OSPF baseline network topology to support the multicast case study are listed next for all routers in the multicast test bed with highlights added as needed to call out changes to existing configuration stanzas (see Listings 4.1 to 4.5).

Listing 4.1: Multicast Case Study Configuration for *r1*

```
[edit]
lab@r1# show protocols pim
dense-groups {
    224.0.1.39/32;
    224.0.1.40/32;
}
rp {
```

```
    auto-rp discovery;
}
interface all {
    mode sparse-dense;
    version 1;
}
interface fxp0.0 {
    disable;
```

Listing 4.2: Multicast Case Study Configuration for *r2*

```
[edit]
lab@r2# show protocols pim
dense-groups {
    224.0.1.39/32;
    224.0.1.40/32;
}
rp {
    auto-rp discovery;
}
interface all {
    mode sparse-dense;
    version 1;
}
interface fxp0.0 {
    disable;
}
```

Listing 4.3: Multicast Case Study Configuration for *r3*

```
[edit]
lab@r3# show routing-options
static {
    route 10.0.200.0/24 {
        next-hop 10.0.1.102;
        no-readvertise;
    }
}
aggregate {
    route 10.0.0.0/16;
}
autonomous-system 65412;
multicast {
    scope auto-rp-announce {
```

```
        prefix 224.0.1.39/32;
        interface fe-0/0/2.0;
    }
    scope auto-rp-discover {
        prefix 224.0.1.40/32;
        interface fe-0/0/2.0;
    }
}

[edit]
lab@r3# show protocols pim
dense-groups {
    224.0.1.39/32;
    224.0.1.40/32;
}
import block-224.8.8.8;
rp {
    local {
        address 10.0.3.3;
        group-ranges {
            224.0.0.0/8;
        }
    }
    auto-rp announce;
}
interface all {
    mode sparse-dense;
    version 1;
}
interface fxp0.0 {
    disable;
}

[edit]
lab@r3# show protocols msdp
group t1 {
    import block-224.8.8.8;
    local-address 172.16.0.13;
    peer 172.16.0.14;
}

[edit]
```

```
lab@r3# show policy-options policy-statement block-224.8.8.8
term 1 {
    from {
        route-filter 224.8.8.8/32 exact reject;
    }
}
```

Listing 4.4: Multicast Case Study Configuration for *r4*

```
[edit]
lab@r4# show routing-options
static {
    route 10.0.200.0/24 {
        next-hop 10.0.1.102;
        no-readvertise;
    }
}
aggregate {
    route 10.0.0.0/16;
}
autonomous-system 65412;
multicast {
    scope auto-rp-announce {
        prefix 224.0.1.39/32;
        interface fe-0/0/0.0;
    }
    scope auto-rp-discover {
        prefix 224.0.1.40/32;
        interface fe-0/0/0.0;
    }
    scope block-224.1.1.1 {
        prefix 224.1.1.1/32;
        interface fe-0/0/0.0;
    }
}

[edit]
lab@r4# show protocols pim
dense-groups {
    224.0.1.39/32;
    224.0.1.40/32;
}
import block-224.8.8.8;
rp {
```

```
    local {
        address 10.0.3.4;
        group-ranges {
            224.0.0.0/8;
        }
    }
    auto-rp mapping;
}
interface all {
    mode sparse-dense;
    version 1;
}
interface fxp0.0 {
    disable;
}

[edit]
lab@r4# show protocols msdp
group t1 {
    import block-224.8.8.8;
    local-address 10.0.3.4;
    peer 130.130.0.1;
}

[edit]
lab@r4# show protocols igmp
interface fe-0/0/0.0 {
    version 3;
    static {
        group 224.2.2.2 {
            source 10.0.5.200;
        }
    }
}

[edit]
lab@r4# show policy-options policy-statement block-224.8.8.8
term 1 {
    from {
        route-filter 224.8.8.8/32 exact reject;
    }
}
```

Listing 4.5: Multicast Case Study Configuration for *r5*

```
[edit]
lab@r5# show interfaces fe-0/0/3
unit 0 {
    family inet {
        address 172.16.30.2/24;
    }
}

[edit]
lab@r5# show protocols pim
dense-groups {
    224.0.1.39/32;
    224.0.1.40/32;
}
rp {
    auto-rp mapping;
}
interface all {
    mode sparse-dense;
    version 1;
}
interface fxp0.0 {
    disable;
}

[edit]
lab@r5# show protocols igmp
interface fe-0/0/3.0 {
    static {
        group 224.1.1.1;
    }
}
```

r6 was not part of the multicast test bed. No changes were made to its configuration as part of the chapter's case study.

r7 was not part of the multicast test bed. No changes were made to its configuration as part of the chapter's case study.

Spot the Issues: Review Questions

1. Is this a valid configuration for r4 according to the case study requirements?

```
[edit protocols]
lab@r4# show pim
dense-groups {
    224.0.1.39/32;
    224.0.1.40/32;
}
rp {
    local {
        address 10.0.3.4;
        group-ranges {
            224.0.0.0/3;
        }
    }
    auto-rp mapping;
}
interface all {
    mode sparse-dense;
    version 1;
}
interface fxp0.0 {
    disable;
}
```

2. Interdomain multicast is not working in the case study topology. Can you locate the problem in r3's configuration?

```
[edit protocols pim]
lab@r3# show
dense-groups {
    224.0.1.39/32;
    224.0.1.40/32;
}
import block-224.8.8.8;
rp {
    local {
        address 10.0.3.3;
```

```
        group-ranges {
            224.0.0.0/8;
        }
    }
    auto-rp announce;
}
interface fxp0.0 {
    disable;
}
interface fe-0/0/0.0 {
    mode sparse-dense;
    version 1;
}
interface fe-0/0/1.0 {
    mode sparse-dense;
    version 1;
}
interface so-0/2/0.100 {
    mode sparse-dense;
    version 1;
}
interface at-0/1/0.0 {
    mode sparse-dense;
    version 1;
}
interface lo0.0 {
    mode sparse-dense;
    version 1;
}
```

3. Why is the bootstrap protocol not working between r2 and r4?

```
[edit]
lab@r2# show protocols pim
interface all {
    mode sparse;
    version 1;
}
interface fxp0.0 {
    disable;
}
```

```
[edit]
lab@r4# show protocols pim
rp {
    bootstrap-priority 10;
    local {
        address 10.0.3.4;
    }
}
interface all {
    mode sparse;
    version 1;
}
interface fxp0.0 {
    disable;
}
```

4. r3 is filtering all SA messages with the case study–based MSDP configuration shown here. Can you spot the issue?

```
[edit]
lab@r3# show protocols msdp
import block-224.8.8.8;
group t1 {
    local-address 172.16.0.13;
    peer 172.16.0.14;
}

[edit]
lab@r3# show policy-options policy-statement block-224.8.8.8
term 1 {
    from {
        route-filter 224.8.8.8/32 exact;
    }
}
then reject;
```

5. DVMRP is not working correctly at r5 in the topology shown earlier in Figure 4.2. Can you spot the problem?

```
[edit]
lab@r5# show routing-options
static {
    route 10.0.200.0/24 {
```

```
        next-hop 10.0.1.102;
        no-readvertise;
    }
}
rib-groups {
    dvmrp-rg {
        export-rib inet.2;
        import-rib inet.2;
    }
}
autonomous-system 65412;

[edit]
lab@r5# show protocols dvmrp
rib-group dvmrp-rg;
interface all {
    hold-time 40;
}
interface fxp0.0 {
    disable;
}
```

6. With r4 shut down, r3 has been elected the RP in the topology shown earlier in Figure 4.6. However, r1 never shows a S,G join for source 10.0.5.200 and group 224.1.1.1, and the multicast traffic is not being delivered to MR1. Can you spot the problem based on this display?

```
lab@r3> show interfaces terse
Interface               Admin Link Proto Local                 Remote
fe-0/0/0                up    up
fe-0/0/0.0              up    up   inet  10.0.4.13/30
fe-0/0/1                up    up
fe-0/0/1.0              up    up   inet  10.0.4.1/30
fe-0/0/2                up    up
fe-0/0/2.0              up    up   inet  172.16.0.13/30
fe-0/0/3                up    down
fe-0/0/3.0              up    down inet  10.0.2.14/30
at-0/1/0                up    up
at-0/1/0.0              up    up   inet  10.0.2.2/30
at-0/1/1                up    down
so-0/2/0                up    up
so-0/2/0.100            up    up   inet  10.0.2.5/30
so-0/2/1                up    down
```

```
so-0/2/2                up    down
so-0/2/3                up    down
dsc                     up    up
fxp0                    up    up
fxp0.0                  up    up   inet  10.0.1.3/24
fxp1                    up    up
fxp1.0                  up    up   tnp   4
gre                     up    up
ipip                    up    up
lo0                     up    up
lo0.0                   up    up   inet  10.0.3.3            --> 0/0
lsi                     up    up
mtun                    up    up
pimd                    up    up
pime                    up    up
tap                     up    up
```

Spot the Issues: Answers to Review Questions

1. No. The problem with r4's configuration relates to the incorrect specification of the class D group range as 224/3 instead of 224/4. Although this configuration commits, other routers will not install r4 as an RP with this group range setting. Note that omitting the group-range statement results in a default of 224/4.
2. The problem relates to the lack of PIM support on r3's fe-0/0/2 interface. Although r3 will receive MSDP SAs with this configuration, it will be unable to forward S,G joins to T1, and this prevents the flow of multicast from MS2.
3. The bootstrap protocol requires PIM version 2. Both r2 and r3 are set for version 1. Note that with this configuration, r4 elects itself the BSR and displays itself as a bootstrap-learned RP; the problem is that other routers in the test bed do not share r4's view.
4. The issue with this configuration lies in the *block-224.8.8.8* policy, and the fact that the reject action was not added as part of term 1. As written, the reject action is considered a final (unnamed) term that matches on all SA messages due to the lack of explicit match criteria in the final term. A corrected policy is shown here:

```
[edit]
lab@r3# show policy-options policy-statement block-224.8.8.8
term 1 {
    from {
        route-filter 224.8.8.8/32 exact;
    }
    then reject;
}
```

5. The problem in this example is a lack of interface RIB group definition under the routing-options stanza, which prevents the installation of interface routes in the inet.2 table. The presence of interface routes is necessary in inet.2 to accommodate RPF checks, as shown next:

```
[edit]
lab@r5# run show multicast rpf 10.0.2.0/30
Multicast RPF table: inet.2, 11 entries



[edit]
lab@r5#
```

To correct the problem, create an interface RIB group as shown here:

```
[edit]
lab@r5# show routing-options
interface-routes {
    rib-group inet interface-rib;
}
static {
    route 10.0.200.0/24 {
        next-hop 10.0.1.102;
        no-readvertise;
    }
}
rib-groups {
    interface-rib {
        import-rib [ inet.0 inet.2 ];
    }
    dvmrp-rg {
        export-rib inet.2;
        import-rib inet.2;
    }
}
autonomous-system 65412;
```

With interface routes in the `inet.2` table, RPF checks will succeed.

```
[edit]
lab@r5# run show multicast rpf 10.0.2.0/30
Multicast RPF table: inet.2, 21 entries

10.0.2.0/30
    Protocol: Direct
    Interface: at-0/2/1.0

10.0.2.1/32
    Protocol: Local
```

6. In this case, the problem is a lack of Tunnel Services PIC hardware at r3. Although r3 displays a pimd and pime interface, the lack of TS PIC hardware means that r3 cannot back up these virtual interfaces. Without the TS PIC, r3 cannot perform register message de-encapsulation, so r5 never receives traffic on the shared tree, and therefore r5 never initiates a S,G join back to the source. After re-installing the TS PIC in r3, the presence of a PIM de-encapsulation interface is confirmed.

```
lab@r3> show interfaces terse
Interface               Admin Link Proto Local           Remote
fe-0/0/0                up    up
. . .
mt-0/3/0                up    up
pd-0/3/0                up    up
pd-0/3/0.32768          up    up   inet
pe-0/3/0                up    up
. . .
tap                     up    up
```

Chapter

5

IPv6

JNCIE LAB SKILLS COVERED IN THIS CHAPTER:

- ✓ **IPv6 Addressing and Neighbor Discovery**
- ✓ **IPv6 IGP support**
  - RIPng, OSPF3, and IS-IS
- ✓ **IPv6 BGP Support**
  - EBGP
  - IBGP
- ✓ **Tunneling IPv6**

This chapter exposes the reader to a variety of JNCIE-level configuration scenarios designed to test the candidate's understanding of IPv6 support in JUNOS software release 5.6. It is assumed that the reader already possesses a working knowledge of IPv6 protocols to the extent covered in the *JNCIS Study Guide* (Sybex, 2003).

Juniper Networks routing platforms running JUNOS software release 5.6 or later support a number of IPv6 protocols and standards, including automatic address configuration (as described in RFC 2462), neighbor discovery, Multicast Listener Discovery (MLD), Routing Information Protocol-Next Generation (RIPng), OSPF version 3 (OSPF3), native EBGP peering using Multiprotocol BGP (MBGP), and support for IPv6 over IPv4 tunnels. JUNOS software also supports the IS-IS extensions needed to accommodate IPv6 routing.

As with previous chapters, examples of key operational mode command output is provided to allow the comparison of your network's operation to that of a known good example. Examples of baseline configuration modifications that are known to meet all case study requirements are provided at the end of the case study for all routers in the IPv6 test bed.

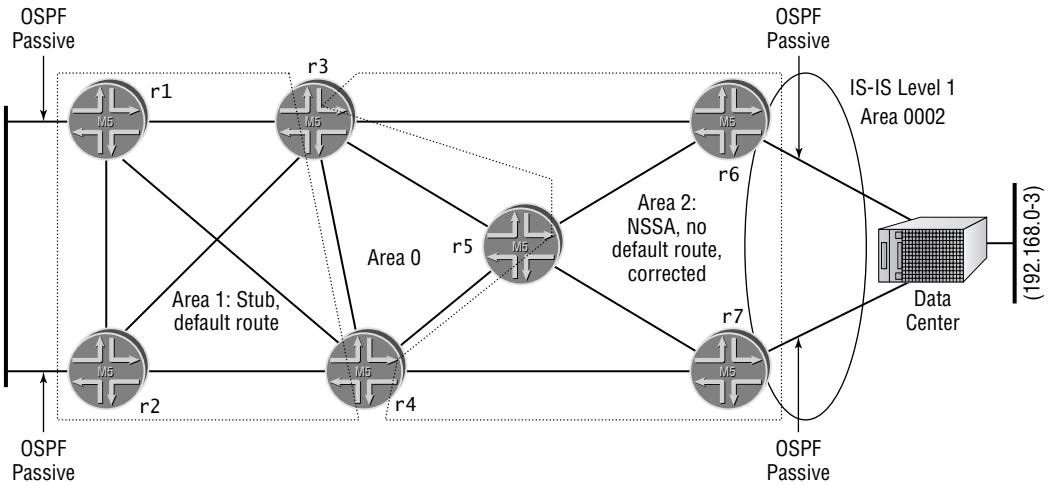
The configuration scenarios demonstrated in the chapter body are based on the OSPF baseline configuration as discovered in the body of Chapter 1, "Network Discovery and Verification." If you are unsure as to the state of your test bed, you should take a few moments to load up and confirm the operation of the OSPF IGP discovery configuration before proceeding. Figure 5.1 reviews the OSPF IGP baseline topology.

Note that some of the routers in the JNCIE test bed are not brought into play during the course of this chapter. In many cases, a JNCIE candidate is expected to configure a given functionality on a subset of routers in an effort to save time while the candidate still has to demonstrate the desired skill set.

IPv6 Addressing and Router Advertisements

One of the main motivations for deploying IPv6 is its expanded addressing capabilities (128 bits versus 32), which alleviates concerns about the diminishing IPv4 address space while also reducing the need for Network Address/Port Address Translation (NAT/PAT) devices. IPv6 supports unicast, multicast, and AnyCast addressing. AnyCast addresses allow a packet to be routed to the nearest member of an AnyCast group. AnyCast addresses can not be distinguished from conventional unicast addresses without explicit AnyCast group configuration; AnyCast group membership is not supported in JUNOS software as of release 5.6.

FIGURE 5.1 Summary of OSPF IGP discovery



Notes:

Loopback addresses have not been assigned to specific areas (lo0 address advertised in Router LSA in all areas).

Passive OSPF interfaces on P1 and data center segments.

No authentication or route summarization in effect; summaries (LSA type 3) allowed in all areas.

Redistribution of OSPF default route to data center from both `r6` and `r7` was broken. Fixed with default-metric command on `r3`, `r4`, and `r5`.

Data center router running IS-IS, Level 1. `r6` and `r7` compatibly configured and adjacent.

Redistribution of 192.168.0/24 through 192.168.3/24 into OSPF from IS-IS by both `r6` and `r7`.

Adjustment to IS-IS level 1 external preference to ensure `r6` and `r7` always prefer IS-IS level 1 externals over OSPF externals.

All adjacencies up and full reachability confirmed.

Sub-optimal routing detected at the data center router for some locations. This is the result of random nexthop choice for its default route. Considered to be working as designed; no action taken.

Unicast and multicast addresses support scoping, which limits the effective range of the corresponding address. For example, a unicast address can be considered to have a global, site-local, or link-local scope; the scope of an IPv6 address is determined by the coding of the Format Prefix (FP), which is the first 10 bits in the address. A global address has a worldwide scope while a link-local address can be used only on a particular link. A site-local address is analogous to the use of private addressing (RFC 1918), in that the address is significant only within a particular site. Note that multicast addresses support 16 scoping levels, and that broadcast is facilitated through multicast addressing.

JUNOS software automatically creates a link-local address for any interface that is enabled for IPv6 operation. One or more global or site-local addresses can also be assigned to each IPv6 interface as needed. Note that explicitly assigning an IPv6 address does not negate the automatic generation of a link-local address. When assigning a unicast address, you can specify all 128 bits manually, or you can have the router automatically compute a 64-bit identifier portion using the IEEE's Extended Unique Identifier (EUI-64) format. Note that RFC 2373, "IP Version 6

Addressing Architecture," states that all IPv6 unicast addresses beginning with a Format Prefix of 001 through 111 are required to have a 64-bit interface identifier in EUI-64 format. This requirement explains why operators often configure their routers to automatically compute their 64-bit interface identifiers with the `eui-64` keyword, as demonstrated later in this chapter. You can not use EUI-64 on loopback interfaces because M-series and T-series routers require that a 128-bit prefix be assigned to the loopback interface.

The IPv6 loopback address is 0:0:0:0:0:0:0:1. The IPv6 loopback address functions identically to the 127.0.0.1 address used in IPv4 and is not always required. Note that the IPv6 loopback address can be represented as *::1* using standard IPv6 address notation.

The IPv6 neighbor discovery protocol provides IPv6 with the functionality associated with IPv4's ARP, ICMP redirect, and router discovery protocols; the ability to detect duplicate addresses is important when considering that IPv6 devices often deploy some type of address auto-configuration. Neighbor solicitation messages, which are used for IPv6 ARP and duplicate address detection, are enabled by default in the release 5.6 JUNOS software used to develop this book. You must specifically configure the router to generate router advertisement messages to communicate various parameters and timers to attached hosts when needed, however. Router advertisement messages are used to communicate parameters to attached hosts to assist them in auto-configuration.

You should verify the operation of the OSPF baseline network in accordance with the steps outlined in previous chapters, being sure that you confine your activities to the subset of routers that make up the IPv6 test bed. Because `r6`, `r7`, and the EBGP peers are currently not involved in the IPv6 test bed, you can expect to see a few active BGP sessions and some missing OSPF routes.

After confirmation of the OSPF baseline network as complete, your IPv6 configuration scenario begins with the following configuration requirements:

- Configure IPv6 addressing according to Figure 5.2.
- Configure `r1` and `r2` so that IPv6 hosts attached to the FEC0:0:5:0::/64 subnet can auto-configure a site-local address.

FIGURE 5.2 IPv6 topology and addressing

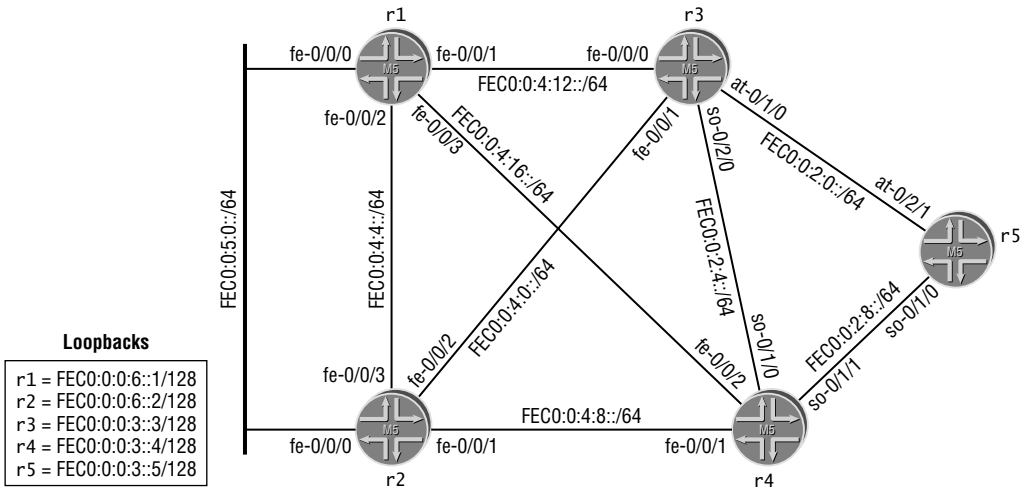


Figure 5.2 provides the additional details you need to complete the IPv6 addressing and neighbor discovery scenario. Note that the IPv6 addressing plan is designed to produce IPv6 addresses that are "similar" to the existing IPv4 addressing in the test bed. The initial addressing plan does not involve IPv4-compatible IPv6 addressing, however.

Assigning IPv6 Addresses

You begin IPv6 address assignment at `r5` by configuring its loopback interface with support for the `inet6` family using the address shown in Figure 5.2:

```
[edit interfaces lo0 unit 0]
lab@r5# set family inet6 address fec0:0:0:3::5/128
```

IPv6 addressing is now added to `r5`'s ATM and POS interfaces:

```
[edit interfaces]
lab@r5# edit so-0/1/0 unit 0

[edit interfaces so-0/1/0 unit 0]
lab@r5# set family inet6 address fec0:0:2:8::/64 eui-64

[edit interfaces so-0/1/0 unit 0]
lab@r5# up 2

[edit interfaces]
lab@r5# edit at-0/2/1 unit 0

[edit interfaces at-0/2/1 unit 0]
lab@r5# set family inet6 address fec0:0:2:0::/64 eui-64
```

The `eui-64` keyword instructs the router to automatically compute the 64-bit interface identifier portion for these interfaces. The requirements posed in this example do not mandate the use of EUI-64 addresses such that manual assignment of the full 128-bit address is also possible. One drawback to using EUI-64 addressing is that you need to know the interface's 48-bit identifier in order to know what address will be computed. In many cases, the operator displays the computed address (using an Out of Band management network in some cases) so that it can be copied into a capture buffer for subsequent use in ping and traceroute commands; a manually assigned address can adhere to a locally administered scheme that allows *a priori* determination of what address is in effect on any given interface.

The configuration changes are displayed:

```
[edit]
lab@r5# show interfaces so-0/1/0
encapsulation ppp;
unit 0 {
```

```
    family inet {
        address 10.0.2.9/30;
    }
    family inet6 {
        address fec0:0:2:8::/64 {
            eui-64;
        }
    }
}

[edit]
lab@r5# show interfaces at-0/2/1
atm-options {
    vpi 0 {
        maximum-vcs 64;
    }
}
unit 0 {
    point-to-point;
    vci 50;
    family inet {
        address 10.0.2.1/30;
    }
    family inet6 {
        address fec0:0:2:0::/64 {
            eui-64;
        }
    }
}

[edit]
lab@r5# show interfaces lo0
unit 0 {
    family inet {
        address 10.0.3.5/32;
    }
    family inet6 {
        address fec0:0:0:3::5/128;
    }
}
```

Be sure that you assign the IPv6 addressing shown earlier in Figure 5.2 to the remaining routers in the IPv6 test bed before proceeding to the verification section. Do not forget to commit your changes!

Configuring Router Advertisements

Although IPv6 router advertisements are not difficult to configure in JUNOS software, the wording of your objective intentionally omits the words "router advertisement" to verify that the candidate is familiar with the functions performed by router advertisement messages as defined by RFC 2461, "Neighbor Discovery for IP Version 6 (IPv6)." By default, router advertisement messages are disabled on M-series and T-series routers. Router advertisements are configured at the `[edit protocols router-advertisement]` hierarchy.

To meet the behavior specified in this example, you need to configure `r1` and `r2` to generate messages that advertise the FEC0:0:5:0::/64 prefix as usable for stateless address configuration (based on EUI-64). Note that the router only advertises a prefix out a particular interface when it matches a subnet that is assigned to that interface. The following command configures `r1` for the required behavior:

```
[edit protocols router-advertisement]
lab@r1# set interface fe-0/0/0 prefix fec0:0:5:0::/64
```

By default, the autonomous-configuration and on-link flags are set as needed for auto-configuration of an IPv6 address, as is required by this scenario. If desired, you can explicitly list the `on-link` and `autonomous` arguments, or their opposites (`no-on-link`, `no-autonomous`), when you define a prefix for inclusion in router advertisement messages. The configuration changes are displayed at `r1`:

```
[edit protocols router-advertisement]
lab@r1# show
interface fe-0/0/0.0 {
    prefix fec0:0:5:0::/64;
}
```

Note that a similar configuration is required at `r2`. Make sure that you commit all changes before proceeding to the next section.

Verifying IPv6 Addressing

IPv6 address verification proceeds as with IPv4 addressing. You begin by displaying the assigned addresses to confirm the correct association of IPv6 address to physical interfaces and logical units. The following display confirms that `r4` has been correctly configured according to the information shown earlier in Figure 5.2:

```
[edit interfaces]
lab@r4# run show interfaces terse
```

```
Interface               Admin Link Proto Local                 Remote
fe-0/0/0                up    down
fe-0/0/0.0              up    down inet  172.16.0.5/30
fe-0/0/1                up    up
fe-0/0/1.0              up    up   inet  10.0.4.9/30
                                   inet6 fe80::290:69ff:fe6b:3001/64
                                         fec0:0:4:8:290:69ff:fe6b:3001/64
fe-0/0/2                up    up
fe-0/0/2.0              up    up   inet  10.0.4.17/30
                                   inet6 fe80::290:69ff:fe6b:3002/64
                                         fec0:0:4:16:290:69ff:fe6b:3002/64
fe-0/0/3                up    down
fe-0/0/3.0              up    down inet  10.0.2.18/30
so-0/1/0                up    up
so-0/1/0.100            up    up   inet  10.0.2.6/30
                                   inet6 fe80::2a0:a5ff:fe28:d36/64
                                         fec0:0:2:4:2a0:a5ff:fe28:d36/64
so-0/1/1                up    up
so-0/1/1.0              up    up   inet  10.0.2.10/30
                                   inet6 fe80::2a0:a5ff:fe28:d36/64
                                         fec0:0:2:8:2a0:a5ff:fe28:d36/64
so-0/1/2                up    up
so-0/1/3                up    down
gr-0/3/0                up    up
. . .
dsc                     up    up
fxp0                    up    up
fxp0.0                  up    up   inet  10.0.1.4/24
fxp1                    up    up
fxp1.0                  up    up   tnp   4
gre                     up    up
ipip                    up    up
lo0                     up    up
lo0.0                   up    up   inet  10.0.3.4            --> 0/0
                                   inet6 fe80::2a0:a5ff:fe28:d36
                                         fec0:0:0:3::4
lsi                     up    up
. . .
```

Note that link-local addresses have automatically been added to all interfaces enabled for IPv6. A link-local address is associated with a FP of 0xFE8 and the resulting address always begins with a fe80::/64 prefix. In the case of the link-local addressing, the 64-bit interface identifier

is coded according to the EUI-64 specification. For point-to-point interfaces, the router "borrows" a 48-bit MAC address from its fxp0 interface, as shown next:

```
[edit interfaces]
lab@r4# run show interfaces fxp0 | match Hardware
  Current address: 00:a0:a5:28:0d:36, Hardware address: 00:a0:a5:28:0d:36
```

It is interesting to note that even though the same fxp0 derived 48-bit MAC address is used for all point-to-point interfaces on the router, the Universal/Local (U/L) bit is set to a *1* to indicate a global interface identifier. The net result is that `r4` assigns identical link-local addresses to each of its point-to-point interfaces, which is not a problem due to their link-local nature:

```
[edit interfaces]
lab@r4# run show interfaces so-0/1/0 terse
Interface               Admin Link Proto Local                 Remote
so-0/1/0                up    up
so-0/1/0.100            up    up   inet  10.0.2.6/30
                                  inet6 fe80::2a0:a5ff:fe28:d36/64
                                        fec0:0:2:4:2a0:a5ff:fe28:d36/64

[edit interfaces]
lab@r4# run show interfaces so-0/1/1 terse
Interface               Admin Link Proto Local                 Remote
so-0/1/1                up    up
so-0/1/1.0              up    up   inet  10.0.2.10/30
                                  inet6 fe80::2a0:a5ff:fe28:d36/64
                                        fec0:0:2:8:2a0:a5ff:fe28:d36/64
```

After visually confirming the addressing at the remaining routers, you conduct IPv6 ping testing to ensure that no mistakes have been made, and to verify that each pair of routers share a common IPv6 subnet. Note that the lack of an IPv6-capable IGP limits your ping testing to the link-local and site-local addressing associated with directly connected neighbors. The use of EUI-64 based addressing means that you need to determine the remote interface's IPv6 address before you can correctly target your pings.

In this example, the IPv6 addressing associated with `r3`'s ATM interface is displayed so that the desired address can be copied into the terminal emulation program's capture buffer for easy recall when ping testing is subsequently performed at `r5`:

```
[edit interfaces]
lab@r3# run show interfaces at-0/1/0 terse
Interface               Admin Link Proto Local                 Remote
at-0/1/0                up    up
at-0/1/0.0              up    up   inet  10.0.2.2/30
                                  inet6 fe80::2a0:a5ff:fe3d:234/64
                                        fec0:0:2:0:2a0:a5ff:fe3d:234/64
```

Once in the capture buffer, the desired address is easily recalled at r5 for ping testing of the ATM link:

```
[edit interfaces]
lab@r5# run ping fec0:0:2:0:2a0:a5ff:fe3d:234 count 2
PING6(56=40+8+8 bytes) fec0:0:2:0:2a0:a5ff:fe28:116e -->
   fec0:0:2:0:2a0:a5ff:fe3d:234
16 bytes from fec0:0:2:0:2a0:a5ff:fe3d:234, icmp_seq=0 hlim=64 time=1.293 ms
16 bytes from fec0:0:2:0:2a0:a5ff:fe3d:234, icmp_seq=1 hlim=64 time=1.644 ms

--- fec0:0:2:0:2a0:a5ff:fe3d:234 ping6 statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 1.293/1.468/1.644 ms
```

The ping test succeeds, and thereby confirms that r3 and r4 have been assigned compatible IPv6 addressing, at least as far as the ATM link is concerned. You must include the interface switch when attempting to ping the link-local address associated with the remote end of a point-to-point link. This is because there is no Layer 3–to–Layer 2 address mapping functionality (ARP) on point-to-point links, which results in the need to identify the correct egress interface as shown next:

```
[edit]
lab@r5# run ping fe80::2a0:a5ff:fe3d:234 count 1
PING6(56=40+8+8 bytes) fe80::2a0:a5ff:fe28:116e --> fe80::2a0:a5ff:fe3d:234

--- fe80::2a0:a5ff:fe3d:234 ping6 statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

The ping to r3's link-local address (fe80::2a0:a5ff:fe3d:234) from r5 fails because r5 is not able to associate the target address with the correct egress interface. Using the interface switch resolves the issue:

```
[edit]
lab@r5# run ping fe80::2a0:a5ff:fe3d:234 count 1 interface at-0/2/1
PING6(56=40+8+8 bytes) fe80::2a0:a5ff:fe28:116e --> fe80::2a0:a5ff:fe3d:234
16 bytes from fe80::2a0:a5ff:fe3d:234, icmp_seq=0 hlim=64 time=1.399 ms

--- fe80::2a0:a5ff:fe3d:234 ping6 statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 1.399/1.399/1.399 ms
```

The address resolution function associated with broadcast interfaces negates the need for the interface switch when testing Ethernet interfaces. To confirm this behavior, you determine the link-local address associated with r1's fe-0/0/1 interface:

```
[edit protocols router-advertisement]
lab@r1# run show interfaces fe-0/0/1 terse
```

```
Interface               Admin Link Proto Local                 Remote
fe-0/0/1                up    up
fe-0/0/1.0              up    up   inet  10.0.4.14/30
                                  inet6 fe80::2a0:c9ff:fe6f:7b3e/64
                                        fec0:0:4:12:2a0:c9ff:fe6f:7b3e/64
```

Note that pings initiated at r3 that are targeted at the link-local address assigned to r1's fe-0/0/1 interface succeed without use of the `interface` switch:

```
[edit]
lab@r3# run ping fe80::2a0:c9ff:fe6f:7b3e count 1
PING6(56=40+8+8 bytes) fe80::290:69ff:fe6d:9800 --> fe80::2a0:c9ff:fe6f:7b3e
16 bytes from fe80::2a0:c9ff:fe6f:7b3e, icmp_seq=0 hlim=64 time=0.569 ms

--- fe80::2a0:c9ff:fe6f:7b3e ping6 statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.569/0.569/0.569 ms
```

The IPv6 address resolution cache at r3 is found to contain an entry for the link-local address of r1's fe-0/0/1 interface as a result of the IPv6 ping traffic:

```
[edit]
lab@r3# run show ipv6 neighbors
IPv6 Address                Linklayer Address  State       Exp Rtr Interface
fe80::2a0:c9ff:fe6f:7b3e    00:a0:c9:6f:7b:3e  reachable   28  yes fe-0/0/0.0
```

After generating pings to the site-local address of r1's fe-0/0/1 interface, the IPv6 neighbor cache is also found to contain an entry for r1's site-local addressing:

```
[edit]
lab@r3# run ping fec0:0:4:12:2a0:c9ff:fe6f:7b3e count 1
PING6(56=40+8+8 bytes) fec0:0:4:12:290:69ff:fe6d:9800 -->
   fec0:0:4:12:2a0:c9ff:fe6f:7b3e
16 bytes from fec0:0:4:12:2a0:c9ff:fe6f:7b3e, icmp_seq=0 hlim=64 time=0.583 ms

--- fec0:0:4:12:2a0:c9ff:fe6f:7b3e ping6 statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.583/0.583/0.583 ms

[edit]
lab@r3# run show ipv6 neighbors
IPv6 Address                Linklayer Address  State       Exp Rtr Interface
fe80::2a0:c9ff:fe6f:7b3e    00:a0:c9:6f:7b:3e  stale       0   yes fe-0/0/0.0
fec0:0:4:12:2a0:c9ff:fe6f:7b3e
                            00:a0:c9:6f:7b:3e  reachable   29  yes fe-0/0/0.0
```

Although not shown here, you can assume that all routers in the IPv6 test bed are confirmed as being able to ping the site-local address associated with all directly attached neighbors. Before moving to the next section, you decide to conduct IPv6 traceroute testing between r3 and r1:

```
[edit]
lab@r3# run traceroute fec0:0:4:0:2a0:c9ff:fe6f:7aff
traceroute6 to fec0:0:4:0:2a0:c9ff:fe6f:7aff (fec0:0:4:0:2a0:c9ff:fe6f:7aff)
   from fec0:0:4:0:290:69ff:fe6d:9801, 30 hops max, 12 byte packets
```

As expected, the traceroute succeeds and displays a single hop in the form of the target address. From the captures shown in this section, it should be clear that IPv6 connectivity is tested and confirmed in the same way, and with the same tools, as venerable IPv4 protocol. In fact, once they get over the shock of those cumbersome addresses, most operators quickly find that their IPv4 skills can be rapidly and effectively brought to bear on all kinds of IPv6 configuration and troubleshooting tasks.

Because the IPv6 neighbor cache is, for all intents and purposes, an ARP cache, you should not expect to see entries for neighbors that are attached with point-to-point links when you issue a `show ipv6 neighbors` command. This is because ARP is not used on these interface types.

Verifying IPv6 Router Advertisements

The verification of router advertisement functionality at r1 and r2 is rather straightforward. You begin by determining that r1 is sending router advertisement messages out its fe-0/0/0 interface:

```
[edit]
lab@r1# run show ipv6 router-advertisement
Interface: fe-0/0/0.0
  Advertisements sent: 3, last sent 00:01:33 ago
  Solicits received: 0
  Advertisements received: 3
  Advertisement from fe80::2a0:c9ff:fe69:c1c0, heard 00:00:32 ago
    Managed: 0
    Other configuration: 0
    Reachable time: 0 ms
    Default lifetime: 1800 sec
    Retransmit timer: 0 ms
    Current hop limit: 64
```

The display confirms that r1 is sending and receiving router advertisements on its fe-0/0/0 interface; this provides a good indication that r2 has been correctly configured to generate router advertisement messages. Though not shown here, you can assume that similar output is observed at r2. The output shown, combined with the knowledge that the FEC0:0:5:0::/64

prefix has been correctly configured under the [edit protocols router-advertisement interface fe-0/0/0] portion of the configuration hierarchy, completes the verification steps for the IPv6 addressing and neighbor discovery configuration scenario. Tracing router advertisement messages provides additional validation. By way of example, the tracing configuration shown next is added to r1:

```
[edit protocols router-advertisement]
lab@r1# show
traceoptions {
    file ra;
    flag all;
}
interface fe-0/0/0.0 {
    prefix fec0:0:5:0::/64;
}
```

This results in the following trace output:

```
[edit protocols router-advertisement]
lab@r1#
May  8 23:32:07 task_timer_dispatch: calling Router-Advertisement_timer, late
   by 0.008
May  8 23:32:07 task_job_create_foreground: create job timer for task Router-
   Advertisement
May  8 23:32:07 task_timer_dispatch: returned from Router-Advertisement_timer,
   rescheduled in 0
May  8 23:32:07 foreground dispatch running job timer for task Router-
   Advertisement
May  8 23:32:07 ipv6_ra_send_advertisement: sending advertisement for ifl 2 to
   ff02::1
May  8 23:32:07 (2002) sending advertisement for ifl 2
May  8 23:32:07         ifa 0x84d9b68 fec0:0:5:0:260:94ff:fe51:e932/64
May  8 23:32:07         --> sent 56 bytes
May  8 23:32:07 task_timer_uset: timer Router-Advertisement_timer <Touched
   SpawnJob> set to offset 16 jitter 20 at 23:32:21
May  8 23:32:07 foreground dispatch completed job timer for task Router-
   Advertisement
May  8 23:32:14 ipv6_ra_receive_advertisement: received advertisement from
   fe80::2a0:c9ff:fe69:c1c0
May  8 23:32:14 ipv6_ra_receive_advertisement: task Router-Advertisement src
   fe80::2a0:c9ff:fe69:c1c0 dst ff02::1 hdr 0x854c000 count 24 intf 0x8570000
```

The highlights in the output show that prefix advertisements are occurring for the FEC0:0:5:0::/64 prefix. The capture also shows that advertisement messages are being received from r2. The results shown in this section confirm that you have met the requirements posed for the IPv6 addressing and router advertisement section!

IPv6 Addressing and Neighbor Discovery Summary

To be effective with IPv6 configuration and fault isolation, a JNCIE candidate must have a strong grasp of the concepts behind IPv6 addressing. This section demonstrated how link-local addresses are automatically created for each IPv6-enabled interface using a FP of 1111 1110 10 and the EUI-64 format. The section also showed how site-local or global IPv6 addresses can be manually assigned to an interface, both with and without the use of automatically computed EUI-64 based interface identifiers. IPv6 testing and verification with common utilities such as ping and traceroute was also demonstrated in this section.

IPv6 uses neighbor discovery to replicate functionality that is provided by a number of IPv4 protocols. This functionality includes ARP, duplicate address detection, redirection, router discovery, and mechanisms to convey auto-configuration parameters to IPv6 hosts. This section contained a typical router advertisement configuration task, and also showed operational mode commands and tracing output that confirmed the operation and configuration of router advertisement messages.

IPv6 and IGP Support

As of JUNOS software release 5.6, Juniper Networks M-series and T-series routers provide support for IPv6 routing using RIPng, OSPF version 3, IS-IS, and static routing.

The good news is that the steps needed to configure RIPng or OSPF3 are almost identical to those needed for their IPv4 counterparts. The even better news is that the JUNOS implementation of IS-IS "just works" for IPv6. In fact, IS-IS requires explicit configuration only when you *do not* want it to advertise IPv6 routes associated with the interfaces that it is configured to operate on.

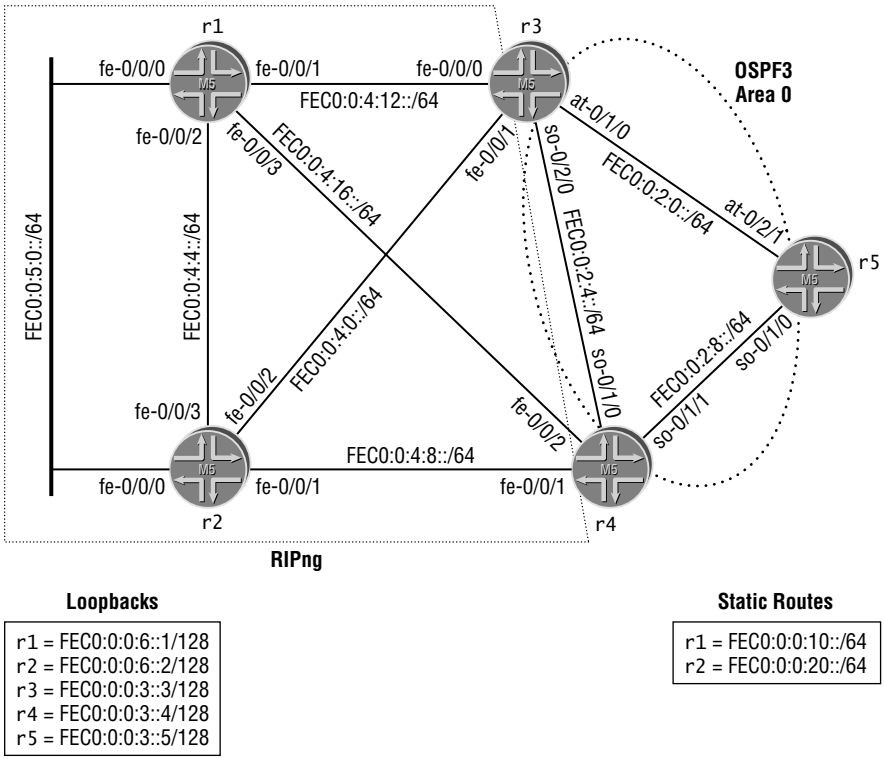
This section demonstrates the use of RIPng and OSPF3, and the routing policy needed to redistribute IPv6 routes between the two protocols. IS-IS support of IPv6 is demonstrated in the chapter's case study. Figure 5.3 details the specifics of your RIPng and OSPF3 configuration scenario.

Based on Figure 5.3, you can see that your IPv6 IGP task involves the definition of static IPv6 routes at `r1` and `r2`, the configuration of RIPng (and related policy) at `r1` through `r4`, and the configuration of OSPF3 (and related policy) on `r3`, `r4`, and `r5`. A JNCIE candidate should immediately recognize that, even though the primary goal is the deployment of IPv6 capable IGPs, this scenario also involves the complexities associated with any mutual route redistribution topology.

To complete this section, you must alter your configurations to meet these criteria:

- Configure RIPng on `r1–r4`.
- Define and redistribute the static routes associated with `r1` and `r2` into RIPng.
- Configure OSPF3 on routers `r3–r5`.
- Your network must have full IPv6 reachability (including loopback addresses) despite the failure of any interface or link.
- You may define only the static routes shown in Figure 5.3.

FIGURE 5.3 IPv6 OSPF3 and RIPng



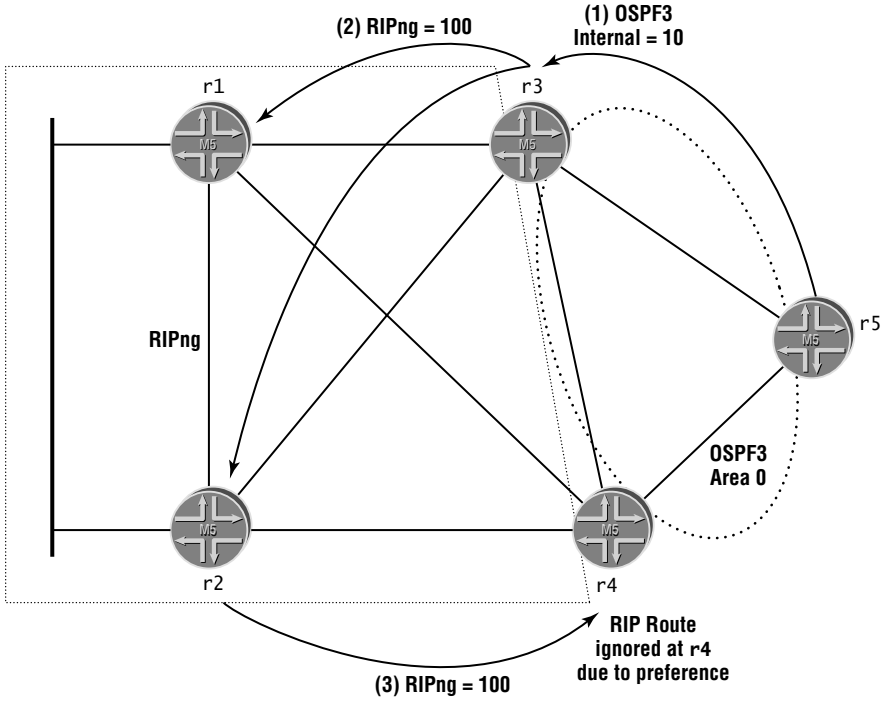
Configuring RIPng

Before jumping into your IPv6 IGP configuration, it behooves you to take a moment to think about your overall route redistribution plan. At this stage in your career, you should be keenly aware that mutual route distribution occurring at multiple points can easily lead to routing problems unless the appropriate precautions are taken. In this example, the default protocol preference settings for RIPng and OSPF3 internal routes result in "the right behavior" with no policy or route preference modification needed. Figure 5.4 shows the situation from the perspective of an OSPF3 internal route being redistributed by `r3` into the RIPng domain.

At Step 1 in Figure 5.4, we see an internal OSPF3 route that originates at `r5` arriving at `r3`; this route has a preference setting of 10. At Step 2, `r3` redistributes this route into the RIPng domain, now with a protocol preference of 100 (although not shown in the figure, `r4` also redistributes the route into RIPng in the test bed). At Step 3, we see `r2` advertising the route to `r4` (the route is not sent back to `r3` due to split horizon). In this case, `r4` does not consider the route as *active* because of its higher preference value when compared to the OSPF3 internal route. A similar situation occurs for routes that originate in the RIPng domain. In this case, such a route arrives at `r3` and `r4` with a preference setting of 100. When redistributed into OSPF3 by `r3` and `r4`, the OSPF3 external route has a preference setting of 150. Once again, the default

behavior causes both r3 and r4 to "ignore" the OSPF3 version of the route in preference to the RIPng version.

FIGURE 5.4 OSPF3 to RIPng redistribution



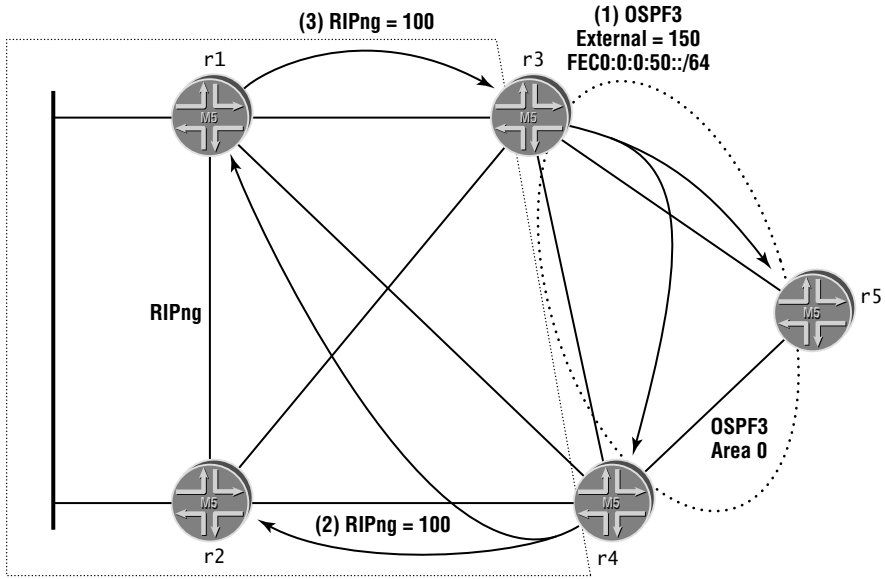
While the default preference settings seem to work fine for internal routes, problems will occur when an OSPF3 external route is redistributed into the RIPng domain, as illustrated in Figure 5.5.

At Step 1 in Figure 5.5, we see r3 redistributing a static route into the OSPF3 domain with a preference of 150. At Step 2, the OSPF3 external route is redistributed back into the RIPng domain by r4 with a preference of 100. At Step 3, the route RIPng version of the route is advertised by r1 back to r3. At this point, r3 still has a local static route defined, so the "echoed" route causes no immediate problems:

```
[edit]
lab@r3# run show route fec0:0:0:30::/64

inet6.0: 30 destinations, 50 routes (30 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

fec0:0:0:30::/64    *[Static/5] 00:00:55, metric 0
                        Reject
```

```
[RIPng/100] 00:05:56, metric 3, tag 0
  to fe80::2a0:c9ff:fe6f:7b3e via fe-0/0/0.0
> to fe80::2a0:c9ff:fe6f:7aff via fe-0/0/1.0
```

FIGURE 5.5 External route redistribution problems



The problems kick in when the static route is removed from r3's configuration. Once the local static definition is removed, r3 installs the RIPng version as the active route. Sloppy RIPng to OSPF3 redistribution policy can result in the RIPng version of the route being redistributed by r3 into OSPF3 as an external route. Once this occurs, we find ourselves with a nonexistent route that does not die or age out:

```
[edit]
lab@r3# delete routing-options rib inet6.0 static route fec0:0:0:30::/64

[edit]
lab@r3# commit
commit complete

[edit]
lab@r3# run show route fec0:0:0:30::/64

inet6.0: 30 destinations, 49 routes (30 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

```
fec0:0:0:30::/64   *[RIPng/100] 00:08:38, metric 3, tag 0
                      to fe80::2a0:c9ff:fe6f:7b3e via fe-0/0/0.0
                    > to fe80::2a0:c9ff:fe6f:7aff via fe-0/0/1.0
```

To prevent problems such as these, you need to be selective in your RIPng and OSPF3 redistribution policies, ensuring that routes are only redistributed in a single direction. While you could play it extra safe by carefully controlling route redistribution in both directions, this example makes use of a selective policy for redistribution in the direction of RIPng to OSPF3 only being selective in one direction or the other, which is sufficient to prevent problems in this case. You start RIPng configuration at `r1` by defining a RIPng group called *`r1-r4`* that includes all of `r1`'s Fast Ethernet interfaces:

```
[edit protocols ripng]
lab@r1# set group r1-r4 neighbor fe-0/0/0
```

The previous command should be repeated as needed to list all of r1's Fast Ethernet interfaces. Next, you associate the *`r1-r4`* RIPng group with an export policy, which in this example is called *`ripng-export`*. Note that an export policy is almost always needed with the JUNOS software implementation of RIP and RIPng, even when the goal is to simply re-advertise routes learned through RIPng itself:

```
[edit protocols ripng]
lab@r1# set group r1-r4 export ripng-export
```

The resulting RIPng configuration stanza is shown next:

```
[edit protocols ripng]
lab@r1# show
group r1-r4 {
    export ripng-export;
    neighbor fe-0/0/0.0;
    neighbor fe-0/0/1.0;
    neighbor fe-0/0/2.0;
    neighbor fe-0/0/3.0;
}
```

You now define the IPv6 static route that is owned by `r1`. This route points to `reject` in this example, but you could just as well point it to a `discard` next hop:

```
[edit routing-options]
lab@r1# set rib inet6.0 static route fec0:0:0:10::/64 reject
```

The modified `routing-option` stanza is displayed for confirmation:

```
[edit routing-options]
lab@r1# show
rib inet6.0 {
    static {
        route fec0:0:0:10::/64 reject;
    }
}
```

```
static {
    route 10.0.200.0/24 {
        next-hop 10.0.1.102;
        no-readvertise;
    }
}
aggregate {
    route 10.0.0.0/16;
}
autonomous-system 65412;
```

The final configuration steps at `r1` involve the definition of the *`ripng-export`* policy. In this case, the policy is written to re-advertise routes learned from RIPng, to redistribute the FEC0:0:0:10::/64 static route into RIPng, and to advertise the site-local addressing associated with direct routes:

```
[edit policy-options policy-statement ripng-export]
lab@r1# set term 1 from protocol ripng

[edit policy-options policy-statement ripng-export]
lab@r1# set term 1 then accept

[edit policy-options policy-statement ripng-export]
lab@r1# set term 2 from protocol static

[edit policy-options policy-statement ripng-export]
lab@r1# set term 2 from route-filter fec0:0:0:10::/64 exact

[edit policy-options policy-statement ripng-export]
lab@r1# set term 2 then accept

[edit policy-options policy-statement ripng-export]
lab@r1# set term 3 from protocol direct

[edit policy-options policy-statement ripng-export]
lab@r1# set term 3 from route-filter fec0::/16 orlonger

[edit policy-options policy-statement ripng-export]
lab@r1# set term 3 then accept
```

The completed *`ripng-export`* policy is displayed next:

```
[edit policy-options policy-statement ripng-export]
lab@r1# show
```

```
term 1 {
    from protocol ripng;
    then accept;
}
term 2 {
    from {
        protocol static;
        route-filter fec0:0:0:10::/64 exact;
    }
    then accept;
}
term 3 {
    from {
        protocol direct;
        route-filter fec0::/16 orlonger;
    }
    then accept;
}
```

Term *3* in the *ripng-export* policy is necessary to effect the advertisement of the router's direct (interface) routes, which includes its loopback address, into RIPng. Note that by default RIPng does not advertise any routes, even those associated with the interfaces that RIPng is configured to run on! The configuration for `r2` is virtually identical to that of `r1` and is therefore not shown here. The only real difference between their configurations relates to the fact that `r2` has a FEC0:0:0:20::/64 static route.

The changes made to `r4` in support of RIPng, and the redistribution of its OSPF3 and direct routes into RIPng, are displayed:

```
[edit]
lab@r4# show protocols ripng
group r1-r4 {
    export ripng-export;
    neighbor fe-0/0/1.0;
    neighbor fe-0/0/2.0;
}

[edit]
lab@r4# show policy-options policy-statement ripng-export
term 1 {
    from protocol ripng;
    then accept;
}
```

```
term 2 {
    from protocol ospf;
    then accept;
}
term 3 {
    from {
        protocol direct;
        route-filter fec0::/16 orlonger;
    }
    then accept;
}
```

The primary difference between the policy-related configurations of `r1` and `r4` is `r4`'s need to match on OSPF routes for redistribution into RIPng. The RIPng configuration of `r3` is very similar to that of `r4`, and is therefore not shown here.

Verifying RIPng

The verification of RIPng is quite similar to the approach taken for RIP; you begin by confirming that RIPng is running on the correct interfaces and that you are receiving and sending the expected routes. Confirmation begins at `r3` with the determination that RIPng is running on its fe-0/0/0 and fe-0/0/1 interfaces:

```
[edit]
lab@r3# run show ripng neighbor
                    Source                          Dest              In
Neighbor     State  Address                         Address  Send Recv Met
--------     -----  -------                         -------  ---- ---- ---
fe-0/0/1.0      Up  fe80::290:69ff:fe6d:9801        ff02::9  yes yes    1
fe-0/0/0.0      Up  fe80::290:69ff:fe6d:9800        ff02::9  yes yes    1
```

The output confirms that RIPng is configured to send and receive updates on the expected interfaces. If desired, you can view the RIPng routes being advertised out a particular interface with a `show route advertising-protocol` command. Note that you need to specify the local router's link-local address for the "neighbor" component of the command:

```
[edit]
lab@r3# run show interfaces fe-0/0/0 terse
Interface               Admin Link Proto Local                 Remote
fe-0/0/0                up    up
fe-0/0/0.0              up    up   inet  10.0.4.13/30
                                  inet6 fe80::290:69ff:fe6d:9800/64
                                        fec0:0:4:12:290:69ff:fe6d:9800/64
```

`r3`'s link-local address is first displayed so that it can be copied into your terminal emulation program's capture buffer for use in the `show route advertising-protocol` command:

```
[edit]
lab@r3# run show route advertising-protocol ripng fe80::290:69ff:fe6d:9800
```

```
inet6.0: 25 destinations, 30 routes (25 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

fec0:0:0:3::3/128  *[Direct/0] 03:49:34
                    > via lo0.0
fec0:0:0:6::2/128  *[RIPng/100] 01:25:12, metric 2, tag 0
                    > to fe80::2a0:c9ff:fe6f:7aff via fe-0/0/1.0
fec0:0:0:20::/64   *[RIPng/100] 01:25:12, metric 2, tag 0
                    > to fe80::2a0:c9ff:fe6f:7aff via fe-0/0/1.0
fec0:0:2::/64      *[Direct/0] 03:47:37
                    > via at-0/1/0.0
fec0:0:2:4::/64    *[Direct/0] 03:47:37
                    > via so-0/2/0.100
fec0:0:4::/64      *[Direct/0] 03:47:37
                    > via fe-0/0/1.0
                    [RIPng/100] 01:25:12, metric 2, tag 0
                    > to fe80::2a0:c9ff:fe6f:7aff via fe-0/0/1.0
fec0:0:4:8::/64    *[RIPng/100] 01:25:12, metric 2, tag 0
                    > to fe80::2a0:c9ff:fe6f:7aff via fe-0/0/1.0
fec0:0:4:12::/64   *[Direct/0] 03:47:37
                    > via fe-0/0/0.0
                    [RIPng/100] 01:14:19, metric 2, tag 0
                    > to fe80::2a0:c9ff:fe6f:7b3e via fe-0/0/0.0
```

The output confirms that the expected routes are being set from `r3` to `r1`. You can use the `receiving-protocol` version of the command to determine the routes you receive from a given neighbor. In this case, you must specify the link-local address of the remote peer:

```
[edit policy-options policy-statement ripng-export]
lab@r1# run show interfaces fe-0/0/1 terse
Interface               Admin Link Proto Local                 Remote
fe-0/0/1                up    up
fe-0/0/1                up    up   inet  10.0.4.14/30
                                  inet6 fe80::2a0:c9ff:fe6f:7b3e/64
                                        fec0:0:4:12:2a0:c9ff:fe6f:7b3e/64
```

Knowing that the link-local address for `r1`'s fe-0/0/1 interface is fe80::2a0:c9ff:fe6f:7b3e/64 allows you to determine what RIPng routes `r3` is receiving from `r1`:

```
[edit]
lab@r3# run show route receive-protocol ripng fe80::2a0:c9ff:fe6f:7b3e

inet.0: 25 destinations, 27 routes (25 active, 0 holddown, 0 hidden)
```

```
inet6.0: 24 destinations, 29 routes (24 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

fec0:0:0:3::4/128  *[RIPng/100] 00:21:04, metric 3, tag 0
                      to fe80::2a0:c9ff:fe6f:7b3e via fe-0/0/0.0
                    > to fe80::2a0:c9ff:fe6f:7aff via fe-0/0/1.0
fec0:0:0:6::1/128  *[RIPng/100] 01:16:29, metric 2, tag 0
                    > to fe80::2a0:c9ff:fe6f:7b3e via fe-0/0/0.0
fec0:0:0:10::/64   *[RIPng/100] 01:27:22, metric 2, tag 0
                    > to fe80::2a0:c9ff:fe6f:7b3e via fe-0/0/0.0
fec0:0:2:8::/64    *[RIPng/100] 00:21:04, metric 3, tag 0
                    > to fe80::2a0:c9ff:fe6f:7b3e via fe-0/0/0.0
                      to fe80::2a0:c9ff:fe6f:7aff via fe-0/0/1.0
fec0:0:4:4::/64    *[RIPng/100] 01:27:22, metric 2, tag 0
                    > to fe80::2a0:c9ff:fe6f:7b3e via fe-0/0/0.0
                      to fe80::2a0:c9ff:fe6f:7aff via fe-0/0/1.0
fec0:0:4:12::/64    [RIPng/100] 01:16:29, metric 2, tag 0
                    > to fe80::2a0:c9ff:fe6f:7b3e via fe-0/0/0.0
fec0:0:4:16::/64   *[RIPng/100] 01:16:29, metric 2, tag 0
                    > to fe80::2a0:c9ff:fe6f:7b3e via fe-0/0/0.0
fec0:0:5::/64      *[RIPng/100] 01:16:29, metric 2, tag 0
                    > to fe80::2a0:c9ff:fe6f:7b3e via fe-0/0/0.0
```

The routes being advertised by RIPng confirm that your *ripng-export* policy correctly advertises RIPng and direct routes with IPv6 addresses beginning with FEC0::/16. As a final check on the operation of RIPng, you display the routes being learned from that protocol:

```
[edit]
lab@r4# run show route protocol ripng

inet.0: 25 destinations, 27 routes (25 active, 0 holddown, 0 hidden)

inet6.0: 24 destinations, 29 routes (24 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

fec0:0:0:3::3/128  *[RIPng/100] 00:24:06, metric 3, tag 0
                    > to fe80::2a0:c9ff:feb2:f8cb via fe-0/0/1.0
                      to fe80::2a0:c9ff:fe69:a806 via fe-0/0/2.0
fec0:0:0:6::1/128  *[RIPng/100] 01:19:38, metric 2, tag 0
                    > to fe80::2a0:c9ff:fe69:a806 via fe-0/0/2.0
```

```
fec0:0:0:6::2/128  *[RIPng/100] 01:41:57, metric 2, tag 0
                    > to fe80::2a0:c9ff:feb2:f8cb via fe-0/0/1.0
fec0:0:0:10::/64   *[RIPng/100] 01:41:57, metric 2, tag 0
                    > to fe80::2a0:c9ff:fe69:a806 via fe-0/0/2.0
fec0:0:0:20::/64   *[RIPng/100] 01:41:57, metric 2, tag 0
                    > to fe80::2a0:c9ff:feb2:f8cb via fe-0/0/1.0
fec0:0:2::/64      *[RIPng/100] 00:24:06, metric 3, tag 0
                      to fe80::2a0:c9ff:feb2:f8cb via fe-0/0/1.0
                    > to fe80::2a0:c9ff:fe69:a806 via fe-0/0/2.0
fec0:0:4::/64      *[RIPng/100] 01:41:08, metric 2, tag 0
                    > to fe80::2a0:c9ff:feb2:f8cb via fe-0/0/1.0
fec0:0:4:4::/64    *[RIPng/100] 01:41:08, metric 2, tag 0
                    > to fe80::2a0:c9ff:feb2:f8cb via fe-0/0/1.0
                      to fe80::2a0:c9ff:fe69:a806 via fe-0/0/2.0
fec0:0:4:8::/64     [RIPng/100] 01:41:08, metric 2, tag 0
                    > to fe80::2a0:c9ff:feb2:f8cb via fe-0/0/1.0
fec0:0:4:12::/64   *[RIPng/100] 01:19:38, metric 2, tag 0
                    > to fe80::2a0:c9ff:fe69:a806 via fe-0/0/2.0
fec0:0:4:16::/64    [RIPng/100] 01:19:38, metric 2, tag 0
                    > to fe80::2a0:c9ff:fe69:a806 via fe-0/0/2.0
fec0:0:5::/64      *[RIPng/100] 01:19:38, metric 2, tag 0
                    > to fe80::2a0:c9ff:fe69:a806 via fe-0/0/2.0
ff02::9/128        *[RIPng/100] 00:24:14, metric 1
                      MultiRecv
```

The output obtained from `r3` and `r4` indicates that the routers in the RIPng domain are properly configured and operational. If time permits, you can perform some additional spot checks, such as traceroutes to loopback addresses, to make sure that your forwarding paths are optimal:

```
[edit]
lab@r2# run traceroute fec0:0:0:6::1
traceroute6 to fec0:0:0:6::1 (fec0:0:0:6::1) from fec0:0:4:4:2a0:c9ff:fe6f:700d,
   30 hops max, 12 byte packets
 1  fec0:0:0:6::1 (fec0:0:0:6::1)  0.399 ms  0.204 ms  0.113 ms


[edit]
lab@r2# run traceroute fec0:0:0:3::3
traceroute6 to fec0:0:0:3::3 (fec0:0:0:3::3) from fec0:0:4:0:2a0:c9ff:fe6f:7aff,
   30 hops max, 12 byte packets
 1  fec0:0:0:3::3 (fec0:0:0:3::3)  1.006 ms  0.434 ms  0.369 ms
```

With RIPng route advertisement and optimal forwarding among `r1` through `r4` verified, the confirmation of the RIPng domain is complete.

Configuring OSPF3

You start OSPF3 configuration at r5 by defining an ospf3 stanza for area 0:

```
[edit protocols ospf3]
lab@r5# set area 0 interface so-0/1/0

[edit protocols ospf3]
lab@r5# set area 0 interface at-0/2/1.0

[edit protocols ospf3]
lab@r5# set area 0 interface lo0.0 passive
```

Note that unlike IPv4 and its OSPF version 2, you must run OSPF3 on the router's lo0 interface if you want the loopback address advertised into OSPF3. It is recommended that you enable the passive operation when running OSPF3 on a lo0 interface to reduce compute cycles. Note that export policy is not needed on r5 for this configuration scenario. The completed ospf3 stanza is displayed at r5:

```
[edit protocols ospf3]
lab@r5# show
area 0.0.0.0 {
    interface lo0.0 {
        passive;
    }
    interface at-0/2/1.0;
    interface so-0/1/0.0;
}
```

The configuration of OSPF3 on r3 and r4 is very similar to that shown for r5; the primary difference is the need to include an export policy that redistributes RIPng and direct routes into OSPF3. r3's OSPF3 configuration is shown next:

```
[edit]
lab@r3# show protocols ospf3
export ospf3-export;
area 0.0.0.0 {
    interface so-0/2/0.100;
    interface lo0.0 {
        passive;
    }
    interface at-0/1/0.0;
}
```

The highlight calls out the application of an *ospf3-export* policy, which is displayed next. Note that the policy makes use of route filter statements to match on, and subsequently accept, only those routes that have originated in the RIPng domain. This precaution prevents the

redistribution problems described at the beginning of this section:

```
[edit]
lab@r3# show policy-options policy-statement ospf3-export
term 1 {
    from {
        route-filter fec0:0:4::/46 orlonger;
        route-filter fec0:0:0:6::/64 orlonger;
    }
    then accept;
}
```

In this example, term *1* in the *ospf3-export* policy is written to match on the interface routes and loopback routes that exist within the RIPng domain, thereby serving to prevent the redistribution of routes that originate in the OSPF3 domain *back* into the OSPF3 domain. Although not shown, r4 has a similar OSPF3 configuration and an identical *ospf3-export* policy. Note that the differences between loopback and interface address assignments result in the need for the two route filter statements shown in this example.

Verifying OSPF3 and Route Redistribution

As with RIP vs. RIPng, the verification of OSPF3 proceeds in much the same way as the verification of OSPF. Specifically, you need to confirm adjacency status and the proper advertisement of routes. You begin at r5 by verifying its OSPF3 adjacency status:

```
[edit protocols ospf3]
lab@r5# run show ospf3 neighbor
ID              Interface             State     Pri   Dead
10.0.3.3        at-0/2/1.0            Full      128   34
  Neighbor-address fe80::2a0:a5ff:fe3d:234
10.0.3.4        so-0/1/0.0            Full      128   34
  Neighbor-address fe80::2a0:a5ff:fe28:d36
```

The display confirms that both of r5's OSPF3 adjacencies are in the established state. Next, you confirm that OSPF3 is advertising the loopback addresses of all routers in the IPv6 test bed. This determination also helps to validate the operation of the *ospf3-export* policy at r3 and r4 because the loopback addresses of r1 and r2 begin as RIPng routes.

```
[edit]
lab@r5# run show route protocol ospf | match /128
fec0:0:0:3::3/128  *[OSPF/10] 00:17:57, metric 1
fec0:0:0:3::4/128  *[OSPF/10] 00:18:06, metric 1
fec0:0:0:3::5/128   [OSPF/10] 00:44:37, metric 0
fec0:0:0:6::1/128  *[OSPF/150] 00:04:55, metric 2, tag 0
fec0:0:0:6::2/128  *[OSPF/150] 00:04:55, metric 2, tag 0
ff02::5/128        *[OSPF/10] 02:27:34, metric 1
```

As hoped for, all five of the loopback addresses are present. Note that the internal route to r5's own loopback address is not active due to the route also being present as a direct route. The

overall status of OSPF3 routes is now displayed at r5. Note the use of CLI match functionality that eliminates the display of IPv4-based OSPF routes:

```
[edit]
lab@r5# run show route protocol ospf | match fec0
fec0:0:0:3::3/128  *[OSPF/10] 00:18:32, metric 1
fec0:0:0:3::4/128  *[OSPF/10] 00:18:41, metric 1
fec0:0:0:3::5/128   [OSPF/10] 00:45:12, metric 0
fec0:0:0:6::1/128  *[OSPF/150] 00:05:30, metric 2, tag 0
fec0:0:0:6::2/128  *[OSPF/150] 00:05:30, metric 2, tag 0
fec0:0:2::/64       [OSPF/10] 00:45:12, metric 1
fec0:0:2:4::/64    *[OSPF/10] 00:18:32, metric 2
fec0:0:2:8::/64     [OSPF/10] 00:45:12, metric 1
fec0:0:4::/64      *[OSPF/150] 00:18:32, metric 0, tag 0
fec0:0:4:4::/64    *[OSPF/150] 00:18:32, metric 2, tag 0
fec0:0:4:8::/64    *[OSPF/150] 00:18:41, metric 0, tag 0
fec0:0:4:12::/64   *[OSPF/150] 00:18:32, metric 0, tag 0
fec0:0:4:16::/64   *[OSPF/150] 00:18:41, metric 0, tag 0
fec0:0:5::/64      *[OSPF/150] 00:18:32, metric 2, tag 0
```

From the display, you can see that all of the expected routes are present, including the prefixes that originate within the RIPng domain. A `show route` command for the FEC0:0:5:0/64 prefix indicates that load balancing, and consequently the required redundancy, is in effect:

```
lab@r5> show route fec0:0:5:0::/64

inet6.0: 21 destinations, 25 routes (21 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

fec0:0:5::/64      *[OSPF/150] 00:18:51, metric 2, tag 0
                      via so-0/1/0.0
                    > via at-0/2/1.0
```

As with plain old OSPF, the OSPF3 database can be displayed, as shown next:

```
lab@r5> show ospf3 database

    OSPF3 link state database, area 0.0.0.0
 Type       ID               Adv Rtr           Seq         Age  Cksum  Len
Router      0.0.0.0          10.0.3.3        0x8000001d   482  0xc1f0  56
Router      0.0.0.0          10.0.3.4        0x80000010   464  0xcfef  56
Router     *0.0.0.0          10.0.3.5        0x80000018    22  0x9324  56
IntraArPfx  0.0.0.1          10.0.3.3        0x80000015   482  0x7be1  76
IntraArPfx  0.0.0.1          10.0.3.4        0x8000000a   464  0x3627  76
IntraArPfx *0.0.0.1          10.0.3.5        0x8000000d   622  0xe07a  76
    OSPF3 AS SCOPE link state database
```

```
 Type       ID               Adv Rtr          Seq          Age  Cksum  Len
Extern      0.0.0.1          10.0.3.3         0x80000009   668  0x6f72  36
Extern      0.0.0.2          10.0.3.3         0x80000005   381  0x28ce  36
Extern      0.0.0.3          10.0.3.3         0x80000009   119  0xb235  36
Extern      0.0.0.4          10.0.3.3         0x80000009    68  0x608a  36
Extern      0.0.0.8          10.0.3.3         0x80000003  1255  0xce3   36
Extern      0.0.0.10         10.0.3.3         0x80000003  1255  0x7563  36
Extern      0.0.0.11         10.0.3.3         0x80000001   482  0xf9aa  44
Extern      0.0.0.12         10.0.3.3         0x80000001   482  0xa98   44
Extern      0.0.0.1          10.0.3.4         0x80000008   673  0xb635  36
Extern      0.0.0.2          10.0.3.4         0x80000008   383  0xa933  36
Extern      0.0.0.3          10.0.3.4         0x80000008   123  0x2ec0  36
Extern      0.0.0.4          10.0.3.4         0x80000008    73  0x5983  36
Extern      0.0.0.5          10.0.3.4         0x80000007  1262  0x5496  36
Extern      0.0.0.10         10.0.3.4         0x80000006  1262  0xdb0f  36
Extern      0.0.0.11         10.0.3.4         0x80000001   464  0xf3af  44
Extern      0.0.0.12         10.0.3.4         0x80000001   464  0x49d   44
Extern     *0.0.0.1          10.0.3.5         0x80000003   322  0x8f1c  36

    OSPF3 Link-Local link state database, interface at-0/2/1.0
 Type       ID               Adv Rtr           Seq         Age  Cksum  Len
Link        0.0.0.2          10.0.3.3         0x80000007  1253  0x71e3  56
Link       *0.0.0.3          10.0.3.5         0x80000008  1222  0xf02c  56

    OSPF3 Link-Local link state database, interface so-0/1/0.0
 Type       ID               Adv Rtr           Seq         Age  Cksum  Len
Link        0.0.0.3          10.0.3.4         0x80000006  1262  0x351   56
Link       *0.0.0.2          10.0.3.5         0x80000008   922  0x2ce9  56
```

Note that the OSPF3 database contains intra-area prefix LSAs, which are not present in OSPF version 2. These LSAs, which advertise internal prefixes, are needed because OSPF3 router and network LSAs do not carry any prefix information.

Moving along, you now display the routes associated with the OSPF3 domain at `r2` to quickly confirm that OSPF3 to RIPng redistribution is working as expected:

```
[edit]
lab@r2# run show route fec0:0:2::/48

inet6.0: 27 destinations, 33 routes (26 active, 1 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

fec0:0:2::/64      *[RIPng/100] 00:25:32, metric 2, tag 0
                       to fe80::290:69ff:fe6b:3001 via fe-0/0/1.0
                    > to fe80::290:69ff:fe6d:9801 via fe-0/0/2.0
```

```
fec0:0:2:4::/64    *[RIPng/100] 00:25:32, metric 2, tag 0
                      to fe80::290:69ff:fe6b:3001 via fe-0/0/1.0
                    > to fe80::290:69ff:fe6d:9801 via fe-0/0/2.0
fec0:0:2:8::/64    *[RIPng/100] 00:25:32, metric 2, tag 0
                      to fe80::290:69ff:fe6b:3001 via fe-0/0/1.0
                    > to fe80::290:69ff:fe6d:9801 via fe-0/0/2.0
```

The output lists the link addresses associated with the OSPF3 domain, as expected, so attention shifts to the loopback addresses assigned to `r3` through `r5`:

```
[edit]
lab@r2# run show route fec0:0:0:3::/64

inet6.0: 27 destinations, 33 routes (26 active, 1 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

fec0:0:0:3::3/128  *[RIPng/100] 00:25:44, metric 2, tag 0
                      to fe80::290:69ff:fe6b:3001 via fe-0/0/1.0
                    > to fe80::290:69ff:fe6d:9801 via fe-0/0/2.0
fec0:0:0:3::4/128  *[RIPng/100] 00:25:44, metric 2, tag 0
                      to fe80::290:69ff:fe6b:3001 via fe-0/0/1.0
                    > to fe80::290:69ff:fe6d:9801 via fe-0/0/2.0
fec0:0:0:3::5/128  *[RIPng/100] 00:25:37, metric 2, tag 0
                      to fe80::290:69ff:fe6b:3001 via fe-0/0/1.0
                    > to fe80::290:69ff:fe6d:9801 via fe-0/0/2.0
```

All loopback addresses associated with the OSPF3 domain are also confirmed. Traceroute testing is performed from `r5` to destinations in the RIPng domain to provide final verification:

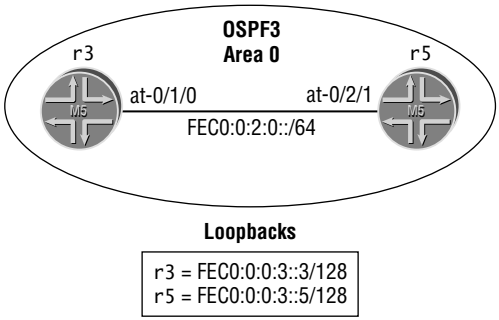
```
lab@r5> traceroute fec0:0:0:6::1
traceroute6 to fec0:0:0:6::1 (fec0:0:0:6::1) from fec0:0:2:0:2a0:a5ff:fe28:116e,
   30 hops max, 12 byte packets
 1  fec0:0:2:0:2a0:a5ff:fe3d:234 (fec0:0:2:0:2a0:a5ff:fe3d:234)  1.518 ms
    1.189 ms  0.798 ms
 2  fec0:0:0:6::1 (fec0:0:0:6::1)  0.581 ms  0.704 ms  0.82 ms

lab@r5> traceroute fec0:0:0:6::2
traceroute6 to fec0:0:0:6::2 (fec0:0:0:6::2) from fec0:0:2:8:2a0:a5ff:fe28:116e,
   30 hops max, 12 byte packets
 1  fec0:0:2:8:2a0:a5ff:fe28:d36 (fec0:0:2:8:2a0:a5ff:fe28:d36)  0.763 ms
    0.623 ms  0.561 ms
 2  fec0:0:0:6::2 (fec0:0:0:6::2)  0.479 ms  0.461 ms  0.431 ms
```

Before proceeding to the next section, you should verify that loopback connectivity is working between all routers in the IPv6 test bed, and that optimal forwarding paths are being used. The results shown in this section indicate that you have achieved the criteria specified for RIPng, OSPF3, and route redistribution.

Real World Scenario

Troubleshooting an OSPF3 Problem

You can not get OSPF3 to form an adjacency between r3 and r5 as shown in the following graphic. The relevant configuration of r3 is shown here.




```
[edit]
lab@r3# show interfaces
at-0/1/0 {
    atm-options {
        vpi 0 {
            maximum-vcs 64;
        }
    }
    unit 0 {
        point-to-point;
        vci 50;
        family inet6 {
            address fec0:0:2:0::/64 {
                eui-64;
            }
        }
    }
}
lo0 {
    unit 0 {
        family inet6 {
            address fec0:0:0:3::3/128;
        }
    }
}

[edit]
lab@r3# show protocols ospf3
traceoptions {
    file ospf3;
    flag error detail;
    flag hello detail;
    flag packets detail;
}
```

```
area 0.0.0.0 {
    interface lo0.0 {
        passive;
    }
    interface at-0/1/0.0;
}

[edit]
lab@r3# show routing-options
static {
    route 10.0.200.0/24 {
        next-hop 10.0.1.102;
        no-readvertise;
    }
}
aggregate {
    route 10.0.0.0/16;
}
autonomous-system 65412;
```

Even though ping testing is successful between the routers, and `r3` displays the correct interfaces as enabled for OSPF3, there is no adjacency to `r5` and there are no local OSPF3 database entries.

```
[edit]
lab@r3# run show ospf3 interface
Interface             State     Area            DR-ID           BDR-ID        Nbrs
at-0/1/0.0            PtToPt    0.0.0.0         0.0.0.0         0.0.0.0          0
lo0.0                 DRother   0.0.0.0         0.0.0.0         0.0.0.0          0

[edit]
lab@r3# run show ospf3 neighbor

[edit]
lab@r3# run show ospf3 database
```

The general dearth of output generated by the OSPF3 tracing configuration leaves much to be desired.

```
[edit]
lab@r3# run restart routing
Routing protocol daemon started, pid 1275

[edit]
lab@r3#
May  8 20:02:20 trace_on: Tracing to "/var/log/ospf3" started
```

Based on this information, can you identify the problem? If you spotted the fact that `r3` has no source for the 32-bit router ID (RID) required by both OSPF and OSPF3, then you are truly a spectacle of human engineering! Many JNCIE candidates are surprised to find that an IPv6 routing protocol requires the presence of an IPv4-based RID for proper operation. To resolve this problem, you can assign an IPv4 address to one of the router's interfaces, or you can manually assign the RID. The latter approach is demonstrated here:

```
[edit]
lab@r3# set routing-options router-id 10.0.3.3
```

```
[edit]
lab@r3# commit
commit complete

[edit]
lab@r3#
May  8 20:06:22 OSPF trigger router LSA build for area 0.0.0.0
May  8 20:06:22 OSPF trigger router intra-area-prefix LSA build
May  8 20:06:22 OSPF trigger router LSA build for area 0.0.0.0
May  8 20:06:22 OSPF trigger router intra-area-prefix LSA build
May  8 20:06:22 OSPF trigger link LSA build for intf at-0/1/0.0
May  8 20:06:22 OSPF sent Hello fe80::2a0:a5ff:fe3d:234 -> ff02::5 (at-0/1/0.0, IFL 9)
. . .

*** monitor and syslog output disabled, press ESC-Q to enable ***

[edit]
lab@r3# run show ospf3 neighbor
ID              Interface             State     Pri   Dead
10.0.1.5        at-0/1/0.0            Full      128   36
  Neighbor-address fe80::2a0:a5ff:fe28:116e
```

Summary of IPv6 IGP Support

An IGP is an IGP. This section demonstrated that an engineer familiar with the configuration of either RIP or OSPF should not experience a great deal of difficulty when working with their next-generation counterparts. Besides the need to explicitly define a 32-bit router ID, or have a suitable IPv4 address on the router's lo0 interface, there is really no trick to configuring OSPF3 and analyzing its operation. Note that for RIPng, you need to use the link-local address associated with the remote or local router when issuing `show route receive-protocol ripng` or `show route advertising-protocol ripng` commands, respectively.

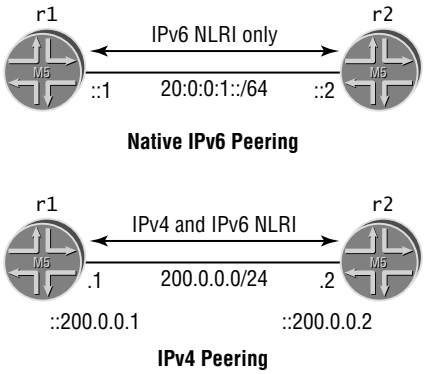
Both RIP and OSPF have special IPv6 versions that are configured separately from their IPv4 counterparts. IS-IS, on the other hand, uses TLVs (Type Length Values) to carry IP prefixes in an opaque fashion. As a result, the same IS-IS instance can handle IPv4, IPv6, or both IPv4 and IPv6. As noted previously, specific configuration is only needed for IS-IS when you do not want it to advertise IPv6 routes associated with the interfaces that it has been configured to run on. IS-IS support of IPv6 routing is demonstrated in the case study but was not covered in this section.

BGP Support for IPv6

The 5.6 JUNOS software release used to develop this book offers BGP support for IPv6 in two ways: through native IPv6 peering, which supports IPv6 Network Layer Reachability Information (NLRI) only, or through the advertisement of IPv6 NLRI over an IPv4-based peering session. The latter approach can support both IPv4 and IPv6 NLRI over the IPv4-based BGP peering

session, but an IPv4-compatible IPv6 address is required for resolution of the BGP next hops associated with the IPv6 NLRI. Figure 5.6 provides a graphical depiction of the options for BGP support of IPv6.

FIGURE 5.6 IPv6 BGP support options



The top portion of Figure 5.6 shows a native IPv6 BGP peering session that supports IPv6 NLRI only. In contrast, the lower portion of the figure shows an IPv4-based peering session that can support both IPv4 and IPv6 NLRI when so configured. Note that an IPv4-compatible IPv6 address, which matches the IPv4 peering address, must be assigned to the peering interface to accommodate next hop resolution on the IPv6 routes. This requirement applies to interface and lo0-based peerings equally. An IPv4-compatible IPv6 address is written as ::*w.x.y.z*. This notation indicates that the first 96 bits of the 128-bit address are all zeros.

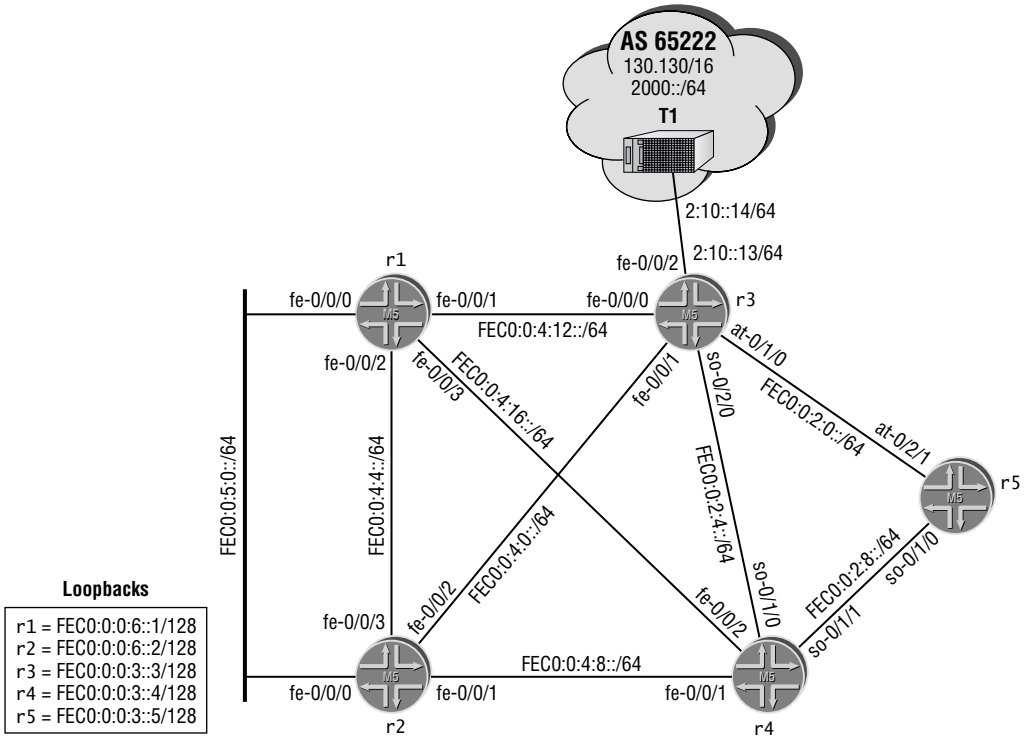
To complete this section, you must configure `r3` according to these requirements:

- Configure a native IPv6 EBGP peering session between `r3` and T1.
- Define and advertise aggregate routes for your AS's IPv6 address space to the T1 peer.
- Ensure that all routers in the IPv6 test bed receive the IPv6 routes advertised by T1 through their IBGP session to `r3`.
- No additional static routes are permitted.
- You may have only one IBGP peering session between each pair of routers within your AS.
- Ensure that there is connectivity between the routers in your AS and the 2000:0:0:0::/64 prefix advertised by the T1 peer.
- Your IPv6-related configuration can not adversely affect the IPv4 aspects of your network.
- You do not have access to the T1 peer.

Refer to Figure 5.7 for the addressing details needed to complete this scenario.

Figure 5.7 indicates that you need to define a native IPv6 EBGP peering session at `r3` based on the IPv6 addressing shown for the T1 peer. The restriction that you have only one IBGP peering session among the routers in the IPv6 test bed requires that you reconfigure the IBGP session to permit the advertisement of IPv6 NLRI over the existing IPv4-based peering sessions.

FIGURE 5.7 BGP and IPv6



Configuring Native IPv6 EBGP Peering

You start your configuration at `r3` by defining the native IPv6 EBGP peering session to T1:

```
[edit protocols bgp group ext-ipv6]
lab@r3# set type external

[edit protocols bgp group ext-ipv6]
lab@r3# set peer-as 65222

[edit protocols bgp group ext-ipv6]
lab@r3# set neighbor 2:10::14
```

Note that the neighbor address specified is the global IPv6 address shown for the T1 peer in Figure 5.7. The *ext-ipv6* peering stanza is displayed next:

```
[edit protocols bgp group ext-ipv6]
lab@r3# show
type external;
```

```
peer-as 65222;
neighbor 2:10::14;
```

To back up the BGP configuration, you need to enable IPv6 and assign the correct IPv6 address (as shown in Figure 5.7) to the fe-0/0/2 interface on r3. Note that although EBGP peering to link-local addresses is supported by JUNOS software release 5.6, current best practice (and RFC 2545, section 2) discourages the use of link-local based EBGP peering because using link-local addressing for EBGP peering mandates the use of a next hop self policy:

```
[edit interfaces fe-0/0/2]
lab@r3# set unit 0 family inet6 address 2:10::13/64
```

r3's fe-0/0/2 interface configuration is displayed with the changes highlighted:

```
[edit]
lab@r3# show interfaces fe-0/0/2
unit 0 {
    family inet {
        address 172.16.0.13/30;
    }
    family inet6 {
        address 2:10::13/64;
    }
}
```

Be sure to commit your changes when satisfied with your modifications.

Verifying Native IPv6 EBGP Peering

After committing the changes on r3, you begin confirmation of the native IPv6 EBGP peering session with execution of a `show bgp summary` command:

```
[edit]
lab@r3# run show bgp summary
Groups: 3 Peers: 8 Down peers: 2
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
inet.0                 2          2          0          0          0          0
inet6.0                1          1          0          0          0          0
Peer            AS  InPkt  OutPkt  OutQ  Flaps Last Up/Dwn State|#Active/
   Received/Damped...
172.16.0.14  65222     72      74     0      0       34:13 Establ
  inet.0: 2/2/0
2:10::14     65222     29      31     0      0       13:17 Establ
  inet6.0: 1/1/0
10.0.3.4     65412      1       4     0      0          26 Establ
  inet.0: 0/0/0
10.0.3.5     65412      1       4     0      0          22 Establ
  inet.0: 0/0/0
```

```
10.0.6.1      65412      2       6      0      0          34 Establ
  inet.0: 0/0/0
10.0.6.2      65412      1       4      0      0          30 Establ
  inet.0: 0/0/0
10.0.9.6      65412      0       0      0      0          46 Active
10.0.9.7      65412      0       0      0      0          46 Active
```

The highlights in the display confirm that the EBGP session to 2:10::14 has been successfully established, and that a single prefix has been received over the peering session. Note that the received prefix has been installed into the `inet6.0` routing table. Your next command displays the IPv6 NLRI advertised by the T1 peer:

```
[edit]
lab@r3# run show route receive-protocol bgp 2:10::14

inet.0: 28 destinations, 30 routes (28 active, 0 holddown, 0 hidden)

inet6.0: 31 destinations, 48 routes (31 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
* 2000::/64               2:10::14             0                  65222 I
```

The output verifies the receipt of a single IPv6 prefix in the form of 2000::/64. A quick ping test is conducted to verify that `r3` has connectivity to host 1 on the 2000::/64 prefix advertised by T1:

```
[edit]
lab@r3# run ping 2000::1 count 2
PING6(56=40+8+8 bytes) 2:10::13 --> 2000::1
16 bytes from 2000::1, icmp_seq=0 hlim=64 time=0.567 ms
16 bytes from 2000::1, icmp_seq=1 hlim=64 time=0.451 ms

--- 2000::1 ping6 statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.451/0.509/0.567 ms
```

Excellent! The test passes with flying colors; things sure seem to be going well for you on the IPv6 front. Because all routers in your AS must ultimately have connectivity to the 2000::/64 prefix, you decide to test reachability when the packet is sourced from an internal address, in this case the FEC0:0:0:3::3/128 address that is assigned to `r3`'s lo0 interface:

```
[edit]
lab@r3# run ping 2000::1 count 2 source fec0:0:0:3::3
PING6(56=40+8+8 bytes) fec0:0:0:3::3 --> 2000::1

--- 2000::1 ping6 statistics ---
2 packets transmitted, 0 packets received, 100% packet loss
```

Noting the failure, you are reminded that you have not yet defined and advertised an aggregate route for your AS's IPv6 addressing space to the T1 peer, which results in T1's inability to route packets back to r3 when they are sourced from any FEC0::/16 prefix. You decide to define the aggregate route for your network's IPv6 address space:

```
[edit routing-options rib inet6.0]
lab@r3# set aggregate route fec0::/16
```

And you also decide to move on to the routing policy required to effect the advertisement of the aggregate route to the T1 peer. The resulting *ipv6-agg* policy is shown next:

```
[edit]
lab@r3# show policy-options policy-statement ipv6-agg
term 1 {
    from {
        protocol aggregate;
        route-filter fec0::/16 exact;
    }
    then accept;
}
```

You apply the *ipv6-agg* policy to the *ext-ipv6* peer group as export, and commit the changes:

```
[edit protocols bgp group ext-ipv6]
lab@r3# set protocols bgp group ext-ipv6 export ipv6-agg

[edit]
lab@r3# commit
commit complete
```

It is easy to confirm that the FEC0::/16 aggregate is being correctly advertised to the T1 peer:

```
[edit]
lab@r3# run show route advertising-protocol bgp 2:10::14

inet6.0: 31 destinations, 48 routes (31 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
* 2000::/64               2:10::14                                65222 I
* fec0::/16               Self                                    I
```

The display confirms that the aggregate route is being sent to T1, which means that pings sourced from one of r3's internal interfaces and targeted at the 2000::/64 prefix should now be possible:

```
[edit]
lab@r3# run ping 2000::1 count 2 source fec0:0:0:3::3
PING6(56=40+8+8 bytes) fec0:0:0:3::3 --> 2000::1
16 bytes from 2000::1, icmp_seq=0 hlim=64 time=0.614 ms
16 bytes from 2000::1, icmp_seq=1 hlim=64 time=0.529 ms
```

```
--- 2000::1 ping6 statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.529/0.571/0.614 ms
```

Excellent! The results confirm that you have configured a native IPv6 EBGP peering session that appears to be fully functional. Before proceeding to the next section, you decide to display the neighbor status for the T1 peer:

```
[edit]
lab@r3# run show bgp neighbor 2:10::14
Peer: 2:10::14+179    AS 65222 Local: 2:10::13+4410   AS 65412
  Type: External    State: Established    Flags: <>
  Last State: OpenConfirm   Last Event: RecvKeepAlive
  Last Error: None
  Export: [ ipv6-agg ]
  Options: <Preference HoldTime AdvertiseInactive PeerAS Refresh>
  Holdtime: 90 Preference: 170
  Number of flaps: 0
  Peer ID: 130.130.0.1      Local ID: 10.0.3.3         Active Holdtime: 90
  Keepalive Interval: 30
  Local Interface: fe-0/0/2.0
  NLRI advertised by peer: inet6-unicast
  NLRI for this session: inet6-unicast
  Peer supports Refresh capability (2)
  Table inet6.0 Bit: 20001
    RIB State: BGP restart is complete
    Send state: in sync
    Active prefixes:            1
    Received prefixes:          1
    Suppressed due to damping:  0
  Last traffic (seconds): Received 8    Sent 9    Checked 9
  Input messages:  Total 59     Updates 4       Refreshes 0     Octets 1299
  Output messages: Total 61     Updates 4       Refreshes 0     Octets 1354
  Output Queue[1]: 0
```

Of note in the resulting display is the indication that the native IPv6 peering session supports IPv6 NLRI only. Also worth noting is that the BGP protocol requires a 32-bit router ID (just like OSPF3). In this case, the router's IPv4-based loopback address is used as the RID for the IPv6 EBGP peering session. If no IPv4 addresses are available on the router, you need to manually assign a RID to enable proper BGP operation.

Configuring IBGP Peering to Support IPv6

With native IPv6 EBGP peering confirmed at `r3`, you move on to the modifications needed to support IPv6 NLRI over your existing IBGP peering sessions. Be aware that changing

the address families configured on a BGP peering session results in temporary disruption to the session while it is being torn down and reestablished. Because this is a non-production network, and because time is always tight in a lab-based examination, you opt for a somewhat heavy-handed approach that configures the new address families at the BGP group level. In a production network, you might consider making this type of change at the neighbor level so that only one BGP session is affected at any given time. You begin on `r3` by adding the IPv6 and IPv4 unicast address families to the existing *`int`* peer group:

```
[edit protocols bgp group int]
lab@r3# set family inet6 unicast

[edit protocols bgp group int]
lab@r3# set family inet4 unicast
```

Note that the default `inet4 unicast` family is disabled when another address family is explicitly configured. Because the restrictions posed in this example result in exam point loss if your IPv4 infrastructure is negatively affected by your IPv6-related configuration, the explicit listing of the `inet4 unicast` family is a significant aspect of a successful configuration. You display the IBGP peer group with highlights added to call out your modifications:

```
[edit protocols bgp group int]
lab@r3# show
type internal;
local-address 10.0.3.3;
family inet {
    unicast;
}
family inet6 {
    unicast;
}
export nhs;
neighbor 10.0.6.1;
neighbor 10.0.6.2;
neighbor 10.0.3.4;
neighbor 10.0.3.5;
neighbor 10.0.9.6;
neighbor 10.0.9.7;
```

The changes are committed at `r3`, and after a few moments the status of the IBGP peering session to `r1` is displayed:

```
[edit]
lab@r3# run show bgp neighbor 10.0.6.1
Peer: 10.0.6.1+179    AS 65412 Local: 10.0.3.3+4587   AS 65412
  Type: Internal    State: Established    Flags: <>
  Last State: OpenConfirm   Last Event: RecvKeepAlive
  Last Error: None
```

```
  Export: [ nhs ]
  Options: <Preference LocalAddress HoldTime AdvertiseInactive AddressFamily
     Refresh>
  Address families configured: inet-unicast inet6-unicast
  Local Address: 10.0.3.3 Holdtime: 90 Preference: 170
  Number of flaps: 0
  Peer ID: 10.0.6.1         Local ID: 10.0.3.3         Active Holdtime: 90
  Keepalive Interval: 30
  NLRI advertised by peer: inet-unicast
  NLRI for this session: inet-unicast
  Peer supports Refresh capability (2)
  Table inet.0 Bit: 10000
    RIB State: BGP restart is complete
    Send state: in sync
    Active prefixes:            0
    Received prefixes:          0
    Suppressed due to damping:  0
  Last traffic (seconds): Received 14   Sent 21   Checked 21
  Input messages:  Total 17     Updates 0       Refreshes 0     Octets 323
  Output messages: Total 26052  Updates 26044   Refreshes 0     Octets 2675071
  Output Queue[0]: 0
```

A key aspect of the resulting display is the indication that `r3` has proposed both the `inet-unicast` and `inet6-unicast` address families while `r1` is proposing only the default `inet-unicast` family. The result is a session with support for the IPv4 address family only. Moving to `r1`, you make similar changes before displaying its modified BGP stanza:

```
[edit protocols bgp group int]
lab@r1# show
type internal;
local-address 10.0.6.1;
family inet {
    unicast;
}
family inet6 {
    unicast;
}
neighbor 10.0.6.2;
neighbor 10.0.3.3;
neighbor 10.0.3.4;
neighbor 10.0.3.5;
neighbor 10.0.9.6;
neighbor 10.0.9.7;;
```

After the IBGP session to r3 is reestablished, you confirm that both address families are now in effect:

```
[edit protocols bgp group int]
lab@r1# run show bgp neighbor 10.0.3.3
Peer: 10.0.3.3+179    AS 65412 Local: 10.0.6.1+3975   AS 65412
  Type: Internal    State: Established    Flags: <>
  Last State: OpenConfirm   Last Event: RecvKeepAlive
  Last Error: None
  Options: <Preference LocalAddress HoldTime AddressFamily Refresh>
  Address families configured: inet-unicast inet6-unicast
  Local Address: 10.0.6.1 Holdtime: 90 Preference: 170
  Number of flaps: 0
  Peer ID: 10.0.3.3         Local ID: 10.0.6.1         Active Holdtime: 90
  Keepalive Interval: 30
  NLRI advertised by peer: inet-unicast inet6-unicast
  NLRI for this session: inet-unicast inet6-unicast
  Peer supports Refresh capability (2)
  Table inet.0 Bit: 10001
    RIB State: BGP restart is complete
    Send state: in sync
    Active prefixes:            2
    Received prefixes:          2
    Suppressed due to damping:  0
  Table inet6.0 Bit: 20001
    RIB State: BGP restart is complete
    Send state: in sync
    Active prefixes:            1
    Received prefixes:          1
    Suppressed due to damping:  0
  Last traffic (seconds): Received 28   Sent 28   Checked 28
  Input messages:  Total 4      Updates 3       Refreshes 0     Octets 226
  Output messages: Total 3      Updates 0       Refreshes 0     Octets 91
  Output Queue[0]: 0
  Output Queue[1]: 0
```

As required, the IBGP session now indicates the required support for both the IPv4 and IPv6 address families. As you begin to catch yourself thinking "this IPv6 stuff is not so bad," you remember that you have not yet demonstrated connectivity from r1 to external IPv6 destinations. You display the 2000::/64 route to see where things stand from the perspective of r1:

```
[edit]
lab@r1# run show route 2000::/64

inet6.0: 28 destinations, 34 routes (27 active, 0 holddown, 1 hidden)
```

Hmmm, nothing is returned, but you happen to catch the indication that there is a single hidden route. Feeling that further investigation is warranted, you display hidden routes at r1:

```
[edit]
lab@r1# run show route hidden detail

inet.0: 106047 destinations, 106047 routes (106047 active, 0 holddown, 0 hidden)

inet6.0: 28 destinations, 34 routes (27 active, 0 holddown, 1 hidden)
2000::/64 (1 entry, 0 announced)
         BGP    Preference: 170/-101
                Next hop type: Unusable
                State: <Hidden Int Ext>
                Local AS: 65412 Peer AS: 65412
                Age: 25:57      Metric: 0
                Task: BGP_65412.10.0.3.3+4140
                AS path: 65222 I
                Localpref: 100
                Router ID: 10.0.3.3
```

The display confirms that the 2000::/64 route has been advertised by r3 to r1, and that r1 has hidden the route due to an unusable next hop. A `show route resolution` command is executed to glean additional details:

```
[edit]
lab@r1# run show route resolution unresolved detail
Table inet.0
Table inet6.0
2000::/64
        Protocol Nexthop: 2:10::14
        Indirect nexthop: 0 -
```

The highlight in the display calls out the initial problem: your existing next hop self policy on r3 has not altered the default BGP next hop on the 2000::/64 route. Because you are not running an IGP instance on r3's fe-0/0/2 interface, other routers cannot resolve the current BGP next hop and so the route is hidden. A look at the *nhs* policy on r3 quickly exposes the problem:

```
[edit]
lab@r3# show policy-options policy-statement nhs
term 1 {
    from {
        protocol bgp;
        neighbor 172.16.0.14;
    }
    then {
```

```
        next-hop self;
    }
}
```

In this case, term 1 of the policy is not matching on the IPv6 peering address associated with T1. The *nhs* policy is redisplayed after the necessary adjustments have been made:

```
[edit policy-options policy-statement nhs]
lab@r3# show
term 1 {
    from {
        protocol bgp;
        neighbor 172.16.0.14;
    }
    then {
        next-hop self;
    }
}
term 2 {
    from {
        protocol bgp;
        neighbor 2:10::14;
    }
    then {
        next-hop self;
    }
}
```

After the changes are committed at r3, you again display the 2000::/64 route at r1:

```
[edit]
lab@r1# run show route 2000::/64

inet6.0: 28 destinations, 34 routes (27 active, 0 holddown, 1 hidden)
```

Oddly enough, the route seems to remain hidden:

```
[edit]
lab@r1# run show route hidden detail

inet.0: 106069 destinations, 106069 routes (106069 active, 0 holddown, 0 hidden)

inet6.0: 28 destinations, 34 routes (27 active, 0 holddown, 1 hidden)
2000::/64 (1 entry, 0 announced)
         BGP    Preference: 170/-101
                Next hop type: Unusable
```

```
State: <Hidden Int Ext>
Local AS: 65412 Peer AS: 65412
Age: 13         Metric: 0
Task: BGP_65412.10.0.3.3+4140
AS path: 65222 I
Localpref: 100
Router ID: 10.0.3.3
```

You decide to again view the route resolution table on r1 to determine if the changes made to the *nhs* policy are taking effect:

```
[edit]
lab@r1# run show route resolution unresolved detail
Table inet.0
Table inet6.0
2000::/64
        Protocol Nexthop: ::10.0.3.3
        Indirect nexthop: 0 -
```

By comparing this display to the output shown previously, it becomes clear that the *nhs* policy at r3 is working as designed. After all, the next hop has changed from 2000::14 to ::10.0.3.3. So what is the problem with the route? If you are wondering why the new next hop has a funky "::" at the front, then you are definitely getting warm! The ::10.0.3.3 prefix is an IPv4-compatible IPv6 address that is used for the BGP next hop of IPv6 routes that are exchanged over IPv4-based peering sessions. To unhide the 2000::/64 route, you must find a way to get a ::10.0.3.3 route installed at r1 that points to r3. Because the rules of engagement in this example prohibit the use of additional static routes, you opt to assign the required IPv4-compatible IPv6 address to r3's lo0 interface:

```
[edit interfaces]
lab@r3# set lo0 unit 0 family inet6 address ::10.0.3.3
```

Because this scenario involves the unidirectional advertisement of IPv6 prefixes over an IPv4-based peering session (from r3 to all other internal peers), it is not strictly necessary that you assign, and then advertise, IPv4 compatible addresses to the other routers in the test bed. It is suggested that you preconfigure this functionality because it is likely that a subsequent scenario will involve the advertisement of IPv6 routes from other IBGP speakers.

The OSPF3 protocol, which is running on the lo0 interface of r3, automatically advertises the new lo0 address, where it will be picked up by r4 and redistributed into the RIPng domain to r1 and r2. The problem is that this behavior results in an additional hop to the ::10.0.3.3 prefix from the perspective of r1, as shown here:

```
[edit]
lab@r1# run traceroute ::10.0.3.3
```

```
traceroute6 to ::10.0.3.3 (::10.0.3.3) from fec0:0:0:6::1, 30 hops max, 12 byte
   packets
 1  fec0:0:4:16:290:69ff:fe6b:3002 (fec0:0:4:16:290:69ff:fe6b:3002)  0.433 ms
    0.387 ms  0.288 ms
 2  ::10.0.3.3 (::10.0.3.3)  0.494 ms  0.489 ms  0.391 ms
```

To resolve this problem, the *ripng-export* policy at r3 is modified:

```
[edit policy-options policy-statement ripng-export]
lab@r3# set term 3 from route-filter ::10.0.3.3 exact
```

The initial policy modification is displayed next with added highlights:

```
[edit policy-options policy-statement ripng-export]
lab@r3# show
term 1 {
    from protocol ripng;
    then accept;
}
term 2 {
    from protocol ospf;
    then accept;
}
term 3 {
    from {
        protocol direct;
        route-filter fec0::/16 orlonger;
        route-filter ::10.0.3.3/128 exact;
    }
    then accept;
}
```

To prevent r3 from re-advertising the IPv4-compatible address owned by r4 into the RIPng domain (which could lead to extra hops), a new term is added to the *ripng-export* policy at r3:

```
[edit policy-options policy-statement ripng-export]
lab@r3# set term 1a from route-filter ::10.0.3.4 exact reject
```

It is critical that the new term *1a* be evaluated *before* the term *2*; the CLI `insert` function is used to correctly sequence the terms.

```
[edit policy-options policy-statement ripng-export]
lab@r3# insert term 1a after term 1
```

The modified policy is displayed next with changes highlighted:

```
[edit policy-options policy-statement ripng-export]
lab@r3# show
term 1 {
```

```
    from protocol ripng;
    then accept;
}
term 1a {
    from {
        route-filter ::10.0.3.4/128 exact reject;
    }
}
term 2 {
    from protocol ospf;
    then accept;
}
term 3 {
    from {
        protocol direct;
        route-filter fec0::/16 orlonger;
        route-filter ::10.0.3.3/128 exact;
    }
    then accept;
}
```

The `reject` action associated with the ::10.0.3.4 route in term *1a* prevents `r3` from redistributing the IPv4-compatible IPv6 address of `r4` into the RIPng domain when it is learned through OSPF3. Similar policy changes are needed at `r4`. The goal here is to ensure that `r1` and `r2` do not receive equal-cost RIPng routes for the IPv4-compatible addresses that are assigned to the lo0 interfaces of `r3` and `r4`. The following capture shows `r1`'s view of the ::10.0.3.3 route before the suggested policy changes are committed at `r4`:

```
[edit]
lab@r1# run show route ::10.0.3.3

inet6.0: 29 destinations, 35 routes (29 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

::10.0.3.3/128     *[RIPng/100] 00:23:22, metric 2, tag 0
                    > to fe80::290:69ff:fe6d:9800 via fe-0/0/1.0
                      to fe80::290:69ff:fe6b:3002 via fe-0/0/3.0
```

With `r1` currently displaying two equal-cost RIPng routes to the ::10.0.3.3 prefix, the policy changes highlighted next are committed at `r4`:

```
[edit policy-options policy-statement ripng-export]
lab@r4# show
term 1 {
```

```
    from protocol ripng;
    then accept;
}
term 1a {
    from {
        route-filter ::10.0.3.3/128 exact reject;
    }
}
term 2 {
    from protocol ospf;
    then accept;
}
term 3 {
    from {
        protocol direct;
        route-filter fec0::/16 orlonger;
        route-filter ::10.0.3.4/128 exact;
    }
    then accept;
}
```

After the changes are committed at r4, r1 displays optimal routing to the IPv4-compatible addresses associated with r3, r4, and r5:

```
[edit]
lab@r1# run show route ::10.0.3.3

inet6.0: 31 destinations, 37 routes (31 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

::10.0.3.3/128     *[RIPng/100] 01:06:59, metric 2, tag 0
                    > to fe80::290:69ff:fe6d:9800 via fe-0/0/1.0

[edit]
lab@r1# run show route ::10.0.3.4

inet6.0: 31 destinations, 37 routes (31 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

::10.0.3.4/128     *[RIPng/100] 00:26:06, metric 2, tag 0
                    > to fe80::290:69ff:fe6b:3002 via fe-0/0/3.0
```

```
[edit]
lab@r1# run show route ::10.0.3.5

inet6.0: 31 destinations, 37 routes (31 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

::10.0.3.5/128      *[RIPng/100] 00:01:30, metric 2, tag 0
                       to fe80::290:69ff:fe6d:9800 via fe-0/0/1.0
                     > to fe80::290:69ff:fe6b:3002 via fe-0/0/3.0
```

The two equal-cost routes to the IPv4-compatible address at `r5` are expected, and their presence helps to validate the redundancy aspects of your IPv6 network. Final verification of your changes comes in the form of successful traceroute testing from `r1` to the 2000::/64 route:

```
[edit]
lab@r1# run traceroute 2000::1
traceroute6 to 2000::1 (2000::1) from fec0:0:0:6::1, 30 hops max, 12 byte
   packets
 1  fec0:0:4:12:290:69ff:fe6d:9800 (fec0:0:4:12:290:69ff:fe6d:9800)  0.429 ms
    0.312 ms  0.27 ms
 2  2000::1 (2000::1)  0.21 ms  0.175 ms  0.158 ms
```

Before proceeding, you should assign IPv4-compatible addresses to the loopback interfaces of the remaining routers, and you must modify the *`int`* peer group on all IBGP speakers to support both IPv4 and IPv6 using the commands shown previously for `r1` and `r3`. Note that the IPv4-compatible IPv6 addresses are not strictly necessary until such time that the IBGP speaker begins advertising IPv6 NLRI. It is suggested you assign the IPv4-compatible addresses now, however, in anticipation of such behavior in a later scenario. The changes made to `r5` are shown here with added highlights:

```
[edit]
lab@r5# show interfaces lo0
unit 0 {
    family inet {
        address 10.0.3.5/32;
    }
    family inet6 {
        address fec0:0:0:3::5/128;
        address ::10.0.3.5/128;
    }
}

[edit]
lab@r5# show protocols bgp group int
type internal;
local-address 10.0.3.5;
```

```
family inet {
    unicast;
}
family inet6 {
    unicast;
}
neighbor 10.0.6.1;
neighbor 10.0.6.2;
neighbor 10.0.3.3;
neighbor 10.0.3.4;
neighbor 10.0.9.6;
neighbor 10.0.9.7;
```

Do not forget that policy modifications are needed at `r1` and `r2` to facilitate the advertisement of their IPv4-compatible loopback addresses into the RIPng domain, and from there, into the OSPF3 domain by `r3` and `r4`, for ultimate use by `r5`. The following highlighted change to `r1`'s *`ripng-export`* policy should also be made at `r2`:

```
[edit]
lab@r1# show policy-options policy-statement ripng-export
term 1 {
    from protocol ripng;
    then accept;
}
term 2 {
    from {
        protocol static;
        route-filter FEC0:0:0:10::/64 exact;
    }
    then accept;
}
term 3 {
    from {
        protocol direct;
        route-filter fec0::/16 orlonger;
        route-filter ::10.0.6.1/128 exact;
    }
    then accept;
}
```

With `r1` and `r2` now advertising their IPv4-compatible loopback address into RIPng, the OSPF3 export policy on `r3` and `r4` must be modified to accept the IPv4-compatible addresses for redistribution into the OSPF3 domain. The highlighted change shown here

for r4 is needed on r3 also:

```
[edit policy-options policy-statement ospf3-export]
lab@r4# show
term 1 {
    from {
        route-filter fec0:0:4::/46 orlonger;
        route-filter fec0:0:0:6::/64 orlonger;
        route-filter ::10.0.6.0/122 orlonger;
    }
    then accept;
}
```

After the changes are committed, equal-cost routes to the IPv4-compatible addresses used by r1 and r2 are confirmed at r5:

```
[edit]
lab@r5# run show route ::10.0.6.2

inet6.0: 25 destinations, 30 routes (25 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

::10.0.6.2/128     *[OSPF/150] 00:04:01, metric 2, tag 0
                      via so-0/1/0.0
                    > via at-0/2/1.0

[edit]
lab@r5# run show route ::10.0.6.1

inet6.0: 25 destinations, 30 routes (25 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

::10.0.6.1/128     *[OSPF/150] 00:04:03, metric 2, tag 0
                      via so-0/1/0.0
                    > via at-0/2/1.0
```

Verifying IBGP Peering and IPv6 Support

You begin verification by confirming that all IBGP sessions are established and that no hidden routes are present:

```
lab@r4> show bgp summary
Groups: 2 Peers: 7 Down peers: 3
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
inet.0            105921     105921          0          0          0          0
inet6.0                1          1          0          0          0          0
```

```
Peer           AS  InPkt  OutPkt  OutQ  Flaps Last Up/Dwn State|#Active/
   Received/Damped...
172.16.0.6   65010      0       0     0      0     4:58:01 Idle
10.0.3.3     65412  21994     250     0      6     2:03:48 Establ
  inet.0: 105921/105921/0
  inet6.0: 1/1/0
10.0.3.5     65412     25      26     0      2       11:58 Establ
  inet.0: 0/0/0
  inet6.0: 0/0/0
10.0.6.1     65412    595     597     0      0     4:57:25 Establ
  inet.0: 0/0/0
  inet6.0: 0/0/0
10.0.6.2     65412     23      24     0      1       10:41 Establ
  inet.0: 0/0/0
  inet6.0: 0/0/0
10.0.9.6     65412      0       0     0      0     4:58:01 Active
10.0.9.7     65412      0       0     0      0     4:58:01 Active

lab@r4> show route hidden

inet.0: 105946 destinations, 105948 routes (105946 active, 0 holddown, 0 hidden)

inet6.0: 32 destinations, 52 routes (32 active, 0 holddown, 0 hidden)
```

The `active` sessions in the previous display relate to peers that are not part of the current IPv6 test bed and are therefore of no concern. The lack of hidden routes is a very good sign, as is the presence of established IBGP sessions that make use of both the `inet.0` and `inet6.0` routing tables, because this indicates that all IBGP sessions have been correctly configured for IPv4 and IPv6 support.

For final verification, traceroutes are performed to external and internal destinations. Make sure that you also test IPv4 connectivity to catch any possible mistakes before exam grading commences:

```
lab@r4> traceroute 2000::1
traceroute6 to 2000::1 (2000::1) from ::10.0.3.4, 30 hops max, 12 byte packets
 1  ::10.0.3.3 (::10.0.3.3)  1.035 ms  0.719 ms  0.694 ms
 2  * * *
 3  *^C
```

Hmm, the traceroute failure from `r4` to the 2000::/64 prefix provides a less than auspicious start. The highlighted portion of the capture shows that the packet is being sourced from the IPv4-compatible address assigned to `r4`'s loopback address.

When present, IPv4-compatible addresses are preferred over IPv6 addresses for purposes of primary address determination on a given interface.

You suspect that the problem relates to the aggregate route being advertised from r3 to T1. Specifically, you are concerned that the FEC0::/16 aggregate does not encompass the IPv4-compatible address space now deployed in the IPv6 test bed. To test your theory, you source the packet from one of the site-local addresses owned by r4:

```
lab@r4> traceroute 2000::1 source fec0:0:0:3::4
traceroute6 to 2000::1 (2000::1) from fec0:0:0:3::4, 30 hops max, 12 byte
   packets
 1  fec0:0:2:4:2a0:a5ff:fe3d:234 (fec0:0:2:4:2a0:a5ff:fe3d:234)  0.768 ms
    0.703 ms  0.563 ms
 2  2000::1 (2000::1)  0.496 ms  0.501 ms  0.447 ms
```

The traceroute succeeds when sourced from a FEC0::/16 prefix. Although not clearly mandated by the rules governing this configuration scenario, you make the highlighted changes on r3 to effect the advertisement of a ::10.0.0.0/112 aggregate route to T1:

```
[edit]
lab@r3# show routing-options rib inet6.0
aggregate {
    route fec0::/16;
    route ::10.0.0.0/112;
}

[edit]
lab@r3# show policy-options policy-statement ipv6-agg
term 1 {
    from {
        protocol aggregate;
        route-filter fec0::/16 exact;
        route-filter ::10.0.0.0/112 orlonger;
    }
    then accept;
}
```

After the changes are committed, traceroutes succeed with the default source address selection of the IPv4-compatible address at r4. Various traceroutes are performed to both IPv4 and IPv6 destinations:

```
lab@r4> traceroute 2000::1
traceroute6 to 2000::1 (2000::1) from ::10.0.3.4, 30 hops max, 12 byte packets
 1  ::10.0.3.3 (::10.0.3.3)  0.963 ms  0.718 ms  0.696 ms
 2  2000::1 (2000::1)  0.489 ms  0.518 ms  0.452 ms
```

Verification proceeds by testing other IPv6 and IPv4 destinations:

```
lab@r4> traceroute 130.130.0.1
traceroute to 130.130.0.1 (130.130.0.1), 30 hops max, 40 byte packets
```

```
 1  10.0.2.5 (10.0.2.5)  0.757 ms  0.569 ms  0.502 ms
 2  130.130.0.1 (130.130.0.1)  0.467 ms  0.463 ms  0.428 ms

lab@r4> traceroute fec0:0:0:6::1
traceroute6 to fec0:0:0:6::1 (fec0:0:0:6::1) from
   fec0:0:4:16:290:69ff:fe6b:3002, 30 hops max, 12 byte packets
 1  fec0:0:0:6::1 (fec0:0:0:6::1)  1.009 ms  0.556 ms  0.411 ms

lab@r4> traceroute ::10.0.6.1
traceroute6 to ::10.0.6.1 (::10.0.6.1) from ::10.0.3.4, 30 hops max,
   12 byte packets
 1  ::10.0.6.1 (::10.0.6.1)  0.633 ms  0.435 ms  0.41 ms
```

Traceroutes to internal IPv6, IPv4, and IPv4-compatible IPv6 destinations succeed. Although not shown here for the sake of brevity, you can assume that all internal destinations have been confirmed from all routers in the IPv6 test bed, which completes your IPv6 and BGP support configuration scenario.

Tunneling IPv6

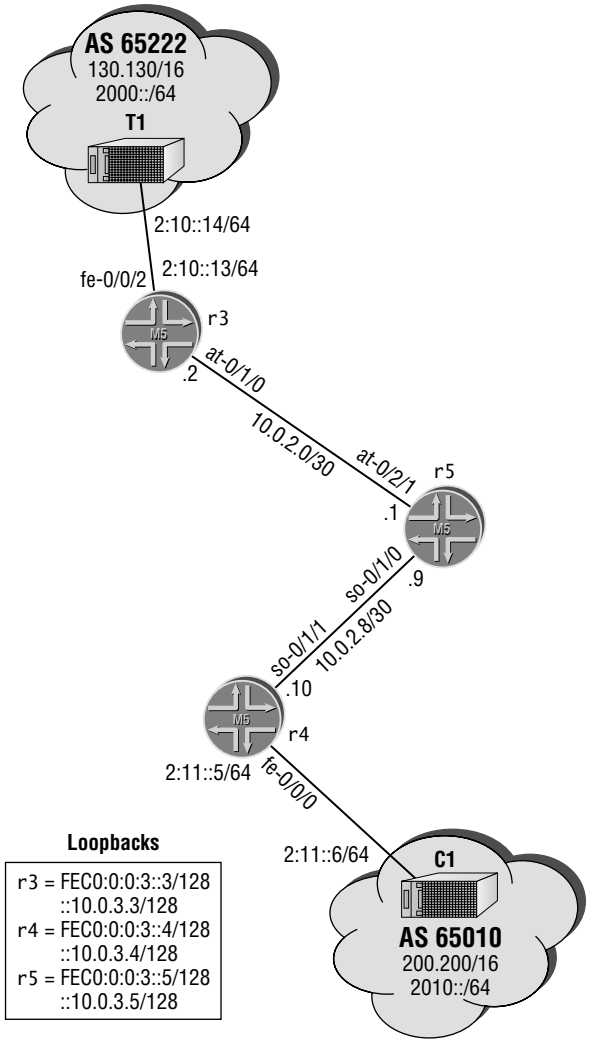
This section demonstrates configuration and verification of IPv4 tunnels that support the transport of IPv6 datagrams. IPv6 tunneling is a significant feature because it is one of the principal mechanisms that allow for graceful migration from IPv4 to IPv6; with tunnels, islands of IPv6 connectivity can be interconnected across IPv4-only infrastructure. As with PIM sparse mode rendezvous points and first hop routers, a Tunnel Services (TS) PIC is required in the routers that form your IP over IP (IP-IP) tunnel endpoints. You can assume that the TS PICs deployed in Chapter 4 are still present in `r3` and `r4`. Figure 5.8 provides the topology details needed to complete this section.

Figure 5.8 clearly shows the challenge that confronts you; you must find a way to provide transit IPv6 routing services across a node (`r5`) that does not offer any IPv6 support. You will be deactivating the Frame Relay link between `r3` and `r4` to ensure that IPv6 traffic is forced to transit `r5`. You need not concern yourself with `r1` and `r2` in this configuration scenario.

To complete this section, you must configure your test bed according to these requirements:

- Reload the OSPF baseline configuration on `r5`.
- Deactivate the Frame Relay link between `r3` and `r4` and the internal facing Fast Ethernet links at `r3` and `r4`.
- Establish an IP-IP tunnel between `r3` and `r4`.
- Provide transit IPv6 routing services between T1 and C1 over an IP-IP tunnel; this traffic must transit `r5`.
- No static routes (IPv4 or IPv6) are permitted on `r3` or `r4`.

FIGURE 5.8 IPv6 over IP-IP tunnels



Preliminary Configuration

You begin with the preliminary configuration changes needed to comply with the scenario's restrictions and to establish the native IPv6 peering session between `r4` and C1. The first set of commands restores and commits the OSPF baseline configuration from Chapter 1 into `r5`:

```
[edit]
lab@r5# load override r5-baseline
load complete
```

After committing the changes on r5, you move on to deactivate the Frame Relay link between r3 and r4, and the internal-facing Fast Ethernet links at r3 and r4, in accordance with the requirements of this configuration example:

```
[edit]
lab@r3# deactivate interfaces so-0/2/0

[edit]
lab@r3# deactivate interfaces fe-0/0/0

[edit]
lab@r3# deactivate interfaces fe-0/0/1
```

Although not shown, similar `deactivate` commands should be entered at r4 also. Note that deactivating the POS-based Frame Relay link at either r3 or r4 is sufficient for the purposes of this example. The next sequence of commands configures the native IPv6 EBGP peering session between r4 and C1, beginning with the configuration of r4's fe-0/0/0 interface:

```
[edit interfaces fe-0/0/0]
lab@r4# set unit 0 family inet6 address 2:11::5/64
```

And now the native IPv6 EBGP peering session is defined and committed at r4:

```
[edit protocols bgp group ext-v6]
lab@r4# set type external peer-as 65010

[edit protocols bgp group ext-v6]
lab@r4# set neighbor 2:11::6

[edit protocols bgp group ext-v6]
lab@r4# commit
commit complete
```

The IPv6 EBGP peering–related changes made to r4's configuration are displayed next with highlights:

```
[edit]
lab@r4# show interfaces fe-0/0/0
unit 0 {
    family inet {
        address 172.16.0.5/30;
    }
    family inet6 {
        address 2:11::5/64;
    }
}

[edit]
lab@r4# show protocols bgp group ext-v6
```

```
type external;
export ipv6-agg;
peer-as 65010;
neighbor 2:11::6;
```

A few minutes after the `commit`, you confirm EBGP session status between `r4` and C1:

```
[edit]
lab@r4# run show bgp neighbor 2:11::6
Peer: 2:11::6+1673    AS 65010 Local: 2:11::5+179     AS 65412
  Type: External    State: Established    Flags: <>
  Last State: OpenConfirm   Last Event: RecvKeepAlive
  Last Error: None
  Options: <Preference HoldTime AdvertiseInactive PeerAS Refresh>
  Holdtime: 90 Preference: 170
  Number of flaps: 0
  Peer ID: 200.200.0.1      Local ID: 10.0.3.4         Active Holdtime: 90
  Keepalive Interval: 30
  Local Interface: fe-0/0/0.0
  NLRI advertised by peer: inet6-unicast
  NLRI for this session: inet6-unicast
  Peer supports Refresh capability (2)
  Table inet6.0 Bit: 20001
    RIB State: BGP restart is complete
    Send state: in sync
    Active prefixes:            1
    Received prefixes:          1
    Suppressed due to damping:  0
  Last traffic (seconds): Received 16   Sent 15   Checked 15
  Input messages:  Total 11     Updates 2       Refreshes 0     Octets 344
  Output messages: Total 12     Updates 2       Refreshes 0     Octets 356
  Output Queue[1]: 0

[edit]
lab@r4# run show route 2010::/64

inet6.0: 31 destinations, 37 routes (31 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

2010::/64          *[BGP/170] 00:02:00, MED 0, localpref 100
                      AS path: 65010 I
                    > to 2:11::6 via fe-0/0/0.0
```

The command's output confirms that the IPv6 EBGP peering session has been established, and that r4 is receiving a 2010::/64 route from the C1 peer. Be aware that the 2010::/64 route, when received by r3, will be hidden due to its inability to resolve the route's next hop. This condition will persist until the appropriate next hop self policy is added to r4:

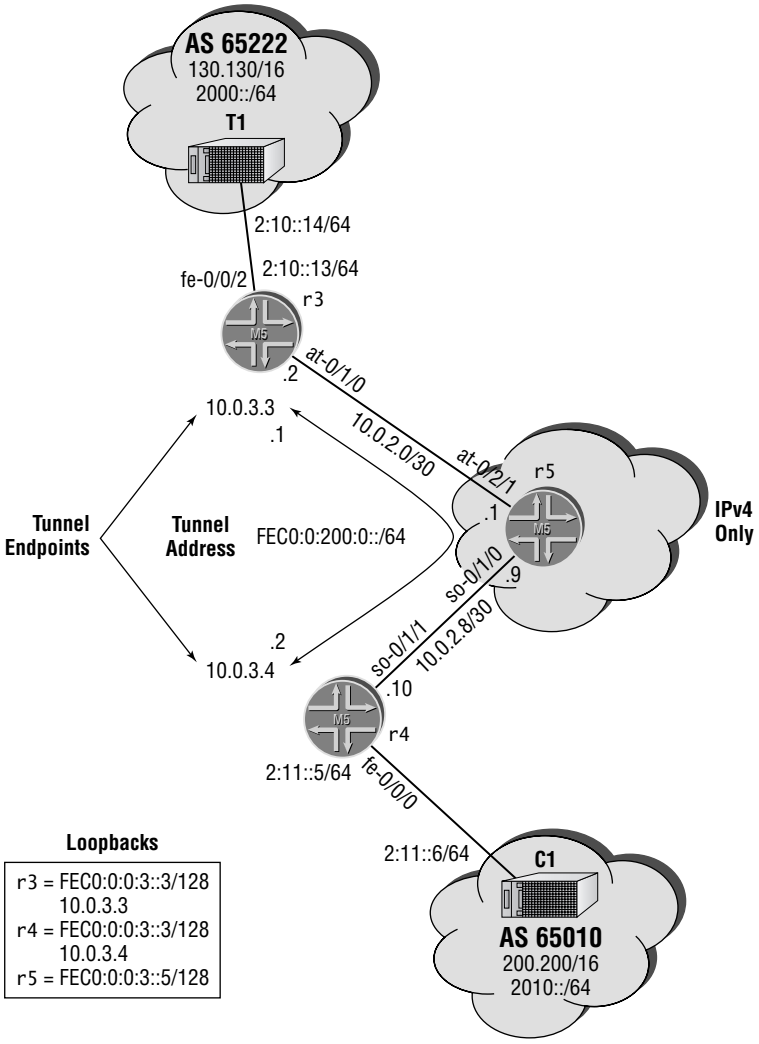
```
[edit]
lab@r3# run show route 2010::/64 hidden detail

inet6.0: 33 destinations, 41 routes (32 active, 0 holddown, 1 hidden)
2010::/64 (1 entry, 0 announced)
         BGP    Preference: 170/-101
                Next hop type: Unusable
                State: <Hidden Int Ext>
                Local AS: 65412 Peer AS: 65412
                Age: 4:20       Metric: 0
                Task: BGP_65412.10.0.3.4+2567
                AS path: 65010 I
                Localpref: 100
                Router ID: 10.0.3.4

[edit]
lab@r3# run show route resolution unresolved
Table inet.0
Table inet6.0
2010::/64
        Protocol Nexthop: 2:11::6
        Indirect nexthop: 0 -
```

Note that the IPv4-compatible IPv6 address that was assigned to r4's lo0 interface as part of the previous section is critical to the overall success of this scenario. With the IPv4-compatible address already present at r4, a simple next hop self policy (or a passive OSPF3 instance on the fe-0/0/0 interface) is all that is needed to make the 2010::/64 route active at r3. With preliminary configuration confirmed, you move to the next section, which details the actual tunnel configuration.

IP-IP Tunnel Configuration

IP-IP tunnels are normally established between loopback addresses for maximum reliability. The tunnel itself might or might not be numbered. A numbered tunnel is configured in this example because numbered interfaces make troubleshooting more straightforward. Figure 5.9 illustrates your tunnel configuration plan.

FIGURE 5.9 IP-IP tunnel configuration



Before you configure the IP-IP tunnel, the TS PIC's position is verified:

```
[edit]
lab@r3# run show chassis fpc pic-status
Slot 0 Online
  PIC 0    4x F/E, 100 BASE-TX
  PIC 1    2x OC-3 ATM, MM
  PIC 2    4x OC-3 SONET, MM
  PIC 3    1x Tunnel
```

You can assume that r5 also has its TS PIC installed in slot 3. The following commands configure the IP-IP tunnel at r3. Note that port 0 must be specified when configuring a TS PIC due to its lack of physical ports:

```
[edit]
lab@r3# edit interfaces ip-0/3/0

[edit interfaces ip-0/3/0]
lab@r3# set unit 0 tunnel source 10.0.3.3

[edit interfaces ip-0/3/0]
lab@r3# set unit 0 tunnel destination 10.0.3.4
```

Next, you assign the IPv6 family to the tunnel to enable IPv6 support. Note that IPv6 addressing is also assigned at this time:

```
[edit interfaces ip-0/3/0]
lab@r3# set unit 0 family inet6 address fec0:0:200::1/64
```

The configuration changes at r4 are similar to those made at r3:

```
[edit interfaces ip-0/3/0]
lab@r4# show
unit 0 {
    tunnel {
        source 10.0.3.4;
        destination 10.0.3.3;
    }
    family inet6 {
        address fec0:0:200::2/64;
    }
}
```

The IP-IP tunnel configuration shown in this example is unnumbered from the perspective of the IPv4 protocol.

Confirming IP-IP Tunnel Operation

After committing the changes, the IP-IP tunnel status is determined:

```
[edit interfaces ip-0/3/0]
lab@r4# run show interfaces ip-0/3/0
Physical interface: ip-0/3/0, Enabled, Physical link is Up
  Interface index: 23, SNMP ifIndex: 36
  Type: IPIP, Link-level type: IP-over-IP, MTU: Unlimited, Speed: 800mbps
  Device flags   : Present Running
  Interface flags: SNMP-Traps
  Input rate     : 0 bps (0 pps)
  Output rate    : 0 bps (0 pps)
```

```
  Logical interface ip-0/3/0.0 (Index 11) (SNMP ifIndex 40)
    Flags: Point-To-Point SNMP-Traps IP-Header
       10.0.3.3:10.0.3.4:4:df:64:00000000
    Encapsulation: IPv4-NULL
  Input packets : 61
  Output packets: 63
    Protocol inet6, MTU: 1480
      Flags: None
      Addresses, Flags: Is-Preferred
        Destination: fe80::/64, Local: fe80::2a0:a5ff:fe28:d36
      Addresses, Flags: Is-Preferred Is-Primary
        Destination: fec0:0:200::/64, Local: fec0:0:200::2
```

The display confirms that the IP-IP tunnel has been established between the loopback addresses of r3 and r4 and that the tunnel has been configured to support the IPv6 family with IPv6 addressing assigned according to Figure 5.9. A quick ping test is initiated at r4 while traffic is monitored at r3 on both the ip-0/3/0 and at-0/2/1 interfaces to confirm operation of the tunnel's data plane:

```
[edit interfaces ip-0/3/0]
lab@r4# run ping fec0:0:200::1
PING6(56=40+8+8 bytes) fec0:0:200::2 --> fec0:0:200::1
16 bytes from fec0:0:200::1, icmp_seq=0 hlim=64 time=1.086 ms
16 bytes from fec0:0:200::1, icmp_seq=1 hlim=64 time=0.934 ms
. . .

[edit]
lab@r3# run monitor traffic interface ip-0/3/0
verbose output suppressed, use <detail> or <extensive> for full protocol decode
Listening on ip-0/3/0, capture size 96 bytes

18:22:59.714406  In fec0:0:200::2 > fec0:0:200::1: icmp6: echo request
18:23:00.714834  In fec0:0:200::2 > fec0:0:200::1: icmp6: echo request
18:23:01.714769  In fec0:0:200::2 > fec0:0:200::1: icmp6: echo request
^C
3 packets received by filter
0 packets dropped by kernel
```

The output confirms the receipt of ICMPv6 echo request packets on r3's ip-0/3/0 tunnel interface. You now monitor r3's at-0/1/0 interface to confirm IP-IP encapsulation of IPv6 datagrams:

```
[edit]
lab@r3# run monitor traffic interface at-0/1/0
verbose output suppressed, use <detail> or <extensive> for full protocol decode
Listening on at-0/1/0, capture size 96 bytes
```

```
18:24:05.196439 Out fe80::2a0:a5ff:fe3d:234 > ff02::5:  OSPFv3-hello 36: rtrid
   10.0.3.3 backbone [hlim 1]
. . .
18:24:05.714843 Out IP 10.0.3.3 > 10.0.3.4: fec0:0:200::1 > fec0:0:200::2:
   icmp6: echo reply (encap)
. . .
18:24:06.715321 Out IP 10.0.3.3 > 10.0.3.4: fec0:0:200::1 > fec0:0:200::2:
   icmp6: echo reply (encap)
```

The results confirm that the IPv6 numbered IP-IP tunnel is operational between r3 and r4.

Adjusting IBGP and EBGP Policy

With the native IPv6 EBGP peering sessions established between r4 and C1, and the establishment of bidirectional IP-IP tunnel functionality between r3 and r4 (bidirectional tunnel functionality is achieved with the configuration of two unidirectional IP-IP tunnels), you anticipate that a few modifications to r4's IBGP and EBGP policies and the definition of some IPv6 aggregate routes are all that separate you from a successful completion of the IPv6-over-IPv4 tunneling task.

You start at r4 with the configuration needed to create, and then advertise, IPv6 aggregate routes for your AS to the C1 peer. This step is necessary to ensure that C1 can respond to ping and traceroute requests that originate within your AS. Note that this configuration is already in place at r3 from a previous section. The changes made to r4's configuration are shown next with added highlights:

```
[edit]
lab@r4# show routing-options rib inet6.0
aggregate {
    route fec0::/16;
    route ::10.0.0.0/112;
}

[edit]
lab@r4# show policy-options policy-statement ipv6-agg
term 1 {
    from {
        protocol aggregate;
        route-filter ::10.0.0.0/112 exact;
        route-filter fec0::/16 exact;
    }
    then accept;
}
[edit]
lab@r4# show protocols bgp group ext-v6
type external;
export ipv6-agg;
```

```
peer-as 65010;
neighbor 2:11::6;
```

The *nhs* policy is modified at r4 so that it also overwrites the BGP next hop on the 2010::/64 route before it is advertised to IBGP peers. The changes made to r4's *nhs* policy are called out here with highlights:

```
[edit policy-options policy-statement nhs]
lab@r4# show
term 1 {
    from {
        protocol bgp;
        neighbor 172.16.0.6;
    }
    then {
        next-hop self;
    }
}
term 2 {
    from neighbor 2:11::6;
    then {
        next-hop self;
    }
}
```

The results are confirmed at r3 by verifying that the 2010::/64 route is no longer hidden:

```
[edit]
lab@r3# run show route 2010::/64 detail

inet6.0: 17 destinations, 22 routes (17 active, 0 holddown, 0 hidden)
2010::/64 (1 entry, 1 announced)
        *BGP    Preference: 170/-101
                Source: 10.0.3.4
                Next hop type: Reject
                Protocol next hop: ::10.0.3.4 Indirect next hop: 85c30a8 124
                State: <Active Int Ext>
                Local AS: 65412 Peer AS: 65412
                Age: 1:30       Metric: 0       Metric2: 0
                Task: BGP_65412.10.0.3.4+179
                Announcement bits (3): 0-KRT 5-BGP.0.0.0.0+179 6-Resolve inet6.0
                AS path: 65010 I
                Localpref: 100
                Router ID: 10.0.3.4
```

The good news is that the modifications you made to r4's *nhs* policy have resulted in the 2010::/64 route being made active at r3. The bad news relates to the route's Reject next hop indication, which seems more than a little strange. Deciding to investigate further, you uncover a valuable clue when you direct pings to the ::10.0.3.4 BGP next hop associated with the 2010::/64 route:

```
[edit]
lab@r3# run ping ::10.0.3.4
PING6(56=40+8+8 bytes) ::10.0.3.3 --> ::10.0.3.4
ping: sendmsg: No route to host
ping6: wrote ::10.0.3.4 16 chars, ret=-1
ping: sendmsg: No route to host
ping6: wrote ::10.0.3.4 16 chars, ret=-1
^C
--- ::10.0.3.4 ping6 statistics ---
2 packets transmitted, 0 packets received, 100% packet loss
```

Displaying the route to the BGP next hop sheds additional light on the problem:

```
[edit]
lab@r3# run show route ::10.0.3.4

inet6.0: 17 destinations, 22 routes (17 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

::10.0.0.0/112     *[Aggregate/130] 03:34:37
                      Reject
```

The lack of an IPv6-capable IGP between r3 and r4 results in a black hole when the longest match for the ::10.0.3.4 route ends up being the locally defined ::10.0.0.0/112 aggregate. This situation could easily be remedied with a static route on r3 and r4 that points packets addressed to the remote BGP next hop to the IP-IP tunnel endpoint. However, the use of static routing is prohibited by the rules of engagement set forth for this example. The motivation for restricting static routes is to verify that the JNCIE candidate is comfortable enabling an IPv6 IGP over an IPv4 tunnel.

Many JNCIE candidates shy away from running an IGP across a tunnel because of previous experience with route recursion problems. Recursion problems surface in these cases when the IGP forms an adjacency over the tunnel and subsequently learns that the shortest path to the remote tunnel endpoint is the tunnel itself. Attempting to install the tunnel as the forwarding next hop for itself creates a route recursion problem that results in the teardown of the tunnel (and loss of the IGP adjacency that had formed over the tunnel). This behavior can be seen next in the context of an IPv4 tunnel that is enabled for OSPF routing:

```
[edit]
lab@r3# show interfaces ip-0/3/0
unit 0 {
```

```
    tunnel {
        source 10.0.3.3;
        destination 10.0.3.4;
    }
    family inet {
        address 10.0.20.1/30;
    }
}
[edit]
lab@r3# show protocols ospf
traceoptions {
    file ospf;
}
area 0.0.0.0 {
    interface so-0/2/0.100;
    interface at-0/1/0.0;
    interface ip-0/3/0.0;
}
```

Once the configuration is committed, the effects of IGP flap can be see in the results of ping exchanges, and in the OSPF trace output (the tracing configuration has no flags enabled to minimize clutter):

```
[edit]
lab@r3# run ping 10.0.3.4
PING 10.0.3.4 (10.0.3.4): 56 data bytes
64 bytes from 10.0.3.4: icmp_seq=0 ttl=254 time=1.306 ms
ping: sendto: Network is down
ping: sendto: Network is down
ping: sendto: Network is down
ping: sendto: Network is down
64 bytes from 10.0.3.4: icmp_seq=5 ttl=254 time=1.321 ms
64 bytes from 10.0.3.4: icmp_seq=6 ttl=254 time=1.666 ms
64 bytes from 10.0.3.4: icmp_seq=7 ttl=254 time=1.654 ms
. . .
64 bytes from 10.0.3.4: icmp_seq=14 ttl=254 time=1.617 ms
64 bytes from 10.0.3.4: icmp_seq=15 ttl=254 time=1.053 ms
ping: sendto: Network is down
ping: sendto: Network is down
^C
--- 10.0.3.4 ping statistics ---
18 packets transmitted, 12 packets received, 33% packet loss
round-trip min/avg/max/stddev = 1.053/3.284/22.398/5.767 ms
```

```
[edit protocols ospf]
lab@r3#
*** monitor and syslog output enabled, press ESC-Q to disable ***
May 12 20:11:07 RPD_OSPF_NBRUP: OSPF neighbor 10.0.20.2 (ip-0/3/0.0) state
   changed from Exchange to Full due to DBD exchange complete
May 12 20:11:08 RPD_OSPF_NBRDOWN: OSPF neighbor 10.0.20.2 (ip-0/3/0.0) state
   changed from Full to Down due to Kill all neighbors
May 12 20:11:18 RPD_OSPF_NBRUP: OSPF neighbor 10.0.20.2 (ip-0/3/0.0) state
   changed from Init to ExStart due to Two way communication established
May 12 20:11:23 RPD_OSPF_NBRUP: OSPF neighbor 10.0.20.2 (ip-0/3/0.0) state
   changed from Exchange to Full due to DBD exchange complete
May 12 20:11:23 RPD_OSPF_NBRDOWN: OSPF neighbor 10.0.20.2 (ip-0/3/0.0) state
   changed from Full to Down due to Kill all neighbors
. . .
```

You can, however, run an IPv6 IGP across an IPv4 tunnel without experiencing recursion problems. This is because the tunnel endpoints are IPv4 based, which means that the IPv6-based IGP does not attempt to install the tunnel as the next hop to itself because the IPv6 IGP never actually learns the tunnel endpoints over the tunnel itself. Keeping this in mind, you boldly enable the OSPF3 instance to run on the tunnel interface at `r3` (a similar command is needed on `r4` but is not shown here):

```
[edit]
lab@r3# set protocols ospf3 area 0 interface ip-0/3/0
```

After committing the changes, OSPF3 adjacency is confirmed:

```
[edit]
lab@r4# run show ospf3 neighbor
ID              Interface             State     Pri   Dead
10.0.3.3        ip-0/3/0.0            Full      128   35
  Neighbor-address fe80::2a0:a5ff:fe3d:234
```

With the adjacency established, routing between the IPv4-compatible IPv6-based loopback addresses (which are used at the BGP next hops for routes learned from the T1 and C1 EBGP peerings) is possible:

```
[edit]
lab@r4# run show route ::10.0.3.3

inet6.0: 19 destinations, 23 routes (19 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

::10.0.3.3/128     *[OSPF/10] 00:04:39, metric 1
                    > via ip-0/3/0.0
[edit]
lab@r4# run ping ::10.0.3.3 source ::10.0.3.4
```

```
PING6(56=40+8+8 bytes) ::10.0.3.4 --> ::10.0.3.3
16 bytes from ::10.0.3.3, icmp_seq=0 hlim=64 time=1.211 ms
16 bytes from ::10.0.3.3, icmp_seq=1 hlim=64 time=1.061 ms
16 bytes from ::10.0.3.3, icmp_seq=2 hlim=64 time=1.454 ms
16 bytes from ::10.0.3.3, icmp_seq=3 hlim=64 time=1.354 ms
16 bytes from ::10.0.3.3, icmp_seq=4 hlim=64 time=1.256 ms
^C
--- ::10.0.3.3 ping6 statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 1.061/1.267/1.454 ms
```

Confirming IPv6-over-IPv4 Tunnel Operation

The configuration steps that have brought you to this point involved the confirmation of various aspects of your IPv6-over-IPv4 tunnel configuration. The commands and output shown next demonstrate (and review) key operational checks. You begin by confirming the presence of the 2000::/64 and the 2010::/64 routes:

```
[edit]
lab@r4# run show route 2000::/64 detail

inet6.0: 19 destinations, 23 routes (19 active, 0 holddown, 0 hidden)
2000::/64 (1 entry, 1 announced)
        *BGP    Preference: 170/-101
                Source: 10.0.3.3
                Next hop: via ip-0/3/0.0, selected
                Protocol next hop: ::10.0.3.3 Indirect next hop: 85c8348 66
                State: <Active Int Ext>
                Local AS: 65412 Peer AS: 65412
                Age: 17:20      Metric: 0       Metric2: 1
                Task: BGP_65412.10.0.3.3+179
                Announcement bits (3): 0-KRT 4-BGP.0.0.0.0+179 5-Resolve inet6.0
                AS path: 65222 I
                Localpref: 100
                Router ID: 10.0.3.3

[edit]
lab@r4# run show route 2010::/64 detail

inet6.0: 19 destinations, 23 routes (19 active, 0 holddown, 0 hidden)
2010::/64 (1 entry, 1 announced)
```

```
*BGP    Preference: 170/-101
        Source: 2:11::6
        Next hop: 2:11::6 via fe-0/0/0.0, selected
        State: <Active Ext>
        Local AS: 65412 Peer AS: 65010
        Age: 2:00:03    Metric: 0
        Task: BGP_65010.2:11::6+3681
        Announcement bits (3): 0-KRT 4-BGP.0.0.0.0+179 5-Resolve inet6.0
        AS path: 65010 I
        Localpref: 100
        Router ID: 200.200.0.1
```

Both routes are present and considered active. Note that the 2000::/64 route identifies a BGP next hop of ::10.0.3.3 and a forwarding next hop in the form of the IP-IP tunnel. Reachability of the tunnel endpoints (as learned through the OSPF3 protocol), as well as the IPv6-related addressing is confirmed:

```
[edit]
lab@r4# run ping 10.0.3.3 count 2
PING 10.0.3.3 (10.0.3.3): 56 data bytes
64 bytes from 10.0.3.3: icmp_seq=0 ttl=254 time=1.664 ms
64 bytes from 10.0.3.3: icmp_seq=1 ttl=254 time=1.449 ms

--- 10.0.3.3 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 1.449/1.556/1.664/0.107 ms

[edit]
lab@r4# run ping fec0:0:20::1 count 2
PING6(56=40+8+8 bytes) ::10.0.3.4 --> fec0:0:20::1
ping: sendmsg: No route to host
ping6: wrote fec0:0:20::1 16 chars, ret=-1
ping: sendmsg: No route to host
ping6: wrote fec0:0:20::1 16 chars, ret=-1

--- fec0:0:20::1 ping6 statistics ---
2 packets transmitted, 0 packets received, 100% packet loss

[edit]
lab@r4# run ping fec0:0:200::1 count 2
PING6(56=40+8+8 bytes) fec0:0:200::2 --> fec0:0:200::1
```

```
16 bytes from fec0:0:200::1, icmp_seq=0 hlim=64 time=1.525 ms
16 bytes from fec0:0:200::1, icmp_seq=1 hlim=64 time=1.388 ms

--- fec0:0:200::1 ping6 statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 1.388/1.456/1.525 ms
```

The IP-IP tunnel is operational, and the IPv4-based endpoints and assigned IPv6 addresses are confirmed as reachable. You now check the tunnel's OSPF3 adjacency status, and the ability to route through the tunnel to the IPv4-compatible IPv6 loopback addresses from the perspective of r3:

```
[edit]
lab@r3# run show ospf3 neighbor
ID              Interface             State     Pri   Dead
10.0.3.4        ip-0/3/0.0            Full      128   37
  Neighbor-address fe80::2a0:a5ff:fe28:d36

[edit]
lab@r3# run show route ::10.0.3.4

inet6.0: 19 destinations, 25 routes (19 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

::10.0.3.4/128     *[OSPF/10] 00:18:00, metric 1
                    > via ip-0/3/0.0

[edit]
lab@r3# run traceroute ::10.0.3.4
traceroute6 to ::10.0.3.4 (::10.0.3.4) from ::10.0.3.3, 30 hops max, 12 byte
   packets
 1  ::10.0.3.4 (::10.0.3.4)  1.985 ms  0.897 ms  1.325 ms
```

All of the results shown thus far indicate that the IP-IP tunnel is operational for the transport of IPv6 packets. The final confirmation comes in the form of a successful traceroute from C1 to the 2000::/64 route that is advertised by the T1 router. Note that care must be taken to source the traceroute from the 2010::1/128 address assigned to C1's loopback address because the 2:11::/64 EBGP peering address is not advertised within your AS, or to the T1 peer:

```
[edit]
lab@r4# run telnet 2010::1
Trying 2010::1...
Connected to 2010::1.
Escape character is '^]'.

c1 (ttyp0)
```

```
login: lab
Password:
Last login: Mon May 12 09:38:40 on ttyd0

--- JUNOS 5.6R2.4 built 2003-02-14 23:22:39 UTC

lab@c1> traceroute 2000::1 source 2010::1
traceroute6 to 2000::1 (2000::1) from 2010::1, 30 hops max, 12 byte packets
 1  2:11::5 (2:11::5)  0.461 ms  0.336 ms  0.33 ms
 2  ::10.0.3.3 (::10.0.3.3)  1.253 ms  0.691 ms  0.824 ms
```

Excellent! The successful traceroute between the IPv6 prefixes owned by the T1 and C1 peers, combined with the output gleaned from previous verification steps, confirms that you have configured IPv6-over-IPv4 tunnels in accordance with the restrictions posed in this example. It bears mentioning that you will not be able to monitor transit traffic at `r3` or `r4` to verify that the IPv6 traffic is actually being tunneled inside of IP-IP. This is because traffic monitoring is possible only for traffic that originates or terminates at that router's RE. You can further confirm that transit IPv6 traffic is "being tunneled" by monitoring packet counts on the IP-IP interface while generating lots of transit IPv6 traffic by adding the `rapid` switch to the pings initiated at T1 or C1.

IPv6 Tunneling Summary

The tunneling of IPv6 traffic over network elements that support IPv4 only is an important capability that is critical to the eventual migration of the public Internet from its existing IPv4 infrastructure to one that is based on IPv6. Juniper Networks M-series and T-series routing platforms support IPv6-over-IPv4 tunnels when equipped with a TS PIC. This section demonstrated the configuration of an IPv6 numbered IP-IP tunnel that originated and terminated on IPv4-based loopback addresses. While static routing is a possibility (depending on the restrictions posed in your particular scenario), this section demonstrated how an IPv6-capable IGP can be enabled over the IPv4-based IP-IP tunnel to facilitate the exchange of routes, including the remote router's IPv4-compatible IPv6 address, which was needed for BGP next hop resolution in this example.

Summary

This chapter provided various examples of JNCIE-level IPv6 configuration scenarios. The chapter demonstrated manual and EUI-64 based IPv6 address assignment and described how link-local addresses are automatically created for IPv6-enabled interfaces. The chapter went on to demonstrate RIPng, OSPF3, and IPv6-related routing policy in the context of mutual IPv6 route redistribution; examples of operational mode commands that help to validate the operation of RIPng or OSPF3 were also shown and described. The BGP section detailed support for native

IPv6 peering, and the use of an IPv4-based BGP peering session that supported the advertisement of both IPv6 and IPv4 NLRI. In the latter case, the need for an IPv4-compatible IPv6 address that matches the IPv4 peering address for next hop resolution of the IPv6 NLRI was discussed and demonstrated.

The chapter body concluded with an IPv6-over-IPv4 tunnel example in which two islands of IPv6 connectivity were interconnected across an IPv4-only cloud using an IP-IP tunnel. In this case, the use of a next hop self policy and an IPv4-based IBGP peering session that supported IPv6 NLRI resulted in BGP next hops that were set to the advertising peer's IPv4-compatible address. Resolution of the IPv4-compatible BGP next hops was made possible by running an IPv6 IGP across the IPv4-based IP-IP tunnel.

In the end, dealing with IPv6 is not really that difficult, at least not once you get over the initial pain of dealing with its lengthy addressing. Although not shown in the chapter body, IPv6-based firewall filters are also supported in JUNOS software. You configure an IPv6-based firewall filter using techniques and approaches that are very similar to those detailed in Chapter 3. While IPv6 currently plays a subordinate role to its older sibling, indications are that the future of the Internet belongs to IPv6. As IPv6 deployments continue to grow, the well-prepared JNCIE candidate will dedicate commensurately more time to mastering the subject to ensure that they are ready to deploy IPv6 when called upon to do so.

Case Study: IPv6

The chapter case study is designed to simulate a JNCIE-level IPv6 configuration scenario. To keep things interesting, you will be adding your IPv6 configuration to the IS-IS baseline configuration that was discovered and documented in the Chapter 1 case study. The IS-IS baseline topology is once again shown in Figure 5.10 so you can reacquaint yourself with it.

It is expected that a prepared JNCIE candidate will be able to complete this case study in approximately one hour, with the resulting network meeting the majority of the specified behaviors and operational characteristics.

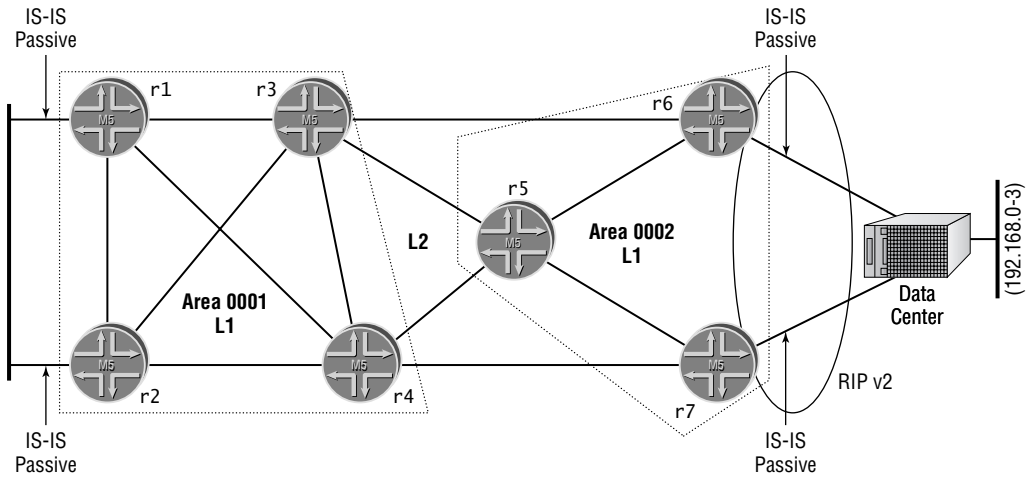
Listings that identify the changes made to the baseline configurations of all five routers in the IPv6 test bed are provided at the end of the case study for comparison with your own configurations. To accommodate differing configuration approaches, various operational mode commands are included in the case study analysis to permit the comparison of your network to that of a known good example.

To complete the case study, your network must be configured to meet these criteria:

- A single IPv6 static (or generated) route is permitted on both `r3` and `r4`.
- Add your IPv6 configuration to the IS-IS baseline network.
- All links and all loopback address must be reachable.
- You must use IS-IS as your IPv6 IGP.
- Establish the IPv6 EBGP peering sessions shown and advertise IPv6 reachability for the IPv6 routes owned by your AS.

- EBGP links are limited to a single BGP session.
- You must use native IPv6 peering for IBGP sessions.
- IPv6 pings and traceroutes from external peers to internal IPv6 destinations must be supported.
- Ensure that you provide transit IPv6 services to the T1, P1, and C1 peers. Note that all routes advertised by the T1 peer are filtered from the P1 router at `r1` and `r2`.
- Your IPv6-related configuration can not adversely impact your existing IPv4 infrastructure.
- No single link or router failure can isolate `r1` and `r2`.

FIGURE 5.10 IS-IS discovery findings



Notes:

Multi-level IS-IS, Areas 0001 and 0002 with ISO NET based on router number.

lo0 address of `r3` and `r4` not injected into Area 0001 to ensure optimal forwarding between 10.0.3.3 and 10.0.3.4.

Passive setting on `r5`'s core interfaces for optimal Area 0002-to-core routing.

No authentication or route summarization. Routing policy at `r5` to leak L1 externals (DC routes) to L2.

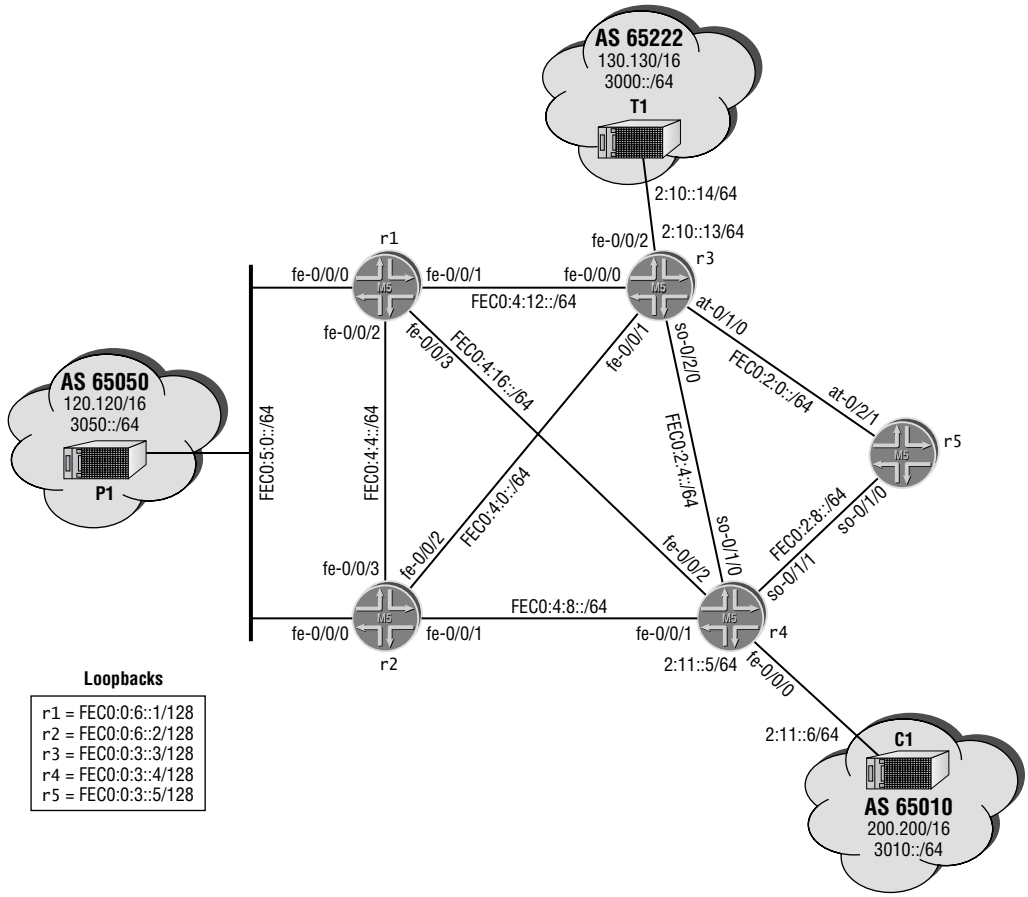
Redistribution of static default route to data center from both `r6` and `r7`. Redistribution of 192.168.0/24 through 192.168.3/24 routes from RIP into IS-IS by both `r6` and `r7`.

All adjacencies are up, reachability problem discovered at `r1` and `r2` caused by local aggregate definition. Corrected through IBGP policy to effect 10.0/16 route advertisement from `r3` and `r4` to `r1` and `r2`; removed local aggregate from `r1` and `r2`.

Suboptimal routing detected at the data center and at `r1`/`r2` for some locations. This is the result of random nexthop choice for data center's default, and the result of `r1` and `r2`'s preference for `r3`'s RID over `r4` with regard to the 10.0/16 route. This is considered normal behavior, so no corrective actions are taken.

You can assume that the T1, C1, and P1 routers are correctly configured, and that you are not permitted to modify or view their configurations. You may telnet to these routers to perform connectivity testing as needed. Refer to Figure 5.11 for IPv6 addressing and topology specifics.

FIGURE 5.11 IPv6 case study



IPv6 Case Study Analysis

To save space, initial verification of the baseline network is not performed here. The IS-IS based baseline network is assumed to be operational at this point. Each configuration requirement for the case study is matched to one or more valid router configurations and, as appropriate, examples of operational mode commands and sample output that serve to confirm that the network's operation adheres to the specified behavior.

The IPv6 case study analysis begins with these criteria because they serve to establish baseline IPv6 functionality within your network:

- A single IPv6 static (or generated) route is permitted on both `r3` and `r4`.
- Add your IPv6 configuration to the IS-IS baseline network.
- All link and loopback addresses must be reachable.

- You must use IS-IS as your IPv6 IGP.
- No single link or router failure can isolate r1 and r2.

Because the case study is performed on the IS-IS baseline network discovered in Chapter 1, and because the JUNOS software IS-IS implementation automatically advertises IPv6 prefixes that are associated with IS-IS interfaces, no explicit IS-IS configuration is necessary to support IPv6. However, you need to define and redistribute IPv6 static default routes into the Level 1 area from the attached routers (r3 and r4) to meet the connectivity requirements posed. A static (or generated) IPv6 default route is necessary because the JUNOS software version deployed in the IPv6 test bed does not automatically install an IPv6 default route based on the setting of the attached bit. Your restrictions prevent the addition of static routes to r1 and r2, so you need to define the default route on r3 and r4 and use policy to inject the route into the Level 1 area. While static/generated route could be defined locally on r1 and r2, the case study restrictions do not permit this approach.

Your configuration begins with assignment of the IPv6 addressing shown earlier in Figure 5.11. In this example, EUI-64 addressing is configured, although manual assignment of the interface identifier portion is permissible, given the restrictions in place. The initial changes made to r3 are shown next with highlights added:

```
[edit]
lab@r3# show interfaces fe-0/0/0
unit 0 {
    family inet {
        address 10.0.4.13/30;
    }
    family iso;
    family inet6 {
        address fec0:4:12:0::/64 {
            eui-64;
        }
    }
}

[edit]
lab@r3# show interfaces fe-0/0/1
unit 0 {
    family inet {
        address 10.0.4.1/30;
    }
    family iso;
    family inet6 {
        address fec0:4:0:0::/64 {
            eui-64;
```

```
        }
    }
}

[edit]
lab@r3# show interfaces at-0/1/0
atm-options {
    vpi 0 {
        maximum-vcs 64;
    }
}
unit 0 {
    point-to-point;
    vci 50;
    family inet {
        address 10.0.2.2/30;
    }
    family iso;
    family inet6 {
        address fec0:2:0:0::/64 {
            eui-64;
        }
    }
}

[edit]
lab@r3# show interfaces so-0/2/0
dce;
encapsulation frame-relay;
unit 100 {
    dlci 100;
    family inet {
        address 10.0.2.5/30;
    }
    family iso;
    family inet6 {
        address fec0:2:4:0::/64 {
            eui-64;
        }
    }
}
```

```
[edit]
lab@r3# show interfaces lo0
unit 0 {
    family inet {
        address 10.0.3.3/32;
    }
    family iso {
        address 49.0001.3333.3333.3333.00;
    }
    family inet6 {
        address fec0:0:3::3/128;
    }
}
```

Note that no changes are required to the protocols stanza in the IS-IS baseline configuration to support IPv6 routing. In fact, explicit configuration of IS-IS is required only when you do *not* desire support for IPv4 and IPv6; in such cases you use the `no-ipv6-routing` or `no-ipv4-routing` keyword at the `[edit protocols isis]` hierarchy. The IPv6 addressing changes required in the remaining routers are similar to those shown for `r3`.

With IPv6 addressing now configured on the internal interfaces of all routers, you move on to the "full connectivity with optimal paths" requirement by configuring an IPv6 static default route on your L1/L2-attached routers. You also adjust/create the necessary IS-IS policy to ensure that the default route is advertised to Level 1 routers `r1` and `r2`. Because `r1` and `r2` will see two equal-cost paths for the default route, there is a potential for inefficient forwarding out of the Level 1 area; this is the expected behavior and is therefore considered normal. The changes made at `r4` to facilitate the creation and advertisement of the IPv6 default route are shown here with added highlights. Although not shown, similar changes are also made at `r3`:

```
[edit]
lab@r4# show routing-options rib inet6.0
static {
    route ::/0 reject;
}

[edit]
lab@r4# show policy-options policy-statement v6-default
term 1 {
    from {
        protocol static;
        route-filter ::0/0 exact;
    }
    to level 1;
    then accept;
}
```

```
[edit]
lab@r4# show protocols isis export
export v6-default;
```

In this example, the `to level 1` condition is added to the *`v6-default`* policy to avoid the redistribution of the IPv6 default route into the backbone (Level 2) area. Based on the specifics, this precaution is not strictly necessary, but this author considers it good form so it is included here. After the changes are committed on both `r3` and `r4`, the default route, and the resulting connectivity to backbone destinations, is confirmed within the L1 area:

```
[edit]
lab@r1# run show route table inet6

inet6.0: 19 destinations, 22 routes (19 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

::/0                *[IS-IS/160] 00:01:19, metric 10
                       to fe80::290:69ff:fe6d:9800 via fe-0/0/1.0
                     > to fe80::290:69ff:fe6b:3002 via fe-0/0/3.0
fe80::/64           *[Direct/0] 02:30:23
                     > via fe-0/0/3.0
                     [Direct/0] 02:30:23
                     > via fe-0/0/2.0
                     [Direct/0] 02:30:23
                     > via fe-0/0/1.0
                     [Direct/0] 02:30:23
                     > via fe-0/0/0.0
[edit]
lab@r1# run traceroute fec0:0:3::5
traceroute6 to fec0:0:3::5 (fec0:0:3::5) from fec0:4:16:0:2a0:c9ff:fe69:a806,
   30 hops max, 12 byte packets
 1  fec0:4:16:0:290:69ff:fe6b:3002 (fec0:4:16:0:290:69ff:fe6b:3002)  13.215 ms
    0.399 ms  0.289 ms
 2  fec0:0:3::5 (fec0:0:3::5)  0.527 ms  0.424 ms  0.384 ms
```

The presence of the two equal-cost next hops for the IPv6 default route indicates that `r3` and `r4` are correctly configured to redistribute the default route into the Level 1 IS-IS area; having two next hops is necessary to meet the redundancy requirements posed for `r1` and `r2`. The successful traceroute to the IPv6 loopback address of `r5` confirms that the default route is working at `r1`. The following capture shows that `r1`'s choice of the next hops associated with the default route can produce an extra hop to some destinations (`r3`'s loopback address, in this case). This behavior is expected and is not considered an issue:

```
[edit]
lab@r1# run traceroute fec0:0:3::3
traceroute6 to fec0:0:3::3 (fec0:0:3::3) from fec0:4:16:0:2a0:c9ff:fe69:a806,
   30 hops max, 12 byte packets
```

```
 1  fec0:4:16:0:290:69ff:fe6b:3002 (fec0:4:16:0:290:69ff:fe6b:3002)  0.428 ms
    0.317 ms  0.276 ms
 2  fec0:0:3::3 (fec0:0:3::3)  0.528 ms  0.393 ms  0.372 ms
```

You now confirm that IS-IS is correctly advertising reachability for the loopback addresses of all routers in the IPv6 test bed into the backbone area:

```
[edit interfaces]
lab@r5# run show route protocol isis | match /128
fec0:0:3::3/128    *[IS-IS/18] 03:36:24, metric 10
fec0:0:3::4/128    *[IS-IS/18] 03:36:23, metric 10
fec0:0:6::1/128    *[IS-IS/18] 02:58:50, metric 20
fec0:0:6::2/128    *[IS-IS/18] 03:36:24, metric 20
```

Although not shown for the sake of brevity, you can assume that all internal IPv6 destinations have been confirmed as reachable by L1 and L1/L2 routers. With initial IPv6 connectivity confirmed operational, you move on to the EBGP peering–related case study requirements:

- Establish the IPv6 EBGP peering sessions shown and advertise IPv6 reachability for the routes owned by your AS.
- EBGP links are limited to a single BGP session.

The stipulation that you may have only one BGP session to each EBGP peer means that you need to reconfigure the existing EBGP sessions to enable support for IPv6 NLRI. The following highlights call out the changes made to `r3`'s configuration to support its EBGP peering session with the T1 peer:

```
[edit]
lab@r3# show interfaces fe-0/0/2
unit 0 {
    family inet {
        address 172.16.0.13/30;
    }
    family inet6 {
        address ::172.16.0.13/126;
    }
}

[edit]
lab@r3# show protocols bgp group ext
import ebgp-in;
family inet {
    unicast;
}
family inet6 {
    unicast;
```

```
}
export ebgp-out;
neighbor 172.16.0.14 {
    peer-as 65222;
}
```

Note that an IPv4-compatible IPv6 address that matches the IPv4 EBGP peering address has been assigned to r3's fe-0/0/2 interface. This is a critical aspect of the configuration; without this address, the 3000::/64 route advertised by T1 will be discarded (not hidden) at r3 due to the EBGP peer being "unexpectedly remote." A 126-bit mask has been configured on the IPv4-compatible address to approximate the /30 IPv4 addressing. After committing the changes, the EBGP session to T1 and the receipt of the 3000::/64 route are confirmed at r3:

```
[edit]
lab@r3# run show bgp neighbor 172.16.0.14
Peer: 172.16.0.14+4938 AS 65222 Local: 172.16.0.13+179 AS 65412
  Type: External    State: Established    Flags: <>
  Last State: OpenConfirm   Last Event: RecvKeepAlive
  Last Error: None
  Export: [ ebgp-out ] Import: [ ebgp-in ]
  Options: <Preference HoldTime AdvertiseInactive AddressFamily PeerAS Refresh>
  Address families configured: inet-unicast inet6-unicast
  Holdtime: 90 Preference: 170
  Number of flaps: 4
  Error: 'Cease' Sent: 2 Recv: 2
  Peer ID: 130.130.0.1      Local ID: 10.0.3.3         Active Holdtime: 90
  Keepalive Interval: 30
  Local Interface: fe-0/0/2.0
  NLRI advertised by peer: inet-unicast inet6-unicast
  NLRI for this session: inet-unicast inet6-unicast
  Peer supports Refresh capability (2)
  Table inet.0 Bit: 10003
    RIB State: BGP restart is complete
    Send state: in sync
    Active prefixes:            1
    Received prefixes:          1
    Suppressed due to damping:  0
  Table inet6.0 Bit: 20000
    RIB State: BGP restart is complete
    Send state: in sync
    Active prefixes:            1
    Received prefixes:          1
    Suppressed due to damping:  0
  Last traffic (seconds): Received 26   Sent 26   Checked 26
```

```
  Input messages:  Total 17     Updates 3       Refreshes 0     Octets 481
  Output messages: Total 19     Updates 4       Refreshes 0     Octets 552
  Output Queue[0]: 0
  Output Queue[1]: 0

[edit]
lab@r3# run show route receive-protocol bgp 172.16.0.14

inet.0: 28 destinations, 28 routes (28 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
* 130.130.0.0/16          172.16.0.14          0                  65222 I

iso.0: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)

inet6.0: 27 destinations, 31 routes (27 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
* 3000::/64               ::172.16.0.14        0                  65222 I
```

Similar configuration changes are needed at r1 and r2 to support the EBGP session to P1, and at r4 for its EBGP session to C1. The modified configuration for r2 is shown here with the changes highlighted:

```
[edit]
lab@r2# show interfaces fe-0/0/0
unit 0 {
    family inet {
        address 10.0.5.2/24;
    }
    family iso;
    family inet6 {
        address fec0:5:0:0::/64 {
            eui-64;
        }
        address ::10.0.5.2/120;
    }
}

lab@r2# show protocols bgp group p1
type external;
family inet {
    unicast;
}
family inet6 {
    unicast;
```

```
}
export ebgp-out;
neighbor 10.0.5.254 {
    peer-as 65050;
```

The IPv4-compatible address ::10.0.5.2 is associated with a 120-bit mask to best approximate the /24 addressing in effect on this subnet for the IPv4 protocol. The r2-to-P1 peering session and receipt of the 3050::/64 IPv6 route is confirmed at r2:

```
[edit]
lab@r2# run show bgp summary
Groups: 2 Peers: 7 Down peers: 2
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
inet.0                 7          5          0          0          0          0
inet6.0                1          1          0          0          0          0
Peer           AS  InPkt  OutPkt  OutQ  Flaps Last Up/Dwn State|#Active/
                                                    Received/Damped...
10.0.3.3    65412    648     636     0      3       17:10 Establ
  inet.0: 2/2/0
10.0.3.4    65412    721     720     0      0     5:58:37 Establ
  inet.0: 2/3/0
10.0.3.5    65412    717     720     0      0     5:58:33 Establ
  inet.0: 0/0/0
10.0.5.254  65050     10      10     0      0        1:37 Establ
  inet.0: 1/1/0
  inet6.0: 1/1/0
10.0.6.1    65412    717     720     0      1     4:20:17 Establ
  inet.0: 0/1/0
10.0.9.6    65412      0       0     0      0     6:00:01 Connect
10.0.9.7    65412      0       0     0      0     6:00:01 Connect

[edit]
lab@r2# run show route 3050::/64 detail

inet6.0: 22 destinations, 25 routes (22 active, 0 holddown, 0 hidden)
3050::/64 (1 entry, 1 announced)
        *BGP    Preference: 170/-101
                Source: 10.0.5.254
                Next hop: ::10.0.5.254 via fe-0/0/0.0, selected
                State: <Active Ext>
                Local AS: 65412 Peer AS: 65050
                Age: 1:49       Metric: 0
                Task: BGP_65050.10.0.5.254+179
```

```
Announcement bits (2): 0-KRT 3-BGP.0.0.0.0+179
AS path: 65050 I
Localpref: 100
Router ID: 120.120.0.1
```

Before proceeding, similar changes are required at `r1` and `r4` to support IPv6 NLRI advertisements over their IPv4-based EBGP peering sessions. With EBGP peering configured and confirmed, you move on to the IBGP-related aspect of the case study:

- You must use native IPv6 peering for IBGP sessions.

This requirement's wording indicates that you must establish native IPv6 IBGP peering sessions that mirror the existing IPv4-based IBGP topology. You must use care to ensure that your changes do not delete or disable the existing IPv4 IBGP sessions because the test bed's IPv4 operation can not be impacted by your IPv6 configuration changes. The changes shown next for `r5` add the new IPv6 loopback address–based peering sessions; similar changes are needed on the remaining routers in the IPv6 test bed.

```
[edit]
lab@r5# show protocols bgp group int-v6
group int-v6 {
    type internal;
    local-address fec0:0:3:0::5;
    neighbor fec0:0:3:0::3;
    neighbor fec0:0:3:0::4;
    neighbor fec0:0:6:0::1;
    neighbor fec0:0:6:0::2;
}
```

Note that `r6` and `r7` are not reflected in the *`int-v6`* peer group because they are not in play in the current IPv6 test bed. This author feels that the use of a text editor and `load merge terminal` operations are well suited to the task of adding this (more or less) common functionality to the remaining routers. When all routers in the test bed have been modified, the IBGP session status is confirmed at `r4`:

```
[edit]
lab@r4# run show bgp summary
Groups: 5 Peers: 11 Down peers: 2
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
inet.0            105855     105853          0          0          0          0
inet6.0                4          3          0          0          0          0
Peer            AS   InPkt  OutPkt  OutQ  Flaps Last Up/Dwn State|#Active/
                                                      Received/Damped...
172.16.0.6   65010   74141   73511     0      0       24:04 Establ
  inet.0: 2/3/0
  inet6.0: 1/1/0
```

```
fec0:0:6::1  65412       19       21     0      0        8:31 Establ
  inet6.0: 1/1/0
fec0:0:3::5  65412       17       19     0      0        8:23 Establ
  inet6.0: 0/0/0
fec0:0:3::3  65412       18       19     0      0        8:26 Establ
  inet6.0: 1/1/0
fec0:0:6::2  65412       17       19     0      0        8:19 Establ
  inet6.0: 0/1/0
10.0.3.3     65412   121852      732     0      3     1:03:38 Establ
  inet.0: 105850/105850/0
10.0.3.5     65412      810      817     0      0     6:44:52 Establ
  inet.0: 0/0/0
10.0.6.1     65412      617      623     0      1     5:06:58 Establ
  inet.0: 1/1/0
10.0.6.2     65412      812      818     0      0     6:45:01 Establ
  inet.0: 0/1/0
10.0.9.6     65412        0        0     0      0     6:46:53 Active
10.0.9.7     65412        0        0     0      0     6:46:53 Active
```

As expected, all IBGP sessions are in the established state (excepting those relating to the absent `r6` and `r7`). The highlights call out that there are now two IBGP sessions between `r3` and `r5`—one that supports IPv4 and another for IPv6. Sensing a light at the end of this case study, you find yourself addressing the remaining stipulations:

- IPv6 pings and traceroutes from external peers to internal IPv6 destinations must be supported.
- Ensure that you provide transit IPv6 services to the T1, P1, and C1 peers. Note that all routes advertised by the T1 peer are filtered from the P1 router at `r1` and `r2`.
- Your IPv6-related configuration can not adversely impact your existing IPv4 infrastructure.

The remaining stipulations are somewhat catchall in nature; they are designed to validate the overall operational aspects of your IPv6 configuration. To achieve the prescribed behavior, you must define IPv6 aggregate routes and adjust your BGP-related policies to correctly set the BGP next hop for routes learned from EBGP peers and to advertise your aggregate routes to your EBGP peers. For example, the current state of the network results in a black hole at `r4` for the 3000::/64 route, which stems from the lack of an appropriate next hop self policy at `r3`:

```
[edit]
lab@r4# run show route 3000::/64 detail

inet6.0: 30 destinations, 35 routes (30 active, 0 holddown, 0 hidden)
3000::/64 (1 entry, 1 announced)
        *BGP    Preference: 170/-101
                Source: fec0:0:3::3
                Next hop type: Reject
```

```
                Protocol next hop: ::172.16.0.14 Indirect next hop: 84d0bd0 80
                State: <Active Int Ext>
                Local AS: 65412 Peer AS: 65412
                Age: 7:41       Metric: 0       Metric2: 0
                Task: BGP_65412.fec0:0:3::3+179
                Announcement bits (3): 0-KRT 1-BGP.0.0.0.0+179 2-Resolve inet6.0
                AS path: 65222 I
                Communities: 65412:420
                Localpref: 100
                Router ID: 10.0.3.3

[edit]
lab@r4# run show route ::172.16.0.14

inet6.0: 30 destinations, 35 routes (30 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

::/0               *[Static/5] 02:28:08, metric 0
                      Reject
```

You do not need a next hop self policy at `r1` and `r2` due to the passive IS-IS instance they run on their fe-0/0/0 interfaces (as with their IPv4-based peering). The following highlights call out the changes made to `r3`'s configuration to correctly set the BGP next hop and to advertise IPv6 aggregates for your AS to its EBGP peer, and to `r1` and `r2`:

```
[edit]
lab@r3# show routing-options rib inet6.0
static {
    route ::0/0 reject;
}
aggregate {
    route fec0::/16;
    route ::10.0.0.0/112;
}

[edit]
lab@r3# show protocols bgp group int-v6
type internal;
local-address fec0:0:3:0::3;
export nhs;
neighbor fec0:0:3:0::5;
neighbor fec0:0:3:0::4;
neighbor fec0:0:6:0::1 {
```

```
    export r1-v6;
}
neighbor fec0:0:6:0::2 {
    export r2-v6;
}

[edit]
lab@r3# show policy-options policy-statement r1-v6
term 1 {
    from {
        protocol aggregate;
        route-filter ::10.0.0.0/112 exact accept;
        route-filter fec0::/16 exact;
    }
    then {
        next-hop fec0:4:12:0:290:69ff:fe6d:9800;
        accept;
    }
}
term 2 {
    from {
        protocol bgp;
        neighbor 172.16.0.14;
    }
    then {
        next-hop self;
    }
}

[edit]
lab@r3# show policy-options policy-statement r2-v6
term 1 {
    from {
        protocol aggregate;
        route-filter ::10.0.0.0/112 exact accept;
        route-filter fec0::/16 exact;
    }
    then {
        next-hop fec0:4::290:69ff:fe6d:9801;
        accept;
```

```
    }
}
term 2 {
    from {
        protocol bgp;
        neighbor 172.16.0.14;
    }
    then {
        next-hop self;
    }
}
```

Note that the *r1-v6* and *r2-v6* export policies at r3 function to advertise the IPv6 aggregate routes to r1 and r2, so that they can in turn re-advertise the aggregate routes to the P1 router. It is worth focusing some attention on the way the BGP next hops are being set on the aggregate routes; the ::10.0.0.0/112 aggregate gets no special treatment, and therefore ends up with a BGP next hop of FEC0:0:3:0::3 at r1:

```
[edit]
lab@r1# run show route protocol bgp ::10.0.0.0/112 detail

inet6.0: 26 destinations, 30 routes (26 active, 0 holddown, 0 hidden)
::10.0.0.0/112 (1 entry, 1 announced)
        *BGP    Preference: 170/-101
                Source: fec0:0:3::3
                Next hop: fec0:4:12:0:290:69ff:fe6d:9800 via fe-0/0/1.0, selected
                Protocol next hop: fec0:0:3::3 Indirect next hop: b73f150 36
                State: <Active Int Ext>
                Local AS: 65412 Peer AS: 65412
                Age: 9:35       Metric2: 0
                Task: BGP_65412.fec0:0:3::3+2460
                Announcement bits (3): 0-KRT 3-BGP.0.0.0.0+179 4-Resolve inet6.0
                AS path: I Aggregator: 65412 10.0.3.3
                Localpref: 100
                Router ID: 10.0.3.3
```

While the default next hop self behavior (as applied to locally originated routes that are redistributed into BGP) works fine for the ::10.0.0.0/112 aggregate, setting the FEC0::/64 route to a next hop of FEC0:0:3:0::3 results in the route being hidden due to the recursion loop that forms when r1 tries to resolve the FEC0:0:3:0::3 next hop through its longest matching route, which, if not hidden, would be the FEC0::/64 route. And this is the very route that needs to resolve the BGP next hop in the first place! Put differently, JUNOS software does not allow a more specific route to resolve through a less specific route.

The approach taken in this example is similar to the existing IPv4 BGP policy; namely, the FEC0::/16 aggregate has its next hop set to the site-local address assigned to r3's fe-0/0/0 interface. This next hop setting allows the route to be made active because r1 has no problem resolving this address within its Level 1 area. In contrast, the *r2-v6* policy correctly sets the BGP next hop based on the site-local address assigned to r3's fe-0/0/1 interface, as shown here:

```
[edit]
lab@r3# run show interfaces fe-0/0/1 terse
Interface               Admin Link Proto Local                 Remote
fe-0/0/1                up    up
fe-0/0/1.0              up    up   inet  10.0.4.1/30
                                  iso
                                  inet6 fe80::290:69ff:fe6d:9801/64
                                        fec0:4::290:69ff:fe6d:9801/64
```

The *nhs* policy itself is unchanged from the IS-IS baseline configuration. The key point here is the reapplication of the *nhs* policy as an export policy to the new *int-vg* peer group. Similar changes are needed at r4 (not shown). The changes needed at r4 include the aggregate route definitions and the policy adjustments needed to make sure they are correctly advertised to r1, r2, and C1. Having the aggregate routes advertised to r1 and r2 from both r3 and r4 is necessary to meet the stated redundancy requirements. Do not forget to address any next hop self issues with the new IPv6 IBGP peer group also. No further modifications are needed at r1 and r2 to comply with the remaining case study requirements.

Real World Scenario

Is the IS-IS Default Route Still Needed?

The changes recently made to the BGP policy on r3 and r4 result in the advertisement of IPv6 aggregate routes with next hops that are resolvable within their IS-IS Level 1 area. This might cause you to feel that the redistribution of an IPv6 default route into their Level 1 area is no longer necessary. After all, at this stage you are seeing output that indicates the IS-IS based default route is no longer even being used:

```
[edit]
lab@r1# run show route fec0:0:3:0::3

inet6.0: 26 destinations, 32 routes (26 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

fec0::/16          *[BGP/170] 00:01:31, localpref 100, from fec0:0:3::3
                      AS path: I
                    > to fec0:4:12:0:290:69ff:fe6d:9800 via fe-0/0/1.0
                    [BGP/170] 00:00:31, localpref 100, from fec0:0:3::4
                      AS path: I
                    > to fec0:4:16:0:290:69ff:fe6b:3002 via fe-0/0/3.0
```

Despite the output, the redistributed IPv6 default route is, in fact, still crucial to the operation of your IPv6 infrastructure. Even though the default route does not seem to be used once the aggregate routes are received through IBGP, there is somewhat of a "chicken and egg" situation at play. You see, without the IS-IS based default route, r1 and r2 can never establish their IBGP sessions to the loopback addresses of r3 and r4. Note that without these IBGP sessions, r1 and r2 never learn the FEC0::/16 aggregate route that ends up overshadowing the less specific 0::/0 IPv6 default route! The fancy BGP policy that strategically sets the next hop on the FEC0::/16 aggregate route so that it is resolvable within the Level 1 IS-IS area never comes into play if the IBGP session to which the policy is applied can never be established!

To verify the remaining case study requirements, a few spot checks are performed. You start at the P1 router, which should receive two BGP routes for your IPv6 aggregates and for the 3010 route owned by C1:

```
lab@P1> show route protocol bgp ::10.0.0.0/112

inet6.0: 12 destinations, 17 routes (12 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

::10.0.0.0/112     *[BGP/170] 01:19:48, localpref 100, from 10.0.5.1
                      AS path: 65412 I
                    > to ::10.0.5.1 via fe-0/0/0.0
                    [BGP/170] 01:17:25, localpref 100, from 10.0.5.2
                      AS path: 65412 I
                    > to ::10.0.5.2 via fe-0/0/0.0

lab@P1> show route protocol bgp fec0::/16

inet6.0: 12 destinations, 17 routes (12 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

fec0::/16          *[BGP/170] 01:19:51, localpref 100, from 10.0.5.1
                      AS path: 65412 I
                    > to ::10.0.5.1 via fe-0/0/0.0
                    [BGP/170] 01:17:28, localpref 100, from 10.0.5.2
                      AS path: 65412 I
                    > to ::10.0.5.2 via fe-0/0/0.0

lab@P1> show route protocol bgp 3010::/64
```

```
inet6.0: 12 destinations, 17 routes (12 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

3010::/64          *[BGP/170] 00:38:44, localpref 100, from 10.0.5.1
                      AS path: 65412 65010 I
                    > to ::10.0.5.1 via fe-0/0/0.0
                    [BGP/170] 00:38:52, localpref 100, from 10.0.5.2
                      AS path: 65412 65010 I
                    > to ::10.0.5.2 via fe-0/0/0.0
```

The output confirms that the expected routes are present at the P1 router, and the dual BGP advertisements serve to confirm that both `r1` and `r2` are advertising the routes to P1. Some quick connectivity tests are performed:

```
lab@P1> ping 10.0.3.5 count 1
PING 10.0.3.5 (10.0.3.5): 56 data bytes
64 bytes from 10.0.3.5: icmp_seq=0 ttl=253 time=0.831 ms

--- 10.0.3.5 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.831/0.831/0.831/0.000 ms
```

IPv4 internal destinations are reachable; what about IPv6?

```
lab@P1> ping fec0:0:3:0::5 count 1
PING6(56=40+8+8 bytes) ::10.0.5.254 --> fec0:0:3::5
16 bytes from fec0:0:3::5, icmp_seq=0 hlim=62 time=1.224 ms

--- fec0:0:3:0::5 ping6 statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 1.224/1.224/1.224 ms
```

Good, P1 can reach internal IPv6 destinations also. You next test connectivity to customer C1 IPv4 and IPv6 destinations:

```
lab@P1> ping 200.200.1.1 count 1
PING 200.200.1.1 (200.200.1.1): 56 data bytes
64 bytes from 200.200.1.1: icmp_seq=0 ttl=253 time=0.418 ms

--- 200.200.1.1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.418/0.418/0.418/0.000 ms

lab@P1> ping 3010::1 count 1
PING6(56=40+8+8 bytes) ::10.0.5.254 --> 3010::1
16 bytes from 3010::1, icmp_seq=0 hlim=62 time=0.415 ms
```

```
--- 3010::1 ping6 statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.415/0.415/0.415 ms
```

Excellent! You have demonstrated IPv4 and IPv6 connectivity from the P1 router in accordance with the remaining case study objectives. Note that P1 does not receive T1's routes, so connectivity is not tested to transit peer locations. You can assume that traceroute testing from P1 to internal and external destinations also returns the expected results. To keep this book interesting, the output of traceroute tests performed at the T1 peer is shown next:

```
lab@T1> traceroute 10.0.3.4 source 130.130.0.1
traceroute to 10.0.3.4 (10.0.3.4) from 130.130.0.1, 30 hops max, 40 byte packets
 1  172.16.0.13 (172.16.0.13)  0.412 ms  0.287 ms  0.274 ms
 2  10.0.3.4 (10.0.3.4)  0.503 ms  0.429 ms  0.418 ms

lab@T1> traceroute 200.200.1.1 source 130.130.0.1
traceroute to 200.200.1.1 (200.200.1.1) from 130.130.0.1, 30 hops max,
   40 byte packets
 1  172.16.0.13 (172.16.0.13)  0.412 ms  0.287 ms  0.274 ms
 2  10.0.2.6 (10.0.2.6)  0.337 ms  0.295 ms  0.291 ms
 3  200.200.1.1 (200.200.1.1)  0.265 ms  0.225 ms  0.222 ms
```

Internal and external IPv4 prefixes are reachable. You move on to test IPv6 connectivity and forwarding paths:

```
lab@T1> traceroute fec0:0:3:0::5 source 3000::1
traceroute6 to fec0:0:3:0::5 (fec0:0:3::5) from 3000::1, 30 hops max,
   12 byte packets
 1  ::172.16.0.13 (::172.16.0.13)  0.478 ms  0.33 ms  0.324 ms
 2  fec0:0:3::5 (fec0:0:3::5)  0.985 ms  0.812 ms  0.846 ms

lab@T1> traceroute 3010::1 source 3000::1
traceroute6 to 3010::1 (3010::1) from 3000::1, 30 hops max, 12 byte packets
 1  ::172.16.0.13 (::172.16.0.13)  0.436 ms  0.331 ms  0.323 ms
 2  ::172.16.0.5 (::172.16.0.5)  0.615 ms  0.484 ms  0.484 ms
 3  3010::1 (3010::1)  0.29 ms  0.249 ms  0.236 ms
```

The results confirm IPv6 forwarding and connectivity from T1 to internal and customer locations.

Congratulations! The operational mode output and IPv6 connectivity demonstrated during the case study indicate that you have deployed an IPv6 configuration that meets all of the requirements and stipulations posed.

IPv6 Case Study Configurations

The changes made to the IS-IS baseline network topology to support the IPv6 case study are listed below in Listings 5.1 through 5.5 for all routers in the test bed with highlights added as needed to call out changes to existing configuration stanzas.

Listing 5.1: IPv6 Case Study Configuration for *r1*

```
[edit]
lab@r1# show interfaces fe-0/0/0
unit 0 {
    family inet {
        address 10.0.5.1/24;
    }
    family iso;
    family inet6 {
        address fec0:5:0:0::/64 {
            eui-64;
        }
        address ::10.0.5.1/120;
    }
}

[edit]
lab@r1# show interfaces fe-0/0/1
unit 0 {
    family inet {
        address 10.0.4.14/30;
    }
    family iso;
    family inet6 {
        address fec0:4:12:0::/64 {
            eui-64;
        }
    }
}

[edit]
lab@r1# show interfaces fe-0/0/2
unit 0 {
    family inet {
        address 10.0.4.5/30;
    }
    family iso;
    family inet6 {
        address fec0:4:4:0::/64 {
            eui-64;
        }
```

```
    }
}

[edit]
lab@r1# show interfaces fe-0/0/3
unit 0 {
    family inet {
        address 10.0.4.18/30;
    }
    family iso;
    family inet6 {
        address fec0:4:16:0::/64 {
            eui-64;
        }
    }
}

[edit]
lab@r1# show interfaces lo0
unit 0 {
    family inet {
        address 10.0.6.1/32;
    }
    family iso {
        address 49.0001.1111.1111.1111.00;
    }
    family inet6 {
        address fec0:0:6:0::1/128;
    }
}

[edit]
lab@r1# show protocols bgp
group int {
    type internal;
    local-address 10.0.6.1;
    neighbor 10.0.6.2;
    neighbor 10.0.3.3;
    neighbor 10.0.3.4;
    neighbor 10.0.3.5;
    neighbor 10.0.9.6;
    neighbor 10.0.9.7;
```

```
}
group p1 {
    type external;
    family inet {
        unicast;
    }
    family inet6 {
        unicast;
    }
    export ebgp-out;
    neighbor 10.0.5.254 {
        peer-as 65050;
    }
}
group int-v6 {
    type internal;
    local-address fec0:0:6:0::1;
    neighbor fec0:0:3:0::5;
    neighbor fec0:0:3:0::4;
    neighbor fec0:0:3:0::3;
    neighbor fec0:0:6:0::2;
}
```

Listing 5.2: IPv6 Case Study Configuration for *r2*

```
[edit]
lab@r2# show interfaces fe-0/0/0
unit 0 {
    family inet {
        address 10.0.5.2/24;
    }
    family iso;
    family inet6 {
        address fec0:5:0:0::/64 {
            eui-64;
        }
        address ::10.0.5.2/120;
    }
}

[edit]
lab@r2# show interfaces fe-0/0/1
```

```
unit 0 {
    family inet {
        address 10.0.4.10/30;
    }
    family iso;
    family inet6 {
        address fec0:4:8:0::/64 {
            eui-64;
        }
    }
}

[edit]
lab@r2# show interfaces fe-0/0/2
speed 100m;
unit 0 {
    family inet {
        address 10.0.4.2/30;
    }
    family iso;
    family inet6 {
        address fec0:4:0:0::/64 {
            eui-64;
        }
    }
}

[edit]
lab@r2# show interfaces fe-0/0/3
unit 0 {
    family inet {
        address 10.0.4.6/30;
    }
    family iso;
    family inet6 {
        address fec0:4:4:0::/64 {
            eui-64;
        }
    }
}
```

```
[edit]
lab@r2# show protocols bgp
group int {
    type internal;
    local-address 10.0.6.2;
    neighbor 10.0.6.1;
    neighbor 10.0.3.3;
    neighbor 10.0.3.4;
    neighbor 10.0.3.5;
    neighbor 10.0.9.6;
    neighbor 10.0.9.7;
}
group p1 {
    type external;
    family inet {
        unicast;
    }
    family inet6 {
        unicast;
    }
    export ebgp-out;
    neighbor 10.0.5.254 {
        peer-as 65050;
    }
}
group int-v6 {
    type internal;
    local-address fec0:0:6:0::2;
    neighbor fec0:0:3:0::5;
    neighbor fec0:0:3:0::4;
    neighbor fec0:0:3:0::3;
    neighbor fec0:0:6:0::1;
}
```

Listing 5.3: IPv6 Case Study Configuration for *r3*

```
[edit]
lab@r3# show interfaces fe-0/0/0
unit 0 {
    family inet {
        address 10.0.4.13/30;
    }
    family iso;
```

```
    family inet6 {
        address fec0:4:12:0::/64 {
            eui-64;
        }
    }
}

[edit]
lab@r3# show interfaces fe-0/0/1
unit 0 {
    family inet {
        address 10.0.4.1/30;
    }
    family iso;
    family inet6 {
        address fec0:4:0:0::/64 {
            eui-64;
        }
    }
}

[edit]
lab@r3# show interfaces fe-0/0/2
unit 0 {
    family inet {
        address 172.16.0.13/30;
    }
    family inet6 {
        address ::172.16.0.13/126;
    }
}

[edit]
lab@r3# show interfaces at-0/1/0
atm-options {
    vpi 0 {
        maximum-vcs 64;
    }
}
unit 0 {
    point-to-point;
```

```
    vci 50;
    family inet {
        address 10.0.2.2/30;
    }
    family iso;
    family inet6 {
        address fec0:2:0:0::/64 {
            eui-64;
        }
    }
}

[edit]
lab@r3# show interfaces so-0/2/0
dce;
encapsulation frame-relay;
unit 100 {
    dlci 100;
    family inet {
        address 10.0.2.5/30;
    }
    family iso;
    family inet6 {
        address fec0:2:4:0::/64 {
            eui-64;
        }
    }
}

[edit]
lab@r3# show interfaces lo0
unit 0 {
    family inet {
        address 10.0.3.3/32;
    }
    family iso {
        address 49.0001.3333.3333.3333.00;
    }
    family inet6 {
        address fec0:0:3::3/128;
    }
}
```

```
[edit]
lab@r3# show routing-options
rib inet6.0 {
    static {
        route ::0/0 reject;
    }
    aggregate {
        route fec0::/16;
        route ::10.0.0.0/112;
    }
}
static {
    route 10.0.200.0/24 {
        next-hop 10.0.1.102;
        no-readvertise;
    }
}
aggregate {
    route 10.0.0.0/16;
}
autonomous-system 65412;

[edit]
lab@r3# show protocols bgp
advertise-inactive;
group int {
    type internal;
    local-address 10.0.3.3;
    export nhs;
    neighbor 10.0.6.1 {
        export r1;
    }
    neighbor 10.0.6.2 {
        export r2;
    }
    neighbor 10.0.3.4;
    neighbor 10.0.3.5;
    neighbor 10.0.9.6;
    neighbor 10.0.9.7;
}
```

```
group ext {
    import ebgp-in;
    family inet {
        unicast;
    }
    family inet6 {
        unicast;
    }
    export ebgp-out;
    neighbor 172.16.0.14 {
        peer-as 65222;
    }
}
group int-v6 {
    type internal;
    local-address fec0:0:3:0::3;
    export nhs;
    neighbor fec0:0:3:0::5;
    neighbor fec0:0:3:0::4;
    neighbor fec0:0:6:0::1 {
        export r1-v6;
    }
    neighbor fec0:0:6:0::2 {
        export r2-v6;
    }
}

[edit]
lab@r3# show protocols isis export
export v6-default;

[edit]
lab@r3# show policy-options policy-statement ebgp-out
term 1 {
    from {
        protocol aggregate;
        route-filter 10.0.0.0/16 exact;
    }
    then accept;
}
term 2 {
```

```
    from {
        protocol aggregate;
        route-filter fec0::/16 exact;
        route-filter ::10.0.0.0/112 exact;
    }
    then accept;
}

[edit]
lab@r3# show policy-options policy-statement v6-default
term 1 {
    from {
        protocol static;
        route-filter ::0/0 exact;
    }
    to level 1;
    then accept;
}

[edit]
lab@r3# show policy-options policy-statement r1-v6
term 1 {
    from {
        protocol aggregate;
        route-filter ::10.0.0.0/112 exact accept;
        route-filter fec0::/16 exact;
    }
    then {
        next-hop fec0:4:12:0:290:69ff:fe6d:9800;
        accept;
    }
}
term 2 {
    from {
        protocol bgp;
        neighbor 172.16.0.14;
    }
    then {
        next-hop self;
    }
}
```

```
[edit]
lab@r3# show policy-options policy-statement r2-v6
term 1 {
    from {
        protocol aggregate;
        route-filter ::10.0.0.0/112 exact accept;
        route-filter fec0::/16 exact;
    }
    then {
        next-hop fec0:4::290:69ff:fe6d:9801;
        accept;
    }
}
term 2 {
    from {
        protocol bgp;
        neighbor 172.16.0.14;
    }
    then {
        next-hop self;
    }
}
```

Listing 5.4: IPv6 Case Study Configuration for *r4*

```
[edit]
lab@r4# show interfaces fe-0/0/0
unit 0 {
    family inet {
        address 172.16.0.5/30;
    }
    family inet6 {
        address ::172.16.0.5/126;
    }
}

[edit]
lab@r4# show interfaces fe-0/0/1
unit 0 {
    family inet {
        address 10.0.4.9/30;
    }
```

```
    family iso;
    family inet6 {
        address fec0:4:8:0::/64 {
            eui-64;
        }
    }
}

[edit]
lab@r4# show interfaces fe-0/0/2
unit 0 {
    family inet {
        address 10.0.4.17/30;
    }
    family iso;
    family inet6 {
        address fec0:4:16:0::/64 {
            eui-64;
        }
    }
}

[edit]
lab@r4# show interfaces so-0/1/0
encapsulation frame-relay;
unit 100 {
    dlci 100;
    family inet {
        address 10.0.2.6/30;
    }
    family iso;
    family inet6 {
        address fec0:2:4:0::/64 {
            eui-64;
        }
    }
}

[edit]
lab@r4# show interfaces so-0/1/1
```

```
encapsulation ppp;
unit 0 {
    family inet {
        address 10.0.2.10/30;
    }
    family iso;
    family inet6 {
        address fec0:2:8:0::/64 {
            eui-64;
        }
    }
}

[edit]
lab@r4# show interfaces lo0
unit 0 {
    family inet {
        address 10.0.3.4/32;
    }
    family iso {
        address 49.0001.4444.4444.4444.00;
    }
    family inet6 {
        address fec0:0:3::4/128;
    }
}

[edit]
lab@r4# show routing-options
rib inet6.0 {
    static {
        route ::/0 reject;
    }
    aggregate {
        route ::10.0.0.0/112;
        route fec0::/16;
    }
}
static {
    route 10.0.200.0/24 {
```

```
        next-hop 10.0.1.102;
        no-readvertise;
    }
}
aggregate {
    route 10.0.0.0/16;
}
autonomous-system 65412;

[edit]
lab@r4# show protocols bgp
advertise-inactive;
group int {
    type internal;
    local-address 10.0.3.4;
    export nhs;
    neighbor 10.0.6.1 {
        export r1;
    }
    neighbor 10.0.6.2 {
        export r2;
    }
    neighbor 10.0.3.3;
    neighbor 10.0.3.5;
    neighbor 10.0.9.6;
    neighbor 10.0.9.7;
}
group c1 {
    type external;
    family inet {
        unicast;
    }
    family inet6 {
        unicast;
    }
    export ebgp-out;
    neighbor 172.16.0.6 {
        peer-as 65010;
    }
}
```

```
group int-v6 {
    type internal;
    local-address fec0:0:3:0::4;
    export nhs;
    neighbor fec0:0:3:0::5;
    neighbor fec0:0:6:0::2 {
        export r2-v6;
    }
    neighbor fec0:0:3:0::3;
    neighbor fec0:0:6:0::1 {
        export r1-v6;
    }
}

[edit]
lab@r4# show protocols isis export
export v6-default;

[edit]
lab@r4# show policy-options policy-statement ebgp-out
term 1 {
    from {
        protocol aggregate;
        route-filter 10.0.0.0/16 exact;
    }
    then accept;
}
term 2 {
    from {
        protocol aggregate;
        route-filter fec0::/16 exact;
        route-filter ::10.0.0.0/112 exact;
    }
    then accept;
}

[edit]
lab@r4# show policy-options policy-statement v6-default
term 1 {
    from {
        protocol static;
        route-filter ::0/0 exact;
```

```
    }
    to level 1;
    then accept;
}

[edit]
lab@r4# show policy-options policy-statement r1-v6
term 1 {
    from {
        protocol aggregate;
        route-filter ::10.0.0.0/112 exact accept;
        route-filter fec0::/16 exact;
    }
    then {
        next-hop fec0:4:16:0:290:69ff:fe6b:3002;
        accept;
    }
}
term 2 {
    from {
        protocol bgp;
        neighbor 172.16.0.6;
    }
    then {
        next-hop self;
    }
}

[edit]
lab@r4# show policy-options policy-statement r2-v6
term 1 {
    from {
        protocol aggregate;
        route-filter ::10.0.0.0/112 exact accept;
        route-filter fec0::/16 exact;
    }
    then {
        next-hop fec0:4:8:0:290:69ff:fe6b:3001;
        accept;
    }
}
```

```
term 2 {
    from {
        protocol bgp;
        neighbor 172.16.0.6;
    }
    then {
        next-hop self;
    }
}
```

Listing 5.5: IPv6 Case Study Configuration for *r5*

```
[edit]
lab@r5# show interfaces at-0/2/1
atm-options {
    vpi 0 {
        maximum-vcs 64;
    }
}
unit 0 {
    point-to-point;
    vci 50;
    family inet {
        address 10.0.2.1/30;
    }
    family iso;
    family inet6 {
        address fec0:2:0:0::/64 {
            eui-64;
        }
    }
}

[edit]
lab@r5# show interfaces so-0/1/0
encapsulation ppp;
unit 0 {
    family inet {
        address 10.0.2.9/30;
    }
    family iso;
    family inet6 {
        address fec0:2:8:0::/64 {
            eui-64;
```

```
        }
    }
}

[edit]
lab@r5# show interfaces lo0
unit 0 {
    family inet {
        address 10.0.3.5/32;
    }
    family iso {
        address 49.0002.5555.5555.5555.00;
    }
    family inet6 {
        address fec0:0:3::5/128;
    }
}

[edit]
lab@r5# show protocols bgp
group int {
    type internal;
    local-address 10.0.3.5;
    neighbor 10.0.6.1;
    neighbor 10.0.6.2;
    neighbor 10.0.3.3;
    neighbor 10.0.3.4;
    neighbor 10.0.9.6;
    neighbor 10.0.9.7;
}
group int-v6 {
    type internal;
    local-address fec0:0:3:0::5;
    neighbor fec0:0:3:0::3;
    neighbor fec0:0:3:0::4;
    neighbor fec0:0:6:0::1;
    neighbor fec0:0:6:0::2;
}
```

r6 was not part of the IPv6 test bed. No changes were made to its configuration as part of this chapter's case study.

r7 was not part of the IPv6 test bed. No changes were made to its configuration as part of this chapter's case study.

Spot the Issues: Review Questions

1. Does this EBGP configuration for r3 meet the requirement posed in this chapter's case study?

```
[edit protocols bgp group ext]
lab@r3# show
import ebgp-in;
family inet6 {
    unicast;
}
export ebgp-out;
neighbor 172.16.0.14 {
    peer-as 65222;
}
```

2. Why does this case study configuration from r4 make it seem that no routes are ever advertised by the C1 peer?

```
[edit]
lab@r4# show interfaces fe-0/0/0
unit 0 {
    family inet {
        address 172.16.0.5/30;
    }
    family inet6;
}

[edit]
lab@r4# show protocols bgp group c1
type external;
family inet {
    unicast;
}
family inet6 {
    unicast;
}
export ebgp-out;
neighbor 172.16.0.6 {
    peer-as 65010;
}
```

3. Why is this router advertisement configuration not advertising the configured prefix?

```
[edit]
lab@r1# show interfaces fe-0/0/0
unit 0 {
    family inet {
        address 10.0.5.1/24;
    }
    family inet6 {
        address fec0:5:0::/64 {
            eui-64;
        }
    }
}

[edit]
lab@r1# show protocols router-advertisement
interface fe-0/0/0.0 {
    prefix fec0:0:5:0::/64;
}
```

4. Can you explain the commit error stemming from this configuration?

```
[edit routing-options]
lab@r4# show rib inet6
static {
    route ::/0 reject;
}
aggregate {
    route ::10.0.0.0/112;
    route fec0::/16;
}

[edit routing-options]
lab@r4# commit check
[edit routing-options rib inet6]
  rib inet6
  RT: invalid rib inet6

error: configuration check-out failed
```

Spot the Issues: Answers to Review Questions

1. No. This configuration does not explicitly list the `inet unicast` family, so the resulting EBGP session will not support IPv4 NLRI. You must explicitly configure the `inet` family when any other address families are also explicitly configured.
2. You must configure an IPv4-compatible IPv6 address that matches the IPv4 peering address when supporting IPv6 NLRI over an IPv4-based BGP session. The lack of a ::172.16.0.5 address assignment on `r4`'s fe-0/0/0 interface causes `r4` to discard (not hide) the IPv6 routes it receives from the C1 peer.
3. A router advertisement advertises a configured prefix only when that prefix is assigned to the advertising interface. In this case, a typo has resulted in the configured prefix not matching any prefixes assigned to `r1`'s fe-0/0/0 interface.
4. The correct RIB for IPv6 routes is `inet6.0`, not `inet6`. Although the CLI allows you to specify user-defined RIBs, the presence of IPv6 route entries forces you to use a standard name for the IPv6 RIB.

Chapter

6

Class of Service

JNCIE LAB SKILLS COVERED IN THIS CHAPTER:

- ✓ **Packet Classification and Forwarding Classes**
  - Behavior Aggregate and Multifield
- ✓ **Rewrite and Packet Marking**
  - Loss Priority
- ✓ **Schedulers**
  - RED Profiles

This chapter exposes the reader to several JNCIE-level Class of Service (CoS) configuration tasks that validate the JNCIE candidate's practical understanding of the CoS capabilities and features associated with JUNOS software release 5.6. It is assumed that the reader already possesses a working knowledge of general CoS/QoS concepts as well as IP Differential Services (DiffServ) code points and per-hop behaviors (PHBs) to the extent covered in the *JNCIS Study Guide* (Sybex, 2003).

Juniper Networks M-series and T-series routing platforms support a rich set of features that facilitate the deployment of IP networks that support specialized traffic handling. In general, you deploy CoS when the default (Best Effort (BE)) packet handling behavior fails to provide a specific application with the required level of performance in the face of limited resources, or when service level agreements (SLAs) dictate that premium services classes are to be afforded a level of treatment above and beyond lesser-service classes. When properly designed and deployed, a JUNOS software–based CoS solution permits integrated service support over IP-based networks, which in turn allows network operators to converge their networks around the well-understood and ubiquitous IP technology. Replacing multiple overlay networks (ATM, X.25, Frame Relay, leased line, POTS) with a single backbone technology offers many economic and operational advantages; in the end, one network is simply less expensive to own and operate.

Note that the syntax for JUNOS software CoS configuration changed dramatically starting with release 4.4. In keeping with the 5.6R2.4 JUNOS software release used to develop this book, only the "new-style" configuration syntax is documented and demonstrated. Also of note are the functional and operational differences between the original and "Enhanced" Flexible PIC Concentrators (FPCs), which became available for M-series routing platforms circa the 4.4 JUNOS software release. Enhanced FPCs are associated with M-series routing platforms only; T-series FPCs have always offered CoS-related functionality that is generally equal to (or better than) the capabilities of the E-FPC. E-FPCs make use of the third-generation I/O manager ASIC (the B-chip) to offer additional capabilities and enhanced operational mode output that greatly simplifies the task of monitoring the effects of a CoS configuration. The reader is encouraged to consult the JUNOS software documentation set for complete details on the capabilities and differences between the FPC, E-FPC, and the M-series and T-series platforms. Because E-FPCs are now standard issue on all M-series platforms, the configuration and operational mode command examples in this chapter are based on E-FPC capabilities. Use the `show chassis hardware` command to determine if your M-series router is equipped with an E-FPC:

```
[edit]
lab@r5# run show chassis hardware
```

```
Hardware inventory:
Item             Version  Part number  Serial number     Description
Chassis                                50525             M5
Midplane         REV 03   710-002650   HF1636
Power Supply A   Rev 04   740-002497   LL14364           AC
Power Supply B   Rev 04   740-002497   LL14375           AC
Display          REV 04   710-001995   AL2316
Routing Engine                         6d0000078e2d5401  RE-2.0
FEB              REV 06   710-003311   HK9932            E-FEB
FPC 0
  PIC 0          REV 04   750-002992   HB2103            4x F/E, 100 BASE-TX
  PIC 1          REV 03   750-002971   HD2855            4x OC-3 SONET, MM
  PIC 2          REV 03   750-002977   HC3250            2x OC-3 ATM, MM
```

The presence of an Enhanced FPC is identified by the `E-FEB` designation evident in the sample output, which was taken from an M5 platform.

The configuration scenarios demonstrated in the chapter body and case study are based on the OSPF baseline configuration as discovered in the case study of Chapter 1. If you are unsure as to the state of your test bed, you should take a few moments to load up and confirm the operation of the OSPF discovery configuration. Note that some of the routers in the JNCIE test bed are not brought into play during the course of this chapter. In many cases, a candidate is expected to configure functionality on a subset of routers in an effort to save time while still validating the candidate's practical skill set.

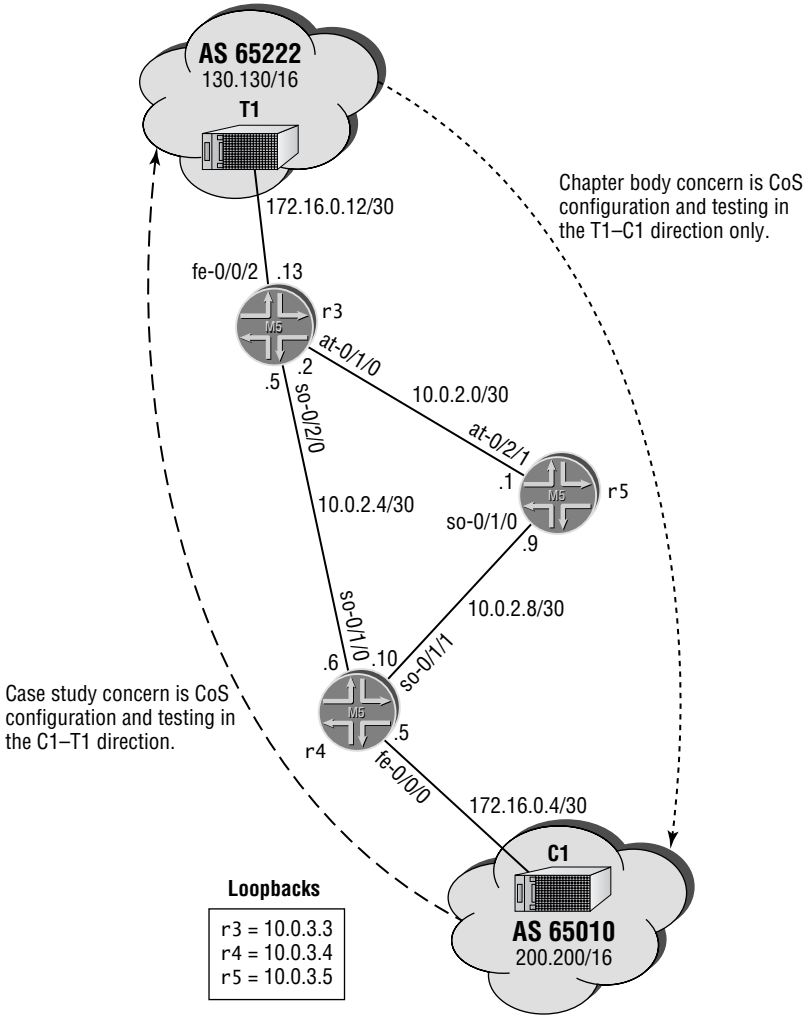
Operational mode command output examples are provided throughout the chapter so that you can compare your network's operation to that of a known good example. Baseline configuration changes that are known to meet all case study requirements are provided at the end of the case study for all routers in the CoS test bed.

Unlike other topics in this book series, which readily accommodated a piece-by-piece approach whereby subsets of functionality could be individually addressed and verified, with CoS, it is often an "all or nothing" proposition. This is to say that a functional CoS configuration often involves numerous changes in numerous places before there can be any hope of a "working" CoS solution. To best accommodate these unique characteristics, while still presenting information in a modular and logical sequence, the approach taken in this chapter deviates from previous chapters in the following ways:

- An overall description of the CoS goals detailed in the chapter body is provided to afford the reader with a big-picture view of where the chapter is going, and what needs to be done.
- The chapter body focuses on a simplex CoS solution, which is to say that the configuration and confirmation examples are performed with a strict focus on traffic flowing in a single direction.
- The chapter case study has the candidate complete the CoS solution by configuring matching CoS functionality in the return direction.

Figure 6.1 details the CoS test bed, which currently comprises `r3` through `r5` and EBGP peers T1 and C1.

FIGURE 6.1 CoS topology



As you progress through this chapter, it helps to keep in mind that your overall goal is to configure the CoS test bed to support an integrated services offering for transit traffic comprising Voice over IP (VoIP) and conventional Internet traffic. To minimize the chapter's complexity (and size), the chapter body concentrates on CoS configuration and confirmation in the T1-to-C1 direction *only*. It is easy to get confused with CoS; keeping the big picture in your mind will generally help to keep you, and your configurations, on track.

A complete listing of the CoS configuration scenario's configuration criteria and restrictions is provided here to give the reader an overview of what needs to be configured and tested.

The application specifics and configuration requirements for the CoS scenario are listed here in their entirety:

- SIP signaling (VoIP) uses TCP/UDP, port 5060.

- RTP media channels use UDP with port assignments in the range of 16,384–32,767.
- Classify all VoIP traffic as EF.
- Ensure that network control traffic continues to be classified as NC.
- Classify all remaining traffic with IP precedence 0 as BE.
- Police BE traffic to 1Mbps with excess data marked for discard.
- Your classification design must tolerate the failure of any single core interface or link.
- Configure `r3` so that traffic received from the T1 peer is classified according to a DSCP-based BA at `r4` and `r5`.
- Ensure that traffic received from the T1 peer is consistently classified by all network elements.
- Ensure that `r4` and `r5` are able to differentiate BE traffic received from the T1 peer based on its compliance with the configured policer.
- You must use DSCP-based classification at `r4` and `r5`.
- Configure schedulers in the CoS test bed according to these criteria:
  - BE traffic limited to 10 percent of interface bandwidth.
  - Ensure that BE traffic never exceeds the configured rate, even when other queues are empty.
  - Configure the EF class to get 30 percent of interface bandwidth.
  - Configure the EF class for a maximum queuing delay of 100 milliseconds.
  - Ensure that the EF class has precedence over all other classes, regardless of the EF queue's current bandwidth credit.
  - Make sure your scheduler configuration does not adversely affect the operation of your routing protocols.
- Drop 1 percent of the BE traffic with a low loss priority at 70 percent buffer fill.
- Drop 10 percent of the BE traffic with a high loss priority at 50 percent buffer fill.
- Do not alter the default RED profiles in effect for the EF and NC forwarding classes.
- Ensure that RED only acts on TCP-based BE traffic.

Again note that the above requirements are to be configured and tested in the direction of T1 to C1 only!

Packet Classification and Forwarding Classes

Packets supporting special services and applications must be recognized and mapped to a corresponding forwarding class at ingress so the desired per-hop behavior (PHB) can be brought to bear on that packet. Currently M-series and T-series routing platforms offer support for four

independent forwarding classes (FCs). These include Expedited Forwarding (EF), Assured Forwarding (AF), Best Effort (BE), and Network Control (NC). Support for additional forwarding classes is achieved by aggregating other classes into one of these four FCs. It is sometimes helpful to recall that forwarding classes used to be called "output queues" in previous JUNOS software CoS terminology; in effect, the end result of classification is the identification of an output queue for a particular packet.

Devices that sit at the edge of a network usually classify packets according to codings that are located in multiple packet header fields. Multifield classification is normally performed at the network's edge due to the general lack of DiffServ Code Point (DSCP) or IP precedence support in end-user applications; common codings of the IP precedence or the DiffServ fields makes classification based on the traffic's behavior aggregate (BA) unavailable for ingress traffic. Multifield classification is performed in JUNOS software using firewall filters and their associated match conditions (see Chapter 3 for coverage of JUNOS software firewall filters).

BA classification, on the other hand, is based on IP precedence, IEEE 802.1P, DiffServ DSCPs, or MPLS EXP bit settings. Because the fixed-length nature of BA-based classification is computationally more efficient than a multifield classification approach, core devices are normally configured to perform BA classification due to the higher traffic volumes they are expected to handle. In most cases, you need to rewrite a given marker (IP precedence, DiffServ DSCP, IEEE 802.1P, or MPLS EXP settings) at the ingress node to accommodate BA classification actions by core and egress devices. Rewrite actions are needed when the ingress traffic is not already coded with BA settings that are compatible with the configuration of core devices, which typically is the case given that most TCP/IP applications use default settings for the BA-related fields in the packets (and frames) they generate.

In addition to MF- and BA-based packet classification, JUNOS software also supports ingress interface and destination address–based classification mechanisms.

In addition to associating packets to forwarding classes, classification is also used to identify a packet's loss priority (PLP). The PLP indication is used by schedulers in conjunction with the RED algorithm to control packet discard during periods of congestion.

Multifield Classification

It is assumed that the OSPF baseline network is operational, at least as it would regard `r3`, `r4`, `r5`, and the EBGP peers T1 and C1.You begin the CoS configuration scenario at `r3` by configuring it to perform multifield classification on packets received from the T1 peer according to the criteria listed next.

- SIP signaling (VoIP) uses TCP/UDP, port 5060.
- RTP media channels use UDP with port assignments in the range of 16,384–32,767.
- Classify all VoIP traffic as EF.
- Ensure that network control traffic continues to be classified as NC.
- Classify all remaining traffic with IP precedence 0 as BE.
- Police BE traffic to 1Mbps with excess data marked for discard.

Configuring Multifield Classification

You begin configuration at r3 by defining a firewall filter called *classify* that is written to match on the transport protocol and ports identified in the packets received from T1 with the intent of classifying the traffic into the forwarding classes specified by your criteria. The first term, which is called *sip* in this case, classifies SIP signaling messages:

```
[edit]
lab@r3# edit firewall filter classify

[edit firewall filter classify]
lab@r3# set term sip from protocol udp

[edit firewall filter classify]
lab@r3# set term sip from protocol tcp

[edit firewall filter classify]
lab@r3# set term sip from port 5060

[edit firewall filter classify]
lab@r3# set term sip then forwarding-class expedited-forwarding
```

Firewall Filters and Address Families

The JUNOS software release 5.6 offers support for firewall filters based on the IPv6 and MPLS families. The syntax shown in this section and in Chapter 4 is based on the historical support for IPv4 firewall filters only. To offer support for additional address families, the firewall filter syntax is changing to include the protocol family. While the older syntax is still supported in the 5.6 release, you should become accustomed to the new syntax at some point. A sample of the newer firewall filter syntax is shown here:

```
[edit firewall]
lab@r2# show
family inet {
    filter new-syntax {
        term sip {
            from {
                protocol [ udp tcp ];
                port 5060;
            }
            then forwarding-class expedited-forwarding;
        }
    }
}
```

Note that port directionality is avoided by using the `port` keyword, which causes a match when either the source port (replies), the destination port (requests), or both are coded to 5060, and that the `forwarding-class` action modified negates the need for an explicit `accept` action. The second term in the *`classify`* firewall filter deals with the VoIP media channels that use UDP-based transport:

```
[edit firewall filter classify]
lab@r3# set term rtp from protocol udp

[edit firewall filter classify]
lab@r3# set term rtp from port 16384-32767

[edit firewall filter classify]
lab@r3# set term rtp then forwarding-class expedited-forwarding
```

The final term in the *`classify`* filter ensures that all remaining traffic is classified as BE and policed in accordance with your restrictions:

```
[edit firewall filter classify]
lab@r3# set term be then policer be-policer
```

With the filter complete, the *`be-policer`* is defined:

```
[edit firewall]
lab@r3# set policer be-policer if-exceeding bandwidth-limit 1m

[edit firewall]
lab@r3# set policer be-policer if-exceeding burst-size-limit 15000

[edit firewall]
lab@r3# set policer be-policer then loss-priority high
```

Given that an explicit burst size value was not specified, the policer's burst tolerance is set to the recommended value for a low-speed interface, which is ten times the interface's MTU. For a high-speed interface, such as an OC-192, the recommended burst size is the transmit rate of the interface times 3–5 milliseconds. The completed *`classify`* filter is displayed:

```
[edit firewall]
lab@r3# show filter classify
term sip {
    from {
        protocol [ udp tcp ];
        port 5060;
    }
    then {
        forwarding-class expedited-forwarding;
        accept;
    }
}
```

```
term rtp {
    from {
        protocol udp;
        port 16384-32767;
    }
    then {
        forwarding-class expedited-forwarding;
        accept;
    }
}
term be {
    then policer be-policer;
}
```

The related *be-policer* is also displayed for visual confirmation:

```
[edit firewall]
lab@r3# show policer be-policer
if-exceeding {
    bandwidth-limit 1m;
    burst-size-limit 15k;
}
then loss-priority high;
```

The fact that the *be* term does not include a `forwarding-class` action modifier, and that no explicit treatment of network control (NC) traffic is provided in the *classify* filter, warrants some additional attention here. Although you could add explicit support for the classification of network control traffic, and all remaining IP traffic, the default IP precedence classifier correctly classifies the remaining traffic with no added typing! To confirm this theory, you display the default classifiers currently in effect for r3's fe-0/0/2 interface:

```
lab@r3# run show class-of-service interface fe-0/0/2
Physical interface: fe-0/0/2, Index: 14
  Scheduler map: <default>, Index: 1

  Logical interface: fe-0/0/2.0, Index: 7
    Object         Name                  Type                      Index
    Rewrite        exp-default           exp                           2
    Classifier     ipprec-compatibility  ip                            5
```

The display confirms that the `ipprec-compatibility` classifier is in effect by default. You use the `show class-of-service classifier` *name* command to view the classifier's mappings:

```
[edit]
lab@r3# run show class-of-service classifier name ipprec-compatibility
```

```
Classifier: ipprec-compatibility, Code point type: inet-precedence, Index: 5
  Code point         Forwarding class                  Loss priority
  000                best-effort                       low
  001                best-effort                       high
  010                best-effort                       low
  011                best-effort                       high
  100                best-effort                       low
  101                best-effort                       high
  110                network-control                   low
  111                network-control                   high
```

The highlighted entries confirm that traffic with IP precedence setting of 0 is correctly classified as BE while NC traffic, with precedence values of 6 or 7, is properly classified as NC. Satisfied with the syntax and structure of the *classify* filter and *be-policer*, you apply the filter as an input on r3's fe-0/0/2 interface:

```
[edit]
lab@r3# show interfaces fe-0/0/2
unit 0 {
    family inet {
        filter {
            input classify;
        }
        address 172.16.0.13/30;
    }
}
```

JUNOS software permits the classification of a given packet using both a BA and multifield classifier. The use of the multifield *classifier* filter combined with the presence of the default IP precedence classifier means that just such a situation will be in effect when you commit the changes. In these situations, the BA classifier is processed first, followed in turn by the multifield classifier. This ordering is significant because it means that a multifield classifier can overwrite the classification made by a BA classifier when both techniques have been brought to bear on the same packet.

Confirming Multifield Classification

To confirm that your multifield classifier is working correctly, you need to monitor the queue counters at r3 for the *egress* interface used when forwarding traffic received from the T1 peer. Note that displaying the queue counters for the fe-0/0/2 interface will tell you nothing about your ingress classification at r3, because queuing occurs only at egress on M-series and T-series routers. You begin with a traceroute that confirms traffic from T1 to C1 currently egresses r3 via its so-0/2/0.100 interface:

```
[edit]
lab@T1-P1# run traceroute 200.200.0.1 source 130.130.0.1
```

```
traceroute to 200.200.0.1 (200.200.0.1) from 130.130.0.1, 30 hops max, 40 byte
   packets
 1  172.16.0.13 (172.16.0.13)  0.432 ms  0.284 ms  0.272 ms
 2  10.0.2.6 (10.0.2.6)  0.362 ms  0.302 ms  0.292 ms
 3  200.200.0.1 (200.200.0.1)  0.248 ms  0.223 ms  0.217 ms
```

With the egress interface identified, you clear its associated queue counters:

```
[edit]
lab@r3# run clear interfaces statistics so-0/2/0
```

Before looking at the actual queue counts, you decide to confirm the default forwarding class to queue number assignment so that you can adequately predict which queues should be used by the VoIP, NC, and other traffic:

```
[edit]
lab@r3# run show class-of-service forwarding-class
Forwarding class                  Queue       Fabric priority
  best-effort                         0       low
  expedited-forwarding                1       low
  assured-forwarding                  2       low
  network-control                     3       low
```

The output displays the default forwarding class to queue mappings; if desired, you can rename the forwarding class name associated with the queues supported on your hardware (all M-series and T-series platforms support at least four output queues). Note that assigning a new class name to a system queue does not alter the default classification or scheduling that is applicable to that queue. CoS configurations are complicated enough, so unless it is required by the specifics of your scenario it is suggested that you not alter the default class names or queue number associations.

The default forwarding class-to-queue mappings shown indicate that VoIP traffic, which is classified as EF by the *classify* filter, should be placed into queue 2. Keeping all this in mind, you display the queue counts on the so-0/2/0 interface:

```
[edit]
lab@r3# run show interfaces so-0/2/0 detail | find "Queue counters"
  Queue counters:       Queued packets  Transmitted packets      Dropped packets
    0 best-effort                    1                    1                    0
    1 expedited-fo                   0                    0                    0
    2 assured-forw                   0                    0                    0
    3 network-cont                 115                  115                    0
  SONET alarms    : None
  SONET defects   : None
. . .
```

Noting that the counter for the EF queue (queue number 1) is currently zeroed out, you generate test traffic from the T1 router. Verifying the UDP aspects of the *classify* filter

is complicated by your inability to generate UDP traffic with arbitrary port values. Therefore, you limit confirmation testing to conventional Internet traffic and the TCP-based VoIP signaling:

```
[edit]
lab@T1-P1# run ping 200.200.0.1 count 5 source 130.130.0.1
PING 200.200.0.1 (200.200.0.1): 56 data bytes
64 bytes from 200.200.0.1: icmp_seq=0 ttl=253 time=0.366 ms
64 bytes from 200.200.0.1: icmp_seq=1 ttl=253 time=0.249 ms
64 bytes from 200.200.0.1: icmp_seq=2 ttl=253 time=0.239 ms
64 bytes from 200.200.0.1: icmp_seq=3 ttl=253 time=0.241 ms
64 bytes from 200.200.0.1: icmp_seq=4 ttl=253 time=0.245 ms

--- 200.200.0.1 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.239/0.268/0.366/0.049 ms

[edit]
lab@T1-P1# run telnet 200.200.0.1 source 130.130.0.1 port 5060
Trying 200.200.0.1...
telnet: connect to address 200.200.0.1: Connection refused
telnet: Unable to connect to remote host
```

These commands should have generated five non-VoIP packets and a single TCP segment that was sent to port 5060. Proper classification is now confirmed by once again displaying `r3`'s so-0/2/0 interface queue counters:

```
[edit]
lab@r3# run show interfaces so-0/2/0 detail | find "Queue counters"
  Queue counters:       Queued packets  Transmitted packets      Dropped packets
    0 best-effort                    6                    6                    0
    1 expedited-fo                   1                    1                    0
    2 assured-forw                   0                    0                    0
    3 network-cont                 117                  117                    0
  SONET alarms    : None
  SONET defects   : None
. . .
```

Both the `best-effort` and `expedited-fo` queue counters have been incremented as expected, considering the traffic that was generated at T1. To confirm operation of the *`be-policer`*, flood pings are generated from the T1 peer while the policer's out-of-profile counter is displayed at `r3`; you clear the firewall counter at `r3` before commencing the test (not shown):

```
[edit]
lab@T1-P1# run ping 200.200.0.1 size 1200 rapid count 100 source 130.130.0.1
```

```
PING 200.200.0.1 (200.200.0.1): 1200 data bytes
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !!!!!!!!!!!!!!!!!!!
--- 200.200.0.1 ping statistics ---
100 packets transmitted, 100 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.755/0.761/0.914/0.016 ms

[edit]
lab@r3# run show firewall
Filter: classify
Policers:
Name                                                Packets
be-policer-be                                          77
```

The non-zero packet count associated with the *be-policer* confirms that 77 of the 100 packets exceeded the policer's parameters, and were therefore marked with a high discard probability. The 100 percent success rate for the ping traffic indicates that the policer is properly configured to pass traffic that is outside of the policing profile.

Although the UDP-related VoIP classification was not tested, knowing that your multifield classification filter operates correctly for the VoIP-related TCP traffic and for the non-VoIP traffic, combined with visual inspection of the UDP match criteria in the *classify* filter, provides sufficient confirmation that your multifield classification filter complies with all specified criteria.

BA Classification

With multifield classification in place at r3, your attention shifts to the need for BA-based classification at transit and egress routers r5 and r4, respectively. Note that successful BA classification at r4 and r5 ultimately relies on DSCP rewrite marker functionality, which has not yet been put into place at r3. The BA classification criteria are as follows:

- Configure DSCP-based classification at r4 and r5.
- Ensure that traffic received from the T1 peer is consistently classified by all network elements.
- Your classification design must tolerate the failure of any single core interface or link.

Configuring BA Classification

The criteria indicate a need for DSCP-based classification at r4 and r5 that will yield a consistent classification of traffic across your network. Your configuration must also take into account the need to operate around the failure of any single core interface or link. You begin at r4 by displaying the default DSCP classification map:

```
[edit class-of-service]
lab@r4# run show classifier name dscp-default
Classifier: dscp-default, Code point type: dscp, Index: 1
```

Code point	Forwarding class	Loss priority
000000	best-effort	low
000001	best-effort	low
000010	best-effort	low
000011	best-effort	low
000100	best-effort	low
000101	best-effort	low
000110	best-effort	low
000111	best-effort	low
001000	best-effort	low
001001	best-effort	low
001010	assured-forwarding	low
001011	best-effort	low
001100	assured-forwarding	high
001101	best-effort	low
001110	assured-forwarding	high
001111	best-effort	low
010000	best-effort	low
010001	best-effort	low
010010	best-effort	low
010011	best-effort	low
010100	best-effort	low
010101	best-effort	low
010110	best-effort	low
010111	best-effort	low
011000	best-effort	low
011001	best-effort	low
011010	best-effort	low
011011	best-effort	low
011100	best-effort	low
011101	best-effort	low
011110	best-effort	low
011111	best-effort	low
100000	best-effort	low
100001	best-effort	low
100010	best-effort	low
100011	best-effort	low
100100	best-effort	low
100101	best-effort	low
100110	best-effort	low
100111	best-effort	low

```
  101000            best-effort                    low
  101001            best-effort                    low
  101010            best-effort                    low
  101011            best-effort                    low
  101100            best-effort                    low
  101101            best-effort                    low
  101110            expedited-forwarding           low
  101111            best-effort                    low
  110000            network-control                low
  110001            best-effort                    low
  110010            best-effort                    low
  110011            best-effort                    low
  110100            best-effort                    low
  110101            best-effort                    low
  110110            best-effort                    low
  110111            best-effort                    low
  111000            network-control                low
  111001            best-effort                    low
  111010            best-effort                    low
  111011            best-effort                    low
  111100            best-effort                    low
  111101            best-effort                    low
  111110            best-effort                    low
  111111            best-effort                    low
```

The `dscp-default` classifier maps all 64 possible DSCPs to one of the four supported traffic classes. You note that none of the criteria specified thus far warrants a deviation from the default DSCP classifications that are shown earlier. Assuming for the moment that `r3` will be configured with a compatible set of DSCP rewrite markers, the only action needed to meet the requirements of this section is to associate the `dscp-default` BA classifier with the `r3`-facing interfaces at `r4` and `r5`. The default DSCP classifier is now associated with `r4`'s core-facing interfaces:

```
[edit class-of-service interfaces so-0/1/0]
lab@r4# set unit 100 classifiers dscp default

[edit class-of-service interfaces so-0/1/1]
lab@r4# set unit 0 classifiers dscp default
```

The resulting configuration is shown:

```
[edit class-of-service]
lab@r4# show
interfaces {
    so-0/1/0 {
        unit 100 {
```

```
            classifiers {
                dscp default;
            }
        }
    }
    so-0/1/1 {
        unit 0 {
            classifiers {
                dscp default;
            }
        }
    }
}
```

Note that the default DSCP classifier is referenced with the keyword `default` as opposed to its fully qualified `dscp-default` name; specifying the full name returns a commit error. Also of note is the need for DSCP-based classification on packets arriving on both of `r4`'s core-facing interfaces, which addresses the case of packets being routed through `r5` in the event of a link failure between `r3` and `r4`. The default DSCP classifier must also be configured on `r5`. The `class-of-service` stanza is displayed on `r5` after the modification is put into place:

```
[edit class-of-service interfaces at-0/2/1]
lab@r5# show
at-0/2/1 {
    unit 0 {
        classifiers {
            dscp default;
        }
    }
}
```

Be sure that you commit your changes on `r4` and `r5` before proceeding to the confirmation section.

Confirming BA Classification

At this stage, the confirmation of the BA classifier must be confined to the simple act of verifying that the correct classifier is in effect on the core-facing interfaces of `r4` and `r5`. This is because you can not expect the default DSCP classifier to correctly classify packets until you have configured `r3` to rewrite the DSCP markers on the traffic received from T1 based on its multifield classification. A `show class-of-service interface` *`name`* command is issued at `r4` to verify that DSCP classification is in effect on its so-0/1/0.100 interface:

```
[edit]
lab@r4# run show class-of-service interface so-0/1/0
Physical interface: so-0/1/0, Index: 16
  Scheduler map: <default>, Index: 1
```

```
  Logical interface: so-0/1/0.100, Index: 9
    Object          Name                    Type                        Index
    Rewrite         exp-default             exp                             2
    Classifier      dscp-default            dscp                            1

[edit class-of-service]
lab@r4# run show class-of-service interface so-0/1/1
Physical interface: so-0/1/1, Index: 17
  Scheduler map: <default>, Index: 1

  Logical interface: so-0/1/1.0, Index: 10
    Object          Name                    Type                        Index
    Rewrite         exp-default             exp                             2
    Classifier      dscp-default            dscp                            1
```

Although not shown, you may assume that `r5` displays similar information for its at-0/2/1.0 interface. Once rewrite marker configuration is added to `r3`, you will be able to revisit your DSCP-based BA classification for final verification.

Classification Summary

Packets that require specialized handling must be identified as they ingress the network. In most cases, you will use a firewall filter to perform multifield-based classifications at the network's edge while using the more direct BA-based classification within the network's core. This section demonstrated how a firewall filter, in conjunction with the default IP precedence classifier, can be used to perform multifield classification to identify traffic as belonging to the BE, EF, or NC classes. The section also demonstrated how a classifier, in this case the default DSCP classifier, is applied to an interface. Note that while the E-FPC supports as many as eight classifiers per FPC, you may specify only one classifier per logical interface.

Each of the four traffic classes supported by JUNOS software is mapped to a corresponding output queue. You display the class-to-queue mappings with the `show class-of-service forwarding-class` command. You use the `show class-of-service classifier` command to display the user-defined and default classifiers.

It is important to note that successful BA-based classification normally requires some sort of marker rewrite action at the network's edge. This is because the majority of traffic that is generated by end-user applications makes use of default IP precedence/DSCP values that, if left unmodified, interferes with successful BA-based classification in the core.

Rewrite Markers

As previously mentioned, successful use of BA classification by core devices normally requires some form of marker rewrite at the network edge to accommodate the fact that most applications populate the IP precedence/DSCP field with a default setting, in other words, all zeros.

JUNOS software combined with E-FPC functionality supports the rewriting of IP precedence, DSCP, MPLS EXP, and IEEE 802.1p markers.

Marker rewrite relates to packet loss priority (PLP) and RED related discards because for IP traffic the PLP bit is conveyed between routers using the LSB of the IP precedence field or specified DiffServ code points. For labeled packets, the PLP status is always coded in bit 22 (the LSB of the MPLS EXP field) of the 32-bit shim label when using the original M-series FPC. With Enhanced FPCs or a T-series FPC, you can define an EXP classifier (and rewrite table) that support MPLS PLP status using any bit in the EXP field. Note that for 802.1P-based classification, bit 0 is always used to code the packet's PLP status. In this example, bit 0 of the MPLS EXP field will be used to code PLP status.

In this example, you will be rewriting the DSCP settings of the traffic received from the T1 peer at `r3` according to these requirements:

- Configure `r3` so that traffic received from the T1 peer is classified according to a DSCP-based BA at `r4` and `r5`.
- Ensure that traffic received from the T1 peer is consistently classified by all network elements.

The astute reader will recognize that the last criterion has been duplicated from the previous section. This is because consistent traffic classification relies on the successful interplay between the configuration steps that relate to classification (as detailed in the previous section) and marker rewrite (as covered in this section).

Configuring DSCP Rewrite

The requirements specified indicate a need to configure `r3` so that egress traffic at `r3` (as received from the T1 peer) will be compatible with the DSCP-based BA classification that is now in effect at `r4` and `r5`. Care must be taken to ensure that your configuration yields a consistent classification between the BA and multifield approaches that are both at play in the CoS test bed. You start at `r3` by displaying the default DSCP rewrite table:

```
[edit]
lab@r3# run show class-of-service rewrite-rule type dscp
Rewrite rule: dscp-default, Code point type: dscp, Index: 1
  Forwarding class                Loss priority      Code point
  best-effort                     low                000000
  best-effort                     high               000000
  expedited-forwarding            low                101110
  expedited-forwarding            high               101110
  assured-forwarding              low                001010
  assured-forwarding              high               001100
  network-control                 low                110000
  network-control                 high               111000
```

The `dscp-default` rewrite table maps each supported traffic class and loss priority setting to a particular DSCP value. By comparing the DSCPs shown for the BE, NC, and EF classes to

the default DSCP classification table, as displayed in the previous section, you can readily confirm that the default DSCP rewrite and classifications tables are compatible. By way of example, consider a packet that has been classified as EF upon ingress via the multifield classifier at r3. When the default DSCP rewrite table is applied to r3's egress interfaces, these packets will have their DSCP coded with the value 101110, which is good, because the default DSCP classifier interprets this pattern as expedited forwarding with low loss priority. The key point here is that the default DSCP rewrite table and the default DSCP classification table are inherently consistent, which is important, because a successful CoS design relies on the consistent (and predictable) classification of a given packet across all devices in the network.

To complete the configuration aspects of this section, you need to correctly associate the default DSCP rewrite table to the core-facing interfaces on r3. Before making any configuration changes, you decide to confirm the default rewrite behavior so you can better gauge the effects of your impending changes. You begin by clearing the interface counters at r4 (not shown), which is followed up with the generation of a TCP segment destined to port 5060 at the T1 peer:

```
[edit]
lab@T1-P1# run telnet 200.200.0.1 source 130.130.0.1 port 5060
Trying 200.200.0.1...
telnet: connect to address 200.200.0.1: Connection refused
telnet: Unable to connect to remote host
```

Noting that this traffic is classified as EF by the multifield classifier in place at r3, you display the queue counts for the egress interface at r4:

```
[edit]
lab@r4# run show interfaces fe-0/0/0 detail | find "Queue counters"
  Queue counters:       Queued packets  Transmitted packets      Dropped packets
    0 best-effort                    7                    7                    0
    1 expedited-fo                   0                    0                    0
    2 assured-forw                   0                    0                    0
    3 network-cont                   6                    6                    0
  Active alarms  : None
  Active defects : None

  . . .
```

The highlighted entry indicates that r4 has failed to correctly classify the traffic as belonging to the EF class. Considering that the only rewrite table that is in effect by default is for MPLS EXP bits, this behavior is to be expected. Rewrite functionality can also be confirmed with traffic monitoring. In this case, you modify the target address of the Telnet command at the T1 router while monitoring traffic on the ingress interface at r4 with the `detail` switch. The target address must be changed to reflect a local address at the monitoring node because transit traffic can not be monitored:

```
[edit]
lab@T1-P1# run telnet 10.0.3.4 source 130.130.0.1 port 5060
```

```
Trying 10.0.3.4...
telnet: connect to address 10.0.3.4: Connection refused
telnet: Unable to connect to remote host

[edit]
lab@r4# run monitor traffic interface so-0/1/0 detail
Listening on so-0/1/0, capture size 96 bytes

20:56:02.125705 Out IP (tos 0xc0, ttl 1, id 598, len 68) 10.0.2.6 > 224.0.0.5:
   OSPFv2-hello 48: rtrid 10.0.3.4 backbone E mask 255.255.255.252 int 10
   pri 128 dead 40 nbrs 10.0.3.3
20:56:04.852978  In IP (tos 0x10, ttl 63, id 62037, len 60) 130.130.0.1.3465 >
   10.0.3.4.5060: S 1914879600:1914879600(0) win 16384 <mss 1460,nop,wscale 0,
   nop,nop,timestamp 558487 0> (DF)
20:56:04.853052 Out IP (tos 0x0, ttl 64, id 601, len 40) 10.0.3.4.5060 >
   130.130.0.1.3465: R 0:0(0) ack 1914879601 win 0
20:56:05.412963  In IP (tos 0xc0, ttl 1, id 21661, len 68) 10.0.2.5 > 224.0.0.5:
   OSPFv2-hello 48: rtrid 10.0.3.3 backbone E mask 255.255.255.252 int 10
   pri 128 dead 40 nbrs 10.0.3.4
^C
4 packets received by filter
0 packets dropped by kernel
```

The highlighted portion of the capture shows that the TCP segment sent to port 5060 by the T1 peer is arriving at r4's core-facing interface with a Type of Service (ToS) setting of 0x10. When broken down into binary, this yields a 000 100 00 pattern (the added spacing helps to call out that the first three bits in ToS field code a precedence setting of 0). The next three bits code the Delay, Throughput, and Reliability indicators, which in this case are coded to indicate that delay is the optimal metric for this traffic (telnet, being interactive and character based, is a delay-sensitive application) The last two bits of the precedence/ToS file are reserved and set to zero. The traffic monitoring results also confirm that at this stage the DiffServ code points for VoIP-related TCP traffic, as sent by the T1 peer, are not being correctly written by r3. Note that for backward compatibility all DiffServ code points in the form of *xxx*000 are interpreted as IP ToS settings.

The output shown here provides definitive proof that, until you add the appropriate DSCP rewrite configuration to r3, your DSCP-based classification settings at r4 and r5 will not have the desired effect. Failing to achieve consistent traffic classification will result in exam point loss given the restrictions in effect for this example.

To rectify the situation, you associate the default DSCP rewrite table to the core-facing interfaces at r3:

```
[edit class-of-service interfaces at-0/1/0]
lab@r3# set unit 0 rewrite-rules dscp default
```

In the current CoS test bed, the ATM link between r3 and r5 supports a single VC that is configured for UBR operation with no ATM traffic shaping. When your goal is to provide CoS on a per-VC basis, you should configure VC-level traffic parameters and shaping, as opposed to using I/O manager (B-chip) based CoS. This is because I/O manager CoS does not understand individual VCs, and therefore can not provide the level of granularity needed for a VC-level CoS solution. The CoS solution demonstrated here is workable because a single ATM VC is defined, and only one of the two ports on the ATM PIC are in use (ATM-1 PICs share a single stream among all physical ports, unlike a PoS PIC, which uses one stream per port). Newer "Q-chip" based PICs, such as the ATM-2, can support as many as 4,096 queues per PIC. The presence of a single Frame Relay DLCI on the PoS link between r3 and r4 also means that there will be no issues with head-of-line blocking, as might occur if multiple DLCIs (or VLANs) are defined on a non Q-CHIP PIC.

A similar configuration is also needed for r3's so-0/2/0.100 interface (not shown). The completed class-of-service stanza is displayed at r3 and the changes are committed:

```
[edit class-of-service]
lab@r3# show
interfaces {
    so-0/2/0 {
        unit 100 {
            rewrite-rules {
                dscp default;
            }
        }
    }
    at-0/1/0 {
        unit 0 {
            rewrite-rules {
                dscp default;
            }
        }
    }
}

[edit class-of-service]
lab@r3# commit
commit complete
```

Confirming DSCP Rewrite

Confirming that r3 is correctly rewriting the DiffServ code points for traffic received from the T1 peer is rather straightforward. You start by displaying the rewrite rules that

are currently in effect for r3's core interfaces:

```
[edit class-of-service]
lab@r3# run show class-of-service interface at-0/1/0
Physical interface: at-0/1/0, Index: 16
  Scheduler map: <default>, Index: 1

  Logical interface: at-0/1/0.0, Index: 9
    Object         Name                   Type                       Index
    Rewrite        dscp-default           dscp                           1
    Rewrite        exp-default            exp                            2
    Classifier     ipprec-compatibility   ip                             5

[edit class-of-service]
lab@r3# run show class-of-service interface so-0/2/0
Physical interface: so-0/2/0, Index: 18
  Scheduler map: <default>, Index: 1

  Logical interface: so-0/2/0.100, Index: 10
    Object         Name                   Type                       Index
    Rewrite        dscp-default           dscp                           1
    Rewrite        exp-default            exp                            2
    Classifier     ipprec-compatibility   ip                             5
```

The highlights call out the correct application of the `dscp-default` rewrite table to both of r3's core-facing interfaces. Note that the only rewrite table present by default, the `exp-default` table, remains in effect for MPLS traffic despite the addition of a DiffServ rewrite table. The ultimate proof of your DiffServ rewrite "pudding" comes in the form of an incrementing EF queue counter at r4 when TCP traffic is generated to port 5060. Once again you begin confirmation testing by clearing all counters at r4 (not shown here) to make sure you have clean test results. After clearing the counters at r4, test traffic is again generated from the T1 peer:

```
[edit]
lab@T1-P1# run telnet 200.200.0.1 source 130.130.0.1 port 5060
Trying 200.200.0.1...
telnet: connect to address 200.200.0.1: Connection refused
telnet: Unable to connect to remote host

[edit]
lab@T1-P1# run ping 200.200.0.1 count 5 source 130.130.0.1
PING 200.200.0.1 (200.200.0.1): 56 data bytes
64 bytes from 200.200.0.1: icmp_seq=0 ttl=253 time=0.328 ms
64 bytes from 200.200.0.1: icmp_seq=1 ttl=253 time=0.247 ms
```

```
64 bytes from 200.200.0.1: icmp_seq=2 ttl=253 time=0.239 ms
64 bytes from 200.200.0.1: icmp_seq=3 ttl=253 time=0.240 ms
64 bytes from 200.200.0.1: icmp_seq=4 ttl=253 time=0.248 ms

--- 200.200.0.1 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.239/0.260/0.328/0.034 ms
```

Once the test traffic has been dispatched, the egress queue counters are again displayed at r4:

```
[edit]
lab@r4# run show interfaces fe-0/0/0 detail | find "Queue counters"
  Queue counters:       Queued packets  Transmitted packets      Dropped packets
    0 best-effort                    5                    5                    0
    1 expedited-fo                   1                    1                    0
    2 assured-forw                   0                    0                    0
    3 network-cont                   0                    0                    0
  Active alarms  : None
  Active defects : None

  . . .
```

The queue counts correspond nicely with your expectations, considering the type and volume of traffic you generated at T1. Although the correct DSCP rewrite functionality has been confirmed, you decide to conduct traffic monitoring at r4 (once again changing the telnet target to r4's lo0 address) for additional proof that all is well:

```
[edit]
lab@r4# run monitor traffic interface so-0/1/0 detail
Listening on so-0/1/0, capture size 96 bytes

21:27:39.633747 Out Call Ref: 75, MSG Type: 95 LOCK-SHIFT-5
                IE: 01 Len: 1, LINK VERIFY
                IE: 03 Len: 2, TX Seq: 237, RX Seq: 206
. . .
21:27:49.843860  In IP (tos 0xb8, ttl 63, id 62701, len 60) 130.130.0.1.4547 >
   10.0.3.4.5060: S 1402932182:1402932182(0) win 16384 <mss 1460,nop,wscale 0,
   nop,nop,timestamp 748985 0> (DF)
21:27:49.843923 Out IP (tos 0x0, ttl 64, id 1542, len 40) 10.0.3.4.5060 >
   130.130.0.1.4547: R 0:0(0) ack 1402932183 win 0
21:27:50.173959 Out IP (tos 0xc0, ttl 1, id 1543, len 68) 10.0.2.6 > 224.0.0.5:
   OSPFv2-hello 48: rtrid 10.0.3.4 backbone E mask 255.255.255.252 int 10
   pri 128 dead 40 nbrs 10.0.3.3
. . .
```

```
^C
14 packets received by filter
0 packets dropped by kernel
```

The highlighted entry provides an interesting contrast to the previous traffic monitoring capture. The IP ToS field is now coded to 0xB8, which breaks down as a binary 10111000; according to RFC 2598, "An Expedited Forwarding PHB," this pattern is used to identify the EF forwarding class. Although not shown, it is suggested that you also confirm correct classification and egress queuing behavior at `r5` by temporarily downing the Frame Relay link between `r3` and `r4` so that you can test the queue counters associated with `r5`'s so-0/1/0 interface.

Loss Priority

JUNOS software supports the tagging of packets with a loss priority indicator that functions in a manner similar to Frame Relay's Discard Eligibility (DE) bit, or ATM's Cell Loss Priority (CLP) bit. The trick to understanding the PLP bit lies in the understanding that it manifests itself differently internally versus externally. Internally, the PLP bit is indicated with proprietary data structures that can not be viewed directly; externally, a packet's PLP status is conveyed using specific settings in the IP precedence, MPLS EXP, or DSCP fields (oftentimes, PLP indication in downstream nodes requires marker rewrite tables in the local node), in *combination* with the downstream node's classification table. The last point bears additional focus because the default DSCP and IP precedence rewrite and classification tables do not support external communication of PLP status for the BE and EF traffic classes. This default behavior is in keeping with IETF DiffServ specifications, which leaves such matters to the control of each autonomous DiffServ domain. In contrast, the default MPLS EXP rewrite and classification tables do offer PLP support for all traffic classes with the default configuration (note that the default EXP classification table is not in effect by *default*, but the default MPLS EXP classification table supports PLP status recognition in all four forwarding classes).

Keep in mind that loss priority processing relates to both classification and marker rewrite operations, both of which have been covered in the material that has brought you to this stage. Although PLP handling does not warrant a complete section unto itself, a loss priority scenario is included to help reinforce the operation and configuration of BA classifiers and marker rewrites in the context of a DiffServ application that requires the differentiation of loss priority for the BE traffic class. To complete this section, you must reconfigure the routers in the CoS test bed to meet these additional requirements:

- Ensure that `r4` and `r5` are able to differentiate BE traffic received from the T1 peer based on its compliance with the configured policer.
- You must continue to use DSCP-based classification at `r4` and `r5`.

Configuring Loss Priority

To meet the required behavior, you have to modify the DSCP rewrite configuration at `r3` and the DSCP classification maps in place at `r4` and `r5`. Figure 6.2 helps you to understand what needs to be done by documenting the current handling of BE traffic being received from the T1 peer.

FIGURE 6.2 Default DSCP classification and rewrite functionality

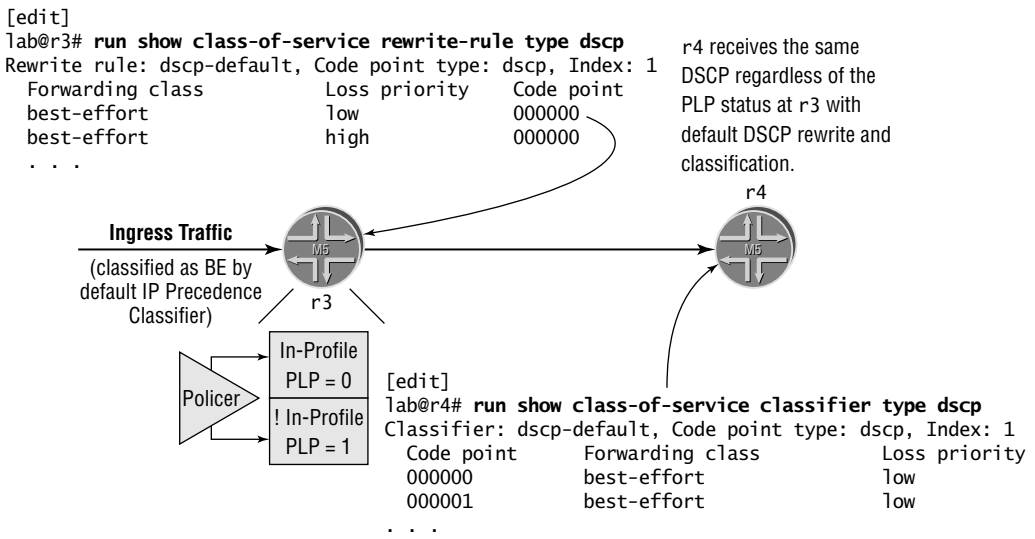


Figure 6.2 shows how non-VoIP traffic (classified as BE by the default IP precedence classifier) outside of the policer's profile is locally tagged with a PLP status of 1. The figure also shows that the default DSCP rewrite table at `r3` results in both in-profile and out-of-profile packets being sent to downstream routers with a DSCP setting of 0000000. The figure goes on to show how the default DSCP classifier in `r4` considers DSCPs 000000 and 000001 as equal, in that they both map to BE traffic with a low loss priority.

The solution demonstrated in this section modifies the DSCP rewrite at `r3` to use 000000 for in-profile traffic and 000001 for traffic in excess of the policer's profile. The changes at `r3` are only one-half of the story, however; you also need to adjust the DSCP classification at `r4` and `r5` to correctly interpret the 000000 and 000001 DSCP settings. You start at `r3` with the definition of a new DSCP rewrite table that is called *`dscp-plp`*:

```
[edit class-of-service]
lab@r3# edit rewrite-rules dscp dscp-plp

[edit class-of-service rewrite-rules dscp dscp-plp]
lab@r3# set import default

[edit class-of-service rewrite-rules dscp dscp-plp]
lab@r3# set forwarding-class best-effort loss-priority low code-point 000000

[edit class-of-service rewrite-rules dscp dscp-plp]
lab@r3# set forwarding-class best-effort loss-priority high code-point 000001
```

```
[edit class-of-service rewrite-rules dscp dscp-plp]
lab@r3# show
import default;
forwarding-class best-effort {
    loss-priority low code-point 000000;
    loss-priority high code-point 000001;
}
```

The `import` statement allows you to pre-populate a given rewrite table with the values present in another table; here the default DSCP rewrite table is used to pre-populate the new *`dscp-plp`* table. The remaining statements specify the desired code points for traffic that is classified as Best Effort, based on the local PLP indication of the traffic's status. The *`dscp-plp`* rewrite table is now applied to *both* of `r3`'s core-facing interfaces:

```
[edit class-of-service interfaces]
lab@r3# set at-0/1/0 unit 0 rewrite-rules dscp dscp-plp

[edit class-of-service interfaces]
lab@r3# set so-0/2/0 unit 100 rewrite-rules dscp dscp-plp
```

The configuration is displayed for visual confirmation of the changes:

```
[edit class-of-service interfaces]
lab@r3# show
so-0/2/0 {
    unit 100 {
        rewrite-rules {
            dscp dscp-plp;
        }
    }
}
at-0/1/0 {
    unit 0 {
        rewrite-rules {
            dscp dscp-plp;
        }
    }
}
```

Satisfied with the changes on `r3`, you commit the candidate configuration and direct your attention to `r5`. Here, a new DSCP-based classifier is created, which, in this example, also happens to be called *`dscp-plp`*:

```
[edit class-of-service classifiers dscp dscp-plp]
lab@r5# set import default
```

```
[edit class-of-service classifiers dscp dscp-plp]
lab@r5# set forwarding-class best-effort loss-priority low code-points 000000

[edit class-of-service classifiers dscp dscp-plp]
lab@r5# set forwarding-class best-effort loss-priority high code-points 000001
```

The newly defined *dscp-plp* classifier is displayed:

```
[edit class-of-service classifiers dscp dscp-plp]
lab@r5# show
import default;
forwarding-class best-effort {
    loss-priority low code-points 000000;
    loss-priority high code-points 000001;
}
```

The *dscp-plp* classifier is correctly applied to the r3-facing interface at r5:

```
[edit class-of-service]
lab@r5# set interfaces at-0/2/1 unit 0 classifiers dscp dscp-plp

[edit class-of-service]
lab@r5# show interfaces at-0/2/1
at-0/2/1 {
    unit 0 {
        classifiers {
            dscp dscp-plp;
        }
    }
}
```

Because a default CoS configuration does not contain any DSCP rewrite functionality, the DSCP marking at r3 should not be overwritten (or returned to a default value) as the traffic transits downstream routers. You could play it "safe" by creating a DSCP rewrite table at r5 that matches the *dscp-plp* table defined at r3 (for application to its so-0/1/0 interface). Such steps are not taken here because the default behavior makes such precautionary steps unnecessary.

Before proceeding to the next section, make sure that you have made similar adjustments to the DSCP classification settings at r4. Do not forget that the new classifier has to be added to both of r4's core-facing interfaces to ensure proper operation in the event of link failure between r3 and r4. The changes made to r4 are displayed for completeness' sake, with highlights added to call out recent changes:

```
[edit class-of-service]
lab@r4# show
classifiers {
    dscp dscp-plp {
```

```
        import default;
        forwarding-class best-effort {
            loss-priority low code-points 000000;
            loss-priority high code-points 000001;
        }
    }
}
interfaces {
    so-0/1/0 {
        unit 100 {
            classifiers {
                dscp dscp-plp;
            }
        }
    }
    so-0/1/1 {
        unit 0 {
            classifiers {
                dscp dscp-plp;
            }
        }
    }
}
```

Verifying Loss Priority

The ability to convey PLP status between routers using DSCP-based classification is confirmed in a number of ways. You start by displaying the new *dscp-plp* rewrite table at r3, while also confirming that it has been correctly applied to r3's output interfaces:

```
[edit]
lab@r3# run show class-of-service rewrite-rule name dscp-plp
Rewrite rule: dscp-plp, Code point type: dscp, Index: 23375
  Forwarding class                Loss priority      Code point
  best-effort                     low                000000
  best-effort                     high               000001
  expedited-forwarding            low                101110
  expedited-forwarding            high               101110
  assured-forwarding              low                001010
  assured-forwarding              high               001100
  network-control                 low                110000
  network-control                 high               111000
```

```
[edit]
lab@r3# run show class-of-service interface at-0/1/0
Physical interface: at-0/1/0, Index: 16
  Scheduler map: <default>, Index: 1

  Logical interface: at-0/1/0.0, Index: 9
    Object         Name                   Type                        Index
    Rewrite        dscp-plp               dscp                        23375
    Rewrite        exp-default            exp                             2
    Classifier     ipprec-compatibility   ip                              5

[edit]
lab@r3# run show class-of-service interface so-0/2/0
Physical interface: so-0/2/0, Index: 18
  Scheduler map: <default>, Index: 1

  Logical interface: so-0/2/0.100, Index: 10
    Object         Name                   Type                        Index
    Rewrite        dscp-plp               dscp                        23375
    Rewrite        exp-default            exp                             2
    Classifier     ipprec-compatibility   ip                              5
```

The displays indicate that the *dscp-plp* rewrite table contains the expected entries, and also shows the correct application of the rewrite table to both of r3's egress interfaces. The added highlights call out the portions of the rewrite table that were changed in support of your loss priority configuration. In the same vein, the contents of the *dscp-plp* classifier, and its correct application to ingress interfaces, are verified at r4:

```
[edit]
lab@r4# run show class-of-service classifier name dscp-plp
Classifier: dscp-plp, Code point type: dscp, Index: 23375
  Code point         Forwarding class                  Loss priority
  000000             best-effort                       low
  000001             best-effort                       high
  000010             best-effort                       low
  000011             best-effort                       low
  . . .
  111110             best-effort                       low
  111111             best-effort                       low

[edit]
lab@r4# run show class-of-service interface so-0/1/0
```

```
Physical interface: so-0/1/0, Index: 16
  Scheduler map: <default>, Index: 1

  Logical interface: so-0/1/0.100, Index: 9
    Object         Name                  Type                       Index
    Rewrite        exp-default           exp                            2
    Classifier     dscp-plp              dscp                       23375

[edit]
lab@r4# run show class-of-service interface so-0/1/1
Physical interface: so-0/1/1, Index: 17
  Scheduler map: <default>, Index: 1

  Logical interface: so-0/1/1.0, Index: 10
    Object         Name                  Type                       Index
    Rewrite        exp-default           exp                            2
    Classifier     dscp-plp              dscp                       23375
```

The captures taken from r4 confirm that consistent DiffServ code points have been configured for packet classification, and that the new classifier has been correctly applied to both of r4's ingress interfaces. In most cases, the confirmation steps shown thus far provide sufficient proof that DSCP marker rewrite and classification has been correctly configured. For those who have to "see to believe," there are a few confirmation options that remain. You could use traffic monitoring (or external test equipment) to capture traffic that egresses r3 in an attempt to spot packets with DSCPs coded to both 000000 and 000001. You could also configure an aggressive drop profile for high loss priority packets so that you can monitor drops in the corresponding queue while test traffic is generated. Yet another option involves the use of a firewall filter that is written to count packets with specific DSCP settings. While all of these approaches have their merits, this author has decided to demonstrate the traffic monitoring approach here. As described previously, your test traffic must be addressed to the device doing the monitoring. In this case, you start with a few small pings destined to r4. The low volume of traffic should not exceed the *be-policer*, so you expect to see DSCP codings of 000000 in the resulting traffic:

```
[edit]
lab@T1-P1# run ping 10.0.3.4 source 130.130.0.1 count 4
PING 10.0.3.4 (10.0.3.4): 56 data bytes
64 bytes from 10.0.3.4: icmp_seq=0 ttl=254 time=0.669 ms
64 bytes from 10.0.3.4: icmp_seq=1 ttl=254 time=0.561 ms
64 bytes from 10.0.3.4: icmp_seq=2 ttl=254 time=0.555 ms
64 bytes from 10.0.3.4: icmp_seq=3 ttl=254 time=0.510 ms

--- 10.0.3.4 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.510/0.574/0.669/0.058 ms
```

The traffic monitoring output that results at r4 is shown next:

```
[edit]
lab@r4# run monitor traffic interface so-0/1/0 detail
Listening on so-0/1/0, capture size 96 bytes

15:43:25.425995  In IP (tos 0x0, ttl 254, id 62966, len 84) 130.130.0.1 >
   10.0.3.4: icmp: echo request
15:43:25.426060 Out IP (tos 0x0, ttl 255, id 58518, len 84) 10.0.3.4 >
   130.130.0.1: icmp: echo reply
15:43:26.426399  In IP (tos 0x0, ttl 254, id 62968, len 84) 130.130.0.1 >
   10.0.3.4: icmp: echo request
15:43:26.426464 Out IP (tos 0x0, ttl 255, id 58521, len 84) 10.0.3.4 >
   130.130.0.1: icmp: echo reply
15:43:26.812786 Out IP (tos 0xc0, ttl 1, id 58522, len 68) 10.0.2.6 >
   224.0.0.5:  OSPFv2-hello 48: rtrid 10.0.3.4 backbone E mask 255.255.255.252
   int 10 pri 128 dead 40 nbrs 10.0.3.3
. . .
```

As expected, the IP ToS value is coded with all zeros indicating low loss priority. The ping test is now adjusted to use large packets and rapid transmission in an attempt to exceed the 1Mbps *best-effort* traffic policer in effect at r3:

```
[edit]
lab@T1-P1# run ping 10.0.3.4 source 130.130.0.1 count 40 size 1400 rapid
PING 10.0.3.4 (10.0.3.4): 1400 data bytes
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
--- 10.0.3.4 ping statistics ---
40 packets transmitted, 40 packets received, 0% packet loss
round-trip min/avg/max/stddev = 1.266/1.284/1.503/0.037 ms
```

The traffic monitoring output observed at r4, which has been edited for brevity, is shown next:

```
[edit]
lab@r4# run monitor traffic interface so-0/1/0 detail
Listening on so-0/1/0, capture size 96 bytes

15:46:41.244891 Out Call Ref: 75, MSG Type: 95 LOCK-SHIFT-5
                IE: 01 Len: 1, LINK VERIFY
                IE: 03 Len: 2, TX Seq: 101, RX Seq: 100

. . .
15:47:28.950328  In IP (tos 0x0, ttl 254, id 63067, len 1428) 130.130.0.1 >
   10.0.3.4: icmp: echo request
15:47:28.950400 Out IP (tos 0x0, ttl 255, id 58691, len 1428) 10.0.3.4 >
   130.130.0.1: icmp: echo reply
```

```
15:47:28.952074  In IP (tos 0x0, ttl 254, id 63068, len 1428) 130.130.0.1 >
   10.0.3.4: icmp: echo request
15:47:28.952116 Out IP (tos 0x0, ttl 255, id 58692, len 1428) 10.0.3.4 >
   130.130.0.1: icmp: echo reply
. . .
15:47:28.968976  In IP (tos 0x0, ttl 254, id 63079, len 1428) 130.130.0.1 >
   10.0.3.4: icmp: echo request
15:47:28.969015 Out IP (tos 0x0, ttl 255, id 58703, len 1428) 10.0.3.4 >
   130.130.0.1: icmp: echo reply
15:47:28.970504  In IP (tos 0x4, ttl 254, id 63080, len 1428) 130.130.0.1 >
   10.0.3.4: icmp: echo request
15:47:28.970547 Out IP (tos 0x4, ttl 255, id 58704, len 1428) 10.0.3.4 >
   130.130.0.1: icmp: echo reply
15:47:28.972036  In IP (tos 0x4, ttl 254, id 63081, len 1428) 130.130.0.1 >
   10.0.3.4: icmp: echo request
15:47:28.972076 Out IP (tos 0x4, ttl 255, id 58705, len 1428) 10.0.3.4 >
   130.130.0.1: icmp: echo reply
. . .
```

The added highlights call out the presence of ToS fields coded to both 0x00 and 0x04. Converting the 0x4 value into binary yields the sequence 0000 0100, which is the DSCP rewrite value configured at `r3` for BE traffic packet with a high loss priority. The operational mode displays and traffic monitoring results demonstrated in this section confirm that you have correctly configured your CoS test bed for the exchange of PLP status between routers using DSCP-based classification.

Rewrite/Marking Summary

The use of BA-based classification by core elements can only succeed when the traffic presented to the core contains the expected settings of the IP precedence, DSCP, MPLS EXP, or IEEE 802.1p code points (depending on which BA classification is in effect). Because most end-user applications will not set these bits in any predictable or consistent manner, you typically have to configure packet header rewrite (marking) functionality at the network's edge to ensure that packets are marked appropriately for BA-based classification action in the core.

Packet marking at the edge works in conjunction with the BA classifiers used by network elements that are downstream. In this section, the default DSCP classifier and rewrite table were shown to yield consistent classification. Albeit without support for PLP status indication for the BE traffic class. Where necessary, you can create custom rewrite tables, which in turn necessitate customized classifiers in downstream devices.

While loss priority is not specific to rewrite and marking, this section also detailed how a packet's PLP status is communicated between routers using various mechanisms such as DSCPs, IP Precedence, or MPLS EXP settings. Communicating PLP status between routers often involves customized marker functionality at the edge in conjunction with a modified classification table in core devices because only the MPLS EXP tables support PLP status for all four traffic classes by default.

Although the examples provided in this section were based on DSCP style markers, the techniques used to classify traffic and rewrite markers when using IP precedence, MPLS EXP, or IEEE 802.1p based CoS are very similar.

Use the `show class-of-service rewrite-rule` command to confirm the settings of the default and any user-configured rewrite tables.

Schedulers

Schedulers define a variety of parameters that control the servicing of egress queues. A scheduler block can specify a queue's priority class, how "leftover" bandwidth should be handled, the queue's transmission rate, the queue's buffer size (to control delay), and RED profiles (for congestion avoidance), among other things. Once a scheduler is defined, you use a `scheduler-map` to associate the scheduler with a given forwarding class (in other words, an output queue). The `scheduler-map` is then linked to one or more interfaces to affect the way in which that interface's queues are serviced.

The current CoS test bed has been configured with DSCP-based classification and rewrite functionality, but the lack of scheduler configuration means that no real CoS is provided in the current test bed. Note that the default scheduler configuration defines the BE forwarding class (queue 0) as having 95 percent of interface bandwidth and buffer space while the NC class (queue 3) is awarded the remaining 5 percent. This means that the current test bed offers no bandwidth or queue space to the EF forwarding class's queue 1! Once schedulers are added to the mix, you can actually expect to see differential service levels based on the traffic's forwarding class because the test bed will have a functional CoS configuration for traffic flowing in the T1-C1 direction.

- BE traffic limited to 10 percent of interface bandwidth.
- Ensure that BE traffic never exceeds the configured rate, even when other queues are empty.
- Configure the EF class to get 30 percent of interface bandwidth.
- Configure the EF class for a maximum queuing delay of 100 milliseconds.
- Ensure that the EF class has precedence over all other classes, regardless of the EF queue's current bandwidth credit.
- Make sure that your scheduler configuration does not adversely affect the operation of your routing protocols.

Configuring Schedulers

You begin scheduler configuration at `r3` with issuance of the commands needed to define the scheduler associated with BE traffic:

```
[edit]
lab@r3# edit class-of-service schedulers best-effort

[edit class-of-service schedulers best-effort]
lab@r3# set transmit-rate percent 10 exact
```

```
[edit class-of-service schedulers best-effort]
lab@r3# set buffer-size percent 10

[edit class-of-service schedulers best-effort]
lab@r3# set priority low
```

The Best Effort scheduler has its priority set to low, which means it will be serviced *only* after all strictly-high data has been sent and all high-priority queues *with positive credit* have been serviced. Because a priority value was not specified, you could have also specified high priority in this example, but the low priority setting seems the best fit, considering your overall goal of supporting real-time VoIP traffic in conjunction with conventional Internet traffic. The queue depth for BE traffic has been set to match the configured bandwidth, which is the recommended default setting. The `exact` keyword associated with the queue's transmit rate is critical in this case; its inclusion ensures that the associated traffic class can never exceed the configured transmission rate, even though all other queues may be empty (or possessing only negative credit). Use the `remainder` keyword to configure a scheduler to make use of unclaimed buffer space, as determined by the buffer settings of other schedulers. Note that the remaining keyword affects only the scheduler's buffer size for notification cells; by default a scheduler is able to avail itself of excess bandwidth when other schedulers are not operating at their full rate.

The *`best-effort`* scheduler is displayed at `r3`:

```
[edit class-of-service schedulers best-effort]
lab@r3# show
best-effort {
    transmit-rate percent 10 exact;
    buffer-size percent 10;
    priority low;
}
```

Scheduler configuration continues at `r3` with definition of the scheduler for the EF class. The completed *`expedited-forwarding`* scheduler is displayed:

```
[edit class-of-service schedulers]
lab@r3# show expedited-forwarding
expedited-forwarding {
    transmit-rate percent 30;
    buffer-size temporal 100000;
    priority strict-high;
}
```

The highlighted entries call out aspects of the EF scheduler that warrant additional focus. The `strict-high` priority setting is necessary to ensure that the EF queues will always be serviced before all other classes, even though the EF queues may have no positive credit. The buffer size has been configured to a depth of 100,000 microseconds through the use of the temporal keyword; this setting addresses the restriction that the EF queue be limited to no more than 100 milliseconds of queuing delay. The next set of commands creates a `scheduler-map` called *`jncie-cos`* that functions to associate the previously defined schedulers with the

BE and EF forwarding classes:

```
[edit class-of-service scheduler-maps]
lab@r3# set jncie-cos forwarding-class expedited-forwarding scheduler
   expedited-forwarding

[edit class-of-service scheduler-maps]
lab@r3# set jncie-cos forwarding-class best-effort scheduler best-effort
```

The *jncie-cos* scheduler-map is displayed for confirmation:

```
[edit class-of-service scheduler-maps]
lab@r3# show
jncie-cos {
    forwarding-class expedited-forwarding scheduler expedited-forwarding;
    forwarding-class best-effort scheduler best-effort;
}
```

The final step in scheduler configuration at r3 relates to the linking of the *jncie-cos* scheduler-map to the output interfaces at r3:

```
[edit class-of-service interfaces]
lab@r3# set at-0/1/0 scheduler-map jncie-cos

[edit class-of-service interfaces]
lab@r3# set so-0/2/0 scheduler-map jncie-cos
```

The resulting modifications are displayed and called out with added highlights:

```
[edit class-of-service interfaces]
lab@r3# show
so-0/2/0 {
    scheduler-map jncie-cos;
    unit 100 {
        rewrite-rules {
            dscp dscp-plp;
        }
    }
}
at-0/1/0 {
    scheduler-map jncie-cos;
    unit 0 {
        rewrite-rules {
            dscp dscp-plp;
        }
    }
}
```

Before proceeding to the confirmation section, similar changes are required at r4 and r5. The modifications made to the configuration of r4 are shown here with highlights. Note that the *jncie-cos* scheduler map is applied to the egress interface (fe-0/0/0) at r4:

```
[edit class-of-service]
lab@r4# show
classifiers {
    dscp dscp-plp {
        import default;
        forwarding-class best-effort {
            loss-priority low code-points 000000;
            loss-priority high code-points 000001;
        }
    }
}
interfaces {
    so-0/1/0 {
        unit 100 {
            classifiers {
                dscp dscp-plp;
            }
        }
    }
    so-0/1/1 {
        unit 0 {
            classifiers {
                dscp dscp-plp;
            }
        }
    }
    fe-0/0/0 {
        scheduler-map jncie-cos;
    }
}
scheduler-maps {
    jncie-cos {
        forwarding-class expedited-forwarding scheduler expedited-forwarding;
        forwarding-class best-effort scheduler best-effort;
    }
}
schedulers {
    best-effort {
        transmit-rate percent 10 exact;
```

```
        buffer-size percent 10;
        priority low;
    }
    expedited-forwarding {
        transmit-rate percent 30;
        buffer-size temporal 100000;
        priority strict-high;
    }
}
```

Confirming Schedulers

Confirmation of your scheduler configuration begins with verification that the *jncie-cos* `scheduler-map` has been correctly associated with each router's output interface, based on traffic flowing in the direction of T1 to C1:

```
[edit]
lab@r3# run show class-of-service interface so-0/2/0
Physical interface: so-0/2/0, Index: 18
  Scheduler map: jncie-cos, Index: 31932

  Logical interface: so-0/2/0.100, Index: 10
    Object         Name                   Type                  Index
    Rewrite        dscp-plp               dscp                  23375
    Rewrite        exp-default            exp                       2
    Classifier     ipprec-compatibility   ip                        5

[edit]
lab@r3# run show class-of-service interface at-0/1/0
Physical interface: at-0/1/0, Index: 16
  Scheduler map: jncie-cos, Index: 31932

  Logical interface: at-0/1/0.0, Index: 9
    Object         Name                   Type                  Index
    Rewrite        dscp-plp               dscp                  23375
    Rewrite        exp-default            exp                       2
    Classifier     ipprec-compatibility   ip                        5
```

The highlighted entries call out the correct application of the *jncie-cos* scheduler to the egress interfaces at `r3`. Although not shown, you may assume that similar confirmation is observed in relation to the output interfaces at `r4` and `r5`. The specific settings associated with a given scheduler can be viewed with a `show class-of-service scheduler-map` command, as executed here on `r5`:

```
[edit]
lab@r5# run show class-of-service scheduler-map
```

```
Scheduler map: jncie-cos, Index: 31932

  Scheduler: best-effort, Forwarding class: best-effort, Index: 61257
    Transmit rate: 10 percent, Rate Limit: exact, Buffer size: 10 percent,
       Priority: low
    Drop profiles:
      Loss priority   Protocol    Index    Name
      Low             non-TCP         1    <default-drop-profile>
      Low             TCP             1    <default-drop-profile>
      High            non-TCP         1    <default-drop-profile>
      High            TCP             1    <default-drop-profile>

  Scheduler: expedited-forwarding, Forwarding class: expedited-forwarding,
     Index: 13946
    Transmit rate: 30 percent, Rate Limit: none, Buffer size: 100000 us,
       Priority: strictly high
    Drop profiles:
      Loss priority   Protocol    Index    Name
      Low             non-TCP         1    <default-drop-profile>
      Low             TCP             1    <default-drop-profile>
      High            non-TCP         1    <default-drop-profile>
      High            TCP             1    <default-drop-profile>
```

The output confirms the correct transmit rate, priority, and buffer size settings for the BE and EF classes. Note that both classes currently use the default RED profile; custom RED profiles are configured in a subsequent step. Displaying the CoS entries installed in the forwarding table provide a different view of the parameters defined in a `scheduler-map`, as shown in this edited capture taken from `r5`:

```
[edit]
lab@r5# run show class-of-service forwarding-table scheduler-map | find so-0/1/0
Interface: so-0/1/0 (Index: 16, Map index: 31932, Num of queues: 2):
  Entry 0 (Scheduler index: 61257, Queue #: 0):
    Tx rate: 0 Kb (10%), Buffer size: 10 percent
    PLP high: 1, PLP low: 1, TCP PLP high: 1, TCP PLP low: 1
    Policy is exact
  Entry 1 (Scheduler index: 13946, Queue #: 1):
    Tx rate: 0 Kb (30%), Buffer size: 0 percent
    Strict-high priority is set
    PLP high: 1, PLP low: 1, TCP PLP high: 1, TCP PLP low: 1
. . .
```

The buffer size for the EF class (queue 1) is set to 0 in the forwarding table display due to the use of a temporal size definition in the EF scheduler. Similar information is also made available with a `show interfaces` command when the `extensive` switch is included, as

shown in this edited capture, which is again taken from r5:

```
[edit]
lab@r5# run show interfaces so-0/1/0 extensive | find "Packet Forwarding"
  Packet Forwarding Engine configuration:
    Destination slot: 0, PLP byte: 1 (0x00)
    CoS transmit queue              Bandwidth              Buffer Priority   Limit
                               %          bps   %          bytes
    0 best-effort             10     15552000  10              0      low   exact
    1 expedited-forwarding    30     46656000   0         100000     high    none
. . .
```

Verifying the forwarding characteristics of a CoS design without access to precision traffic generation and measurement test equipment is extremely difficult. In most cases, you simply have to "believe" that the configured behavior is, in fact, actually occurring. For example, even though this scenario used a relatively low transmission rate of 10 percent in conjunction with the exact keyword for the BE forwarding class, it will be difficult to generate enough ping traffic from another router to force packet loss; this is primarily due to the internal rate limiting in effect on the fxp1 interface that links the RE to the PFE complex. To help prove the validity of your CoS configuration, the transmission and buffer size percentages for the BE scheduler are temporarily reduced to 1 percent at r4. r4 is modified in this example because its Fast Ethernet interface represents the lowest bandwidth link in the CoS domain for traffic flowing in the direction of T1 to C1:

```
[edit class-of-service schedulers best-effort]
lab@r4# set transmit-rate percent 1

[edit class-of-service schedulers best-effort]
lab@r4# set buffer-size percent 1


[edit class-of-service schedulers best-effort]
lab@r4# show
best-effort {
    transmit-rate percent 1 exact;
    buffer-size percent 1;
    priority low;
}
```

With the change committed at r4, you clear r4's interface counters and generate rapid pings targeted to 200.200.0.1 using 15,000-byte packets simultaneously at both the T1 peer *and* r3; even with the extremely low transmit percentage that is now associated with the BE forwarding class, rapid pings generated from only one router do not result in packet loss.

```
[edit]
lab@T1-P1# run ping 200.200.0.1 source 130.130.0.1 count 2000 rapid size 15000
PING 200.200.0.1 (200.200.0.1): 15000 data bytes
```

```
!!!!.!!!!.!!!!.!!!!!!!!!!!!!!!!!!!!!!!!!!!!.!!!!.!!!!.!!!..!!!..!!!!.!!!!.!!!
   ..!!!!.!!!!.!!!!.!!!!.!!!!..!!!!.!!!!..!!!!.!!!!..!!!!!!!!!!!!!!!!!!!^C
--- 200.200.0.1 ping statistics ---
147 packets transmitted, 120 packets received, 18% packet loss
round-trip min/avg/max/stddev = 116.085/135.090/235.346/29.525 ms
```

While the ping traffic is being generated, you examine the queue and drop counters for the fe-0/0/0 interface at r4.

Two alternative ways of displaying this information is shown; the `show interfaces queue` command is particularly handy here, because it shows both the drop counts and the current rate of drops:

```
[edit]
lab@r4# run show interfaces fe-0/0/0 detail | find "Queue counters"
  Queue counters:       Queued packets  Transmitted packets      Dropped packets
    0 best-effort                75926                109322                3575
    1 expedited-fo                   0                     0                   0
    2 assured-forw                   0                     0                   0
    3 network-cont                  37                    37                   0
  Active alarms  : None
  Active defects : None
. . .

[edit]
lab@r4# run show interfaces queue fe-0/0/0 forwarding-class best-effort
Physical interface: fe-0/0/0, Enabled, Physical link is Up
  Interface index: 12, SNMP ifIndex: 25
Forwarding class: best-effort, Queue: 0
  Queued:
    Packets              :                76026                  49 pps
    Bytes                :            114343792              1077792 bps
  Transmitted:
    Packets              :                109546                 110 pps
    Bytes                :            110218480              1006256 bps
    Tail-dropped packets :                    0                    0 pps
    RED-dropped packets  :                 3591                    7 pps
     Low, non-TCP        :                 3591                    7 pps
     Low, TCP            :                    0                    0 pps
     High, non-TCP       :                    0                    0 pps
     High, TCP           :                    0                    0 pps
     Low, non-TCP        :              5757500                95048 bps
     Low, TCP            :                    0                    0 bps
```

```
  High, non-TCP       :                    0                    0 bps
  High, TCP           :                    0                    0 bps
RED-dropped bytes     :              5757500                95048 bps
```

The highlights in the resulting displays confirm that the test traffic is being correctly classified as Best Effort, and that queue drops are occurring. Although this test is far from scientific, the observation of some packet loss for BE traffic provides incremental confirmation that your CoS configuration is working as designed. Be sure that you restore the BE scheduler configuration at `r4` before proceeding to the next section:

```
[edit]
lab@r4# rollback 1
load complete

[edit]
lab@r4# show class-of-service schedulers best-effort
best-effort {
    transmit-rate percent 10 exact;
    buffer-size percent 10;
    priority low;
}

[edit]
lab@r4# commit
commit complete
```

Real World Scenario

A Potential Landmine

You may have noticed that the current scheduler configuration does not include a scheduler definition for network control (NC) traffic. When you apply a `scheduler-map` to an interface, you negate the default scheduler, which was providing queue 0 with 95 percent of the bandwidth (and queue depth) with the remainder going to queue 3 for NC traffic. Because the PFE affords unconfigured traffic classes with a modicum of transmission bandwidth (approximately .04 percent of the interface's WRR is provided to each undefined class) your network control traffic (in other words, IBGP) appears to be functional with the current configuration. However, it is highly recommended that you take the time to define a NC scheduler to ensure that network control packets are not starved for bandwidth during periods of high traffic volume, such as might occur after clearing your BGP neighbors in the face of a full BGP feed. The added highlights call out the modifications made to `r5`'s configuration in support of the NC class:

```
[edit class-of-service]
lab@r5# show
```

```
classifiers {
    dscp dscp-plp {
        import default;
        forwarding-class best-effort {
            loss-priority low code-points 000000;
            loss-priority high code-points 000001;
        }
    }
}
drop-profiles {
    be-low-plp {
        fill-level 70 drop-probability 1;
    }
    be-high-plp {
        fill-level 50 drop-probability 10;
    }
}
interfaces {
    at-0/2/1 {
        unit 0 {
            classifiers {
                dscp dscp-plp;
            }
        }
    }
    so-0/1/0 {
        scheduler-map jncie-cos;
    }
}
scheduler-maps {
    jncie-cos {
        forwarding-class expedited-forwarding scheduler expedited-forwarding;
        forwarding-class best-effort scheduler best-effort;
        forwarding-class network-control scheduler network-control;
    }
}
schedulers {
    best-effort {
        transmit-rate percent 10 exact;
        buffer-size percent 10;
        priority low;
        drop-profile-map loss-priority low protocol tcp drop-profile be-low-plp;
        drop-profile-map loss-priority high protocol tcp drop-profile be-high-plp;
    }
    expedited-forwarding {
        transmit-rate percent 30;
        buffer-size temporal 100000;
        priority strict-high;
    }
    network-control {
        transmit-rate percent 5;
        buffer-size percent 5;
        priority high;
    }
}
```

After committing the change, the NC scheduler is confirmed:

```
[edit class-of-service scheduler-maps]
lab@r5# run show interfaces so-0/1/0 detail extensive | find "Forwarding"
  Packet Forwarding Engine configuration:
    Destination slot: 0, PLP byte: 1 (0x00)
    CoS transmit queue              Bandwidth              Buffer Priority   Limit
                               %          bps   %          bytes
    0 best-effort             10     15552000  10              0      low   exact
    1 expedited-forwarding    30     46656000   0         100000     high    none
    3 network-control          5      7776000   5              0      low    none
. . .
```

It is critical to note that the *network-control* scheduler has been assigned a high priority. This setting is needed to ensure that network control traffic is not starved when excess amounts of EF class traffic are present. A strictly high scheduler can only lose the WRR selection algorithm to another high-priority scheduler; setting the network control scheduler to a high priority prevents problems with your network control traffic. You should make similar changes on r3 and r4 to prevent potential problems with network control traffic before proceeding.

RED Profiles

The RED algorithm is designed to prevent the global synchronization of TCP retransmissions, and the associated congestion-induced back-off procedures that can occur when packets are dropped from the end of a full queue (tail drops). Put simply, RED functions by monitoring the fill level of a queue to anticipate incipient congestion, which is then proactively reduced by performing drops from the head of the queue according to a drop probability that is indexed to the queue's fill level. RED is of most use when dealing with TCP-based applications because TCP reacts to packet loss by reducing the current transmission rate. A scheduler definition can be linked to two independent RED profiles; in such cases one of the profiles is referenced for packets with low loss priority while the other is used for packets with high loss priority.

To complete this section, you must reconfigure the routers in the CoS test bed according to these requirements:

- Drop 1 percent of the BE traffic with a low loss priority at 70 percent buffer fill.
- Drop 10 percent of the BE traffic with a high loss priority at 50 percent buffer fill.
- Do not alter the default RED profiles in effect for the EF and NC forwarding classes.
- Ensure that RED acts only on TCP-based BE traffic.

Configuring RED Profiles

To meet the required behavior, you need to define RED profiles that are linked to the *best-effort* scheduler. You start at r3 with the definition of a RED profile for low loss priority BE traffic:

```
[edit class-of-service drop-profiles]
lab@r3# set be-low-plp fill-level 70 drop-probability 1
```

The *be-low-plp* RED profile is displayed:

```
[edit class-of-service drop-profiles]
lab@r3# show
be-low-plp {
    fill-level 70 drop-probability 1;
}
```

The second RED profile is configured in a similar fashion; the completed profile is then displayed for confirmation:

```
[edit class-of-service drop-profiles]
lab@r3# show be-high-plp
be-high-plp {
    fill-level 50 drop-probability 10;
}
```

The custom RED profiles are now linked to the scheduler for the BE forwarding class:

```
[edit class-of-service]
lab@r3# edit schedulers best-effort

[edit class-of-service schedulers best-effort]
lab@r3#

[edit class-of-service schedulers best-effort]
lab@r3# set drop-profile-map loss-priority low protocol tcp drop-profile
   be-low-plp

[edit class-of-service schedulers best-effort]
lab@r3# set best-effort drop-profile-map loss-priority high protocol tcp
   drop-profile be-high-plp
```

Inclusion of the `tcp` flag ensures that the RED profiles are applied only to traffic that is designated as being TCP based, which is in accordance with the restrictions posed in this example. The modified *best-effort* scheduler is displayed with highlights added to call out recent modifications:

```
[edit class-of-service schedulers]
lab@r3# show
best-effort {
    transmit-rate percent 10 exact;
    buffer-size percent 10;
    priority low;
    drop-profile-map loss-priority low protocol tcp drop-profile be-low-plp;
    drop-profile-map loss-priority high protocol tcp drop-profile be-high-plp;
}
```

```
expedited-forwarding {
    transmit-rate percent 30;
    buffer-size temporal 100000;
    priority strict-high;
}
network-control {
    transmit-rate percent 5;
    buffer-size percent 5;
    priority high;
}
```

The changes are committed on r3, and similar modifications are now made on r4 and r5. The changes made to r4's configuration are shown with added highlights:

```
[edit class-of-service]
lab@r4# show
classifiers {
    dscp dscp-plp {
        import default;
        forwarding-class best-effort {
            loss-priority low code-points 000000;
            loss-priority high code-points 000001;
        }
    }
}
drop-profiles {
    be-low-plp {
        fill-level 70 drop-probability 1;
    }
    be-high-plp {
        fill-level 50 drop-probability 10;
    }
}
interfaces {
    so-0/1/0 {
        unit 100 {
            classifiers {
                dscp dscp-plp;
            }
        }
    }
    so-0/1/1 {
```

```
        unit 0 {
            classifiers {
                dscp dscp-plp;
            }
        }
    }
    fe-0/0/0 {
        scheduler-map jncie-cos;
    }
}
scheduler-maps {
    jncie-cos {
        forwarding-class expedited-forwarding scheduler expedited-forwarding;
        forwarding-class best-effort scheduler best-effort;
        forwarding-class network-control scheduler network-control;
    }
}
schedulers {
    best-effort {
        transmit-rate percent 10 exact;
        buffer-size percent 10;
        priority low;
        drop-profile-map loss-priority low protocol tcp drop-profile be-low-plp;
        drop-profile-map loss-priority high protocol tcp drop-profile be-high-plp;
    }
    expedited-forwarding {
        transmit-rate percent 30;
        buffer-size temporal 100000;
        priority strict-high;
    }
    network-control {
        transmit-rate percent 5;
        buffer-size percent 5;
        priority high;
    }
}
```

Verifying RED Profile

Use the `show class-of-service drop-profile` command to quickly display the default RED profile in addition to any user-configured profiles:

```
[edit]
lab@r4# run show class-of-service drop-profile
```

```
Drop profile: <default-drop-profile>, Type: discrete, Index: 1
  Fill level    Drop probability
         100                 100
Drop profile: be-high-plp, Type: discrete, Index: 19129
  Fill level    Drop probability
          50                  10
Drop profile: be-low-plp, Type: discrete, Index: 45288
  Fill level    Drop probability
          70                   1
```

The display, taken from r4, confirms that the *be-low-plp* and *be-high-plp* profiles correctly reflect the parameters specified for low and high loss priority traffic that is associated with the BE forwarding class. Displaying the scheduler maps with a `show class-of-service scheduler-map` command provides a convenient way of determining the drop profiles that are in use for each forwarding class:

```
[edit]
lab@r4# run show class-of-service scheduler-map
Scheduler map: jncie-cos, Index: 31932

  Scheduler: best-effort, Forwarding class: best-effort, Index: 61257
    Transmit rate: 10 percent, Rate Limit: exact, Buffer size: 10 percent,
       Priority: low
    Drop profiles:
      Loss priority   Protocol    Index    Name
      Low             non-TCP         1    <default-drop-profile>
      Low             TCP         45288    be-low-plp
      High            non-TCP         1    <default-drop-profile>
      High            TCP         19129    be-high-plp

  Scheduler: expedited-forwarding, Forwarding class: expedited-forwarding,
     Index: 13946
    Transmit rate: 30 percent, Rate Limit: none, Buffer size: 100000 us,
       Priority: strictly high
    Drop profiles:
      Loss priority   Protocol    Index    Name
      Low             non-TCP         1    <default-drop-profile>
      Low             TCP             1    <default-drop-profile>
      High            non-TCP         1    <default-drop-profile>
      High            TCP             1    <default-drop-profile>

  Scheduler: network-control, Forwarding class: network-control, Index: 38488
    Transmit rate: 5 percent, Rate Limit: none, Buffer size: 5 percent,
       Priority: high
```

```
Drop profiles:
  Loss priority   Protocol    Index    Name
  Low             non-TCP         1    <default-drop-profile>
  Low             TCP             1    <default-drop-profile>
  High            non-TCP         1    <default-drop-profile>
  High            TCP             1    <default-drop-profile>
```

The highlights call out the correct application of the new RED profiles to TCP traffic belonging to the BE forwarding class only. Note how non-TCP traffic, and the traffic associated with the EF and NC forwarding classes, continues to use the default RED profile. The results shown in this section confirm that you have configured RED and scheduler functionality in accordance with all requirements posed.

Combining this output with the results shown in previous sections of this chapter body indicate that you have met all requirements for the CoS configuration scenario. Congratulations!

Scheduler Summary

Scheduler configuration blocks specify parameters that relate to a queue's priority, transmit rate, buffer depth, and RED profiles. Once configured, you link the scheduler to one or more forwarding classes with a `scheduler-map`. The `scheduler-map` is then associated with one or more output interfaces to put the scheduler into effect for that interface's output queues. The default scheduler has support for the BE and NC classes only. Scheduling EF or AF forwarding classes requires explicit configuration.

You can specify a scheduler's queue depth as a percentage of the available buffer, or as a temporal value that reflects the maximum queuing delay. Generally speaking, time-sensitive traffic such as real-time voice should be configured with a shallow buffer to minimize the delay associated with the traffic; delivering VoIP traffic late is normally worse than not delivering the traffic at all. Low-priority schedulers are only serviced when no high-priority queues have positive credit, and when no strict-priority queue has data pending (regardless of its credit value). Note that the use of strict-priority can result in other queues being starved when there is a high volume of traffic in the corresponding forwarding class.

Do not forget that the application of a user-defined `scheduler-map` to an interface negates the default scheduler that would otherwise be in place. You must therefore use care to ensure that any user-defined schedulers adequately address all forwarding classes expected to egress on the associated interface.

You can confirm scheduler and RED profile settings with the `show class-of-service schedule-map` and `show class-of-service drop-profile` operational mode commands. Additional information about schedules that have been applied to interfaces can be displayed when the `extensive` switch is added to a `show interfaces` command.

Summary

This chapter provided an example of a typical JNCIE-level CoS configuration scenario. The chapter began with a packet classification example that was based on a multifield classifier in

the form of a firewall filter, and went on to show how marker rewrite actions were needed at edge devices to permit DSCP-based BA classification in core devices. Loss priority, and how a packet's PLP status is conveyed between routers, was also demonstrated.

The chapter went on to discuss schedulers, which define transmit rates, queue depths, scheduling priority, and indexes to RED profiles. The scheduler section showed how schedulers are linked to a forwarding class through a `scheduler-map`, and how the `scheduler-map` is also used to associate one or more schedulers with a given output interface. The chapter ended with a RED profile configuration example that resulted in packet discard behavior that was based, at least in part, on the local indication of a packet's loss priority.

Keep in mind that it can be all but impossible to empirically verify the effects of a CoS configuration unless you have access to sophisticated test equipment, or your test bed is equipped with low-speed interfaces (such as a DS1 providing a paltry 1.536Mbps of throughput) in the core. You should therefore be prepared to rely on operational mode command output, and the visual inspection of the CoS-related aspects of your configurations, when validating a CoS configuration in a lab examination setting.

On a final note, the lack of MPLS and IEEE 802.1p based CoS configuration scenarios in this chapter might be seen as a significant omission by some readers. The decision to focus on DSCP-based classification and marker rewrite in this chapter was intended to accommodate the widest CoS audience possible by taking into consideration that the majority of M-series customers that have *actually* deployed CoS have done so based on the DiffServ specifications. On the up side, CoS is CoS, and a classifier is a classifier. It is the opinion of this author that the reader who understands the DSCP-based CoS scenario presented in this chapter will not have significant difficulty applying the techniques and concepts demonstrated in this chapter to an MPLS or IEEE 802.1p based CoS design—assuming, of course, that the reader understands the generic aspects of that technology. To help prove this point, one of the "Spot the Issues" questions at the end of this chapter deals specifically with CoS in a MPLS environment.

Case Study: CoS

The CoS case study takes a unique approach that stems from the fact that an end-to-end CoS solution tends to be an "all or nothing" proposition. For example, rather than starting with a new baseline configuration that is CoS free, in this case the JNCIE candidate begins with the configuration left from the chapter body, with the goal being a replication of the existing CoS functionality in the *opposite* direction. This approach helps to reinforce the directionally significant aspects of CoS while also accommodating a straightforward and pedagogical chapter structure. While you can use the existing CoS configuration to help yourself complete the case study, it is suggested that you perform the case study without the use of cut and paste and with minimal "reverse engineering" of the existing configuration.

The CoS case study is performed on the existing topology; refer back to Figure 6.1 as needed. Once again it is expected that a prepared JNCIE candidate will be able to complete this case study in approximately one hour, with the resulting network meeting the majority of the specified behaviors and operational characteristics.

Listings that identify the changes made to the current configurations of all three routers in the CoS test bed are provided at the end of the case study. To accommodate differing configuration approaches, the output from various operational mode commands is included in the case study analysis section to permit the comparison of your network's operation to that of a known good example.

To complete the CoS case study, the routers comprising the CoS test bed must be configured to meet the criteria specified at the beginning of the chapter. However, you are now concerned with traffic flowing in the C1-to-T1 direction. The CoS configuration requirements listed here have been edited to reflect this change in directionality:

- SIP signaling (VoIP) uses TCP/UDP, port 5060.
- RTP media channels use UDP with port assignments in the range of 16,384–32,767.
- Classify all VoIP traffic as EF.
- Ensure that network control traffic continues to be classified as NC.
- Classify all remaining traffic with IP precedence 0 as BE.
- Your classification design must tolerate the failure of any single core interface or link.
- Police BE traffic to 1Mbps with excess data marked for discard.
- Configure `r4` so that traffic received from the C1 peer is classified according to a DSCP-based BA at `r3` and `r5`.
- Ensure that traffic received from the C1 peer is consistently classified by all network elements.
- Ensure that `r3` and `r5` are able to differentiate BE traffic received from the C1 peer based on its compliance with the configured policer.
- You must use DSCP-based classification at `r3` and `r5`.
- Configure schedulers in the CoS test bed according to these criteria:
  - BE traffic limited to 10 percent of interface bandwidth.
  - Ensure that BE traffic never exceeds the configured rate, even when other queues are empty.
  - Configure the EF class to get 30 percent of interface bandwidth.
  - Configure the EF class for a maximum queuing delay of 100 milliseconds.
  - Ensure that the EF class has precedence over all other classes, regardless of the EF queue's current bandwidth credit.
  - Make sure that your scheduler configuration does not adversely affect the operation of your routing protocols.
- Drop 1 percent of the BE traffic with a low loss priority at 70 percent buffer fill.
- Drop 10 percent of the BE traffic with a high loss priority at 50 percent buffer fill.
- Do not alter the default RED profiles in effect for the EF and NC forwarding classes.
- Ensure that RED acts only on TCP-based BE traffic.

You can assume that the T1 and C1 routers are correctly configured, and that you are not permitted to modify or view their configurations. You may telnet to these routers to perform connectivity testing as needed.

CoS Case Study Analysis

Each configuration requirement for the CoS case study is matched in the following to one or more valid router configurations and, as appropriate, examples of operational mode commands and sample output that confirm the network's operation adheres to all specified behaviors and restrictions.

The CoS case study analysis begins with the following criteria:

- SIP signaling (VoIP) uses TCP/UDP, port 5060.
- RTP media channels using UDP, ports 16,384–32,767.
- Classify all VoIP traffic as EF.
- Ensure that network control traffic continues to be classified as NC.
- Classify all remaining traffic with IP precedence 0 as BE.
- Police BE traffic to 1Mbps with excess data marked for discard.

The first grouping of criteria requires the configuration of multifield classification and traffic policing at the ingress node, which is now `r4`. The *`r4-voip-classifier`* filter shown here meets all requirements specified. Note that this filter uses a slightly different syntax from the filter that is in place at `r3`:

```
[edit firewall]
lab@r4# show
policer 1m {
    if-exceeding {
        bandwidth-limit 1m;
        burst-size-limit 15k;
    }
    then loss-priority high;
}
filter r4-voip-classifier {
    term 1 {
        from {
            protocol [ tcp udp ];
            port 5060;
        }
        then forwarding-class expedited-forwarding;
    }
    term 2 {
        from {
```

```
            protocol udp;
            port 16384-32767;
        }
        then forwarding-class expedited-forwarding;
    }
    term 3 {
        from {
            precedence [ net-control internet-control ];
        }
        then forwarding-class network-control;
    }
    term 4 {
        then {
            policer 1m;
            forwarding-class best-effort;
        }
    }
}

[edit]
lab@r4# show interfaces fe-0/0/0
unit 0 {
    family inet {
        filter {
            input r4-voip-classifier;
        }
        address 172.16.0.5/30;
    }
}
```

The primary difference between the two multifield classifiers relates to terms *3* and *4*, which in this example explicitly classify network control traffic (precedence settings 110 and 111) as Best Effort traffic. The filter in effect at `r3` relies on the default IP precedence classifier for the classification of NC and BE traffic. A quick confirmation of the filter's classification action is achieved by generating test traffic from C1 after clearing the interface counters associated with `r4`'s egress interface (not shown):

```
lab@c1> telnet 130.130.0.1 source 200.200.0.1 port 5060
Trying 130.130.0.1...
telnet: connect to address 130.130.0.1: Connection refused
telnet: Unable to connect to remote host

lab@c1> ping 130.130.0.1 source 200.200.0.1 tos 192 count 200 rapid
```

```
PING 130.130.0.1 (130.130.0.1): 56 data bytes
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
--- 130.130.0.1 ping statistics ---
200 packets transmitted, 200 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.225/0.231/0.366/0.011 ms
```

The (edited) classification results observed at r4 are shown:

```
[edit]
lab@r4# run show interfaces so-0/1/0 detail
Physical interface: so-0/1/0, Enabled, Physical link is Up
  Interface index: 16, SNMP ifIndex: 15, Generation: 15
  Link-level type: Frame-Relay, MTU: 4474, Clocking: Internal, SONET mode,
     Speed: OC3, Loopback: None,
  . . .
  Queue counters:       Queued packets  Transmitted packets      Dropped packets
    0 best-effort                    0                     0                    0
    1 expedited-fo                   1                     1                    0
    2 assured-forw                   0                     0                    0
    3 network-cont                 203                   203                    0
  SONET alarms    : None
  . . .
```

The highlighted queue counters accurately reflect the types of test traffic that was generated. If desired, you can confirm BE classification by omitting the `tos` argument to the previous `ping` command. These results confirm that multifield classification is operational at r4, which brings you to the next set of case study criteria:

- Configure r4 so that traffic received from the C1 peer is classified according to a DSCP-based BA at r3 and r5.
- Ensure that traffic received from the C1 peer is consistently classified by all network elements.
- Ensure that r3 and r5 are able to differentiate BE traffic received from the C1 peer based on its compliance with the configured policer.
- You must use DSCP-based classification at r3 and r5.
- Your classification design must tolerate the failure of any single core interface or link.

To achieve the functionality defined by this grouping of case study requirements, you need to configure DSCP rewrite at r4 and r5, and you need to configure r3 and r5 to perform DSCP-based BA classification on their ingress interfaces to ensure their ability to differentiate BE traffic based on loss priority. The changes made to r4 in support of these requirements are shown next:

```
[edit class-of-service]
lab@r4# show rewrite-rules
dscp r4-dscp-rewrite {
    import default;
```

```
    forwarding-class best-effort {
        loss-priority low code-point 000000;
        loss-priority high code-point 000001;
    }
}

[edit class-of-service]
lab@r4# show interfaces
so-0/1/0 {
    unit 100 {
        classifiers {
            dscp dscp-plp;
        }
        rewrite-rules {
            dscp r4-dscp-rewrite;
        }
    }
}
so-0/1/1 {
    unit 0 {
        classifiers {
            dscp dscp-plp;
        }
        rewrite-rules {
            dscp r4-dscp-rewrite;
        }
    }
}
fe-0/0/0 {
    scheduler-map jncie-cos;
}
```

Note that the *r4-dscp-rewrite* table is correctly applied to both of r4's egress interfaces, as needed to meet the classification redundancy stipulation. The changes made to r3 in support of DSCP-based BA classification are shown here:

```
[edit class-of-service]
lab@r3# show classifiers
dscp r3-dscp-classifier {
    import default;
    forwarding-class best-effort {
        loss-priority low code-points 000000;
        loss-priority high code-points 000001;
```

```
    }
}

[edit class-of-service]
lab@r3# show interfaces
so-0/2/0 {
    scheduler-map jncie-cos;
    unit 100 {
        classifiers {
            dscp r3-dscp-classifier;
        }
        rewrite-rules {
            dscp dscp-plp;
        }
    }
}
at-0/1/0 {
    scheduler-map jncie-cos;
    unit 0 {
        classifiers {
            dscp r3-dscp-classifier;
        }
        rewrite-rules {
            dscp dscp-plp;
        }
    }
}
```

Because r5 already has an appropriate DSCP classifier configured, you need only apply the existing classifier to its r4-facing interface to achieve the functionality required by the current set of case study criteria:

```
[edit class-of-service]
lab@r5# show interfaces so-0/1/0
so-0/1/0 {
    scheduler-map jncie-cos;
    unit 0 {
        classifiers {
            dscp dscp-plp;
        }
    }
}
```

A quick confirmation of the *r4-dscp-rewrite* table's contents, and its correct application to r4's output interfaces, is performed next:

```
[edit class-of-service]
lab@r4# run show class-of-service rewrite-rule name r4-dscp-rewrite
Rewrite rule: r4-dscp-rewrite, Code point type: dscp, Index: 48737
  Forwarding class                Loss priority       Code point
  best-effort                     low                 000000
  best-effort                     high                000001
  expedited-forwarding            low                 101110
  expedited-forwarding            high                101110
  assured-forwarding              low                 001010
  assured-forwarding              high                001100
  network-control                 low                 110000
  network-control                 high                111000

[edit class-of-service]
lab@r4# run show class-of-service interface so-0/1/0
Physical interface: so-0/1/0, Index: 16
  Scheduler map: <default>, Index: 1

  Logical interface: so-0/1/0.100, Index: 9
    Object         Name                 Type                      Index
    Rewrite        r4-dscp-rewrite      dscp                      48737
    Rewrite        exp-default          exp                           2

[edit class-of-service]
lab@r4# run show class-of-service interface so-0/1/1
Physical interface: so-0/1/1, Index: 17
  Scheduler map: <default>, Index: 1

  Logical interface: so-0/1/1.0, Index: 10
    Object         Name                 Type                      Index
    Rewrite        r4-dscp-rewrite      dscp                      48737
    Rewrite        exp-default          exp                           2
    Classifier     dscp-plp             dscp                      23375
```

A similar check is performed at r3 for the contents and application of the *r3-dscp-classifier*:

```
[edit class-of-service]
lab@r3# run show class-of-service classifier name r3-dscp-classifier
Classifier: r3-dscp-classifier, Code point type: dscp, Index: 37820
```

```
  Code point          Forwarding class                    Loss priority
  000000              best-effort                         low
  000001              best-effort                         high
  000010              best-effort                         low
  . . .
  111111              best-effort                         low

[edit class-of-service]
lab@r3# run show class-of-service interface at-0/1/0
Physical interface: at-0/1/0, Index: 16
  Scheduler map: jncie-cos, Index: 31932

  Logical interface: at-0/1/0.0, Index: 9
    Object         Name                  Type                    Index
    Rewrite        dscp-plp              dscp                    23375
    Rewrite        exp-default           exp                         2
    Classifier     r3-dscp-classifier    dscp                    37820

[edit class-of-service]
lab@r3# run show class-of-service interface so-0/2/0
Physical interface: so-0/2/0, Index: 18
  Scheduler map: jncie-cos, Index: 31932

  Logical interface: so-0/2/0.100, Index: 10
    Object         Name                  Type                    Index
    Rewrite        dscp-plp              dscp                    23375
    Rewrite        exp-default           exp                         2
    Classifier     r3-dscp-classifier    dscp                    37820
```

Although not shown here, a similar check of the DSCP classification table and interface application is performed at `r5`. These results indicate that you have correctly configured DSCP rewrite and DSCP-based BA classification according to the case study criteria. If time permits, you can verify that the PLP indication is correctly being set at ingress, and that `r4`'s DSCP rewrite table is operational by monitoring traffic in the core while rapid pings are generated at the C1 peer. Note that these pings must be targeted to the core router that is performing the traffic monitoring. You can tell that PLP setting and rewrite functionality is working when you see a mix of ToS values such those shown next in this edited capture:

```
[edit]
lab@r3# run monitor traffic interface so-0/2/0 detail
Listening on so-0/2/0, capture size 96 bytes

17:55:27.562622 Out IP (tos 0xc0, ttl 1, id 12161, len 68) 10.0.2.5 > 224.0.0.5:
   OSPFv2-hello 48: rtrid 10.0.3.3 backbone E mask 255.255.255.252 int 10
```

```
   pri 128 dead 40 nbrs 10.0.3.4
. . .
17:55:49.145108  In IP (tos 0x0, ttl 254, id 8712, len 1028) 200.200.0.1 >
   10.0.3.3: icmp: echo request
17:55:49.145141 Out IP (tos 0x0, ttl 255, id 12225, len 1028) 10.0.3.3 >
   200.200.0.1: icmp: echo reply
17:55:49.146184  In IP (tos 0x4, ttl 254, id 8713, len 1028) 200.200.0.1 >
   10.0.3.3: icmp: echo request
17:55:49.146218 Out IP (tos 0x4, ttl 255, id 12226, len 1028) 10.0.3.3 >
   200.200.0.1: icmp: echo reply
17:55:49.167686  In IP (tos 0x4, ttl 254, id 8714, len 1028) 200.200.0.1 >
   10.0.3.3: icmp: echo request
17:55:49.167721 Out IP (tos 0x4, ttl 255, id 12227, len 1028) 10.0.3.3 >
   200.200.0.1: icmp: echo reply
17:55:49.168788  In IP (tos 0x0, ttl 254, id 8715, len 1028) 200.200.0.1 >
   10.0.3.3: icmp: echo request
17:55:49.168822 Out IP (tos 0x0, ttl 255, id 12228, len 1028) 10.0.3.3 >
   200.200.0.1: icmp: echo reply
. . .
^C
75 packets received by filter
0 packets dropped by kernel
```

There is no easy way to confirm that the DSCP-based BA classifiers at `r3` and `r5` are actually mapping DSCP to a local loss priority. Visual confirmation of the DSCP classifier table, combined with the knowledge that the classifier is in effect on the ingress interfaces of `r3` and `r5`, provides sufficient confirmation at this time. With marker rewrite and policer-based setting of PLP status confirmed, you move on to address these scheduler-related criteria:

- Configure schedulers in the CoS test bed according to these criteria:
  - BE traffic limited to 10 percent of interface bandwidth.
  - Ensure that BE traffic never exceeds the configured rate, even when other queues are empty.
  - Configure the EF class to get 30 percent of interface bandwidth.
  - Configure the EF class for a maximum queuing delay of 100 milliseconds.
  - Ensure that the EF class has precedence over all other classes, regardless of the EF queue's current bandwidth credit.
  - Make sure your scheduler configuration does not adversely affect the operation of your routing protocols.

Because schedulers were defined in the chapter body that meet the specified criteria, the only requirement needed to meet the case study criteria is to apply the existing `scheduler-map` to the interfaces that act as egress for traffic flowing in the direction of C1 to T1. For added practice, you can opt to define a new scheduler with a different name. The changes made to `r3`

and r5 in support of the "reapply the existing scheduler" approach are shown next:

```
[edit class-of-service]
lab@r3# show interfaces
so-0/2/0 {
    scheduler-map jncie-cos;
    unit 100 {
        classifiers {
            dscp r3-dscp-classifier;
        }
        rewrite-rules {
            dscp dscp-plp;
        }
    }
}
at-0/1/0 {
    scheduler-map jncie-cos;
    unit 0 {
        classifiers {
            dscp r3-dscp-classifier;
        }
        rewrite-rules {
            dscp dscp-plp;
        }
    }
}
fe-0/0/2 {
    scheduler-map jncie-cos;
}

[edit class-of-service]
lab@r5# show interfaces
at-0/2/1 {
    scheduler-map jncie-cos;
    unit 0 {
        classifiers {
            dscp dscp-plp;
        }
    }
}
so-0/1/0 {
    scheduler-map jncie-cos;
    unit 0 {
```

```
        classifiers {
            dscp dscp-plp;
        }
    }
}
```

Note that the *jncie-cos* scheduler-map must be applied to both of r4's egress interfaces (not shown). Confirmation that the scheduler-map is correctly applied to an interface is straightforward; this example is taken from r4:

```
[edit class-of-service]
lab@r4# run show class-of-service interface so-0/1/0
Physical interface: so-0/1/0, Index: 16
  Scheduler map: jncie-cos, Index: 31932

  Logical interface: so-0/1/0.100, Index: 9
    Object         Name                  Type                  Index
    Rewrite        r4-dscp-rewrite       dscp                  48737
    Rewrite        exp-default           exp                       2
    Classifier     dscp-plp              dscp                  23375
```

Use a show interface extensive to quickly confirm the transmit rate and buffer settings for the associated forwarding classes:

```
[edit class-of-service]
lab@r3# run show interfaces fe-0/0/2 extensive | find "Packet Forwarding"
  Packet Forwarding Engine configuration:
    Destination slot: 0
    CoS transmit queue              Bandwidth              Buffer Priority   Limit
                              %           bps   %          bytes
    0 best-effort            10      10000000  10              0      low   exact
    1 expedited-forwarding   30      30000000   0         100000     high    none
    3 network-control         5       5000000   5              0     high    none
  . . .
```

Visual inspection of the scheduler definition, coupled with the captures shown previously, confirm that you have correctly applied a pre-existing scheduler-map to a new set of output interfaces for the routers in the CoS test bed. This brings you to the final set of case study criteria:

- Drop 1 percent of the BE traffic with a low loss priority at 70 percent buffer fill.
- Drop 10 percent of the BE traffic with a high loss priority at 50 percent buffer fill.
- Do not alter the default RED profiles in effect for the EF and NC forwarding classes.
- Ensure that RED acts only on TCP-based BE traffic.

As with the scheduler-related criteria addressed previously, all routers in the CoS test bed already have the required drop profiles configured. Recalling that the custom RED drop profiles are indexed by the *best-effort* scheduler, which in turn is indexed by the *jncie-cos*

scheduler-map, you suddenly realize that you have indirectly achieved the RED-related requirements by virtue of the *jncie-cos* scheduler-map's application to the new set of egress interfaces. Your realization is correct, and better yet, it is easy to confirm.

```
[edit class-of-service]
lab@r5# run show class-of-service scheduler-map
Scheduler map: jncie-cos, Index: 31932

  Scheduler: best-effort, Forwarding class: best-effort, Index: 61257
    Transmit rate: 10 percent, Rate Limit: exact, Buffer size: 10 percent,
       Priority: low
    Drop profiles:
      Loss priority   Protocol    Index    Name
      Low             non-TCP         1    <default-drop-profile>
      Low             TCP         45288    be-low-plp
      High            non-TCP         1    <default-drop-profile>
      High            TCP         19129    be-high-plp

  Scheduler: expedited-forwarding, Forwarding class: expedited-forwarding,
     Index: 13946
    Transmit rate: 30 percent, Rate Limit: none, Buffer size: 100000 us,
       Priority: strictly high
    Drop profiles:
      Loss priority   Protocol    Index    Name
      Low             non-TCP         1    <default-drop-profile>
      Low             TCP             1    <default-drop-profile>
      High            non-TCP         1    <default-drop-profile>
      High            TCP             1    <default-drop-profile>

  Scheduler: network-control, Forwarding class: network-control, Index: 38488
    Transmit rate: 5 percent, Rate Limit: none, Buffer size: 5 percent,
       Priority: low
    Drop profiles:
      Loss priority   Protocol    Index    Name
      Low             non-TCP         1    <default-drop-profile>
      Low             TCP             1    <default-drop-profile>
      High            non-TCP         1    <default-drop-profile>
      High            TCP             1    <default-drop-profile>
```

The output confirms the use of the *be-low-plp* and *be-high-plp* drop profiles by the *jncie-cos* scheduler-map for the BE class only. You next confirm that the *jncie-cos* scheduler-map is correctly applied to the new output interface at r5:

```
[edit class-of-service]
lab@r5# run show class-of-service interface at-0/2/1
Physical interface: at-0/2/1, Index: 21
```

```
  Scheduler map: jncie-cos, Index: 31932

  Logical interface: at-0/2/1.0, Index: 8
    Object         Name                    Type                    Index
    Rewrite        exp-default             exp                         2
    Classifier     dscp-plp                dscp                    23375
```

These results confirm proper drop-profile configuration and application, which brings the case study analysis section to a close. If time permits, it is suggested that you perform traffic monitoring and queue counter analysis for r3's egress interface, which generates test traffic from the C1 peer. Such a test confirms the end-to-end aspects of your CoS configuration.

CoS Case Study Configurations

The changes made to the OSPF baseline network topology to support CoS are listed next for all routers in the CoS test bed. The added highlights in Listings 6.1 through 6.3 call out the CoS-related changes that were added in support of the chapter's case study.

r1 was not part of the CoS test bed. No changes were made to its configuration in support of this chapter's case study.

r2 was not part of the CoS test bed. No changes were made to its configuration in support of this chapter's case study.

Listing 6.1: CoS Case Study Configuration for *r3*

```
[edit]
lab@r3# show class-of-service
classifiers {
    dscp r3-dscp-classifier {
        import default;
        forwarding-class best-effort {
            loss-priority low code-points 000000;
            loss-priority high code-points 000001;
        }
    }
}
drop-profiles {
    be-low-plp {
        fill-level 70 drop-probability 1;
    }
    be-high-plp {
        fill-level 50 drop-probability 10;
    }
}
interfaces {
    so-0/2/0 {
```

```
        scheduler-map jncie-cos;
        unit 100 {
            classifiers {
                dscp r3-dscp-classifier;
            }
            rewrite-rules {
                dscp dscp-plp;
            }
        }
    }
    at-0/1/0 {
        scheduler-map jncie-cos;
        unit 0 {
            classifiers {
                dscp r3-dscp-classifier;
            }
            rewrite-rules {
                dscp dscp-plp;
            }
        }
    }
    fe-0/0/2 {
        scheduler-map jncie-cos;
    }
}
rewrite-rules {
    dscp dscp-plp {
        import default;
        forwarding-class best-effort {
            loss-priority low code-point 000000;
            loss-priority high code-point 000001;
        }
    }
}
scheduler-maps {
    jncie-cos {
        forwarding-class expedited-forwarding scheduler expedited-forwarding;
        forwarding-class best-effort scheduler best-effort;
        forwarding-class network-control scheduler network-control;
    }
}
```

```
schedulers {
    best-effort {
        transmit-rate percent 10 exact;
        buffer-size percent 10;
        priority low;
        drop-profile-map loss-priority low protocol tcp drop-profile be-low-plp;
        drop-profile-map loss-priority high protocol tcp drop-profile be-high-plp;
    }
    expedited-forwarding {
        transmit-rate percent 30;
        buffer-size temporal 100000;
        priority strict-high;
    }
    network-control {
        transmit-rate percent 5;
        buffer-size percent 5;
        priority high;
    }
}

[edit]
lab@r3# show firewall
policer be-policer {
    if-exceeding {
        bandwidth-limit 1m;
        burst-size-limit 15k;
    }
    then loss-priority high;
}
filter classify {
    term sip {
        from {
            protocol [ udp tcp ];
            port 5060;
        }
        then {
            forwarding-class expedited-forwarding;
            accept;
        }
    }
    term rtp {
```

```
        from {
            protocol udp;
            port 16384-32767;
        }
        then {
            forwarding-class expedited-forwarding;
            accept;
        }
    }
    term be {
        then policer be-policer;
    }
}

[edit]
lab@r3# show interfaces fe-0/0/2
unit 0 {
    family inet {
        filter {
            input classify;
        }
        address 172.16.0.13/30;
    }
}
```

Listing 6.2: CoS Case Study Configuration for *r4*

```
[edit]
lab@r4# show class-of-service
classifiers {
    dscp dscp-plp {
        import default;
        forwarding-class best-effort {
            loss-priority low code-points 000000;
            loss-priority high code-points 000001;
        }
    }
}
drop-profiles {
    be-low-plp {
        fill-level 70 drop-probability 1;
    }
```

```
    be-high-plp {
        fill-level 50 drop-probability 10;
    }
}
interfaces {
    so-0/1/0 {
        scheduler-map jncie-cos;
        unit 100 {
            classifiers {
                dscp dscp-plp;
            }
            rewrite-rules {
                dscp r4-dscp-rewrite;
            }
        }
    }
    so-0/1/1 {
        scheduler-map jncie-cos;
        unit 0 {
            classifiers {
                dscp dscp-plp;
            }
            rewrite-rules {
                dscp r4-dscp-rewrite;
            }
        }
    }
    fe-0/0/0 {
        scheduler-map jncie-cos;
    }
}
rewrite-rules {
    dscp r4-dscp-rewrite {
        import default;
        forwarding-class best-effort {
            loss-priority low code-point 000000;
            loss-priority high code-point 000001;
        }
    }
}
```

```
scheduler-maps {
    jncie-cos {
        forwarding-class expedited-forwarding scheduler expedited-forwarding;
        forwarding-class best-effort scheduler best-effort;
        forwarding-class network-control scheduler network-control;
    }
}
schedulers {
    best-effort {
        transmit-rate percent 10 exact;
        buffer-size percent 10;
        priority low;
        drop-profile-map loss-priority low protocol tcp drop-profile be-low-plp;
        drop-profile-map loss-priority high protocol tcp drop-profile be-high-plp;
    }
    expedited-forwarding {
        transmit-rate percent 30;
        buffer-size temporal 100000;
        priority strict-high;
    }
    network-control {
        transmit-rate percent 5;
        buffer-size percent 5;
        priority high;
    }
}

[edit]
lab@r4# show firewall
policer 1m {
    if-exceeding {
        bandwidth-limit 1m;
        burst-size-limit 15k;
    }
    then loss-priority high;
}
filter r4-voip-classifier {
    term 1 {
        from {
            protocol [ tcp udp ];
            port 5060;
```

```
        }
        then forwarding-class expedited-forwarding;
    }
    term 2 {
        from {
            protocol udp;
            port 16384-32767;
        }
        then forwarding-class expedited-forwarding;
    }
    term 3 {
        from {
            precedence [ net-control internet-control ];
        }
        then forwarding-class network-control;
    }
    term 4 {
        then {
            policer 1m;
            forwarding-class best-effort;
        }
    }
}

[edit]
lab@r4# show interfaces fe-0/0/0
unit 0 {
    family inet {
        filter {
            input r4-voip-classifier;
        }
        address 172.16.0.5/30;
    }
}
```

Listing 6.3: CoS Case Study Configuration for *r5*

```
[edit]
lab@r5# show class-of-service
classifiers {
    dscp dscp-plp {
        import default;
```

```
        forwarding-class best-effort {
            loss-priority low code-points 000000;
            loss-priority high code-points 000001;
        }
    }
}
drop-profiles {
    be-low-plp {
        fill-level 70 drop-probability 1;
    }
    be-high-plp {
        fill-level 50 drop-probability 10;
    }
}
interfaces {
    at-0/2/1 {
        scheduler-map jncie-cos;
        unit 0 {
            classifiers {
                dscp dscp-plp;
            }
        }
    }
    so-0/1/0 {
        scheduler-map jncie-cos;
        unit 0 {
            classifiers {
                dscp dscp-plp;
            }
        }
    }
}
scheduler-maps {
    jncie-cos {
        forwarding-class expedited-forwarding scheduler expedited-forwarding;
        forwarding-class best-effort scheduler best-effort;
        forwarding-class network-control scheduler network-control;
    }
}
schedulers {
```

```
    best-effort {
        transmit-rate percent 10 exact;
        buffer-size percent 10;
        priority low;
        drop-profile-map loss-priority low protocol tcp drop-profile be-low-plp;
        drop-profile-map loss-priority high protocol tcp drop-profile be-high-plp;
    }
    expedited-forwarding {
        transmit-rate percent 30;
        buffer-size temporal 100000;
        priority strict-high;
    }
    network-control {
        transmit-rate percent 5;
        buffer-size percent 5;
        priority high;
    }
}
```

Note that `r5`'s configuration does not include any multifield classification or marker rewrite functionality, which is in keeping with its placement in the network's core.

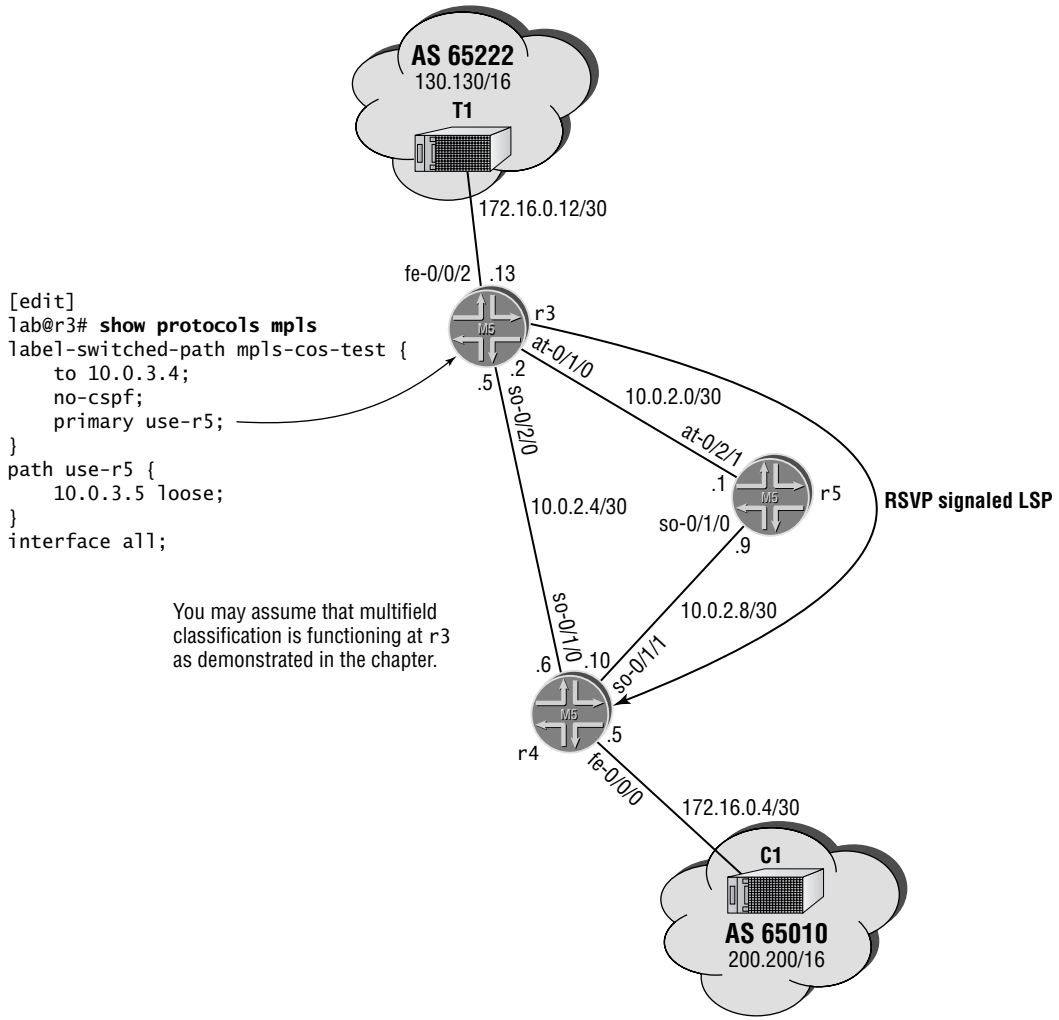
`r6` was not part of the IPv6 test bed. No changes were made to its configuration in support of this chapter's case study.

`r7` was not part of the IPv6 test bed. No changes were made to its configuration in support of this chapter's case study.

Spot the Issues: Review Questions

1. You are trying to configure MPLS CoS according to the topology shown in Figure 6.3.

FIGURE 6.3 CoS topology



The LSP has been established, and traceroutes from T1 to C1 confirm that LSP forwarding is operational. While the queue counters on `r5`'s so-0/1/0 interface show the expected forwarding class mappings, `r4`'s fe-0/0/0 queue counters indicate that all EF and BE traffic is being placed into queue 0. Can you spot the problem given the CoS-related configuration stanzas shown next?

```
[edit]
lab@r3# show class-of-service

[edit]
lab@r3#

[edit]
lab@r4# show class-of-service
interfaces {
    so-0/1/0 {
        unit 0 {
            classifiers {
                exp default;
            }
        }
    }
}

[edit]
lab@r4#
[edit]
lab@r5# show class-of-service
interfaces {
    at-0/2/1 {
        unit 0 {
            classifiers {
                exp default;
            }
        }
    }
}
```

2. Can you think of another way to address the classification problem detailed in the previous question, one that does not involve rewrite functionality at `r5`?

3. This scheduler block is intended to provide real-time traffic, which is classified as EF, with low latency and low loss. The user is complaining about poor media quality. Can you spot any problems with the scheduler configuration?

```
[edit]
lab@r4# show class-of-service schedulers
best-effort {
    transmit-rate percent 80 exact;
    buffer-size percent 15;
```

```
        priority low;
        drop-profile-map loss-priority low protocol tcp drop-profile be-low-plp;
        drop-profile-map loss-priority high protocol tcp drop-profile be-high-plp;
    }
    expedited-forwarding {
        transmit-rate percent 15;
        buffer-size percent 80;
        priority high;
    }
    network-control {
        transmit-rate percent 5;
        buffer-size percent 5;
        priority high;
}
```

4. List the default classifier and rewrite functionality that is associated with an M-series router that has no explicit CoS configuration.

Spot the Issues: Answers to Review Questions

1. This is a tricky question, especially without access to the actual routers! For a hint, consider the operational output obtained at r4:

```
[edit]
lab@r4# run show class-of-service interface so-0/1/1
Physical interface: so-0/1/1, Index: 16
  Scheduler map: <default>, Index: 1

  Logical interface: so-0/1/1.100, Index: 11
    Object         Name                   Type                        Index
    Rewrite        exp-default            exp                             2
    Classifier     ipprec-compatibility   ip                              5
```

If you are thinking that the default EXP classifier will have no effect on the traffic arriving at r4, due to the penultimate hop-popping (PHP) behavior at r5, which results in the receipt of unlabeled packets at r4, then you are definitely getting warm! The only other classifier in place also has no practical effect on the incoming traffic because the default IP precedence classifier supports only the BE and NC forwarding classes. The solution shown next resolves the issue with an IP precedence rewrite function at the penultimate node and a matching IP precedence classifier at the egress node. Note that the lack of an IP precedence rewrite function at r4 will result in C1 receiving IP packets with precedence settings that differ from those generated by T1:

```
[edit class-of-service]
lab@r4# show
classifiers {
    inet-precedence php-classifier {
        import default;
        forwarding-class best-effort {
            loss-priority low code-points 000;
            loss-priority high code-points 001;
        }
        forwarding-class expedited-forwarding {
            loss-priority low code-points 010;
        }
    }
}
interfaces {
    so-0/1/1 {
        unit 0 {
```

```
            classifiers {
                inet-precedence php-classifier;
            }
        }
    }
}

[edit class-of-service]
lab@r5# show
interfaces {
    at-0/2/1 {
        unit 0 {
            classifiers {
                exp default;
            }
        }
    }
    so-0/1/0 {
        unit 0 {
            rewrite-rules {
                inet-precedence php-classifier;
            }
        }
    }
}
rewrite-rules {
    inet-precedence php-classifier {
        import default;
        forwarding-class best-effort {
            loss-priority low code-point 000;
            loss-priority high code-point 001;
        }
        forwarding-class expedited-forwarding {
            loss-priority low code-point 010;
        }
    }
}
```

A code point for EF traffic with high loss priority was not assigned because this traffic is not being marked with a PLP status.

2. Because the problem relates to the default MPLS PHP behavior, another solution is to configure r4 to request an explicit null label, as shown here:

```
[edit]
lab@r4# show protocols mpls
explicit-null;
interface all;

[edit]
lab@r4# show class-of-service
interfaces {
    so-0/1/1 {
        unit 0 {
            classifiers {
                exp default;
            }
        }
    }
}

[edit]
lab@r5# show class-of-service
interfaces {
    at-0/2/1 {
        unit 0 {
            classifiers {
                exp default;
            }
        }
    }
}
```

By configuring the exchange of labeled packets on the LSPs final hop, you allow the default EXP rewrite table, which is in effect on r5's output interface, to work with the explicitly applied default EXP classifier at r4 to correctly classify traffic originating at the T1 peer. Note that their configurations no longer require custom rewrite or classification functionality.

3. The problem with the configuration relates to the buffer depths that have been assigned to the BE and EF classes, respectively. For real-time traffic, loss is often preferred over high level of latency, as the latter normally results in discard by the end-user application anyway. With this configuration, EF traffic could experience a significant queuing delay when the aggregate traffic load for the interface exceeds its output capacity. The changes shown next will reduce queuing delays at the

expense of traffic discard during periods of congestion, which is a behavior that is generally desirable for real-time traffic such as voice:

```
[edit]
lab@r4# show class-of-service schedulers
best-effort {
    transmit-rate percent 80 exact;
    buffer-size percent 80;
    priority low;
    drop-profile-map loss-priority low protocol tcp drop-profile be-low-plp;
    drop-profile-map loss-priority high protocol tcp drop-profile be-high-plp;
}
expedited-forwarding {
    transmit-rate percent 15;
    buffer-size percent 15;
    priority high;
}
network-control {
    transmit-rate percent 5;
    buffer-size percent 5;
    priority high;
}
```

If the increased loss becomes an issue, then you might consider changing the EF class to strict-high priority and/or making transmit percentage adjustments to afford the EF class with a higher transmit capacity.

4. A factory default configuration, which contains no explicit CoS settings, supports packet classification using the `ipprec-compatibility` table and MPLS EXP rewrite using the `exp-default` table only.

Chapter

7

VPNs

JNCIE LAB SKILLS COVERED IN THIS CHAPTER:

- ✓ **Layer 3 VPNs (2547 bis)**
  - Preliminary Configuration
  - PE-CE BGP and Static Routing
  - PE-CE OSPF Routing
- ✓ **Layer 2 VPNs**
  - Draft-Kompella with Non-VRF Internet Access
  - Draft-Martini

This chapter exposes the reader to several JNCIE-level provider provisioned (PP) virtual private networking (VPN) configuration scenarios, all of which employ some form of MPLS technology for forwarding VPN traffic across the provider's backbone. A provider provisioned VPN solution can take the form of a Layer 3 or Layer 2 VPN service offering.

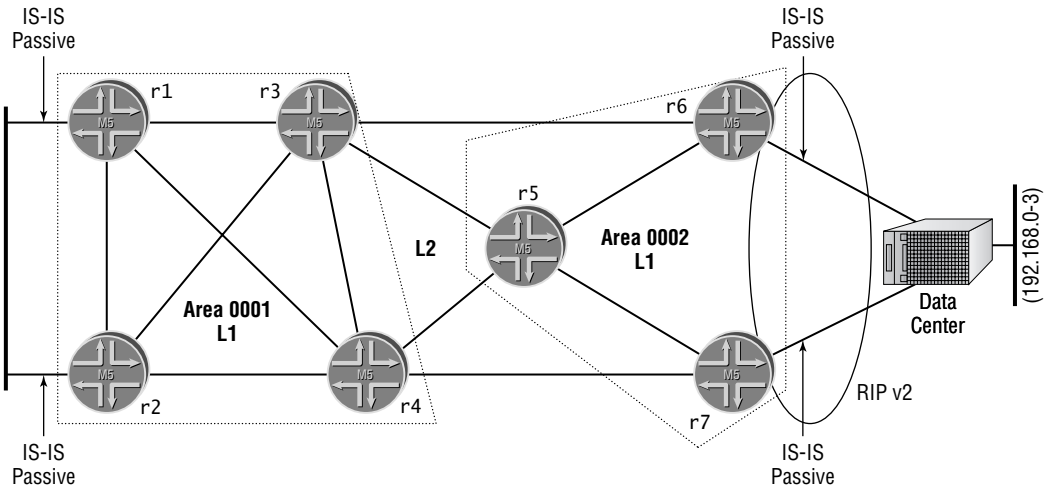
A Layer 3 VPN solution allows a customer to outsource their backbone routing to the service provider. In a Layer 3 solution, the customer edge (CE) and provider edge (PE) devices are Layer 3 peers; they share a common IP subnet and routing protocol. Layer 3 PP VPNs are defined in *BGP/MPLS VPNs*, draft-ietf-ppvpn-rfc2547bis-04. JUNOS software release 5.6 supports Layer 3 VPNs based on either IPv4 or IPv6.

Layer 2 VPN solutions are very similar to a private line or Frame Relay/ATM solution, in that the customer is provided with Layer 2 forwarding services that completely separate the customer's network level and routing protocols from those of the service provider. Because a Layer 2 VPN is protocol agnostic, a L2 VPN service can be deployed to support non-IP and non-routable protocols such as IPX, SNA/APPN, and NetBEUI. Layer 2 PP VPNs are defined in a number of drafts; key among these are *MPLS-based Layer 2 VPNs*, Internet draft draft-kompella-ppvpn-l2vpn-02.txt, and *Transport of Layer 2 Frames Over MPLS*, Internet draft draft-martini-l2circuit-trans-mpls-07.txt.

The VPN solutions supported by JUNOS software release 5.6 employ MPLS forwarding in the data plane. The use of MPLS forwarding across the provider's core enables support for non-routable protocols (L2 VPN solution) and for customers using overlapping or private use–only IP addressing in the context of a Layer 3 VPN solution. The MPLS control plane can make use of RSVP or LDP signaling for establishment of LSPs between PE routers. The VPN control plane is responsible for communicating VPN membership and is based on the use of MP-BGP for 2547 bis (Layer 3) and draft-Kompella (Layer 2) VPNs. In contrast, the draft-Martini approach requires LDP-based signaling in the VPN control plane.

The VPN examples demonstrated in the chapter body are based on the IS-IS baseline topology that was discovered in the Chapter 1 case study. If you are unsure as to the state of your test bed, you should take a few moments to load up and confirm the IS-IS baseline configuration before proceeding; if needed, you should refer to Chapter 1 for suggestions on how to quickly confirm the baseline network's operation and to review the IS-IS baseline topology. It is assumed that all facets of the IS-IS IGP topology are operational at this time. Figure 7.1 displays the results of the IS-IS discovery scenario to help you recall the specifics of the IGP that will support your VPN configurations.

FIGURE 7.1 Summary of IS-IS discovery



Notes:

Multi-level IS-IS, Areas 0001 and 0002 with ISO NET based on router number.

lo0 address of `r3` and `r4` not injected into Area 0001 to ensure optimal forwarding between 10.0.3.3 and 10.0.3.4.

Passive setting on `r5`'s core interfaces for optimal Area 0002-to-core routing.

No authentication or route summarization. Routing policy at `r5` to leak L1 externals (DC routes) to L2.

Redistribution of static default route to data center from both `r6` and `r7`. Redistribution of 192.168.0/24 through 192.168.3/24 routes from RIP into IS-IS by both `r6` and `r7`.

All adjacencies are up, reachability problem discovered at `r1` and `r2` caused by local aggregate definition. Corrected through IBGP policy to effect 10.0/16 route advertisement from `r3` and `r4` to `r1` and `r2`; removed local aggregate from `r1` and `r2`.

Suboptimal routing detected at the data center and at `r1`/`r2` for some locations. This is the result of random nexthop choice for data center's default, and the result of `r1` and `r2`'s preference for `r3`'s RID over `r4` with regard to the 10.0/16 route. This is considered normal behavior, so no corrective actions are taken.

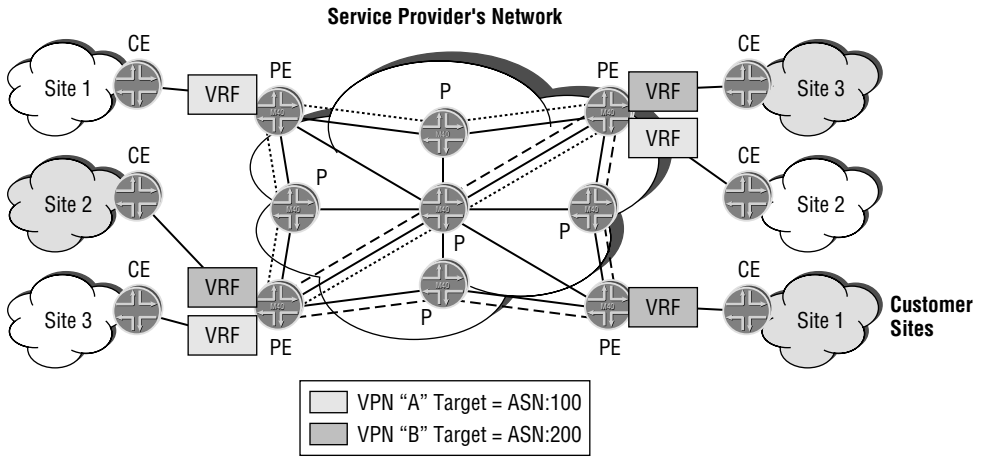
Layer 3 VPNs (2547 bis)

Although it is assumed that the reader possesses a working knowledge of Layer 3 VPN technology to the extent covered in the *JNCIS Study Guide* (Sybex, 2003), a brief review of key 2547 bis terms and concepts is provided in Figure 7.2 for purposes of review.

The figure shows how the provider's edge (PE) routers maintain per-VPN Routing and Forwarding tables called VRFs that house the routes associated with a given VPN site separately from the main routing table. A key concept to scaling a PP VPN solution is the fact that provider (P) routers do not maintain any VPN-specific state. The lack of VPN awareness in P routers is

made possible by the use of MPLS forwarding in the provider's core. The legend in Figure 7.2 illustrates how each collection of sites that constitute a given VPN is normally identified with a common route target (RT). The RT is an extended BGP community that is attached to routes as they are advertised to remote PE routers using VRF export policy; upon receipt, the RT is used in conjunction with VRF import policy to install routes into matching VRFs based on the RT associated with each VRF instance. The RT community can be coded with an IP address or the provider's Autonomous System Number (ASN). Although not shown in the figure, a route distinguisher (RD) is added to the L3 VPN Network Layer Reachability Information (NLRI) advertised by each PE router to ensure the uniqueness of each IPv4 and IPv6 VPN prefix. Recall that VPN customers can deploy local use addressing (RFC 1918) that will result in address overlap between VPNs.

FIGURE 7.2 2547 bis terminology



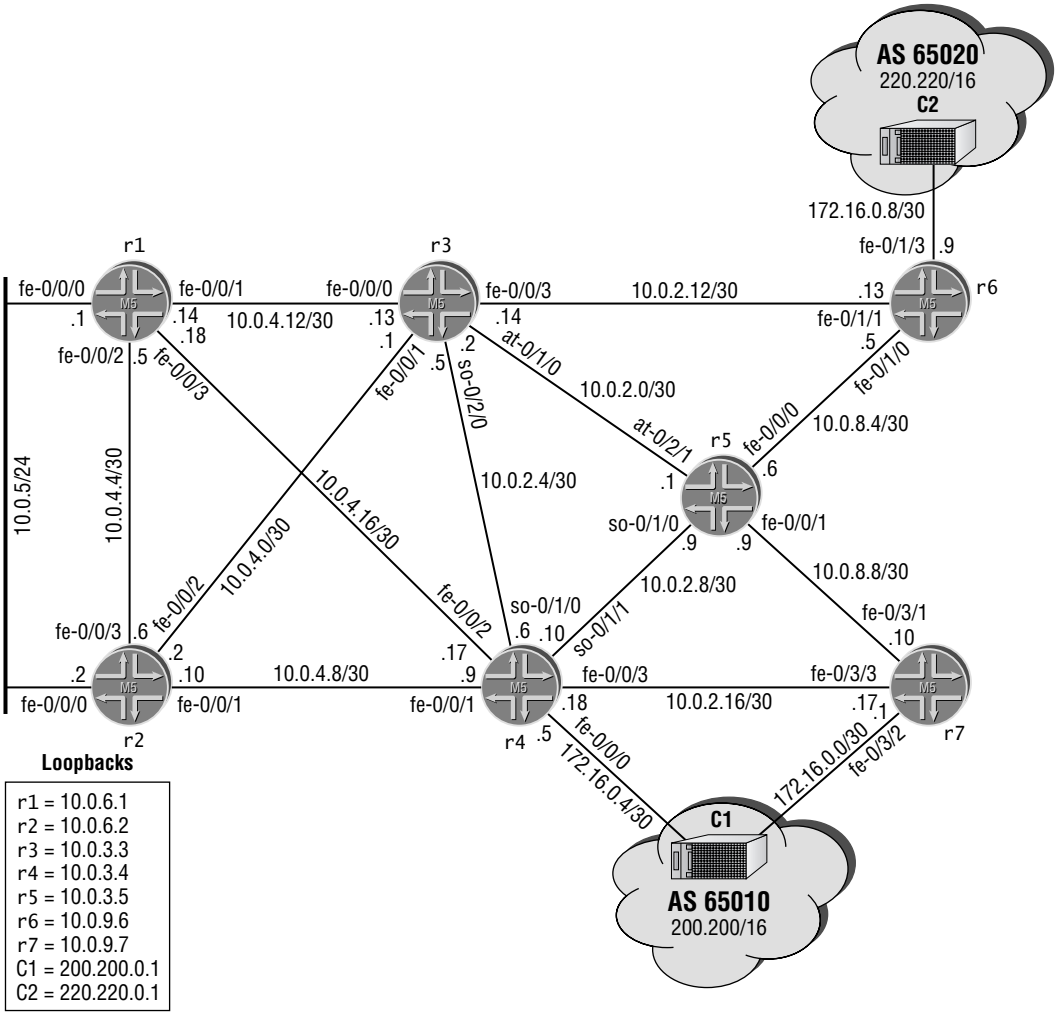
Your Layer 3 VPN configuration scenario begins with the preliminary configuration needed to establish RSVP signaled LSPs and MPLS forwarding in your network. Your preliminary configuration criteria are as follows:

- Enable RSVP signaling on all internal-facing interfaces.
- Establish bidirectional LSPs between PE routers.

Preliminary Configuration

You begin your preliminary configuration task by adding the `mpls` family and RSVP signaling to all internal-facing transit interfaces associated with `r1` through `r7`. This preliminary configuration provides the infrastructure needed to support the Layer 3 VPN configuration that is added in a subsequent step. Refer to Figure 7.3 for the topology specifics needed to complete this configuration task.

FIGURE 7.3 VPN configuration scenario topology



The following commands correctly add the `mpls` family to the internal transit interfaces on `r5`:

```
[edit]
lab@r5# edit interfaces

[edit interfaces]
lab@r5# set at-0/2/1 unit 0 family mpls

[edit interfaces]
lab@r5# set so-0/1/0 unit 0 family mpls
```

```
[edit interfaces]
lab@r5# set fe-0/0/0 unit 0 family mpls

[edit interfaces]
lab@r5# set fe-0/0/1 unit 0 family mpls
```

RSVP signaling and MPLS processing is now enabled for all interfaces, excepting the router's fxp0 OoB interface:

```
[edit]
lab@r5# set protocols mpls interface all

[edit]
lab@r5# set protocols mpls interface fxp0 disable

[edit protocols]
lab@r5# set rsvp interface all

[edit protocols]
lab@r5# set rsvp interface fxp0 disable
```

The changes made to `r5`'s IS-IS baseline configuration are displayed with highlights added to call out changes to existing stanzas:

```
[edit]
lab@r5# show protocols mpls
interface all;
interface fxp0.0 {
    disable;
}

[edit]
lab@r5# show protocols rsvp
interface all;
interface fxp0.0 {
    disable;
}

[edit]
lab@r5# show interfaces
fe-0/0/0 {
    unit 0 {
        family inet {
            address 10.0.8.6/30;
        }
```

```
        family iso;
        family mpls;
    }
}
fe-0/0/1 {
    unit 0 {
        family inet {
            address 10.0.8.9/30;
        }
        family iso;
        family mpls;
    }
}
so-0/1/0 {
    encapsulation ppp;
    unit 0 {
        family inet {
            address 10.0.2.9/30;
        }
        family iso;
        family mpls;
    }
}
at-0/2/1 {
    atm-options {
        vpi 0 {
            maximum-vcs 64;
        }
    }
    unit 0 {
        point-to-point;
        vci 50;
        family inet {
            address 10.0.2.1/30;
        }
        family iso;
        family mpls;
    }
}
fxp0 {
    unit 0 {
        family inet {
```

```
            address 10.0.1.5/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 10.0.3.5/32;
        }
        family iso {
            address 49.0002.5555.5555.5555.00;
        }
    }
}
```

Similar changes are needed in the remaining routers that compose the JNCIE VPN test bed. Note that MPLS processing and RSVP signaling is not enabled on the 10.0.5/24 subnet associated with the fe-0/0/0 interface of `r1` and `r2`. The changes made to `r7`'s configuration are displayed next with highlights:

```
[edit]
lab@r7# show protocols mpls
interface all;
interface fxp0.0 {
    disable;
}

[edit]
lab@r7# show protocols rsvp
interface all;
interface fxp0.0 {
    disable;
}

[edit]
lab@r7# show interfaces
fe-0/3/0 {
    unit 0 {
        family inet {
            address 10.0.8.14/30;
        }
        family iso;
    }
```

```
}
fe-0/3/1 {
    unit 0 {
        family inet {
            address 10.0.8.10/30;
        }
        family iso;
        family mpls;
    }
}
fe-0/3/2 {
    unit 0 {
        family inet {
            address 172.16.0.1/30;
        }
        family mpls;
    }
}
fe-0/3/3{
    unit 0 {
        family inet {
            address 10.0.2.17/30;
        }
        family iso;
        family mpls;
    }
}
fxp0 {
    unit 0 {
        family inet {
            address 10.0.1.7/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 10.0.9.7/32;
        }
        family iso {
```

```
            address 49.0002.7777.7777.7777.00;
        }
    }
}
```

With MPLS processing and RSVP signaling enabled at all routers, you proceed to the definition of the LSPs that are needed between PE routers. You begin at r4 with the configuration of its ingress LSPs that terminate at r6 and r7:

```
[edit protocols mpls]
lab@r4# set label-switched-path r4-r6 to 10.0.9.6 no-cspf

[edit protocols mpls]
lab@r4# set label-switched-path r4-r7 to 10.0.9.7 no-cspf
```

Note that CSPF has been disabled in this example because its use is not required by the scenario's restrictions. Disabling CSPF eliminates the potential for CSPF failures stemming from the fact that there are multiple TED domains in the test bed's multi-level IS-IS topology. Also note that no Explicit Route Objects (ERO) or bandwidth-related constraints are configured, which is again in keeping with the minimum level of functionality required in the preliminary configuration. The modified MPLS stanza is displayed next at r4 with highlights:

```
[edit protocols mpls]
lab@r4# show
label-switched-path r4-r6 {
    to 10.0.9.6;
    no-cspf;
}
label-switched-path r4-r7 {
    to 10.0.9.7;
    no-cspf;
}
interface all;
interface fxp0.0 {
    disable;
}
```

Similar LSP definitions are needed at r6 and r7. The MPLS stanza at r7 is shown here with highlights:

```
[edit protocols mpls]
lab@r7# show
label-switched-path r7-r4 {
    to 10.0.3.4;
    no-cspf;
}
```

```
label-switched-path r7-r6 {
    to 10.0.9.6;
    no-cspf;
}
interface all;
interface fxp0.0 {
    disable;
}
```

Be sure that you `commit` the preliminary configuration changes on all routers before proceeding to the verification section.

Verifying Preliminary Configuration

Verifying your preliminary configuration begins with the determination that RSVP signaling and MPLS processing have been correctly provisioned. The following commands verify that `r3` is correctly configured for basic MPLS and RSVP support:

```
[edit]
lab@r3# run show mpls interface
Interface         State       Administrative groups
fe-0/0/0.0        Up          <none>
fe-0/0/1.0        Up          <none>
fe-0/0/3.0        Up          <none>
at-0/1/0.0        Up          <none>
so-0/2/0.100      Up          <none>

[edit]
lab@r3# run show rsvp interface
RSVP interface: 6 active
                  Active Subscr- Static      Available   Reserved    Highwater
Interface   State resv   iption  BW          BW          BW          mark
fe-0/0/0.0  Up         0   100%  100Mbps     100Mbps     0bps        0bps
fe-0/0/1.0  Up         0   100%  100Mbps     100Mbps     0bps        0bps
fe-0/0/2.0  Up         0   100%  100Mbps     100Mbps     0bps        0bps
fe-0/0/3.0  Up         1   100%  100Mbps     100Mbps     0bps        0bps
at-0/1/0.0  Up         0   100%  155.52Mbps  155.52Mbps  0bps        0bps
so-0/2/0.100Up         1   100%  155.52Mbps  155.52Mbps  0bps        0bps
```

Although not shown, you can assume that all other routers are providing similar indications regarding interface support for MPLS packets and RSVP signaling. Next, you confirm successful establishment of the RSVP signaled LSPs that are associated with `r4`:

```
[edit protocols mpls]
lab@r4# run show rsvp session
Ingress RSVP: 2 sessions
```

```
To              From            State Rt Style Labelin Labelout LSPname
10.0.9.6        10.0.3.4        Up     1  1 FF       -    100002 r4-r6
10.0.9.7        10.0.3.4        Up     1  1 FF       -         3 r4-r7
Total 2 displayed, Up 2, Down 0

Egress RSVP: 2 sessions
To              From            State Rt Style Labelin Labelout LSPname
10.0.3.4        10.0.9.6        Up     0  1 FF       3         - r6-r4
10.0.3.4        10.0.9.7        Up     0  1 FF       3         - r7-r4
Total 2 displayed, Up 2, Down 0

Transit RSVP: 0 sessions
Total 0 displayed, Up 0, Down 0

[edit protocols mpls]
lab@r4#
```

The highlights in the output of the `show rsvp session` command confirm the expected number of ingress and egress sessions, which confirms that all four of the LSPs associated with `r4` have been correctly established. You can assume that a similar display is observed at `r6` and `r7` (not shown).

Verification of your preliminary configuration is complete when all routers in the test bed display support for MPLS processing and RSVP signaling on the required interfaces, and when all RSVP signaled LSPs are successfully established. It is imperative that you have a functional MPLS infrastructure before you proceed to the next section, because PP-VPNs rely on a functional MPLS control and data plane for forwarding VPN traffic. Being able to distinguish between conventional MPLS problems and those that specifically relate to the configuration of a VPN is an invaluable skill for the JNCIE candidate.

Preliminary Configuration Summary

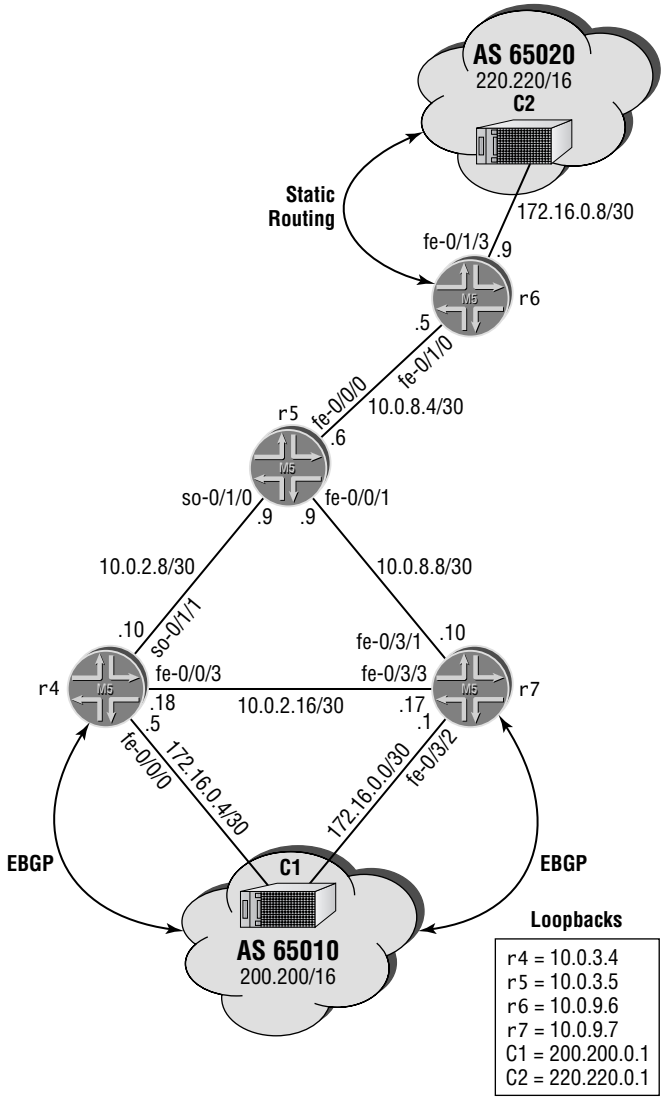
This section involved the configuration and testing of the MPLS infrastructure that will support the Layer 3 VPN configuration added in the following section. While this section demonstrated the configuration of RSVP signaled LSPs, it should be noted that, with very few exceptions, LDP signaled LSPs can also serve to support the Layer 2 and Layer 3 VPN configurations shown in this chapter.

The reader is encouraged to refer back to Chapter 2, "MPLS and Traffic Engineering," for detailed coverage of MPLS configurations and verification techniques.

PE-CE BGP and Static Routing

With the MPLS forwarding and control plane infrastructure in place and confirmed, it is time to get cracking on a Layer 3 VPN configuration. Your first L3 VPN scenario requires a combination of static and BGP routing on the PE-CE links, as shown in Figure 7.4.

FIGURE 7.4 L3 VPN with static and BGP routing



The use of dissimilar routing protocols on the PE-CE constituting the Layer 3 VPN is designed to confirm that the JNCIE candidate is competent with more than one PE-CE routing mechanism; having multiple PE-CE routing protocols also adds complexity to your assignment because you will need distinctly different VPN configurations to support the C1 and C2 devices. To complete this Layer 3 scenario, you must configure the subset of routers shown in Figure 7.4 according to these criteria:

- Establish a L3 VPN providing connectivity between C1 and C2.
- You must support pings that originate and terminate on VRF interfaces.

- Ensure that the VPN is not disrupted by the failure of r4 or r7, or by any internal link/interface failure.
- You must not configure the RD within the VRF instance.
- Use an RD that is based on the PE lo0 address.
- Configure a route target of target:*65412:420*.
- Your VPN configuration can not disrupt existing IPv4 routing and forwarding functionality within your AS.
- You may add two static routes to the configuration of C2.

Initial L3 VPN Configuration: Static and BGP Routing

Configuration of the L3 VPN begins at r6 with the creation of a VRF routing instances called *c2*. The first grouping of commands creates the VRF and associates r6's fe-0/1/3 interface with the VRF instance:

```
[edit]
lab@r6# edit routing-instances c2

[edit routing-instances c2]
lab@r6# set instance-type vrf

[edit routing-instances c2]
lab@r6# set interface fe-0/1/3
```

JUNOS software releases prior to *5.5* required the manual creation of VRF import and export policies, as well as the definition of a named extended community for use as a RT. Although manual VRF policy definition is still supported, use of the vrf-target statement significantly simplifies the configuration of a Layer 3 VPN. The resulting "default VRF policy" attaches the configured RT community to all route advertisements from that VRF and also matches on the specified community for routes received from remote PEs. Because the default VRF policy associated with the vrf-target statement advertises *all* active routes received from the CE, as well as the VRF's static and direct routes, the use of vrf-target supports all of the functionality specified in this scenario. Note that local routes can not be exported using routing policy, and therefore local routes are not exported in conjunction with the vrf-target statement.

Some VPN applications, for example, a hub and spoke topology, require that you attach one RT to the routes being advertised while matching on a different RT in the routes being received. For these applications, use the import and export keywords in conjunction with the vrf-target statement to effect the advertisement of one community while matching on a different community in routes that are received. The VPN's RT is now configured in association with the vrf-target feature according to the criteria specified:

```
[edit routing-instances c2]
lab@r6# set vrf-target target:65412:420
```

The restriction on manual assignment of the RD within the VRF is addressed with automatic RD computation in conjunction with the `route-distinguisher-id` statement. The automatically derived RD is computed by concatinating the IP address specified with a unique identifier that is associated with each VRF instance on the local router. You enter the `route-distinguisher-id` statement at the `[edit routing-options]` hierarchy:

```
[edit routing-options]
lab@r6# set route-distinguisher-id 10.0.9.6
```

The static route used by `r6` when forwarding traffic to the C2 site is now defined:

```
[edit routing-instances c2]
lab@r6# set routing-options static route 220.220/16 next-hop 172.16.0.10
```

The *c2* VRF routing instance configuration is displayed next for visual inspection:

```
[edit routing-instances c2]
lab@r6# show
instance-type vrf;
interface fe-0/1/3.0;
vrf-target target:65412:420;
routing-options {
    static {
        route 220.220.0.0/16 next-hop 172.16.0.10;
    }
}
```

Although additional changes might be required at `r6` to meet all of the specified criteria, you decide to commit the VRF changes at `r6` and direct your attention to the configuration of the *c1* VRF routing instance at `r4`. As with the *c2* instance on `r6`, the *c1* routing instance also makes use of the `vrf-target` statement and its related default VRF import and export policy.

The changes made to `r4`'s configuration in support of the C1-C2 VPN are shown here with highlights added to call out changes to existing configuration stanzas:

```
[edit]
lab@r4# show routing-options
static {
    route 10.0.200.0/24 {
        next-hop 10.0.1.102;
        no-readvertise;
    }
}
aggregate {
    route 10.0.0.0/16;
}
route-distinguisher-id 10.0.3.4;
autonomous-system 65412;
```

```
[edit]
lab@r4# show routing-instances
c1 {
    instance-type vrf;
    interface fe-0/0/0.0;
    vrf-target target:65412:420;
    protocols {
        bgp {
            group c1 {
                type external;
                peer-as 65010;
                neighbor 172.16.0.6;
            }
        }
    }
}
```

Although a similar configuration is needed at `r7` to meet the stated redundancy requirement, you decide to test the waters by committing the changes at `r4` so that you can determine where your existing VPN configuration might need additional tweaking.

Initial L3 VPN Confirmation: Static and BGP Routing

One of the benefits of a L3 VPN is the ability to conduct local testing of the PE-CE VRF link and routing protocols. The fact that the PE and CE interact at the IP layer in a L3 VPN greatly simplifies fault isolation, as you will experience when Layer 2 VPNs are deployed in a later section. You begin initial confirmation by confirming the presence of an active static route at `r6` for the 220.220/16 prefix associated with C2:

```
[edit]
lab@r6# run show route protocol static 220.220/16

c2.inet.0: 3 destinations, 3 routes (3 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

220.220.0.0/16     *[Static/5] 00:41:45
                    > to 172.16.0.10 via fe-0/1/3.0
```

The static route is present and active, so ping testing is conducted from `r6` to C2; note that the pings fail when the `ping` command is not associated with the correct routing instance:

```
[edit]
lab@r6# run ping 220.220.0.1 count 2
PING 220.220.0.1 (220.220.0.1): 56 data bytes
ping: sendto: No route to host
ping: sendto: No route to host
```

```
--- 220.220.0.1 ping statistics ---
2 packets transmitted, 0 packets received, 100% packet loss
```

The pings fail because the egress interface associated with the route is not present in the `inet.0` routing table. Specifying the *c2* routing instance allows `r6` to correctly identify the egress interface (fe-0/1/3) for the test traffic:

```
[edit]
lab@r6# run ping 220.220.0.1 count 2 routing-instance c2
PING 220.220.0.1 (220.220.0.1): 56 data bytes
64 bytes from 220.220.0.1: icmp_seq=0 ttl=255 time=0.321 ms
64 bytes from 220.220.0.1: icmp_seq=1 ttl=255 time=0.172 ms

--- 220.220.0.1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.172/0.246/0.321/0.074 ms
```

The `r6`–C2 ping is successful, which confirms that static routing is working between PE `r6` and CE C2. The status of the EBGP session between `r4` and C1 is now verified:

```
[edit]
lab@r4# run show bgp summary instance c1
Groups: 1 Peers: 1 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
c1.inet.0              0          0          0          0          0          0
Peer               AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn
   State|#Active/Received/Damped...
172.16.0.6      65010      24684         12       0       0        4:23 Establ
   c1.inet.0: 2/3/0
```

Note that the output is limited to the status of BGP sessions associated with the *c1* routing instance by including the `instance` switch. The contents of the *c1* VRF are now displayed to confirm receipt of BGP routes from the C1 peer:

```
[edit]
lab@r4# run show route table c1

c1.inet.0: 5 destinations, 5 routes (4 active, 0 holddown, 1 hidden)
+ = Active Route, - = Last Active, * = Both

172.16.0.4/30      *[Direct/0] 00:08:24
                    > via fe-0/0/0.0
172.16.0.5/32      *[Local/0] 00:08:24
                       Local via fe-0/0/0.0
200.200.0.0/16     *[BGP/170] 00:06:26, MED 0, localpref 100
                       AS path: 65010 I
                    > to 172.16.0.6 via fe-0/0/0.0
```

```
200.200.1.0/24     *[BGP/170] 00:06:26, MED 0, localpref 100
                      AS path: 65010 I
                    > to 172.16.0.6 via fe-0/0/0.0
```

Initial confirmation indicates that both of the PE-CE VRF links are operational. However, you note that the `c1` VRF does not contain any routes associated with the C2 site. Can you identify why VPN NLRI is not being exchanged between the PE routers?

Real World Scenario

Troubleshooting a Layer 3 VPN Problem

You have determined that VPN routes are not being exchanged between PE routers `r4` and `r6` in your initial Layer 3 VPN configuration. Can you spot the issue based on the results of a `show bgp neighbor` display?

```
[edit]
lab@r4# run show bgp neighbor 10.0.9.6
Peer: 10.0.9.6+179    AS 65412 Local: 10.0.3.4+2471   AS 65412
  Type: Internal    State: Established    Flags: <>
  Last State: OpenConfirm   Last Event: RecvKeepAlive
  Last Error: None
  Export: [ nhs ]
  Options: <Preference LocalAddress HoldTime AdvertiseInactive Refresh>
  Local Address: 10.0.3.4 Holdtime: 90 Preference: 170
  Number of flaps: 0
  Peer ID: 10.0.9.6         Local ID: 10.0.3.4         Active Holdtime: 90
  Keepalive Interval: 30
  NLRI advertised by peer: inet-unicast
  NLRI for this session: inet-unicast
  Peer supports Refresh capability (2)
  Table inet.0 Bit: 10000
    RIB State: BGP restart is complete
    Send state: in sync
    Active prefixes:            0
    Received prefixes:          4
    Suppressed due to damping:  0
  Last traffic (seconds): Received 5    Sent 29   Checked 29
  Input messages:  Total 52     Updates 1       Refreshes 0     Octets 1036
  Output messages: Total 54     Updates 2       Refreshes 0     Octets 1106
  Output Queue[0]: 0
```

If you identified the fact that MP-IBGP has not been configured with support for the `inet-vpn` family, then you are operating in a "fully switched on" mode and should congratulate yourself! Candidates often fail to adjust their existing IBGP sessions to support the appropriate VPN family, and then find themselves wasting time manipulating their VRF policies in a futile attempt to evoke the advertisement of VPN NLRI. The changes shown for `r4` correctly provision its MP-IBGP session to `r6` for support of both IPv4 and IPv4 VPN NLRI; note that failing to explicitly configure the default `inet` family along with the `inet-vpn` family disrupts the exiting IPv4 routing functionality because the resulting IBGP session supports only VPN NLRI.

```
[edit protocols bgp group int]
lab@r4# show
type internal;
local-address 10.0.3.4;
export nhs;
neighbor 10.0.6.1 {
    export r1;
}
neighbor 10.0.6.2 {
    export r2;
}
neighbor 10.0.3.3;
neighbor 10.0.3.5;
neighbor 10.0.9.6 {
    family inet {
        unicast;
    }
    family inet-vpn {
        unicast;
    }
}
neighbor 10.0.9.7;
```

Similar changes are needed at r6. Do not forget that the IBGP peering session between r6 and r7 will ultimately need similar modifications. After committing the changes, MP-IBGP support for the inet and inet-vpn families is confirmed, as shown next:

```
[edit protocols bgp group int]
lab@r4# run show bgp neighbor 10.0.9.6 | match NLRI
  NLRI advertised by peer: inet-unicast inet-vpn-unicast
  NLRI for this session: inet-unicast inet-vpn-unicast
```

With MP-IBGP now correctly configured between r4 and r6, you again display the contents of the *c1* VRF at r4:

```
[edit routing-instances c1]
lab@r4# run show route table c1

c1.inet.0: 7 destinations, 7 routes (6 active, 0 holddown, 1 hidden)
+ = Active Route, - = Last Active, * = Both

172.16.0.4/30      *[Direct/0] 02:19:55
                    > via fe-0/0/0.0
172.16.0.5/32      *[Local/0] 02:19:55
                       Local via fe-0/0/0.0
172.16.0.8/30      *[BGP/170] 00:04:39, localpref 100, from 10.0.9.6
                       AS path: I
                    > via so-0/1/1.0, label-switched-path r4-r6
```

```
200.200.0.0/16     *[BGP/170] 01:26:38, MED 0, localpref 100
                      AS path: 65010 I
                    > to 172.16.0.6 via fe-0/0/0.0
200.200.1.0/24     *[BGP/170] 01:26:38, MED 0, localpref 100
                      AS path: 65010 I
                    > to 172.16.0.6 via fe-0/0/0.0
220.220.0.0/16     *[BGP/170] 00:04:39, localpref 100, from 10.0.9.6
                      AS path: I
                    > via so-0/1/1.0, label-switched-path r4-r6
```

The added highlights call out the presence of the 220.220/16 and 172.16.0.8/30 prefixes as BGP routes in r4's *c1* VRF. A `show route advertising protocol` command is issued to confirm that the 220.220/16 route is, in turn, correctly advertised by r4 to the C1 peer:

```
[edit protocols bgp group int]
lab@r4# run show route advertising-protocol bgp 172.16.0.6

[edit protocols bgp group int]
lab@r4#
```

Hmm, the results are not what you had hoped to see. Oddly, a `show route receiving protocol` command, when issued at C1, confirms the proper receipt of the 220.220/16 route from r4:

```
[edit protocols bgp group int]
lab@r4# run telnet routing-instance c1 172.16.0.6
Trying 172.16.0.6...
Connected to 172.16.0.6.
Escape character is '^]'.

c1 (ttyp0)

login: lab
Password:
Last login: Wed Jun  4 13:10:30 from 172.16.0.5

--- JUNOS 5.6R2.4 built 2003-02-14 23:22:39 UTC

lab@c1> show route receive-protocol bgp 172.16.0.5

inet.0: 121179 destinations, 121184 routes (121179 active, 0 holddown, 5 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
* 220.220.0.0/16          172.16.0.5                               65412 I
```

You may have noticed that the telnet session from r4 to C1 made use of the `routing-instance` switch to address the fact that r4's fe-0/0/0 interface is no longer present in its main routing

instance. The issue with the `show route advertising protocol` command relates to the fact that the EBGP session between `r4` and C2 is currently defined twice: once in the *c1* VRF (correctly) and again in the main routing instance (unnecessarily). The result is that `r4` incorrectly indexes the 172.16.0.6 neighbor request against the main routing instance, which returns an empty display due to this EBGP session being in an `idle` state (placing `r4`'s fe-0/0/0 interface into the *c1* VRF prevents its use by the main routing instance). This situation is shown here:

```
[edit protocols bgp group int]
lab@r4# run show bgp summary | match 172.16.0.6
172.16.0.6       65010       31880       30510       0       1       42:00 Idle
172.16.0.6       65010       26727          84       0       0       40:02 Establ
```

The unnecessary BGP neighbor definition is removed from `r4` (not shown) and `r6` to resolve this anomaly:

```
lab@r6# delete protocols bgp group c2
```

After the change is committed at `r4`, the `show route advertising protocol` command returns the expected results:

```
[edit routing-instances c1]
lab@r4# run show route advertising-protocol bgp 172.16.0.6

c1.inet.0: 7 destinations, 7 routes (6 active, 0 holddown, 1 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
* 172.16.0.8/30           Self                                    I
* 200.200.0.0/16          172.16.0.6                              65010 I
* 200.200.1.0/24          172.16.0.6                              65010 I
* 220.220.0.0/16          Self                                    I
```

The confirmation results observed thus far indicate that the C1–C2 VPN is working in accordance with all specified requirements excepting those relating to redundancy; recall that `r7` has not yet had its *c1* VRF configured. Hoping for the best, you telnet to the C2 router to perform end-to-end connectivity testing before bringing `r7` into the mix:

```
[edit]
lab@r6# run telnet routing-instance c2 220.220.0.1
Trying 220.220.0.1...
Connected to 220.220.0.1.
Escape character is '^]'.

C2 (ttyp0)

login: lab
Password:
Last login: Wed Jun  4 13:10:39 from 172.16.0.5

--- JUNOS 5.6R2.4 built 2003-02-14 23:22:39 UTC
```

```
lab@c2> ping 200.200.0.1
PING 200.200.0.1 (200.200.0.1): 56 data bytes
ping: sendto: No route to host
ping: sendto: No route to host
^C
--- 200.200.0.1 ping statistics ---
17 packets transmitted, 0 packets received, 100% packet loss
```

Noting the ping failure, you display the 200.200/16 route at C2:

```
lab@c2> show route 200.200/16

lab@c2>
```

The lack of a 200.200/16 route at C2 stems from the fact that r6 and C2 no longer peer with EBGP; the static routing on the r4-C2 VRF interface means that C2 can not dynamically learn any of the routes that are present in r6's *c2* VRF. The reason that you are permitted to define two static routes at the C2 peer should now be clear: these static routes are needed at C2 to direct traffic associated with 200.200/16 and 172.16.0.4/30 to r6. The static routes are added to C2 and the change is committed:

```
[edit routing-options]
lab@c2# set static route 200.200/16 next-hop 172.16.0.9

[edit routing-options]
lab@c2# set static route 172.16.0.4/30 next-hop 172.16.0.9
```

The definition of a static route for the r4–C1 VRF subnet (172.16.0.4/30) is critical to ensure that traffic can be originated and terminated on the VRF interfaces, which is a requirement in this example; C1, in contrast, learns the 172.16.0.8/30 route associated with the r6–C2 VRF interface through its EBGP session to r4. Once the static routes are committed, you repeat the end-to-end test:

```
lab@c2> ping 200.200.0.1 count 2
PING 200.200.0.1 (200.200.0.1): 56 data bytes
64 bytes from 200.200.0.1: icmp_seq=0 ttl=252 time=0.433 ms
64 bytes from 200.200.0.1: icmp_seq=1 ttl=252 time=0.331 ms

--- 200.200.0.1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.331/0.382/0.433/0.051 ms
```

The pings succeed when sourced from C2's VRF interface. Additional testing confirms that pings also succeed when sourced from C2's loopback address, and when the traffic is targeted at C1's VRF interface:

```
lab@c2> ping 200.200.0.1 count 2 source 220.220.0.1
PING 200.200.0.1 (200.200.0.1): 56 data bytes
64 bytes from 200.200.0.1: icmp_seq=0 ttl=252 time=0.438 ms
64 bytes from 200.200.0.1: icmp_seq=1 ttl=252 time=0.334 ms
```

```
--- 200.200.0.1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.334/0.386/0.438/0.052 ms

lab@c2> ping 172.16.0.6 count 2 source 220.220.0.1
PING 172.16.0.6 (172.16.0.6): 56 data bytes
64 bytes from 172.16.0.6: icmp_seq=0 ttl=252 time=0.433 ms
64 bytes from 172.16.0.6: icmp_seq=1 ttl=252 time=0.325 ms

--- 172.16.0.6 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.325/0.379/0.433/0.054 ms
```

The results shown confirm that you have configured the required VPN connectivity between the C1 and C2 locations. Traceroute testing from the C1 site confirms the presence of MPLS forwarding by P routers:

```
lab@c1> traceroute 220.220.0.1
traceroute to 220.220.0.1 (220.220.0.1), 30 hops max, 40 byte packets
 1  172.16.0.5 (172.16.0.5)  0.417 ms  0.294 ms  0.278 ms
 2  * * *
 3  10.0.2.13 (10.0.2.13)  0.297 ms  0.238 ms  0.243 ms
     MPLS Label=100003 CoS=0 TTL=1 S=1
 4  220.220.0.1 (220.220.0.1)  0.331 ms  0.311 ms  0.301 ms
```

The time-out on the second hop is expected due to the fact that `r5` does not carry any VPN routes, and so can not route the TTL expired message back to the 172.16.0.6 address used by C1 to source its traffic. It should be noted that an E-FPC equipped router copies the TTL value present in the IP header into both the inner and outer MPLS labels when handling traffic received from the attached CE. However, for traffic that is generated locally, an E-FPC PE sets the TTL in the outer MPLS label TTL to the maximum value (255) to avoid P router time-outs:

```
[edit routing-instances c1]
lab@r4# run traceroute routing-instance c1 220.220.0.1
traceroute to 220.220.0.1 (220.220.0.1), 30 hops max, 40 byte packets
 1  10.0.2.13 (10.0.2.13)  0.778 ms  0.591 ms  0.527 ms
     MPLS Label=100003 CoS=0 TTL=1 S=1
 2  220.220.0.1 (220.220.0.1)  0.591 ms  0.602 ms  0.553 ms
```

In contrast, an M-series router that is equipped with a standard FPC can not write the TTL of the received IP packet into both the outer and inner MPLS labels. This results in the outer label having a maximum TTL value for traffic received from the local CE *and* for traffic that is generated locally. This behavior is demonstrated here in the context of `r6` and C2, where `r6` is not equipped with an E-FPC and C2 generates a traceroute to a C1 prefix:

```
lab@c2> traceroute 200.200.0.1
traceroute to 200.200.0.1 (200.200.0.1), 30 hops max, 40 byte packets
```

```
 1  172.16.0.9 (172.16.0.9)  0.248 ms  0.157 ms  0.149 ms
 2  10.0.2.10 (10.0.2.10)  12.934 ms  0.542 ms  0.524 ms
     MPLS Label=100003 CoS=0 TTL=1 S=1
 3  200.200.0.1 (200.200.0.1)  0.317 ms  0.299 ms  0.297 ms
```

The bottom line is that you should or should not expect to see time-outs for P-router hops when conducting end-to-end traceroute testing based on whether the ingress PE is or is not E-FPC equipped. With all aspects of the VPN configurations in effect at `r4` and `r6` confirmed, all that remains to complete the initial Layer 3 VPN scenario is the addition of VRF-related configuration at `r7`. As with `r4` and `r6`, you should also remove the existing EBGP stanza for the C1 peer:

```
[edit protocols bgp]
lab@r7# delete group c1
```

To save space, the actual commands used to configure `r7`'s VRF are not shown. In this example, this author used a text editor to modify the VRF configuration in place at `r4` to reflect the fe-0/3/2 VRF interface and 172.16.0.2 EBGP peering address needed at `r7`. The modified `routing-instance` stanza was then loaded into `r7` using the `load merge terminal` command. The changes made to `r7`'s configuration in support of the initial Layer 3 VPN configuration stanza are shown here with highlights added:

```
[edit]
lab@r7# show routing-options
static {
    route 0.0.0.0/0 reject;
    route 10.0.200.0/24 {
        next-hop 10.0.1.102;
        no-readvertise;
    }
}
aggregate {
    route 10.0.0.0/16;
}
route-distinguisher-id 10.0.9.7;
autonomous-system 65412;

[edit]
lab@r7# show routing-instances
c1 {
    instance-type vrf;
    interface fe-0/3/2.0;
    vrf-target target:65412:420;
    protocols {
```

```
        bgp {
            group c1 {
                type external;
                peer-as 65010;
                neighbor 172.16.0.2;
            }
        }
    }
}
[edit]
lab@r7# show protocols bgp
group int {
    type internal;
    local-address 10.0.9.7;
    export nhs;
    neighbor 10.0.6.1;
    neighbor 10.0.6.2;
    neighbor 10.0.3.3;
    neighbor 10.0.3.4 {
        family inet {
            unicast;
        }
        family inet-vpn {
            unicast;
        }
    }
    neighbor 10.0.3.5;
    neighbor 10.0.9.6 {
        family inet {
            unicast;
        }
        family inet-vpn {
            unicast;
        }
    }
}
```

Adding the `inet-vpn` family to the `r4` peer definition at `r7` provides additional resiliency to failure that is not strictly required in this scenario. To back this up, you need to add the `inet-vpn` family to the peer definition for `r7` at `r4` (not shown). By enabling the advertisement of C1's routes between `r4` and `r7`, you achieve tolerance for the failure of the VRF interface at either `r4` or `r7`.

Do not forget to also adjust the r7 peering definition at r6 to add support for both the inet and inet-vpn families. If desired, you could add the inet-vpn and inet family declarations at the IBGP group level, because peers that do not support the VPN NLRI (for example, r5) will simply negotiate the NLRI that *is* supported during IBGP session establishment. For completeness' sake, the changes made to r6's configuration in support of the VPN configuration at r7 are shown here with highlights:

```
[edit]
lab@r6# show protocols bgp group int
type internal;
local-address 10.0.9.6;
export nhs;
neighbor 10.0.6.1;
neighbor 10.0.6.2;
neighbor 10.0.3.3;
neighbor 10.0.3.4 {
    family inet {
        unicast;
    }
    family inet-vpn {
        unicast;
    }
}
neighbor 10.0.3.5;
neighbor 10.0.9.7 {
    family inet {
        unicast;
    }
    family inet-vpn {
        unicast;
    }
}
```

The confirmation of r7's VRF configuration proceeds in the manner previously shown for r4 and r6; you begin with verification of the EBGP session between r7 and C1:

```
[edit]
lab@r7# run show bgp summary instance c1
Groups: 1 Peers: 1 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
c1.inet.0              8          6          0          0          0          0
Peer               AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn
   State|#Active/Received/Damped...
172.16.0.2      65010         34         35       0       0       15:30 Establ
   c1.inet.0: 3/3/0
```

The EBGP session is correctly established. Traceroute testing is performed to confirm forwarding to both local and remote CE prefixes:

```
[edit]
lab@r7# run traceroute 200.200.0.1 routing-instance c1
traceroute to 200.200.0.1 (200.200.0.1), 30 hops max, 40 byte packets
 1  200.200.0.1 (200.200.0.1)  0.223 ms  0.142 ms  0.102 ms

[edit]
lab@r7# run traceroute 220.220.0.1 routing-instance c1
traceroute to 220.220.0.1 (220.220.0.1), 30 hops max, 40 byte packets
 1  10.0.8.5 (10.0.8.5)  0.402 ms  0.319 ms  0.325 ms
     MPLS Label=100003 CoS=0 TTL=1 S=1
 2  * * *
 3  * *^C
```

The traceroute to the local CE's prefix is successful, but the traceroute to the C2 prefix fails. The fact that traceroutes from C1 to C2 do succeed when sourced from C1's loopback address should shed light on the remaining issue:

```
lab@c1> show route 220.220/16

inet.0: 11 destinations, 21 routes (11 active, 0 holddown, 5 hidden)
+ = Active Route, - = Last Active, * = Both

220.220.0.0/16     *[BGP/170] 00:18:04, localpref 100
                      AS path: 65412 I
                    > to 172.16.0.1 via fe-0/0/0.0
                    [BGP/170] 00:04:05, localpref 100
                      AS path: 65412 I
                    > to 172.16.0.5 via fe-0/0/1.0
```

The `show route` output confirms that C1 is currently forwarding traffic destined to C2 through `r7`, and traceroute testing succeeds when the traffic is sourced from C1's loopback address:

```
lab@c1> traceroute 220.220.0.1 source 200.200.0.1
traceroute to 220.220.0.1 (220.220.0.1) from 200.200.0.1, 30 hops max,
   40 byte packets
 1  172.16.0.1 (172.16.0.1)  0.268 ms  0.261 ms  0.164 ms
 2  10.0.8.5 (10.0.8.5)  0.305 ms  0.272 ms  0.263 ms
     MPLS Label=100003 CoS=0 TTL=1 S=1
 3  220.220.0.1 (220.220.0.1)  0.354 ms  0.338 ms  0.327 ms
```

As an additional hint, think about what address is used when traffic is sourced by `r7`'s *`c1`* routing instance. If you are starting to think that modifications are needed in the static route

definitions at C2 to accommodate the 172.16.0.0/30 address in use on the r7-C1 VRF interface, then you are spot-on! Being that only two static routes are permitted at C2, you alter the existing 172.16.0.4/30 static route to summarize the 172.16.0.0/30 and 172.16.0.4/30 prefixes associated with both of site C1's VRF links:

```
[edit routing-options static]
lab@c2# delete route 172.16.0.4/30

[edit routing-options static]
lab@c2# set route 172.16.0.0/29 next-hop 172.16.0.9
```

After the change is committed at C2, traceroute from r7 is successful:

```
[edit]
lab@r7# run traceroute 220.220.0.1 routing-instance c1
traceroute to 220.220.0.1 (220.220.0.1), 30 hops max, 40 byte packets
 1  10.0.8.5 (10.0.8.5)  0.428 ms  0.290 ms  0.242 ms
     MPLS Label=100003 CoS=0 TTL=1 S=1
 2  220.220.0.1 (220.220.0.1)  0.339 ms  0.304 ms  0.283 ms
```

As a final check on the redundancy aspects of your configuration, you verify that r6 correctly displays two BGP routes for C1 prefixes; recall that a previous display showed that C1 correctly displays two BGP routes for the prefixes associated with the C2 location:

```
[edit]
lab@r6# run show route 200.200/16

c2.inet.0: 8 destinations, 10 routes (8 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

200.200.0.0/16     *[BGP/170] 00:06:59, MED 0, localpref 100, from 10.0.3.4
                      AS path: 65010 I
                    > to 10.0.2.14 via fe-0/1/1.0, label-switched-path r6-r4
                    [BGP/170] 00:31:01, MED 0, localpref 100, from 10.0.9.7
                      AS path: 65010 I
                    > to 10.0.8.6 via fe-0/1/0.0, label-switched-path r6-r7
200.200.1.0/24     *[BGP/170] 00:06:59, MED 0, localpref 100, from 10.0.3.4
                      AS path: 65010 I
                    > to 10.0.2.14 via fe-0/1/1.0, label-switched-path r6-r4
                    [BGP/170] 00:31:01, MED 0, localpref 100, from 10.0.9.7
                      AS path: 65010 I
                    > to 10.0.8.6 via fe-0/1/0.0, label-switched-path r6-r7
```

The results confirm that both r4 and r7 are advertising C1's prefixes to r6, and also show that r6 has correctly installed these prefixes into the *c2* VRF table. Although viewing a VPN's forwarding table is generally not needed when all is working, this author has found that displaying a VPN's forwarding table can prove invaluable when troubleshooting problems in the MPLS forwarding plane. An example of the output provided by a `show route forwarding-table vpn` command is shown here for informational purposes:

```
[edit]
lab@r4# run show route forwarding-table vpn c1
Routing table: c1.inet
Internet:
Destination        Type RtRef Next hop           Type Index NhRef Netif
default            perm     0                    dscd    33     1
172.16.0.0/30      user     0                    indr   117     3
                              10.0.2.17          Push  100000     fe-0/0/3.0
172.16.0.4/30      intf     0                    rslv    68     1 fe-0/0/0.0
172.16.0.4/32      dest     0 172.16.0.4         recv    66     1 fe-0/0/0.0
172.16.0.5/32      intf     0 172.16.0.5         locl    67     2
172.16.0.5/32      dest     0 172.16.0.5         locl    67     2
172.16.0.6/32      dest     1 0:d0:b7:3f:b3:fb   ucst   118     5 fe-0/0/0.0
172.16.0.7/32      dest     0 172.16.0.7         bcst    65     1 fe-0/0/0.0
172.16.0.8/30      user     0                    indr   115     3
                                                 Push 100003, Push 100001(top)
                                                    so-0/1/1.0
172.16.0.12/30     user     0                    indr   117     3
                              10.0.2.17          Push  100000     fe-0/0/3.0
200.200.0.0/16     user     0 172.16.0.6         ucst   118     5 fe-0/0/0.0
200.200.1.0/24     user     0 172.16.0.6         ucst   118     5 fe-0/0/0.0
220.220.0.0/16     user     0                    indr   115     3
                                                 Push 100003, Push 100001(top)
                                                    so-0/1/1.0
224.0.0.0/4        perm     0                    mdsc    34     1
224.0.0.1/32       perm     0 224.0.0.1          mcst    30     1
255.255.255.255/32 perm     0                    bcst    31     1
```

Of particular interest are the forwarding table entries that show two labels. These entries represent VPN routes that have been learned from a remote PE through MP-IBGP. In these cases, the first label (bottom) represents the VRF label attached to the route by the advertising PE; when traffic is received, the remote PE associates this label with a local VRF interface. The second (top) label represents the MPLS transport label, which was assigned by RSVP in this example.

The various output examples and the test results shown in the confirmation section indicate that you have configured a Layer 3 VPN that complies with all restrictions and operational requirements. Congratulations!

The use of `vrf-target` in this example resulted in a default VRF policy that advertised all active routes in the VRF, including the directly connected route for the PE-CE VRF interface. Because the PEs in this example had at least one active route (static or BGP) that pointed to the attached CE as a next hop, the PE was able to pre-populate its Layer 2 rewrite table with the MAC address associated with the attached CE. This, coupled with the automatic export of the PE-CE direct route, resulted in a situation that allowed traffic to originate and terminate on a multi-access PE-CE link. In many cases, you will have trouble sourcing traffic from a multi-access PE-CE VRF interface unless specific steps are taken. Note that the inability to perform end-to-end ping and traceroute testing with traffic sourced from the local VRF interface may not even represent a problem in a production network, or in a lab examination setting for that matter, depending on the specific circumstances at play. Generally speaking, you use the `vt-interface` or `vrf-table-label` configuration options to work around problems with the local origination and termination of VPN traffic when the JUNOS software version or configuration specifics do not yield the behavior observed in this scenario. Creative use of static routes and VRF interface subnetting is another workable, albeit brain-unfriendly workaround. Unfortunately, there are too many JUNOS software version–related enhancements and configuration "what ifs" to allow a full exploration of this topic here. The reader is encouraged to consult the related JUNOS software documentation set for full coverage of the behavior associated with multi-access VRF interfaces and the options available to work around these issues. The related "Why the extra hop?" sidebar contains additional background information on this topic.

Why the Extra Hop?

You may have noticed that traceroutes destined to the remote PE's VRF interface incur an additional hop through the attached CE, as shown in this example taken from `r6`:

```
[edit]
lab@r6# run traceroute routing-instance c2 172.16.0.5
traceroute to 172.16.0.5 (172.16.0.5), 30 hops max, 40 byte packets
 1  10.0.2.6 (10.0.2.6)  0.681 ms  0.538 ms  0.475 ms
     MPLS Label=100009 CoS=0 TTL=1 S=1
 2  172.16.0.6 (172.16.0.6)  0.302 ms  0.276 ms  0.254 ms
 3  172.16.0.5 (172.16.0.5)  0.524 ms  0.509 ms  0.488 ms
```

This behavior, which is expected, stems from the architecture of M-series and T-series routing platforms. Specifically, the IP II ASIC in the egress PE is normally used to index the value of the bottom (VRF) label to a corresponding VRF interface. Because VPN traffic arrives at the egress PE with an MPLS label, its pass through the IP II lookup function is based on the Layer 2 label instead of an IP address. The IP II can not be used to process an MPLS label and an IP address in the same pass, and therefore IP address and IP packet–related functions, such as firewall filtering, are normally not available at the egress PE for VPN traffic. As a result, the PE router simply shoves the (now) native IP packet out the indexed VRF interface to the attached CE, which in this case quickly realizes that the packet is actually addressed to the "other end" of the link. The CE therefore returns the favor by sending the native IP packet back to the PE where it can now be processed as an IP packet by the IP II. This default behavior is normally not a problem, as the vast majority of "real" traffic would wind up pointing to the VRF interface as a next hop anyway, and the extra hop for the exception traffic—in other words, traffic destined to the local PEs VRF interface address—does not break anything per se.

There are cases where IP II processing of egress VPN traffic is desirable, such as when you need JUNOS software firewall filtering functionality at the egress PE or to accommodate the handling of Layer 3 to Layer 2 address mappings needed on multi-access interfaces such as Ethernet. While not an issue in the initial Layer 3 VPN configuration scenario, certain scenarios, such as when multiple CE devices share a common VRF subnet, require the ability to map IP to MAC addresses dynamically to achieve optimal routing from the PE to the local CE devices.

When IP II functionality is needed at the egress PE, consider using the `vrf-table-label` statement when your PE's core-facing interfaces are supported (only point-to-point, non-channelized core-facing interfaces were supported with this option at the time of this writing). When a Tunnel Services (TS) PIC is installed, you can loop egress VPN traffic back through the IP II for a second, Layer 3–based processing run, with the `vt-interface` configuration statement. The specifics of the JNCIE test bed used to develop this book prevent the use of `vrf-table-label` so its configuration, while very straightforward, can not be demonstrated.

PE-CE OSPF Routing

The primary goal of this configuration scenario is to verify that the JNCIE candidate is capable of configuring and troubleshooting a Layer 3 VPN that uses OSPF routing on the PE-CE links. Slight differences in the configuration requirements of this scenario have been added to facilitate the demonstration of additional Layer 3 VPN configuration options, some of which are not specific to PE-CE OSPF routing. The Layer 3 VPN topology for PE-CE OSPF routing is shown in Figure 7.5.

FIGURE 7.5 L3 VPN with OSPF routing

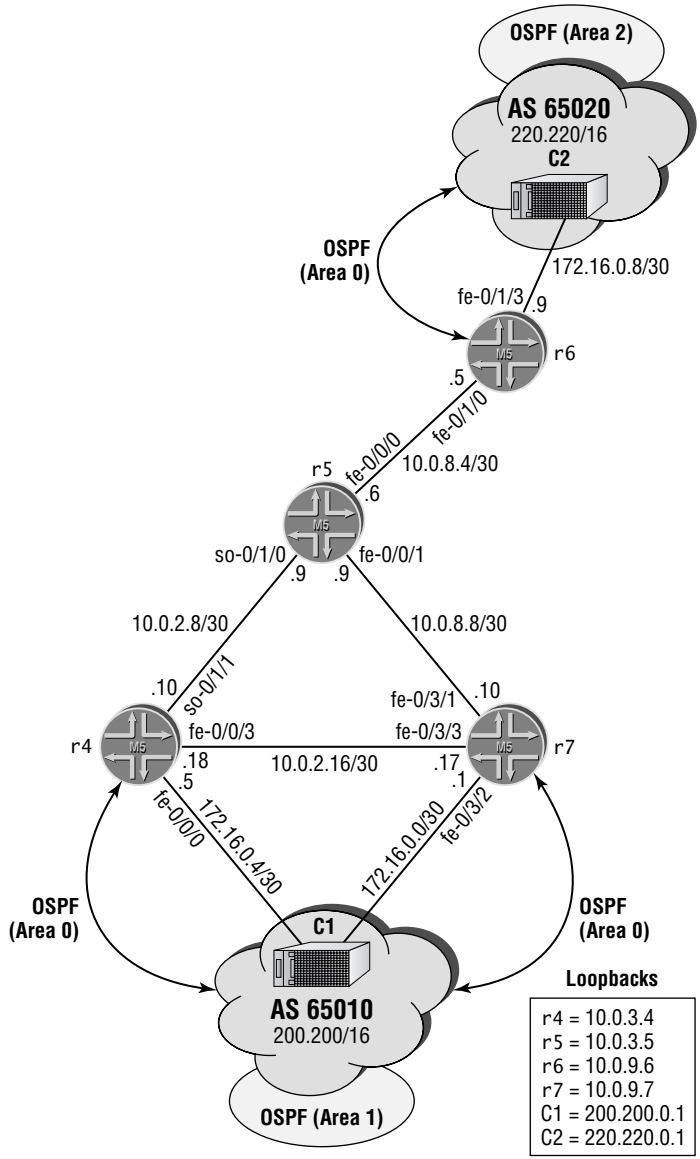


Figure 7.5 shows that you must configure PE routers `r4`, `r6`, and `r7` to support OSPF routing with their attached CEs in area 0. Also of note is that the CE devices have been reconfigured to run OSPF area 0 on their VRF interfaces and their loopback interfaces have been assigned to area 1 and area 2 for C1 and C2, respectively. Both CE routers have a policy in effect to redistribute their /16 prefixes into OSPF. The pertinent portions of the CE router configurations

are shown here in the context of C2:

```
[edit]
lab@c2# show interfaces
fe-0/0/0 {
    unit 0 {
        family inet {
            address 172.16.0.10/30;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 220.220.0.1/32;
        }
    }
}

[edit]
lab@c2# show routing-options
    static {
        route 220.220.0.0/16 discard;
    }

[edit]
lab@c2# show protocols
ospf {
    export stat;
    area 0.0.0.0 {
        interface fe-0/0/0.0;
    }
    area 0.0.0.2 {
        interface lo0.0;
    }
}

[edit]
lab@c2# show policy-options
policy-statement stat {
    from protocol static;
    then accept;
}
```

The key point regarding the configuration of the CE routers is that you can expect C2 to advertise the 172.16.0.8/30 route to PE r6 in a Type 1 (router) LSA while the 220.220.0.1 and 220.220/16 prefixes are advertised with Type 3 (network summary) and Type 5 (AS External) LSAs, respectively. This behavior is confirmed by viewing the OSPF link-state database (LSDB) at C2 for area 0:

```
[edit]
lab@c2# run show ospf database area 0 detail

    OSPF link state database, area 0.0.0.0
 Type       ID               Adv Rtr           Seq      Age  Opt  Cksum  Len
Router   *220.220.0.1      220.220.0.1      0x80000008   277  0x2  0xea26  36
  bits 0x3, link count 1
  id 172.16.0.8, data 255.255.255.252, Type Stub (3)
  TOS count 0, TOS 0 metric 1
Summary *220.220.0.1      220.220.0.1      0x80000005   277  0x2  0x17cc  28
  mask 255.255.255.255
  TOS 0x0, metric 0
    OSPF AS SCOPE link state database
 Type       ID               Adv Rtr           Seq      Age  Opt  Cksum  Len
Extern   *220.220.0.0      220.220.0.1      0x80000004   277  0x2  0x9ac0  36
  mask 255.255.0.0
  Type 2, TOS 0x0, metric 0, fwd addr 0.0.0.0, tag 0.0.0.0
```

To complete the OSPF-based Layer 3 scenario, you must reconfigure the subset of routers shown earlier in Figure 7.5 according to these criteria:

- You must delete the existing routing instances on r4, r6, and r7 before you begin your configuration.
- Establish a L3 VPN providing connectivity between C1 and C2.
- You must support traffic that originates on VRF interfaces.
- Ensure that the VPN is not disrupted by the failure of r4 or r7, or by any internal link/interface failure.
- Your VPN configuration can not disrupt existing IPv4 routing and forwarding functionality.
- Ensure that the /32 route for each CE's loopback interface is received as a Type 5 LSA by the remote CE device.
- Assign an ASN-based RD to each VRF.
- You may not use the vrf-target option.
- Ensure that r4 and r7 will never advertise routes received from C1 back to C1.
- Configure r6 to log a warning when the number of routes in C2's VRF exceeds 100.

You should assume that both CE routers are correctly configured, and that you may access them for purposes of testing connectivity only. As with the static and BGP routing example, initial configuration will focus on r4 and r6. The redundancy provided by r7, and any issues

relating to community tagging and route filtering, are addressed once initial VPN functionality is confirmed between r4 and r6.

MP-IBGP session support for the inet-vpn family between PE routers is left in place from the previous configuration scenario, as is the MPLS forwarding and control plane infrastructure from the preliminary setup task. The OSPF scenario therefore requires only the configuration of VRFs and any related policy. The actual JNCIE lab examination might not afford you the luxury of any pre-configured parameters.

L3 VPN Configuration: OSPF Routing

Your configuration begins at r4 with the deletion of the VRFs that remain from the previous BGP and static routing scenario:

```
[edit]
lab@r4# delete routing-instances

[edit]
lab@r4#
```

It is suggested that you also delete the VRF instance at r6 and r7 at this time (not shown) and commit all changes before proceeding. You begin the OSPF-based VRF configuration at r6 by defining a VRF instance called *c2-ospf*; the initial VRF definition is shown here:

```
[edit routing-instances c2-ospf]
lab@r6# show
instance-type vrf;
interface fe-0/1/3.0;
route-distinguisher 65412:2;
vrf-import c2-import;
vrf-export c2-export;
protocols {
    ospf {
        domain-id 10.0.9.6;
        area 0.0.0.0 {
            interface all;
        }
    }
}
```

An RD that is configured within a VRF takes precedence over any automatically generated RD resulting from the use of route-distinguisher-id. This behavior means there is no need to remove the route-distinguisher-id statement that was added in the previous scenario.

The display confirms that the RD has been manually assigned to the VRF instance in keeping with the restrictions in effect for this scenario; make sure that you assign a unique value for the RD associated with r4 and r7 when their VRFs are defined. The domain-id value configured must also be unique at all PE routers to ensure that the Type 3 summary LSAs, which advertise the CE router's loopback address, are correctly distributed to the remote PE as an AS external (LSA Type 5). When no domain ID is configured, or when the configured value matches, network summary LSAs are distributed to the remote CE as a network summary, which will result in exam point loss for this scenario. In this example, the domain ID is coded, based on the PE's RID to guarantee uniqueness among all PE routers. For proper operation, your VRF export policy must attach the domain ID community to the routes being advertised to the remote PEs.

The prohibition against using the vrf-target option means that you need to manually define the associated VRF policy and RT community. The *c2-ospf* VRF references the *c2-import* and *c2-export* VRF policies that must also be defined. Lastly, note that the OSPF stanza in the *c2-ospf* VRF correctly places r6's fe-0/1/3 interface into OSPF area 0.

The *c2-import* policy is displayed at r6:

```
[edit policy-options policy-statement c2-import]
lab@r6# show
term 1 {
    from {
        protocol bgp;
        community c1-c2-vpn;
    }
    then accept;
}
```

The highlighted portion calls out that the policy is written to match on BGP routes that contain the community named *c1-c2-vpn*. When a protocol-based match condition is included, you must match on the BGP protocol. This is because the PE routers exchange VPN routes through MP-BGP. In this example, the *c1-c2-vpn* community functions as the VPN's RT. You must use care to ensure that a common RT (or the policy needed to match on different RTs) is in place at all PE routers; an explict value for the RT community is not specified in the scenario's rules of engagement. A common RT will be defined on all PEs that serve the C1-C2 VPN in this example:

```
[edit policy-options]
lab@r6# set community c1-c2-vpn members target:65412:69
```

With the VRF import policy confirmed, you move on to the display of the *c2-export* policy, again at r6:

```
[edit policy-options policy-statement c2-export]
lab@r6# show
term 1 {
    from protocol ospf;
    then {
```

```
        community add c1-c2-vpn;
        community add domain;
        accept;
    }
}
term 2 {
    from {
        protocol direct;
        route-filter 172.16.0.8/30 exact;
    }
    then {
        community add c1-c2-vpn;
        accept;
    }
}
```

The key aspects of the *c2-export* policy are the OSPF and direct protocol matching conditions that result in the attachment of the *c1-c2-vpn* RT community to matching routes as they are advertised to remote PEs. The first term catches the routes learned from the C2 router through OSPF while the second term causes the direct route associated with r6's VRF interface to be advertised. Because the PE router will have at least one OSPF route that points to C2 as the next hop, the direct route will be advertised with a VPN label; this behavior is critical to support the stipulation that your design must support VPN traffic that originates on the multi-access VRF interfaces. Also of significance is the attachment of an OSPF domain ID community, which is simply called *domain* in this example. The *domain* community will be compared to the locally configured domain ID value in the remote PE to determine how Type 3 LSAs (network summaries) should be presented to the attached CE. This scenario requires that network summaries be delivered as an AS external (Type 5), so you must ensure that the domain ID that is attached to the OSPF routes does not match the value configured in the remote PE. The Domain ID extended community is defined at r6 in a manner that is based on its loopback address to ensure uniqueness among PE routers:

```
[edit policy-options]
lab@r6# show community domain
members domain-id:10.0.9.6:0;
```

Note that the domain community is not attached to the direct route in this example. While attaching the community causes no harm, it also has no observable effect because the direct route will be identified as an external route, and no amount of domain ID configuration can change a Type 5 into a Type 3. The domain ID only functions to control how routes identified as a network summary are advertised to the local CE.

The final modification at r6 relates to the configuration of a prefix limit for the *c2-ospf* VRF that will generate log warnings when the total number of the routes in the VRF exceeds 100:

```
[edit routing-instances c2-ospf routing-options]
lab@r6# set maximum-routes 100 log-only
```

The key to the maximum route requirement is the need to configure the `maximum-routes` option in the *c2-ospf* VRF as opposed to the main routing instance. The *c2-ospf* VRF is displayed for verification:

```
[edit routing-instances c2-ospf]
lab@r6# show
instance-type vrf;
interface fe-0/1/3.0;
route-distinguisher 65412:2;
vrf-import c2-import;
vrf-export c2-export;
routing-options {
    maximum-routes {
        100;
        log-only;
    }
}
protocols {
    ospf {
        domain-id 10.0.9.6;
        export bgp-ospf;
        area 0.0.0.0 {
            interface all;
        }
    }
}
```

After committing the changes at `r6`, a similar configuration is added to `r4`. The modified portions of `r4`'s configuration are shown next:

```
[edit]
lab@r4# show routing-instances
c1-ospf {
    instance-type vrf;
    interface fe-0/0/0.0;
    route-distinguisher 65412:1;
    vrf-import c1-import;
    vrf-export c1-export;
    protocols {
        ospf {
            domain-id 10.0.3.4;
            area 0.0.0.0 {
```

```
                interface all;
            }
        }
    }
}
```

Note that the RD and Domain ID values in r4's *c1-ospf* VRF are unique when compared to the values in r6's *c2-ospf* VRF. A matching RT community is defined at r4, as is a unique *domain* community:

```
[edit]
lab@r4# show policy-options community c1-c2-vpn
members target:65412:69;

[edit]
lab@r4# show policy-options community domain
members domain-id:10.0.3.4:0;
```

The VRF import and export policy statements on r4 are very similar to those shown for r6:

```
[edit]
lab@r4# show policy-options policy-statement c1-import
term 1 {
    from {
        protocol bgp;
        community c1-c2-vpn;
    }
    then accept;
}

[edit]
lab@r4# show policy-options policy-statement c1-export
term 1 {
    from protocol ospf;
    then {
        community add c1-c2-vpn;
        community add domain;
        accept;
    }
}
term 2 {
    from {
        protocol direct;
        route-filter 172.16.4.0/30 exact;
```

```
    }
    then {
        community add c1-c2-vpn;
        accept;
    }
}
```

Make sure that you commit your changes on r4 before proceeding to the confirmation section.

Initial L3 VPN Confirmation: OSPF Routing

With the changes committed at r4 and r6, you proceed to initial verification so that any configuration problems can be resolved before you make similar mistakes in r7's configuration. Confirmation begins with the determination of the OSPF adjacency status at r4 and r6:

```
[edit]
lab@r4# run show ospf neighbor instance c1-ospf
  Address          Interface              State     ID               Pri  Dead
172.16.0.6         fe-0/0/0.0             Full      200.200.0.1      128    37
```

Although not shown, you can assume that r6 is also fully adjacent with C2. The next command displays the VRF table at PE router r4:

```
[edit]
lab@r4# run show route table c1-ospf

c1-ospf.inet.0: 10 destinations, 10 routes (10 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

172.16.0.0/30      *[OSPF/10] 00:00:34, metric 2
                    > to 172.16.0.6 via fe-0/0/0.0
172.16.0.4/30      *[Direct/0] 02:29:32
                    > via fe-0/0/0.0
172.16.0.5/32      *[Local/0] 02:38:57
                      Local via fe-0/0/0.0
172.16.0.8/30      *[BGP/170] 00:28:54, localpref 100, from 10.0.9.6
                      AS path: I
                    > via so-0/1/1.0, label-switched-path r4-r6
200.200.0.0/16     *[OSPF/150] 00:29:03, metric 0, tag 0
                    > to 172.16.0.6 via fe-0/0/0.0
200.200.0.1/32     *[OSPF/10] 00:29:03, metric 1
                    > to 172.16.0.6 via fe-0/0/0.0
200.200.1.0/24     *[OSPF/150] 00:29:03, metric 0, tag 0
                    > to 172.16.0.6 via fe-0/0/0.0
```

```
220.220.0.0/16     *[BGP/170] 00:28:54, MED 0, localpref 100, from 10.0.9.6
                      AS path: I
                    > via so-0/1/1.0, label-switched-path r4-r6
220.220.0.1/32     *[BGP/170] 00:28:54, MED 1, localpref 100, from 10.0.9.6
                      AS path: I
                    > via so-0/1/1.0, label-switched-path r4-r6
224.0.0.5/32       *[OSPF/10] 02:38:58, metric 1
                      MultiRecv
```

The highlights call out that C2's prefixes, as advertised by `r6` through MP-IBGP, have been correctly installed in the *`c1-ospf`* VRF. The presence of these routes confirms that a matching RT (and support for VPN NLRI) has been correctly configured between `r4` and `r6`. However, just as you start to crack a well-deserved beer, you notice that C2's routes are not present at C1:

```
lab@c1> show route 220.220/16

lab@c1>
```

This is a serious problem. You decide to revisit `r4` to display the contents of the OSPF database associated with the *`c1-ospf`* routing instance:

```
[edit]
lab@r4# run show ospf database instance c1-ospf

    OSPF link state database, area 0.0.0.0
 Type       ID               Adv Rtr           Seq      Age  Opt  Cksum  Len
Router  *172.16.0.5       172.16.0.5       0x80000010     9  0x2  0x3209  36
Router   200.200.0.1      200.200.0.1      0x80000010     8  0x2  0xfecc  48
Network  172.16.0.6       200.200.0.1      0x8000000a     8  0x2  0x802a  32
Summary  200.200.0.1      200.200.0.1      0x80000009     8  0x2  0x5ad5  28
    OSPF AS SCOPE link state database
 Type       ID               Adv Rtr           Seq      Age  Opt  Cksum  Len
Extern   200.200.0.0      200.200.0.1      0x80000008     8  0x2  0xddc9  36
Extern   200.200.1.0      200.200.0.1      0x80000007     8  0x2  0xd4d2  36
```

None of the routes associated with the C2 site are present in the OSPF database at `r4`, which explains why none of the routes are present in the attached C1 device. Although not shown, C2's routes are reflected in the OSPF LSDB for the *`c2-ospf`* instance at `r6`. Based on these symptoms, can you identify the problem?

As a hint, consider that the routes are present as BGP routes in the *`c1-ospf`* VRF at `r4`, and that these routes are not present in the instance's OSPF database at `r4`. This set of symptoms definitely indicates that the problem lies with the local PE router. As a final hint, think about the default export policy for OSPF, and consider the nature of the routes that you want the OSPF instance on `r4` to advertise to the C1 device.

If you are thinking that some form of OSPF export policy is needed on the PE routers to effect the redistribution of BGP routes into OSPF, then you are getting very hot! The highlights added to the following capture call out the changes required to redistribute the BGP routes, as learned from the remote PE, into the OSPF protocol. The changes shown for r4 are also needed at r6:

```
[edit]
lab@r4# show routing-instances c1-ospf protocols ospf
domain-id 10.0.3.4;
export bgp-ospf;
area 0.0.0.0 {
    interface all;
}

[edit]
lab@r4# show policy-options policy-statement bgp-ospf
term 1 {
    from protocol bgp;
    then accept;
}
```

When the changes are committed on both r4 and r6, the OSPF database for the *c1-ospf* VRF is again displayed at r4:

```
[edit]
lab@r4# run show ospf database instance c1-ospf

    OSPF link state database, area 0.0.0.0
 Type       ID               Adv Rtr           Seq      Age  Opt  Cksum  Len
Router  *172.16.0.5       172.16.0.5       0x80000011    11  0x2  0x3602  36
Router   200.200.0.1      200.200.0.1      0x80000010    53  0x2  0xfecc  48
Network  172.16.0.6       200.200.0.1      0x8000000a    53  0x2  0x802a  32
Summary  200.200.0.1      200.200.0.1      0x80000009    53  0x2  0x5ad5  28
    OSPF AS SCOPE link state database
 Type       ID               Adv Rtr           Seq      Age  Opt  Cksum  Len
Extern  *172.16.0.8       172.16.0.5       0x80000001     3  0x2  0xd722  36
Extern   200.200.0.0      200.200.0.1      0x80000008    53  0x2  0xddc9  36
Extern   200.200.1.0      200.200.0.1      0x80000007    53  0x2  0xd4d2  36
Extern  *220.220.0.0      172.16.0.5       0x80000001     3  0x2  0x2ed3  36
Extern  *220.220.0.1      172.16.0.5       0x80000001     3  0x2  0xaad5  36
```

The highlights call out the C2 routes that are now present in the *c1-ospf* routing instance's link-state database, which confirms the operation of the *bgp-ospf* export policy is operating as designed. Note that both the 220.220/16 and the 220.220.0.1/32 prefixes are represented as AS externals (Type 5 LSAs) as required by this scenario's restrictions. A configuration that does not result in mismatched OSPF domain IDs results in the 220.220.0.1 route being represented as a

network summary (Type 3 LSA). These LSAs should now be present in the attached CE's LSDB, and as a result, C1 should now have a route to C2 destinations:

```
lab@c1> show route 220.220/16

inet.0: 11 destinations, 11 routes (11 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

220.220.0.0/16     *[OSPF/150] 00:49:30, metric 0, tag 3489726340
                    > to 172.16.0.5 via fe-0/0/0.0
220.220.0.1/32     *[OSPF/150] 00:39:05, metric 2, tag 3489726340
                    > to 172.16.0.5 via fe-0/0/0.0
```

The routes are present, and are confirmed to be OSPF externals by virtue of the preference setting of 150. End-to-end connectivity, and the ability to generate traffic from the VRF interface, are now confirmed; note that the first traceroute is sourced from C1's VRF interface while the second is sourced from its loopback address:

```
lab@c1> traceroute 220.220.0.1
traceroute to 220.220.0.1 (220.220.0.1), 30 hops max, 40 byte packets
 1  172.16.0.5 (172.16.0.5)  0.398 ms  0.284 ms  0.276 ms
 2  * * *
 3  10.0.8.5 (10.0.8.5)  0.287 ms  0.238 ms  0.233 ms
     MPLS Label=100001 CoS=0 TTL=1 S=1
 4  220.220.0.1 (220.220.0.1)  0.646 ms  0.532 ms  0.528 ms

lab@c1> traceroute 220.220.0.1 source 200.200.0.1
traceroute to 220.220.0.1 (220.220.0.1) from 200.200.0.1, 30 hops max, 40 byte
   packets
 1  172.16.0.5 (172.16.0.5)  0.388 ms  0.281 ms  0.272 ms
 2  * * *
 3  10.0.8.5 (10.0.8.5)  0.276 ms  0.232 ms  0.230 ms
     MPLS Label=100001 CoS=0 TTL=1 S=1
 4  220.220.0.1 (220.220.0.1)  0.626 ms  0.532 ms  0.531 ms
```

As described in the previous scenario, the time-out on the second hop is expected when the ingress node is equipped with an E-FPC. The initial confirmation results indicate that the configuration in place at `r4` and `r6` is meeting all relevant stipulations.

Adding and Confirming Redundancy and Route Filtering

With initial OSPF connectivity confirmed between `r4` and `r6`, you now address the redundancy requirements of the scenario by configuring `r7` to support OSPF interaction with C1. The initial changes made to `r7` are shown here:

```
[edit]
lab@r7# show routing-instances
```

```
c1-ospf {
    instance-type vrf;
    interface fe-0/3/2.0;
    route-distinguisher 65412:1;
    vrf-import c1-import;
    vrf-export c1-export;
    protocols {
        ospf {
            domain-id 10.0.9.7;
            export bgp-ospf;
            area 0.0.0.0 {
                interface all;
            }
        }
    }
}
[edit]
lab@r7# show policy-options community vpn-c1-c2
members target:65412:420;

[edit]
lab@r7# show policy-options community domain
members domain-id:10.0.9.7:0;

[edit]
lab@r7# show policy-options policy-statement c1-import
term 1 {
    from {
        protocol bgp;
        community c1-c2-vpn;
    }
    then accept;
}

[edit]
lab@r7# show policy-options policy-statement c1-export
term 1 {
    from protocol ospf;
    then {
        community add c1-c2-vpn;
```

```
        community add domain;
        accept;
    }
}
term 2 {
    from {
        protocol direct;
        route-filter 172.16.0.0/30 exact;
    }
    then {
        community add c1-c2-vpn;
        accept;
    }
}
[edit]
lab@Tokyo# show policy-options policy-statement bgp-ospf
term filter_c1_routes {
    from community c1;
    then reject;
}
term 1 {
    from protocol bgp;
    then accept;
}
```

After committing the changes, you confirm redundancy with the verification that both `r4` and `r7` are advertising C2's routes as AS externals to C1:

```
lab@c1> show ospf database extern
    OSPF AS SCOPE link state database
 Type       ID               Adv Rtr           Seq      Age  Opt  Cksum  Len
Extern   172.16.0.8       172.16.0.1       0x80000001  367  0x2  0xef0e  36
Extern   172.16.0.8       172.16.0.5       0x80000001  473  0x2  0xd722  36
Extern  *200.200.0.0      200.200.0.1      0x80000008  521  0x2  0xddc9  36
Extern  *200.200.1.0      200.200.0.1      0x80000007  521  0x2  0xd4d2  36
Extern   220.220.0.0      172.16.0.1       0x80000001  367  0x2  0x46bf  36
Extern   220.220.0.0      172.16.0.5       0x80000001  473  0x2  0x2ed3  36
Extern   220.220.0.1      172.16.0.1       0x80000001  367  0x2  0xc2c1  36
Extern   220.220.0.1      172.16.0.5       0x80000001  473  0x2  0xaad5  36
```

The duplicate entries for C2's routes in this display confirm that both `r4` and `r7` are receiving VPN routes from `r6`, and that both PE routers are correctly redistributing the routes into OSPF

as Type 5 LSAs. Redundancy is also confirmed at r6 by observing that two sets of BGP routes exist for prefixes related to the C1 site:

```
[edit]
lab@r6# run show route table c2-ospf 200.200/16

c2-ospf.inet.0: 10 destinations, 15 routes (10 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

200.200.0.0/16     *[BGP/170] 00:10:19, MED 0, localpref 100, from 10.0.3.4
                      AS path: I
                    > to 10.0.2.14 via fe-0/1/1.0, label-switched-path r6-r4
                    [BGP/170] 00:07:47, MED 0, localpref 100, from 10.0.9.7
                      AS path: I
                    > to 10.0.8.6 via fe-0/1/0.0, label-switched-path r6-r7
```

The presence of two BGP routes in the *c1-ospf* VRF table, one from r4 and the other from r7, indicates that both r4 and r7 are correctly configured to redistribute the OSPF routes learned from the C1 device into MP-IBGP. A final confirmation check verifies that r7 has connectivity to the local and remote CE devices:

```
[edit]
lab@r7# run traceroute routing-instance c1-ospf 200.200.0.1
traceroute to 200.200.0.1 (200.200.0.1), 30 hops max, 40 byte packets
 1  200.200.0.1 (200.200.0.1)  0.379 ms  0.201 ms  0.116 ms

[edit]
lab@r7# run traceroute routing-instance c1-ospf 220.220.0.1
traceroute to 220.220.0.1 (220.220.0.1), 30 hops max, 40 byte packets
 1  10.0.8.5 (10.0.8.5)  0.405 ms  0.301 ms  0.245 ms
     MPLS Label=100001 CoS=0 TTL=1 S=1
 2  220.220.0.1 (220.220.0.1)  0.664 ms  0.549 ms  0.525 ms
```

Both of the traceroutes initiated at r7 succeed, which confirms redundancy and brings you to the final requirement of this configuration example. You must now configure r4 and r7 in such a way that you can guarantee that neither router will re-advertise routes that originated at site C1 back to site C1. This requirement can be tricky because the default preference settings for OSPF externals and BGP routes currently result in the desired behavior, which may lead some candidates to assume that no added configuration is necessary. Currently, neither r4 nor r7 is re-advertising routes that originate at site C1 back to C1 because both r4 and r5 prefer the OSPF external route, as learned from C1, to the BGP version of the route that is learned over the MP-IBGP session between the PE routers. This condition is shown here:

```
[edit]
lab@r7# run show route 200.200/16
```

```
c1-ospf.inet.0: 10 destinations, 15 routes (10 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

200.200.0.0/16     *[OSPF/150] 00:04:54, metric 0, tag 0
                    > to 172.16.0.2 via fe0/3/2.0
                    [BGP/170] 00:04:43, MED 0, localpref 100, from 10.0.3.4
                      AS path: I
                    > to 10.0.2.18 via fe-0/3/3.0, label-switched-path r7-r4
```

However, without the addition of route filtering precautions, a temporary change in routing preference for OSPF externals at r4 results in the re-advertisement of the 200.200/16 routes back to C1; this behavior is a clear violation of the restrictions in effect for this scenario, and thereby shows that additional configuration is required:

```
[edit routing-instances c1-ospf protocols ospf]
lab@r4# set external-preference 175

[edit routing-instances c1-ospf protocols ospf]
lab@r4# commit
commit complete

[edit routing-instances c1-ospf protocols ospf]
lab@r4# run show route 200.200/16

c1-ospf.inet.0: 10 destinations, 15 routes (10 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

200.200.0.0/16     *[BGP/170] 00:00:05, MED 0, localpref 100, from 10.0.9.7
                      AS path: I
                    > to 10.0.2.17 via fe-0/0/3.0, label-switched-path r4-r7
                    [OSPF/175] 00:00:05, metric 0, tag 0
                    > to 172.16.0.6 via fe-0/0/0.0
200.200.0.1/32     *[OSPF/10] 00:00:05, metric 1
                    > to 172.16.0.6 via fe-0/0/0.0
                    [BGP/170] 00:00:05, MED 1, localpref 100, from 10.0.9.7
                      AS path: I
                    > to 10.0.2.17 via fe-0/0/3.0, label-switched-path r4-r7
200.200.1.0/24     *[BGP/170] 00:00:05, MED 0, localpref 100, from 10.0.9.7
                      AS path: I
                    > to 10.0.2.17 via fe-0/0/3.0, label-switched-path r4-r7
                    [OSPF/175] 00:00:05, metric 0, tag 0
                    > to 172.16.0.6 via fe-0/0/0.0
```

With the BGP version of the route now active at r4, the *bgp-ospf* export policy results in the route being incorrectly re-advertised back to C1:

```
lab@c1> show ospf database extern detail | match 200.200.0.0
Extern   200.200.0.0      172.16.0.5       0x80000001     3  0x2  0x2406  36
Extern  *200.200.0.0      200.200.0.1      0x80000006   889  0x2  0xe1c7  36
```

The best way to resolve this type of problem is to define a unique *origin* community that is associated with site C1 and attached to BGP updates using VRF export policy. Once the origin community is attached, you can then filter routes between the PE routers using VRF `import` policy, or from the attached CE using routing instance `export` policy. The community and VRF policy changes made to r4 to support community based filtering are shown here with highlights added to call out modifications to existing configuration stanzas:

```
[edit policy-options]
lab@r4# show community c1
members origin:65412:1;
```

Although you can filter on virtually any community value, an origin community is used in this example because it is in keeping with the community's intended use. The modified VRF export policy is displayed here at r4:

```
[edit policy-options]
lab@r4# show policy-statement c1-export
term 1 {
    from protocol ospf;
    then {
        community add c1-c2-vpn;
        community add domain;
        community add c1;
        accept;
    }
}
term 2 {
    from {
        protocol direct;
        route-filter 172.16.0.4/30 exact;
    }
    then {
        community add c1-c2-vpn;
        accept;
    }
}
```

The change made to the *c1-export* policy ensures that the origin community is attached to the routes sent to remote PE routers. The presence of the *c1* community has no effect at r6, because no policy changes relating to the *c1* origin community are in effect there. It is critical to note that VRF import policy is *not* modified at r4 and r7 to reject routes with the *c1* community attached, because doing so would impact the redundancy requirements of the scenario. Your goal is to allow the advertisement of C1's prefixes between the PE routers while *also* preventing the PE routers from re-advertising C1's routes back to site C1. To achieve this behavior, you need to modify the *bgp-ospf* export policy at r4 and r7 as shown next:

```
[edit policy-options]
lab@r4# show policy-statement bgp-ospf
term filter_c1_routes {
    from community c1;
    then reject;
}
term 1 {
    from protocol bgp;
    then accept;
}
```

Note that the `insert` command is used to ensure that the *c1-import* policy's original term *1* is evaluated after the new *filter_c1_routes* term. After committing the changes, you can easily verify the results by examining the OSPF database at C1 to confirm that r4 is no longer re-advertising the 200.200/16 prefix back to C1. Note that the modified preference value for OSPF externals is still in effect at r4 at this time:

```
lab@c1> show ospf database extern detail | match 200.200.0.0
Extern   200.200.0.0      172.16.0.5       0x80000001  3600  0x2  0x2406  36
Extern  *200.200.0.0      200.200.0.1      0x80000006  1383  0x2  0xe1c7  36
```

Immediately after committing the policy changes at r4, you observe that the 200.200/16 external LSA generated by r4 has its age set to 3600, which is the maximum age of an OSPF LSA. This is a good indication that the filtering changes now in effect at r4 are producing the desired behavior. A few moments later, the external LSA that was generated by r4 is flushed from the OSPF database:

```
lab@c1> show ospf database extern detail | match 200.200.0.0
Extern  *200.200.0.0      200.200.0.1      0x80000006  1395  0x2  0xe1c7  36
```

The output confirms that the only external LSA for the 200.200/16 route is the one generated locally by C1. Before proceeding, you should return r4's preference values to their default settings and repeat the confirmation test by modifying the OSPF external preference in r7's routing instance. Although the results are not shown here for brevity's sake, you can assume that the same results are observed when the OSPF external preference is temporarily modified in r7's *c1-ospf* instance.

These results conclude the verification tasks for the Layer 3 VPN with OSPF routing scenario.

Layer 3 VPN Summary

Layer 3 VPNs are based on the concept of per-site routing tables, called VRFs, which house the routes associated with a given VPN in isolation from the routes associated with other VPNs and those in the main routing instance. Layer 3 VPNs based on the 2547 bis model make use of MP-BGP to advertise IPv4 and IPv6 VPN NLRI between PE routers. Each PE router installs the VPN routes into the VRF (or VRFs) identified by the attached route target community according to the associated VRF import policy. At export, VRF export policy attaches one or more RT communities for use by remote PEs upon receiving the route.

This section demonstrated the configuration of Layer 3 VPNs that were based on static, BGP, and OSPF routing on the PE-CE VRF links in JUNOS software. The section also demonstrated recent features that simplify RD and VRF policy configuration, in addition to the manually configured alternatives. The examples in this section also demonstrated how VPN traffic can be sourced and destined to multi-access VRF interfaces when the appropriate VRF export and import policy is in effect and the PE has at least one route in the VRF that identifies the attached CE as the next hop. Although not demonstrated, the use of `vt-interface` and `vrf-table-label`, which provide IP II filtering (and ARP mapping) functions at the egress PE, were described.

Because many operational mode commands default to the main routing instance, you must remember to use the `instance`, `routing-instance` and `vpn` switches with the appropriate *`vpn-name`* argument when performing VPN operational mode analysis and troubleshooting. To be effective with Layer 3 VPNs, the JNCIE candidate must be able to quickly isolate and diagnose problems in the VPN forwarding plane (MPLS signaling and MPLS forwarding, double label push operations, and so on) and in the VPN control plane (MP-BGP, route targets, extended communities, VRF policy, and so on). Throughout this section, the reader was exposed to operational mode commands that are useful in determining the operational status of Layer 3 VPNs on Juniper Networks M-series and T-series platforms.

Layer 2 VPNs (Draft-Kompella and Draft-Martini)

Layer 2 VPNs share many of the same concepts and terms as their Layer 3 counterparts, especially in the case of draft-Kompella solutions, because the control plane is based on the same MP-BGP signaling as is found in the 2547 bis model. The principal difference between

Layer 3 and Layer 2 VPNs is that the PE and CE routers do not share a subnet and do not interact beyond the forwarding of frames based strictly on Layer 2 parameters.

In many ways, you can compare the interaction of the CE and PE devices in a Layer 2 VPN to the interaction of a host system and a transparent bridge. While the transparency of the PE routers and the service provider's network has certain advantages, such as the ability to support non-routable protocols, there are some drawbacks. For example, the fact that the CE router does not interact at the IP layer with the PE router in a Layer 2 solution makes it very difficult to ascertain whether the local PE-CE link is correctly transporting traffic. In effect, you can either ping end-to-end between CE devices, in which case all is well, or you can not. In many cases, the provider will provision a second interface to the CE device for out of band (OoB) management and diagnostic purposes; often this second interface is manifest as a second logical unit on the physical device that is also used to provide Layer 2 VPN connectivity. The presence of a non-VRF interface can also be used to provide a CE device with Internet access when global IP addressing is in effect on the CE.

JUNOS software supports three different types of Layer 2 VPNs: Circuit Cross Connect (CCC), draft-Kompella, and draft-Martini. Of these, the draft-Kompella and draft-Martini solutions enjoy the greatest degree of interest due to their advantages in the scaling and provisioning arenas. Because CCC was a precursor to the full-blown Layer 2 VPN solutions that are now available, the configuration and testing of CCC connections is not covered in this chapter.

As of this writing, it is unclear which Layer 2 VPN standard will dominate in the industry. The draft-Kompella approach, being based on BGP signaling, is very similar to 2547 bis; providers that have deployed 2547 bis Layer 3 VPNs may well deploy a draft-Kompella solution due to the similarities in the way the two VPN technologies are configured and tested. The draft-Kompella approach also offers the ability to pre-provision a VPN so that future site additions do not require changes to the configuration of PE routers that attach to existing sites. On the other hand, the draft-Martini approach makes use of LDP signaling, which can simplify network operations when LDP is used for MPLS signaling *and* Layer 3 VPNs are not being offered. This section provides configuration and testing scenarios for both Layer 2 VPN drafts.

Draft-Kompella

You begin with a draft-Kompella based Layer 2 VPN scenario to leverage the fact that the test bed currently has an RSVP-based MPLS infrastructure in place, and because the configuration is very similar to the Layer 3 VPN examples demonstrated previously. While draft-Kompella VPNs can operate over LDP signaled LSPs, the fact that LDP support is *mandatory* for draft-Martini solutions results in the deployment of LDP in conjunction with the draft-Martini scenario.

Your draft-Kompella VPN scenario requires the configuration of a two-site Layer 2 VPN that supports CE-CE OSPF routing, as shown in Figure 7.6. Your configuration must also provide C1 with access to Internet routes.

FIGURE 7.6 Draft-Kompella Layer 2 VPN

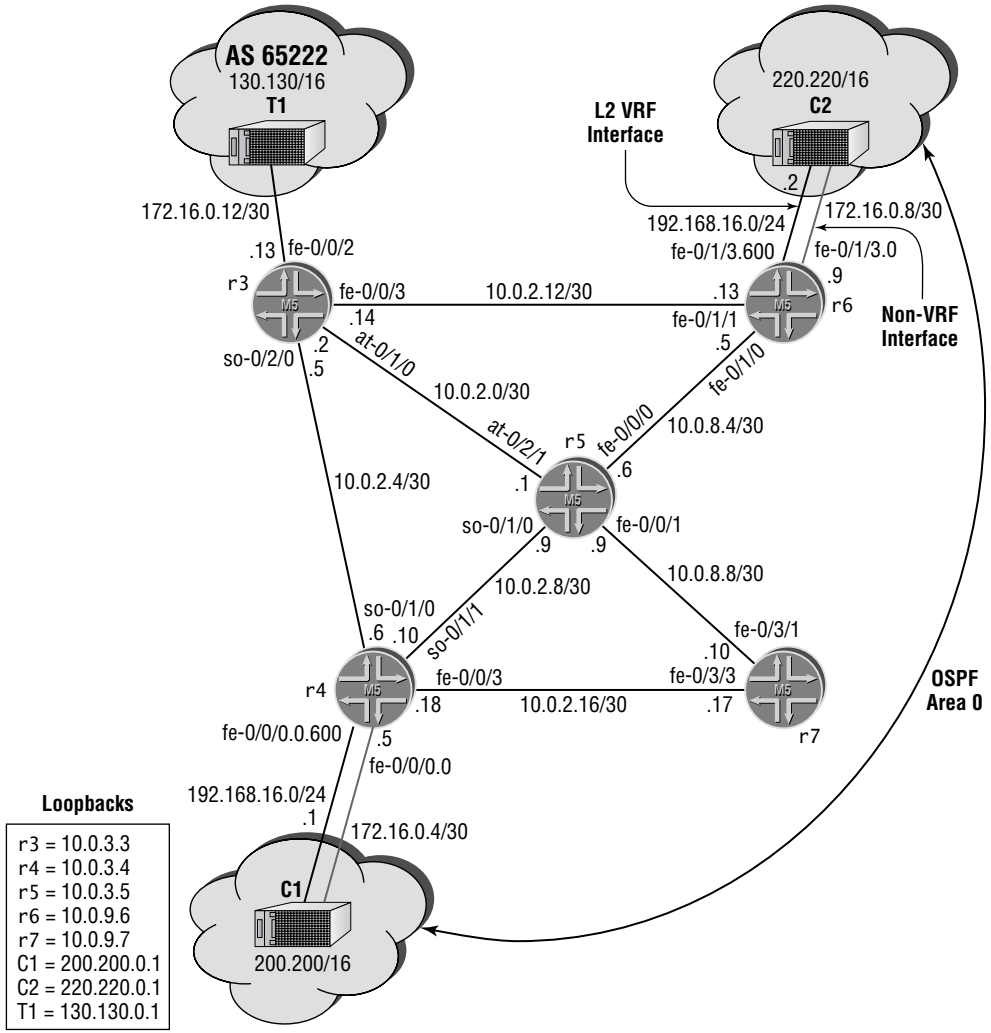


Figure 7.6 shows that the CE devices have been reconfigured to support VLAN tagging on their VRF interfaces, with two logical interfaces defined at each site. In this example, logical unit 0 is provisioned to provide OoB management (and Internet access) between the PE and CE devices while logical unit 600 is used to interconnect the two sites with a Layer 2 VPN. A key point in the figure is the fact that the two CE devices now share a logical IP subnet in the form of 192.168.16/24, with CE1 having host ID 1 and CE 2 being assigned host ID 2. Note that both CE devices have also been configured to run OSPF area 0 on their VRF interface. The relevant portions of the CE device configuration are shown here in the context of C1:

```
[edit]
lab@c1# show interfaces
```

```
fe-0/0/0 {
    vlan-tagging;
    unit 0 {
        vlan-id 1;
        family inet {
            address 172.16.0.6/30;
        }
    }
    unit 600 {
        vlan-id 600;
        family inet {
            address 192.168.16.1/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 200.200.0.1/32;
        }
    }
}

[edit]
lab@c1# show protocols
ospf {
    export stat;
    area 0.0.0.0 {
        interface fe-0/0/0.600;
    }
}
[edit]
lab@c1# show routing-options
    static {
        route 200.200.0.0/16 discard;
        route 200.200.1.0/24 reject;
    }

[edit]
lab@c2# show policy-options
policy-statement stat {
```

```
    from protocol static;
    then accept;
}
```

To complete the first Layer 2 VPN scenario, you must configure the subset of routers shown earlier in Figure 7.6 according to these criteria:

- Delete the routing instance configuration in place at r4, r6, and r7. If desired, you may also delete any VRF policy and related community definitions from the previous Layer 3 scenario.
- Add a second pair of LSPs between r4 and r6 with a 10Mbps bandwidth reservation.
- Without using LDP, establish a L2 VPN providing connectivity between C1 and C2.
- Configure PE routers r4 and r6 to be compatible with the attached CE devices, including the VRF and non-VRF interfaces.
- Your VPN configuration can not disrupt existing IPv4 routing and forwarding functionality within your AS.
- Map the VPN traffic flowing between C1 and C2 to the LSP with reserved bandwidth; you must not change the default LSP metrics to achieve this goal.
- You may add a single static route to the configuration of r4 and C1 to facilitate Internet access when packets are sourced from C1's 200.200/16 net block.
- You must not use the vrf-target option.

Draft-Kompella Configuration

Although not explicitly stated in the objectives, the restriction on LDP use mandates a draft-Kompella VPN solution. A less-than-prepared JNCIE candidate might miss this point and wind up incorrectly provisioning a draft-Martini solution. This scenario is complicated by the need to map Layer 2 VPN traffic to a specific LSP and by the need to provide Internet access to site C1. Once again, you decide to concentrate on establishing basic Layer 2 VPN connectivity before you worry about LSP mapping and Internet access.

You begin your draft-Kompella Layer 2 VPN configuration at r4 with the removal of the existing VRF and VRF-related policy configuration. Although not shown, similar commands are also entered on r6 and r7:

```
[edit]
lab@r4# delete routing-instances

[edit]
lab@r4# delete policy-options policy-statement c1-import

[edit]
lab@r4# delete policy-options policy-statement c1-export

[edit]
lab@r4# delete policy-options community c1-c2-vpn
```

```
[edit]
lab@r4# delete policy-options community domain
```

The next set of commands adds VLAN tagging support on r4's fe-0/0/0 interface and provisions the interface for Layer 2 VPN support:

```
[edit interfaces fe-0/0/0]
lab@r4# set vlan-tagging

[edit interfaces fe-0/0/0]
lab@r4# set encapsulation vlan-ccc
```

Note that vlan-tagging and vlan-ccc encapsulation must be specified at the physical device level; candidates often forget to add a ccc encapsulation at the device level and later experience forwarding plan problems! The next series of statements defines the interface's logical units and associates them with the correct VLAN IDs:

```
[edit interfaces fe-0/0/0]
lab@r4# set unit 0 vlan-id 1

[edit interfaces fe-0/0/0]
lab@r4# set unit 600 vlan-id 600

[edit interfaces fe-0/0/0]
lab@r4# set unit 600 encapsulation vlan-ccc
```

Because VLAN ID 0 is reserved for tagging priority frames, the first available VLAN ID is assigned to the interface's existing logical unit 0; if desired, you could reassign the logical unit to match the assigned VLAN ID, but this is purely an aesthetic matter. Assigning a VLAN ID of 1 is also required to be compatible with the interface configuration of the C1 device. The VRF and non-VRF interface configuration is shown at r4 for confirmation:

```
[edit interfaces fe-0/0/0]
lab@r4# show
vlan-tagging;
encapsulation vlan-ccc;
unit 0 {
    vlan-id 1;
    family inet {
        address 172.16.0.5/30;
    }
}
unit 600 {
    encapsulation vlan-ccc;
    vlan-id 600;
}
```

The logical unit that is associated with the Layer 2 VPN has no protocol families or addressing configured. It is also worth pointing out that you must have matched VLAN IDs at opposite ends of the Layer 2 VPN unless translational cross connect (TCC) is in use. TCC is also known as Layer 2.5 IP–only interworking because it is a Layer 2 VPN solution that supports IP traffic only, due to the striping of Layer 2 framing at ingress. In this example, VLAN ID 600 is specified at both ends to accommodate this behavior.

With the VRF interface configured at `r4`, you move onto the definition of the Layer 2 VPN's VRF table. In this example, the `route-distinguisher-id statement`, which was left over from a previous Layer 3 VPN configuration, is used to create the RD for the *`c1-c2-l2`* routing instance:

```
[edit]
lab@r4# edit routing-instances c1-c2-l2

[edit routing-instances c1-c2-l2]
lab@r4# set instance-type l2vpn

[edit routing-instances c1-c2-l2]
lab@r4# set interface fe-0/0/0.600
```

Note that the *`c1-c2-vpn`* instance type is correctly configured for Layer 2 VPN operation with the `l2vpn` keyword. Because your restrictions prevent the use of the `vrf-target` feature, you must explicitly associate the Layer 2 routing instance with VRF import and export policy:

```
[edit routing-instances c1-c2-l2]
lab@r4# set vrf-import c1-c2-import

[edit routing-instances c1-c2-l2]
lab@r4# set vrf-export c1-c2-export
```

The Layer 2 routing instance's configuration is completed with the definition of the local parameters associated with local site C1:

```
[edit routing-instances c1-c2-l2]
lab@r4# set protocols l2vpn encapsulation-type ethernet-vlan

[edit routing-instances c1-c2-l2]
lab@r4# set protocols l2vpn site c1 site-identifier 1

[edit routing-instances c1-c2-l2]
lab@r4# set protocols l2vpn site c1 interface fe-0/0/0.600
```

The resulting Layer 2 VRF is displayed for visual inspection:

```
[edit routing-instances c1-c2-l2]
lab@r4# show
instance-type l2vpn;
```

```
interface fe-0/0/0.600;
vrf-import c1-c2-import;
vrf-export c1-c2-export;
protocols {
    l2vpn {
        encapsulation-type ethernet-vlan;
        site c1 {
            site-identifier 1;
            interface fe-0/0/0.600;
        }
    }
}
```

In this example, C1 has been assigned a site identifier of *1* because an explicit site ID value was not specified and this number just "makes sense" considering that the PE connects to CE device C1. The site ID assignment must be unique among all sites that make up a single Layer 2 VPN. For completeness' sake, the pre-existing `route-distinguisher-id` configuration is also displayed:

```
[edit]
lab@r4# show routing-options route-distinguisher-id
route-distinguisher-id 10.0.3.4;
```

Before you can commit your changes on `r4`, you must define the VRF import and export policy and the related RT community. You begin with the definition of the RT:

```
[edit policy-options]
lab@r4# set community c1-c2-rt members target:65412:7
```

No specific RT value is specified in your criteria. The value of *7* was chosen in this case because of the "mystic" qualities historically associated with that number; after all, you need all the help you can get with this VPN stuff. The *`c1-c2-import`* and *`c1-c2-export`* policies are now displayed. The similarities between these policies and the ones deployed in the previous Layer 3 VPN examples should be obvious:

```
[edit policy-options]
lab@r4# show policy-statement c1-c2-import
term 1 {
    from {
        protocol bgp;
        community c1-c2-rt;
    }
    then accept;
}

[edit policy-options]
lab@r4# show policy-statement c1-c2-export
```

```
term 1 {
    then {
        community add c1-c2-rt;
        accept;
    }
}
```

The principal difference between the Layer 2 and Layer 3 VRF policies lies in the omission of a protocol-based match condition in the export policy. Because a Layer 2 VPN is protocol agnostic, routes are not housed in the local VRF. Instead, the VRF houses information relating to the local site that is communicated to remote PEs to allow them to compute the labels that are be used when sending or receiving traffic to and from that site. With the VRF interface, the Layer 2 VRF, and the related VRF policy defined, you commit your changes at `r4` and move on to make similar changes at `r6`. The modifications made to its configuration in support of the draft-Kompella Layer 2 VPN scenario are shown next:

```
[edit]
lab@r6# show routing-instances
c1-c2-l2 {
    instance-type l2vpn;
    interface fe-0/1/3.600;
    vrf-import c1-c2-import;
    vrf-export c1-c2-export;
    protocols {
        l2vpn {
            encapsulation-type ethernet-vlan;
            site 2 {
                site-identifier 2;
                interface fe-0/1/3.600;
            }
        }
    }
}
```

Note that, as with `r4`'s configuration, the interface declaration at `r6` correctly specifies the logical unit value of 600. `r6`'s VRF policy is identical to that in place at `r4` and is shown here for completeness:

```
[edit]
lab@r6# show policy-options community c1-c2-rt
members target:65412:7;

[edit]
lab@r6# show policy-options policy-statement c1-c2-import
term 1 {
```

```
    from {
        protocol bgp;
        community c1-c2-rt;
    }
    then accept;
}

[edit]
lab@r6# show policy-options policy-statement c1-c2-export
term 1 {
    then {
        community add c1-c2-rt;
        accept;
    }
}
```

The VRF interface at r6 is configured to use the same VLAN ID as site C1. This point is significant because the translation of VLAN IDs requires a TCC type of encapsulation:

```
[edit]
lab@r6# show interfaces fe-0/1/3
vlan-tagging;
encapsulation vlan-ccc;
unit 0 {
    vlan-id 1;
    family inet {
        address 172.16.0.9/30;
    }
}
unit 600 {
    encapsulation vlan-ccc;
    vlan-id 600;
}
```

You decide to test the waters by committing the changes at r4; this approach allows you to test baseline L2 VPN connectivity so that you can determine if and where your preliminary VPN configuration may require additional tweaking. Any remaining configuration criteria can be dealt with after initial functionality is confirmed.

Initial L2 VPN Confirmation: Draft-Kompella

Confirming the operation of a Layer 2 VPN is made difficult by the inability to use PE-CE pings, or the operation of a PE-CE routing protocol, to validate the local PE-CE configuration and VRF interface. Because this example makes use of a non-VRF logical unit on the PE-CE link to facilitate OoB management between the PE and CE devices, you can conduct PE-CE ping testing and telnet to the CE device. Without the non-VRF interface, you are limited to PE-to-PE and

CE-to-CE types of testing. You begin by verifying the functionality of the OOB network between R6 and C2:

```
[edit]
lab@r6# run ping 172.16.0.10 count 2
PING 172.16.0.10 (172.16.0.10): 56 data bytes
64 bytes from 172.16.0.10: icmp_seq=0 ttl=255 time=0.604 ms
64 bytes from 172.16.0.10: icmp_seq=1 ttl=255 time=0.455 ms

--- 172.16.0.10 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.455/0.529/0.604/0.075 ms
```

The successful ping confirms the overall operational status of the VRF interface device (fe-0/1/3) and provides good indication that the OoB aspects of the PE router's configuration is compatible with the configuration present in the CE device. You can assume that ping testing conducted over the OoB network associated with C1 also succeeds (not shown). Before attempting any end-to-end testing between the CE devices, you display the state of Layer 2 VPN connection on r6:

```
[edit]
lab@r6# run show l2vpn connections
L2VPN Connections:

Legend for connection status (St)   Legend for interface status
OR -- out of range                  Up -- operational
EI -- encapsulation invalid         Dn -- down
EM -- encapsulation mismatch        NP -- no present
CM -- control-word mismatch         DS -- disabled
CN -- circuit not present           WE -- wrong encapsulation
OL -- no outgoing label             UN -- uninitialized
Dn -- down
VC-Dn -- Virtual circuit down
WE -- intf encaps != instance encaps
-> -- only outbound conn is up
<- -- only inbound  conn is up
Up -- operational
XX -- unknown


Instance: c1-c2-l2
Local site: 2 (2)
L2VPN Connections:

. . .

No L2VPN connections found.
```

Hmmm, the output, or more correctly the lack thereof, does not bode well for the end-to-end operation of the C1-C2 VPN. A VPN troubleshooting task is greatly simplified when you can correctly isolate the problem to the VPN's control or forwarding plane. Because this display indicates that no L2VPN connection exists, you can safely eliminate the forwarding plane as a likely cause for this problem; after all, there is no need to forward VPN traffic when there is no VPN! Knowing that a draft-Kompella VPN makes use of MP-BGP signaling to communicate Layer 2 VPN site information and VPN membership (via the RT), you decide to analyze the BGP session status between r4 and r6 to see if you can spot what is wrong:

```
[edit]
lab@r4# run show bgp summary
Groups: 3 Peers: 6 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
inet.0                 0          0          0          0          0          0
bgp.l3vpn.0            0          0          0          0          0          0
bgp.l2vpn.0            0          0          0          0          0          0
Peer         AS  InPkt  OutPkt  OutQ  Flaps Last Up/Dwn State|#Active/Received/
                                                                    Damped...
10.0.3.3  65412    449     451     0      0     3:44:31 Establ
  inet.0: 0/0/0
. . .
10.0.9.6  65412     35      36     0      0       16:37 Establ
  inet.0: 0/0/0
  bgp.l3vpn.0: 0/0/0
. . .
```

The edited capture confirms that the MP-IBGP session between r4 and r6 is established, which is a good sign. However, you happen to notice that the session does not indicate support for the l2vpn family! Without support for Layer 2 VPN NLRI, the PE routers can not exchange information relating to their local site, and this would definitely account for the symptom you are seeing. Recognizing the issue, you add the l2vpn family to the MP-IBGP session between r4 and r6. Note that the change shown here for r4 is also needed at r6:

```
[edit]
lab@r4# show protocols bgp group int neighbor 10.0.9.6
family inet {
    unicast;
}
family inet-vpn {
    unicast;
}
family l2vpn {
    unicast;
}
```

After committing the changes, you confirm the correct NLRI support for draft-Kompella VPNs:

```
[edit]
lab@r4# run show bgp neighbor 10.0.9.6 | match NLRI
  NLRI advertised by peer: inet-unicast inet-vpn-unicast l2vpn
  NLRI for this session: inet-unicast inet-vpn-unicast l2vpn
```

With `l2vpn` NLRI now being exchanged between the PE routers, you expect to see that a Layer 2 VPN connection has been formed:

```
[edit]
lab@r6# run show l2vpn connections
L2VPN Connections:

Legend for connection status (St)   Legend for interface status
OR -- out of range                  Up -- operational
EI -- encapsulation invalid         Dn -- down
EM -- encapsulation mismatch        NP -- no present
CM -- control-word mismatch         DS -- disabled
CN -- circuit not present           WE -- wrong encapsulation
OL -- no outgoing label             UN -- uninitialized
Dn -- down
VC-Dn -- Virtual circuit down
WE -- intf encaps != instance encaps
-> -- only outbound conn is up
<- -- only inbound  conn is up
Up -- operational
XX -- unknown


Instance: c1-c2-l2
Local site: 2 (2)
    connection-site           Type  St     Time last up          # Up trans
    1                         rmt   Up     Jun  6 19:57:12 2003           1
      Local interface: fe-0/1/3.600, Status: Up, Encapsulation: VLAN
      Remote PE: 10.0.3.4, Negotiated control-word: Yes (Null)
      Incoming label: 800000, Outgoing label: 800001
```

Great! The display confirms that a Layer 2 VPN connection is now in place at `r6`, and it shows the connection status as `Up`. The output also shows the VPN labels associated with this L2 VPN connection that terminates at site 1. The draft-Kompella model uses draft-Martini encapsulation in the forwarding plane. Here you can see that a null Martini control word is added to the VPN traffic by default; this can be optioned off to interoperate with JUNOS software versions prior to 5.6, which do not support the Martini control word. You can confirm

the L2 VPN NLRI being exchanged between PE routers with a `show route advertising-protocol` command:

```
[edit]
lab@r6# run show route advertising-protocol bgp 10.0.3.4 detail

inet.0: 37 destinations, 41 routes (37 active, 0 holddown, 0 hidden)
* 192.168.0.0/24 (2 entries, 1 announced)
 BGP group int type Internal
     Nexthop: 10.0.8.1
     MED: 2
     Localpref: 100
     AS path: I
 Communities:

. . .
c1-c2-l2.l2vpn.0: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)

* 10.0.9.6:1:2:1/96 (1 entry, 1 announced)
 BGP group int type Internal
     Route Distinguisher: 10.0.9.6:1
     Label-base : 800000, range : 1, status-vector : 0x0
     Nexthop: Self
     Localpref: 100
     AS path: I
 Communities: target:65412:7 Layer2-info: encaps:VLAN, control flags:2, mtu: 0
```

The highlighted portion of the edited capture confirms that `r6` is advertising the L2 VPN site information for C2 to the PE router that connects to site C1. Note that the NLRI is 96 bits in length and comprises the local PE's 8-byte RD concatenated with a 2-byte site-ID and a 1-byte value that codes the label offset. The final confirmation of the L2 VPN's operational status involves end-to-end testing from the CE devices. You begin with traceroute testing conducted at the C2 location; note that the routing-instance switch is not required when establishing telnet sessions to the CE device because the OoB interface is not associated with any VRFs:

```
[edit]
lab@r6# run telnet 172.16.0.10
Trying 172.16.0.10...
Connected to 172.16.0.10.
Escape character is '^]'.

c2 (ttyp1)
```

```
login: lab
Password:
Last login: Fri Jun  6 20:47:50 from 10.0.1.100

--- JUNOS 5.2R2.3 built 2002-03-23 02:44:36 UTC

lab@c2> traceroute 192.168.16.1
traceroute to 192.168.16.1 (192.168.16.1), 30 hops max, 40 byte packets
 1  192.168.16.1 (192.168.16.1)  0.356 ms  0.271 ms  0.262 ms
```

The traceroute to the remote CE's VRF interface address is successful, and the display shows the single hop that is expected between CE devices in a Layer 2 VPN. Next, the OSPF adjacency status between the C1 and C2 devices is confirmed:

```
lab@c2> show ospf neighbor
  Address          Interface              State     ID               Pri  Dead
192.168.16.1     fe-0/0/0.600               Full      200.200.0.1      128   33

lab@c2> show route protocol ospf

inet.0: 12 destinations, 12 routes (12 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

200.200.0.0/16     *[OSPF/150] 00:17:10, metric 0, tag 0
                    > to 192.168.16.1 via fe-0/0/0.600
200.200.0.1/32     *[OSPF/10] 00:17:10, metric 1
                    > to 192.168.16.1 via fe-0/0/0.600
200.200.1.0/24     *[OSPF/150] 00:17:10, metric 0, tag 0
                    > to 192.168.16.1 via fe-0/0/0.600
224.0.0.5/32       *[OSPF/10] 04:11:58, metric 1
```

The results confirm proper OSPF adjacency formation across the provider's network and the presence of OSPF routes associated with the remote VPN site. Final verification comes with traceroute testing performed between CE loopback addresses:

```
lab@c2> traceroute 200.200.0.1 source 220.220.0.1
traceroute to 200.200.0.1 (200.200.0.1) from 220.220.0.1, 30 hops max, 40 byte
   packets
 1  200.200.0.1 (200.200.0.1)  0.374 ms  0.269 ms  0.259 ms
```

The results obtained in this section confirm that you have successfully established baseline Layer 2 VPN connectivity between C1 and C2. Subsequent sections address the scenario's remaining requirements.

MAPPING L2 VPN TRAFFIC TO AN LSP

To meet the traffic mapping aspects of this example, you need to define a second set of RSVP signaled LSPs between `r4` and `r6`, and make the necessary policy changes to ensure that the

L2 VPN traffic is mapped to the correct LSP. Policy-based LSP mapping is required here because you may not change the default LSP metrics to effect the use of one LSP over another. Note that the techniques shown in this section are equally applicable to Layer 3 VPNs. You begin by defining the new LSP at r4 and r6 (not shown). The modified MPLS stanza at r4 is displayed next:

```
[edit protocols mpls]
lab@r4# show
label-switched-path r4-r6 {
    to 10.0.9.6;
    no-cspf;
}
label-switched-path r4-r7 {
    to 10.0.9.7;
    no-cspf;
}
label-switched-path r4-r6-prime {
    to 10.0.9.6;
    bandwidth 10m;
    no-cspf;
}
interface all;
interface fxp0.0 {
    disable;
}
```

After the changes are committed, the successful establishment of the new LSPs is confirmed at r6:

```
lab@r6# run show rsvp session
Ingress RSVP: 3 sessions
To              From            State Rt Style Labelin Labelout LSPname
10.0.3.4        10.0.9.6        Up     0  1 FF       -   100000 r6-r4
10.0.3.4        10.0.9.6        Up     0  1 FF       -   100004 r6-r4-prime
10.0.9.7        10.0.9.6        Up     0  1 FF       -   100000 r6-r7
Total 3 displayed, Up 3, Down 0

Egress RSVP: 3 sessions
To              From            State Rt Style Labelin Labelout LSPname
10.0.9.6        10.0.3.4        Up     0  1 FF       3        - r4-r6
10.0.9.6        10.0.3.4        Up     0  1 FF       3        - r4-r6-prime
10.0.9.6        10.0.9.7        Up     0  1 FF       3        - r7-r6
Total 3 displayed, Up 3, Down 0

Transit RSVP: 0 sessions
Total 0 displayed, Up 0, Down 0
```

With the LSPs correctly established, you display the current L2 VPN to LSP mapping so that you may better judge the effects of your subsequent policy-based mapping configuration:

```
[edit protocols mpls]
lab@r6# run show route table mpls.0

mpls.0: 5 destinations, 5 routes (5 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0                  *[MPLS/0] 04:24:45, metric 1
                      Receive
1                  *[MPLS/0] 04:24:45, metric 1
                      Receive
2                  *[MPLS/0] 04:24:45, metric 1
                      Receive
800000             *[L2VPN/7] 00:31:46
                    > via fe-0/1/3.600, Pop       Offset: 4
fe-0/1/3.600       *[L2VPN/7] 00:31:46
                    > to 10.0.2.14 via fe-0/1/1.0, label-switched-path r6-r4
                      to 10.0.2.14 via fe-0/1/1.0, label-switched-path r6-r4-
                         prime
```

The display confirms that the L2 VPN's traffic is currently being forwarded over the original LSP with no bandwidth reservation. Because the LSPs have identical metrics, the default VPN to LSP mapping will be random, which is why you must define a policy to ensure that VPN traffic is deterministically mapped to the desired LSP. In this example, the *mapping* policy is based on the presence of a particular RT community. The completed policy is shown at r6; a similar policy is also created at r4 (not shown):

```
[edit policy-options]
lab@r6# show policy-statement mapping
term 1 {
    from community c1-c2-rt;
    then {
        install-nexthop lsp r6-r4-prime;
        accept;
    }
}
term 2 {
    then accept;
}
```

The first term in the mapping policy matches on the specified community with an action of installing the *r6-r4-prime* LSP as the next hop; the accept action in term *1* is critical for proper operation, because without a terminating action in this term the L2 NLRI falls through to the second term, which matches everything. Because the second term does not have a

mapping action, traffic hitting term *2* will be subjected to the default load-balancing behavior. You must now apply the *mapping* policy to the main routing instance's forwarding table; note that many JNCIE candidates incorrectly apply their policy to the VRF's routing instance, where it has absolutely no effect:

```
[edit routing-options]
lab@r6# set forwarding-table export mapping
```

After the changes are committed, the desired VPN to LSP mapping is confirmed at r6:

```
[edit routing-options]
lab@r6# run show route table mpls.0

mpls.0: 5 destinations, 5 routes (5 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0                  *[MPLS/0] 04:34:26, metric 1
                      Receive
1                  *[MPLS/0] 04:34:26, metric 1
                      Receive
2                  *[MPLS/0] 04:34:26, metric 1
                      Receive
800000             *[L2VPN/7] 00:41:27
                    > via fe-0/1/3.600, Pop       Offset: 4
fe-0/1/3.600       *[L2VPN/7] 00:41:27
                      to 10.0.2.14 via fe-0/1/1.0, label-switched-path r6-r4
                    > to 10.0.2.14 via fe-0/1/1.0, label-switched-path r6-r4-
                      prime
```

The display confirms that the L2 VPN traffic is now correctly mapped to the *r6-r4-prime* LSP. Similar results are also observed at r4:

```
[edit]
lab@r4# run show route table mpls.0

mpls.0: 5 destinations, 5 routes (5 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0                  *[MPLS/0] 04:36:25, metric 1
                      Receive
1                  *[MPLS/0] 04:36:25, metric 1
                      Receive
2                  *[MPLS/0] 04:36:25, metric 1
                      Receive
800001             *[L2VPN/7] 00:43:22
                    > via fe-0/0/0.600, Pop       Offset: 4
```

```
fe-0/0/0.600        *[L2VPN/7] 00:43:22
                       via so-0/1/0.100, label-switched-path r4-r6
                     > via so-0/1/0.100, label-switched-path r4-r6-prime
```

PROVIDING INTERNET ACCESS FROM A NON-VRF INTERFACE

Because the CE devices already have a non-VRF interface provisioned, providing C1 with the required access to Internet routes is somewhat trivial. To obtain the desired behavior, you must add a static default route to C1 that directs matching packets out the non-VRF interface to r4. You also need to define a static route for C1's 200.200/16 net block in the main routing instance at r4, and ensure that this route is redistributed into IBGP so that internal and external BGP peers can route packets back to C1. The lack of Internet connectivity at C1 is confirmed before the static route is added:

```
[edit]
lab@c1# run show route 130.130.0.1

[edit]
lab@c1#
```

The static default route is now added to C1 that points to r4's non-VRF interface as the next hop:

```
[edit routing-options]
lab@c1# set static route 0.0.0.0/0 next-hop 172.16.0.5
```

A similar static route is added to r4, and the IBGP export policy at r4 is modified to effect advertisement of the 200.200/16 route:

```
[edit]
lab@r4# show routing-options static
route 10.0.200.0/24 {
    next-hop 10.0.1.102;
    no-readvertise;
}
route 200.200.0.0/16 next-hop 172.16.0.6;

[edit]
lab@r4# show policy-options policy-statement nhs
term 1 {
    from {
        protocol bgp;
        neighbor 172.16.0.6;
    }
    then {
        next-hop self;
    }
```

```
}
term 2 {
    from {
        protocol static;
        route-filter 200.200.0.0/16 exact;
    }
    then {
        next-hop self;
        accept;
    }
}
```

Note that you must take care to correctly set the next hop when advertising the 200.200/16 route from r4; by default the BGP next hop is set to match the static route's 172.16.0.6 next hop, which results in the route being hidden because the 172.16.0.4/30 prefix is not carried within your IGP. After the changes are committed, Internet access is confirmed from C1:

```
lab@c1> show route 130.130.0.0

inet.0: 11 destinations, 11 routes (11 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[Static/5] 00:38:49
                    > to 172.16.0.5 via fe-0/0/0.0

lab@c1> traceroute 130.130.0.1 source 200.200.0.1
traceroute to 130.130.0.1 (130.130.0.1) from 200.200.0.1, 30 hops max, 40 byte
   packets
 1  172.16.0.5 (172.16.0.5)  0.406 ms  0.295 ms  0.276 ms
 2  10.0.2.5 (10.0.2.5)  0.342 ms  0.294 ms  0.297 ms
 3  130.130.0.1 (130.130.0.1)  0.261 ms  0.228 ms  0.224 ms
```

The traceroute to T1 destinations succeeds, and therefore confirms that C1 has the required Internet access for traffic sourced from the 200.200/16 net block. Sourcing the traffic from C1's 172.16.0.6 address will result in failures because this prefix is not carried in your IGP or advertised to EBGP peers. The scenario's criteria do not specify if site C2 should also have Internet access. C2 should not have Internet access at this time even though the static default route at C1 is being redistributed into OSPF:

```
[edit]
lab@c2# run show route protocol ospf

inet.0: 12 destinations, 12 routes (12 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

```
0.0.0.0/0          *[OSPF/150] 00:02:28, metric 0, tag 0
                    > to 192.168.16.1 via fe-0/0/0.600
200.200.0.0/16     *[OSPF/150] 00:02:28, metric 0, tag 0
                    > to 192.168.16.1 via fe-0/0/0.600
200.200.0.1/32     *[OSPF/10] 00:02:28, metric 1
                    > to 192.168.16.1 via fe-0/0/0.600
200.200.1.0/24     *[OSPF/150] 00:02:28, metric 0, tag 0
                    > to 192.168.16.1 via fe-0/0/0.600
224.0.0.5/32       *[OSPF/10] 00:30:05, metric 1
```

Internet access is a problem for C2 because none of the routers in the test bed have a route back to the 220.220/16 net block associated with the traffic that the C2 site generates. You could add a 220.220/16 static route to `r4`, and adjust its *`nhs`* export policy to advertise the route to its IBGP peers to provide C2 with Internet access quite easily, however. Note that in this case traffic from C2 to the Internet will have to bounce off of C1 in a hub-and-spoke fashion.

These results confirm that your draft-Kompella Layer 2 VPN configuration meets all provided restrictions and specified behaviors. Good job!

Draft-Martini

Your draft-Martini VPN scenario requires that you replicate the existing Layer 2 connectivity between C1 and C2 using a draft-Martini solution. The topology details are identical to those specified for the draft-Kompella scenario. Refer back to Figure 7.6 for details as needed.

To complete the draft-Martini VPN scenario, you must reconfigure the subset of routers shown in Figure 7.6 according to these criteria:

- Delete the routing instance configuration in place at `r4` and `r6`. If desired, you can also delete any VRF policy and related community definitions left over from the previous Layer 2 VPN scenario.
- Delete the RSVP stanza and LSP definitions at `r4` and `r6`.
- Establish an L2 VPN providing connectivity between C1 and C2 without adding a `routing-instance` stanza to `r4` or `r6`.
- Your VPN configuration can not disrupt existing IPv4 routing and forwarding functionality.
- Your configuration must tolerate the failure of either SONET interface at `r4`.

Draft-Martini Configuration

Although not explicitly stated in the objectives, the restriction on adding a `routing-instance` stanza to the PE routers imposes a draft-Martini VPN solution. You begin your draft-Martini Layer 2 VPN configuration at `r4` with the removal of the existing VPN and VRF-related policy configuration. Although not shown, similar commands are also entered on `r6`:

```
[edit]
lab@r4# delete routing-instances

[edit]
lab@r4# delete policy-options policy-statement c1-c2-import
```

```
[edit]
lab@r4# delete policy-options policy-statement c1-c2-export

[edit]
lab@r4# delete policy-options community c1-c2-rt

[edit]
lab@r4# delete policy-options policy-statement mapping

[edit]
lab@r4# delete routing-options forwarding-table export
```

If desired, you can also remove the 200.200/16 static route definition and the related *nhs* policy changes from r4 (not shown). RSVP signaling support is now removed from r4 and r6 (not shown):

```
[edit]
lab@r4# delete protocols rsvp
```

Note that you are now beginning your draft-Martini scenario with a PE-CE VPN interface configuration that is left in place from the previous draft-Kompella scenario. Also note that the VPN test bed has MPLS processing and mpls family support on your core-facing interfaces from the preliminary configuration scenario. This means that your configuration tasks will be limited to LDP (with extended neighbor discovery support) and the definition of the Layer 2 circuit that interconnects sites C1 and C2. You begin with the configuration of the ldp stanza on r4:

```
[edit protocols ldp]
lab@r4# set interface lo0

[edit protocols ldp]
lab@r4# set interface so-0/1/0.100

[edit protocols ldp]
lab@r4# set interface so-0/1/1
```

You must run LDP on the router's lo0 interface for LDP extended neighbor discovery to function correctly. Extended neighbor discovery is required to support draft-Martini signaling. LDP is also enabled on both of r4's core-facing interfaces to support the stated redundancy requirements; an interface all statement could have been used in this example. The completed ldp stanza is displayed at r4:

```
[edit protocols ldp]
lab@r4# show
interface so-0/1/0.100;
interface so-0/1/1.0;
interface lo0.0;
```

A similar LDP configuration must be added to r6, and to a subset of the routers in the test bed to ensure that your design can tolerate the failure of either PoS interface at r4. In this example,

you decide that adding LDP support to r3 and r5 meets the level of redundancy required. The LDP stanza for r5 is shown here:

```
[edit]
lab@r5# show protocols ldp
interface fe-0/0/0.0;
interface so-0/1/0.0;
interface at-0/2/1.0;
```

You do not need to enable LDP on r5's lo0 interface because r5's role as a P router in this topology means that it has no need for draft-Martini signaling, and therefore no need for extended neighbor discovery. Enabling LDP on r5's at-0/2/1 interface is necessary to ensure the required redundancy in the direction of r6 to r4; LDP support on the ATM link between r3 and r5 permits the LSP to be routed arround the failure if the PoS link between r3 and r4 should fail. A similar LDP configuration is added to r3:

```
[edit]
lab@r3# show protocols ldp
interface fe-0/0/3.0;
interface at-0/1/0.0;
interface so-0/2/0.100;
```

With the VPN's control plane provisioned, the configuration of the l2circuit that will actually interconnect the two sites rises to the top of your configuration heap. You begin l2circuit definition on r6:

```
[edit protocols l2circuit]
lab@r6# set neighbor 10.0.3.4 interface fe-0/1/3.600 virtual-circuit-id 12
```

The completed L2 circuit definition is displayed:

```
[edit protocols l2circuit]
lab@r6# show
neighbor 10.0.3.4 {
    interface fe-0/1/3.600 {
        virtual-circuit-id 12;
    }
}
```

A similar l2circuit configuration is added to r4. For proper operation, you must ensure that both ends of the Layer 2 circuit use the same circuit-id value; in this case, the value *12* is intended to code "site 1 and site 2," but any unique value can be specified. The l2circuit definition at r4 is shown next:

```
[edit protocols l2circuit]
lab@r4# show
neighbor 10.0.9.6 {
    interface fe-0/0/0.600 {
        virtual-circuit-id 12;
    }
}
```

L2 VPN Confirmation: Draft-Martini

You begin confirmation of the draft-Martini VPN with verification that the LDP-based control (and forwarding) plane is operational. Extended neighbor discovery, and the presence of LSPs between PE router loopback addresses, is confirmed at r4:

```
[edit protocols l2circuit]
lab@r4# run show ldp neighbor
Address            Interface          Label space ID         Hold time
10.0.9.6           lo0.0              10.0.9.6:0               13
10.0.2.5           so-0/1/0.100       10.0.3.3:0               10
10.0.2.9           so-0/1/1.0         10.0.3.5:0               12
```

LDP extended neighbor discovery is verified by the presence of r6's loopback address in the list of LDP neighbors at r4. The presence of LDP signaled LSPs is verified next:

```
[edit protocols l2circuit]
lab@r4# run show route table inet.3

inet.3: 3 destinations, 3 routes (3 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.3.3/32        *[LDP/9] 00:10:06, metric 1
                    > via so-0/1/0.100
10.0.3.5/32        *[LDP/9] 00:17:12, metric 1
                    > via so-0/1/1.0
10.0.9.6/32        *[LDP/9] 00:10:06, metric 1
                    > via so-0/1/0.100, Push 100022
                      via so-0/1/1.0, Push 100229
```

The highlights call out the presence of LDP signaled LSPs that are associated with the loopback address of the egress PE router (r6). The presence of two equal-cost next hops for this LSP indicates that you have met the stated redundancy, at least in the direction of r4 to r6. LSP establishment in the r6 to r4 direction is now verified at r6:

```
[edit protocols l2circuit]
lab@r6# run show route 10.0.3.4

inet.0: 121296 destinations, 121301 routes (121296 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.3.4/32        *[IS-IS/18] 00:15:35, metric 20
                    > to 10.0.2.14 via fe-0/1/1.0

inet.3: 3 destinations, 3 routes (3 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.3.4/32        *[LDP/9] 00:16:36, metric 1
                    > to 10.0.2.14 via fe-0/1/1.0, Push 100025
```

The output shows that an LSP to r4 has been successfully established, and also indicates that r6's IGP route to the egress PE's loopback address is via the L2 interface that links it to r3. This is expected, considering that the 10.0.3.4 Level 2 prefix is not being leaked into r6's Level 1 area in the IS-IS baseline topology. The final redundancy check is performed at r3 with confirmation that the LSP can reroute around failures of its so-0/2/0 interface:

```
[edit]
lab@r3# run show route 10.0.3.4

inet.0: 121352 destinations, 121361 routes (121352 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.3.4/32        *[IS-IS/18] 00:19:44, metric 10
                    > to 10.0.2.6 via so-0/2/0.100

inet.3: 3 destinations, 3 routes (3 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.3.4/32        *[LDP/9] 00:19:15, metric 1
                    > via so-0/2/0.100
```

The outputs shows that r3's current IGP route to 10.0.3.4, and therefore the path of the LDP signaled LSP that egresses at this address, is currently routed over the 10.0.2.4/30 subnet. By deactivating r3's so-0/2/0 interface, LSP failover can be verified:

```
[edit]
lab@r3# deactivate interfaces so-0/2/0

[edit]
lab@r3# commit
commit complete

[edit]
lab@r3# run show route 10.0.3.4

inet.0: 121352 destinations, 121361 routes (121352 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.3.4/32        *[IS-IS/18] 00:02:20, metric 20
                    > to 10.0.2.1 via at-0/1/0.0

inet.3: 3 destinations, 3 routes (3 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.3.4/32        *[LDP/9] 00:02:18, metric 1
                    > via at-0/1/0.0, Push 100226
```

The display confirms that you have met the stated redundancy requirements by virtue of the LSP failing over to the ATM link connecting r3 and r5. Do not forget to activate r3's so-0/2/0 interface once you are satisfied with your network's redundancy behavior!

With the LDP control plane confirmed, you move into verification of the VPN control plane with a display of l2circuit status at r6:

```
[edit protocols l2circuit]
lab@r6# run show l2circuit connections
Layer-2 Circuit Connections:

Legend for connection status (St)   Legend for interface status
EI -- encapsulation invalid         Up -- operational
MM -- mtu mismatch                  Dn -- down
EM -- encapsulation mismatch        NP -- no present
CM -- control-word mismatch         DS -- disabled
OL -- no outgoing label             WE -- wrong encapsulation
Dn -- down                          UN -- uninitialized
VC-Dn -- Virtual circuit Down
Up -- operational
XX -- unknown

Neighbor: 10.0.3.4
    Interface                 Type  St     Time last up          # Up trans
    fe-0/1/3.600 (vc 12)      rmt   Up     Jun  7 19:27:39 2003           1
      Local interface: fe-0/1/3.600, Status: Up, Encapsulation: VLAN
      Remote PE: 10.0.3.4, Negotiated control-word: Yes (Null)
      Incoming label: 100014, Outgoing label: 100005
```

The output, which is very similar to that shown for draft-Kompella based l2vpn connections, indicates that the Layer 2 circuit has been correctly signaled. The even better news is that r4 also indicates successful establishment of the l2circuit at this time:

```
[edit protocols l2circuit]
lab@r4# run show l2circuit connections
Layer-2 Circuit Connections:

Legend for connection status (St)   Legend for interface status
EI -- encapsulation invalid         Up -- operational
MM -- mtu mismatch                  Dn -- down
EM -- encapsulation mismatch        NP -- no present
CM -- control-word mismatch         DS -- disabled
OL -- no outgoing label             WE -- wrong encapsulation
Dn -- down                          UN -- uninitialized
VC-Dn -- Virtual circuit Down
Up -- operational
XX -- unknown
```

```
Neighbor: 10.0.9.6
    Interface                 Type  St     Time last up          # Up trans
    fe-0/0/0.600 (vc 12)      rmt   Up     Jun  7 18:37:38 2003           1
      Local interface: fe-0/0/0.600, Status: Up, Encapsulation: VLAN
      Remote PE: 10.0.9.6, Negotiated control-word: Yes (Null)
```

All results observed thus far indicate that your draft-Martini Layer 2 VPN is operational. Confirmation of the VPN forwarding plane comes with end-to-end testing between CE devices and the determination of proper OSPF adjacency formation:

```
[edit]
lab@r4# run telnet 172.16.0.6
Trying 172.16.0.6...
Connected to 172.16.0.6.
Escape character is '^]'.

c1 (ttyp0)

login: lab
Password:
Last login: Sat Jun  7 10:39:53 on ttyd0

--- JUNOS 5.6R2.4 built 2003-02-14 23:22:39 UTC

lab@c1> show ospf neighbor
  Address          Interface              State      ID               Pri  Dead
192.168.16.2     fe-0/0/0.600           Full       220.220.0.1      128   39
```

The presence of an OSPF adjacency provides a strong indication that the `l2circuit` is operating properly. The presence of OSPF learned routes associated with the remote C2 device is another indication that all is well:

```
lab@c1> show route protocol ospf

inet.0: 11 destinations, 11 routes (11 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

220.220.0.0/16     *[OSPF/150] 00:01:58, metric 0, tag 0
                    > to 192.168.16.2 via fe-0/0/1.600
220.220.0.1/32     *[OSPF/10] 00:01:58, metric 1
                    > to 192.168.16.2 via fe-0/0/1.600
224.0.0.5/32       *[OSPF/10] 00:16:36, metric 1
                      MultiRecv
```

Note that the C1 device still has the default route that was added to support Internet access in the previous draft-Kompella scenario; its presence causes no harm here. The ability to conduct traceroute testing to the VPN interface and the loopback address of the remote CE device

provides final confirmation that your draft-Martini Layer 2 VPN is fully operational in the forwarding plane:

```
lab@c1> traceroute 220.220.0.1
traceroute to 220.220.0.1 (220.220.0.1), 30 hops max, 40 byte packets
 1  220.220.0.1 (220.220.0.1)  0.379 ms  0.281 ms  0.269 ms

lab@c1> traceroute 192.168.16.1
traceroute to 192.168.16.1 (192.168.16.1), 30 hops max, 40 byte packets
 1  192.168.16.1 (192.168.16.1)  0.439 ms  0.248 ms  0.226 ms
```

Proper VPN forwarding combined with previous confirmation of MPLS and VPN signaling means that you have met all stated requirements for the draft-Martini Layer 2 VPN configuration challenge. Good work!

Layer 2 VPN Summary

Layer 2 VPNs based on the draft-Kompella model are configured and tested in much the same way as 2547 bis Layer 3 VPNs. MP-BGP is used to signal VPN membership and you must configure VRFs along with all the trappings in the form of route distinguishers, route targets, and so on. Draft-Martini Layer 2 VPNs, on the other hand, rely on LDP-based signaling; do not make use of RTs, RDs, or VRF policy; and require configuration at the `edit protocols l2circuit` hierarchy as opposed to the `edit routing-instances` hierarchy.

Regardless of what type of Layer 2 VPN is deployed, the nature of the technology results in the appearance of a direct link between the attached CE devices, much as you would expect with a transparent bridge, or the virtual circuit connections associated with ATM or Frame Relay technologies. This behavior has several advantages, such as eliminating customer routes from the service provider's PE routers, and the ability to support non-IP protocols. The drawbacks to Layer 2 VPNs tend to relate to fault isolation because the lack of IP and routing protocol interaction between the CE and PE devices makes it difficult to determine whether there are hardware or configuration problems on the local VRF interface.

This section demonstrated the configuration of Layer 2 VPNs in JUNOS software using both draft-Kompella and draft-Martini solutions. In the case of draft-Kompella, the use of forwarding table export policy to effect the mapping of L2 VPN traffic to a particular LSP, and non-VRF interface based Internet access, was also demonstrated. It bears stressing that the configuration techniques demonstrated to support VPN-to-LSP mapping and Internet access can be used for Layer 3 VPNs.

The configuration and verification of translational cross connect (TCC), which is also known as "Layer 2.5 IP-Only Interworking," was not demonstrated in the chapter body. TCC is supported in CCC, draft-Kompella, and draft-Martini based VPNs to allow for interworking between dissimilar access technologies (or differing VLAN IDs, which are normally required to be the same at both ends of a L2 VPN). Having to interconnect a Frame Relay–based CE device to another site that uses ATM is a classic application for TCC. Because the Fast Ethernet interfaces and the JUNOS software release 5.6 code that is deployed in this VPN test bed do not support a mix of TCC and non-TCC families on a VLAN-tagged interface, the use of TCC eliminates your ability to access the CE devices from the PE router using a non-VRF interface based OoB

network. The need for candidate access to the CE devices combined with the specifics of this author's test bed is the primary reason that a TCC scenario was not included in this chapter:

```
[edit interfaces fe-0/0/0]
lab@r4# show
vlan-tagging;
encapsulation extended-vlan-tcc;
unit 0 {
    vlan-id 1;
    family inet {
        address 10.0.5.1/32;
    }
}
unit 600 {
    encapsulation vlan-ccc;
    vlan-id 600;
    family tcc;
}

[edit interfaces fe-0/0/3]
lab@r4# commit check
[edit interfaces fe-0/0/3 unit 0]
  'family'
     Only the TCC family is allowed on TCC interfaces
error: configuration check-out failed
```

To be effective with Layer 2 VPNs the candidate must be able to quickly isolate and diagnose problems in the VPN forwarding plane (MPLS signaling, and MPLS forwarding, double label push operations, and so on) and in the VPN control plane (MP-BGP, route targets, extended communities, VRF policy, or LDP with targeted hellos). Throughout this section, the reader was exposed to operational mode commands that are useful in determining the operational status of Layer 2 VPNs based on either the Kompella or Martini drafts.

Summary

JNCIE candidates should be prepared to configure a variety of provider-provisioned VPN solutions in their lab exam. Successful candidates will be fluent with BGP and LDP-based VPN solutions, and will possess a keen grasp of the differences between a Layer 3 and a Layer 2 VPN model.

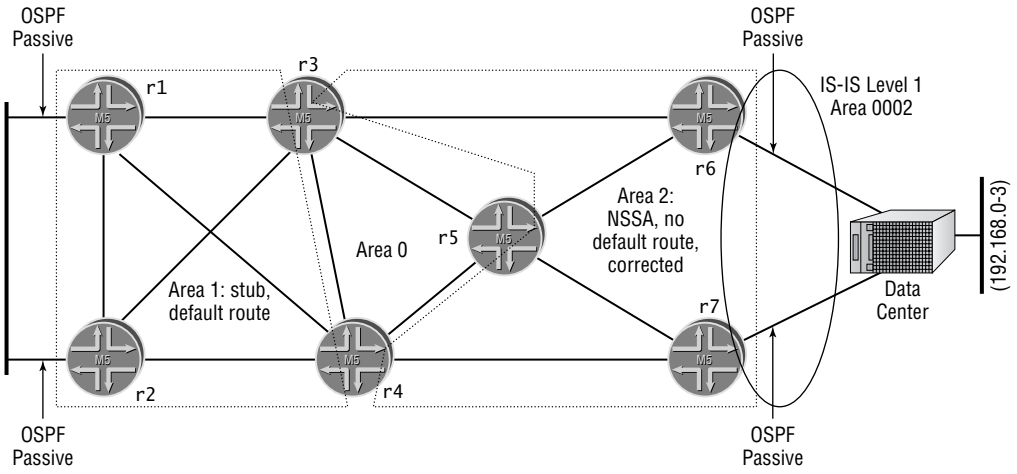
This chapter provided configuration scenarios and verification techniques for Layer 3 VPNs based on the 2547 bis model. Although RSVP signaled LSPs were used to support the VPN's forwarding plane, the LSPs could have been signaled with LDP. Recent enhancements, such as the `vrf-target` and `route-distinguisher-id` statement, which make the provisioning of BGP signaled VPNs simpler, were demonstrated along with the far more manual alternatives.

The chapter went on to demonstrate the configuration and testing of Layer 2 VPNs based on draft-Martini and draft-Kompella. Configuration and testing of draft-Kompella solutions follows many of the same procedures used for Layer 3 VPNs based on 2547 bis; the similarities between 2547 bis and draft-Kompella provide operational benefits when a given provider plans to support both Layer 2 and Layer 3 VPN offerings. In contrast, the draft-Martini solution is usually considered to be far easier to configure because draft-Martini VPNs do not make use of BGP-based signaling, and therefore have no concept of RDs, RTs, and VRF policy.

Case Study: VPNs

The chapter case study approximates a JNCIE-level provider-provisioned VPN scenario. You will be performing your VPN case study using the OSPF baseline configuration that was discovered and documented in the body of Chapter 1. The OSPF baseline topology is shown in Figure 7.7 for reference purposes.

FIGURE 7.7 OSPF discovery findings



Notes:

Loopback addresses have not been assigned to specific areas (lo0 address advertised in Router LSA in all areas).

Passive OSPF interfaces on P1 and data center segments.

No authentication or route summarization in effect; summaries (LSA type 3) allowed in all areas.

Redistribution of OSPF default route to data center from both `r6` and `r7` was broken. Fixed with default-metric command on `r3`, `r4`, and `r5`.

Data center router running IS-IS, Level 1. `r6` and `r7` compatibly configured and adjacent.

Redistribution of 192.168.0/24 through 192.168.3/24 into OSPF from IS-IS by both `r6` and `r7`.

Adjustment to IS-IS level 1 external preference to ensure `r6` and `r7` always prefer IS-IS Level 1 externals over OSPF externals.

All adjacencies up and full reachability confirmed.

Sub-optimal routing detected at the data center router for some locations, and when `r3` and `r4` forward to some Area 2 addresses. This is the result of random nexthop choice for its default route and Area 2 topology specifics. Considered to be working as designed; no action taken.

You should load and commit the baseline OSPF configuration and confirm that your baseline network's OSPF IGP and IBGP peerings are operational before beginning the case study. Problems are not expected in the baseline network at this stage, but it never hurts to verify your starting point in a journey such as this. Note that due to configuration changes in the peripheral routers, you should expect to find that no EBGP sessions are established with the OSPF baseline configuration in place.

Refer to the case study criteria listing and the case study topology that is shown in Figure 7.8 for the information needed to complete the VPN case study. It is expected that a JNCIE candidate will be able to complete this case study in approximately two hours with no major operational problems in the finished work.

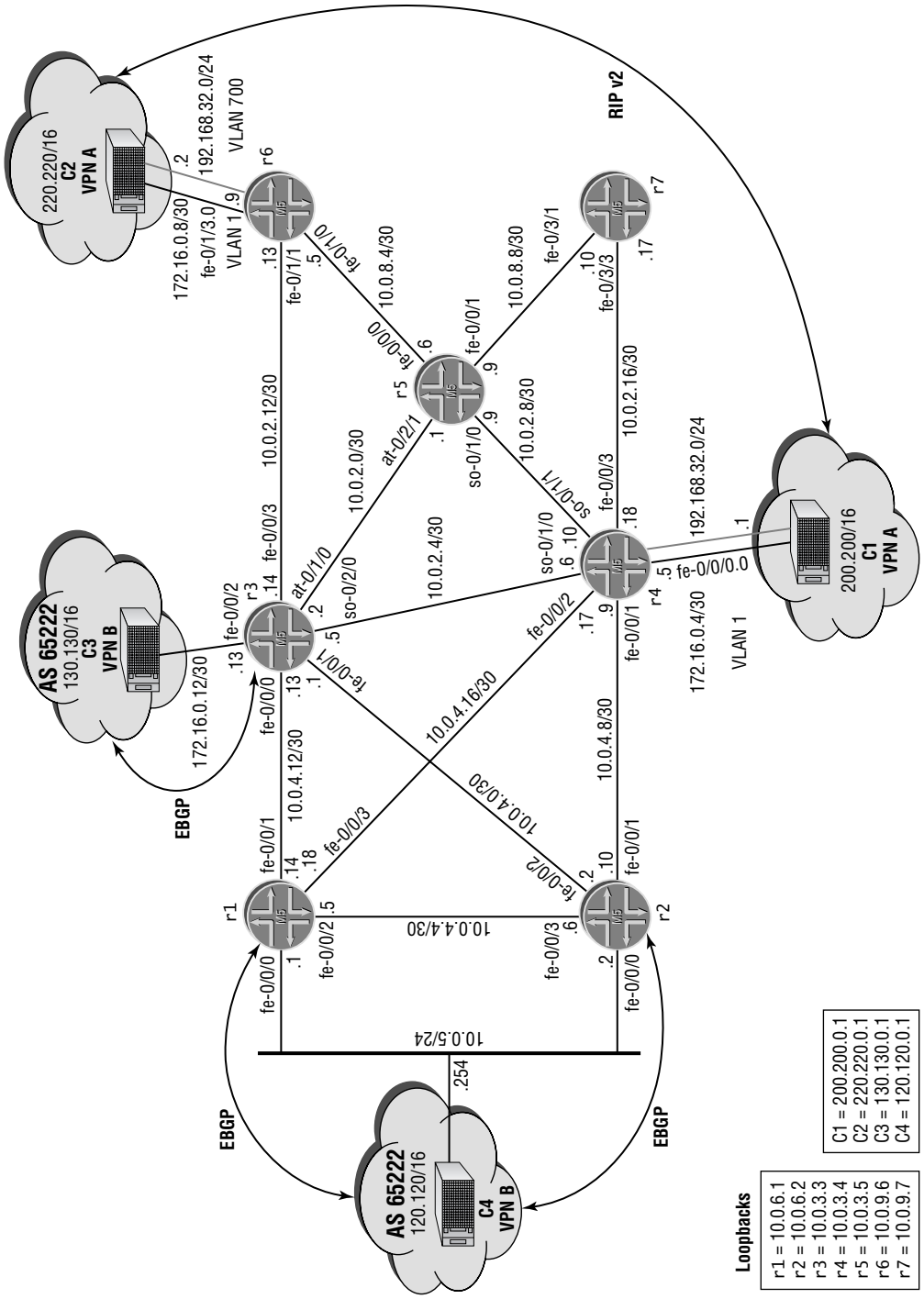
Sample configurations from all routers are provided at the end of the case study for comparison with your own configurations. Because multiple solutions may be possible for a given aspect of the case study, differences in your own solution are not automatically indicative of a mistake. Because you are graded on the overall functionality of your network along with its conformance to the specified criteria, the output from key operational mode commands is also included to allow an operational comparison of your network and that of a known good example.

To complete this case study, your network must be configured to meet the following criteria:

- For VPN A:
  - You may not alter the exiting BGP stanzas on `r4` and `r6`.
  - Ensure that C1 and C2 exchange their respective routes using RIP V2.
  - You may not configure RIP V2 on `r3`, `r5`, or `r7`.
  - You can access the C1 and C2 devices for testing purposes only; you must not modify their configuration.
  - You must have connectivity between C1 and C2.
  - Your VPN must not disrupt or alter the flow of IPv4 packets within your network.
- For VPN B:
  - Ensure that C3 and C4 exchange their respective routes using EBGP.
  - The failure of either `r1` or `r2` can not disable VPN B.
  - You must count all ICMP traffic that egresses `r3`'s fe-0/0/2 interface.
  - You must have connectivity between C3 and C4.
  - You must support traffic that originates or terminates on the multi-access VRF interfaces.
  - Ensure that the loopback addresses of the PE routers are reachable from the customer sites, and from within the VRF instances, without altering loopback address reachability for P routers.
  - Your VPN must not disrupt or alter the flow of IPv4 packets within your network.

You should assume that the customer site routers are correctly configured to advertise their respective routes using the protocols identified in Figure 7.8. Please refer back to Chapter 1, or to your IGP discovery notes, for specifics on the OSPF baseline network as needed. Note that the data center router, and its IS-IS based route redistribution, are not involved in the VPN case study.

FIGURE 7.8 MPLS case study topology



VPN Case Study Analysis

Each configuration requirement for the VPN case study is matched to one or more valid router configurations and, where applicable, the commands that are used to confirm whether your network is operating within the specified case study guidelines. You begin with the grouping of Layer 2 VPN criteria:

- For VPN A:
  - You may not alter the existing BGP stanzas on `r4` and `r6`.
  - Ensure that C1 and C2 exchange their respective routes using RIP V2.
  - You may not configure RIP V2 on `r3`, `r5`, or `r7`.
  - You can access the C1 and C2 devices for testing purposes only; you must not modify their configuration.
  - You must have connectivity between C1 and C2.
  - Your VPN must not disrupt or alter the flow of IPv4 packets within your network.

Although the words "Layer 2" are entirely absent from the criteria listing, you know that some form of L2 VPN solution is required by virtue of C1 and C2 sharing a common IP subnet, and by the indications that you must run a RIP V2 routing protocol between C1 and C2 without enabling RIP on the PE routers. You must now decide which type of Layer 2 VPN you will deploy; in some cases the choice of draft-Kompella vs. draft-Martini will be left to the JNCIE candidate's discretion. In this example, the prohibition against modifying the existing BGP stanzas on `r4` and `r6` compels you to configure a draft-Martini solution because a draft-Kompella VPN can not function without the addition of the `l2vpn` family to the IBGP session between `r4` and `r6`.

You begin with the interface configuration changes needed at `r4` and `r6` to establish OoB connectivity to the VPN A devices. You also configure a logical unit that will support the Layer 2 VPN connection between the C1 and C2 routers. The changes made to `r4` are shown here with added highlights:

```
[edit interfaces fe-0/0/0]
lab@r4# show
vlan-tagging;
encapsulation vlan-ccc;
unit 0 {
    vlan-id 1;
    family inet {
        address 172.16.0.5/30;
    }
}
unit 700 {
    encapsulation vlan-ccc;
    vlan-id 700;
    family ccc;
}
```

The choice of logical unit numbers is not critical on the PE router, but you must configure compatible VLAN IDs for the logical units that support the OoB and Layer 2 VPN. Figure 7.8 specifies the VLAN ID at both sites for the OoB network, but the VLAN ID used to support the VPN is only specified on the r6-C2 link. This "extra rope" is designed to verify if the JNCIE candidate knows that matching VLAN IDs are required for draft-Martini VPNs because the TCC family is currently supported only for CCC and draft-Kompella types of connections. Although not shown, similar configuration changes are made to fe-0/1/3 interface at r6. After committing the changes, OoB connectivity is confirmed:

```
[edit interfaces fe-0/1/3]
lab@r6# run ping 172.16.0.10
PING 172.16.0.10 (172.16.0.10): 56 data bytes
64 bytes from 172.16.0.10: icmp_seq=0 ttl=255 time=0.625 ms
64 bytes from 172.16.0.10: icmp_seq=1 ttl=255 time=0.523 ms
^C
--- 172.16.0.10 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.523/0.574/0.625/0.051 ms
```

The next step in getting VPN A operational involves the configuration of the VPN's forwarding plane. You only need to enable MPLS forwarding on a subnet of the routers and interfaces in the JNCIE test bed to support VPN A, because no redundancy requirements have been posed. In this example, MPLS forwarding support is added to those interfaces on r4, r5, and r6 that constitute what appears to be the most direct path between PE routers r4 and r6.

The changes made to r6 are shown with highlights added:

```
[edit]
lab@r6# show protocols mpls
interface fe-0/1/0.0;

[edit]
lab@r6# show interfaces fe-0/1/0
unit 0 {
    family inet {
        address 10.0.8.5/30;
    }
    family mpls;
}
```

Similar changes are needed at r4 and r5. The changes made to r5 are shown:

```
[edit]
lab@r5# show interfaces fe-0/0/0
unit 0 {
    family inet {
        address 10.0.8.6/30;
    }
```

```
    family mpls;
}

[edit]
lab@r5# show interfaces so-0/1/0
encapsulation ppp;
unit 0 {
    family inet {
        address 10.0.2.9/30;
    }
    family mpls;
}

[edit]
lab@r5# show protocols mpls
interface fe-0/0/0.0;
interface so-0/1/0.0;
```

With the forwarding plane configured, you address the VPN's control plane by configuring LDP to operate on the interfaces at `r4`, `r5`, and `r6` with `mpls` family support. LDP must be enabled on the loopback interfaces of the PE routers to support the extended neighbor discovery required in a draft-Martini VPN. The changes made to `r4` are shown next:

```
[edit]
lab@r4# show protocols ldp
interface so-0/1/1.0;
interface lo0.0;
```

Similar changes are needed at `r5` and `r6`. The changes made to `r5` are also displayed:

```
[edit]
lab@r5# show protocols ldp
interface fe-0/0/0.0;
interface so-0/1/0.0;
```

Although it causes no harm, including LDP support on `r5`'s lo0 interface is not necessary because it has no need for extended LDP neighbor discovery. After committing the changes, you confirm the MPLS forwarding and control plane. You start with confirmation that the expected interfaces are enabled for MPLS processing and labeled packet handling:

```
[edit]
lab@r5# run show mpls interface
Interface        State       Administrative groups
fe-0/0/0.0       Up          <none>
so-0/1/0.0       Up          <none>
```

The sample display from r5 indicates MPLS support has been correctly configured on the interfaces that connect it to r4 and r6. LDP neighbor discovery and session establishment are confirmed next, also at r5:

```
[edit]
lab@r5# run show ldp neighbor
Address            Interface          Label space ID         Hold time
10.0.8.5           fe-0/0/0.0         10.0.9.6:0               12
10.0.2.10          so-0/1/0.0         10.0.3.4:0               11

[edit]
lab@r5# run show ldp session
  Address           State        Connection     Hold time
10.0.3.4            Operational  Open             20
10.0.9.6            Operational  Open             20
```

The displays confirm that r5's LDP instance sees r4 and r6 as neighbors, and that LDP sessions have been correctly established between r5 and its LDP neighbors. Even though the output indicates that LDP signaling is operational, you decide to confirm the successful establishment of the LSPs needed between the L2 VPN PE routers. You begin at r6:

```
[edit]
lab@r6# run show route table inet.3

inet.3: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.3.4/32        *[LDP/9] 00:15:10, metric 1
                    > to 10.0.8.6 via fe-0/1/0.0, Push 100004
10.0.3.5/32        *[LDP/9] 00:16:42, metric 1
                    > to 10.0.8.6 via fe-0/1/0.0
```

The display confirms successful establishment of LDP signaled LSPs that egress at r4 and r5. However, the display at r4 indicates a problem of some sort:

```
[edit]
lab@r4# run show route table inet.3

inet.3: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.3.5/32        *[LDP/9] 00:16:33, metric 1
                    > via so-0/1/1.0
```

Hmm, for some reason r4 has not established an LSP to the loopback address of r6. To correct this problem, the JNCIE candidate must understand LDP signaling and LDP's dependency on

tracking the IGP's preferred path to the LSP's egress point. The problem in this case is that r4 prefers the intra-area OSPF route to 10.0.9.6, as learned in r7's router LSA, over the summary version being advertised into the backbone area from r3 and r5:

```
[edit]
lab@r4# run show route 10.0.9.6

inet.0: 121371 destinations, 121373 routes (121371 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.9.6/32         *[OSPF/10] 00:24:07, metric 3
                     > to 10.0.2.17 via fe-0/0/3.0
```

You now realize that, as with the chapter body, the specifics of the IGP topology results in asymmetric routing between the PE routers:

```
[edit]
lab@r4# run traceroute 10.0.9.6
traceroute to 10.0.9.6 (10.0.9.6), 30 hops max, 40 byte packets
 1  10.0.2.17 (10.0.2.17)  0.775 ms  0.544 ms  0.412 ms
 2  10.0.8.9 (10.0.8.9)  0.715 ms  0.630 ms  0.591 ms
 3  10.0.9.6 (10.0.9.6)  0.820 ms  0.789 ms  0.765 ms

[edit]
lab@r6# run traceroute 10.0.3.4
traceroute to 10.0.3.4 (10.0.3.4), 30 hops max, 40 byte packets
 1  10.0.8.6 (10.0.8.6)  0.731 ms  0.576 ms  0.521 ms
 2  10.0.3.4 (10.0.3.4)  0.849 ms  0.748 ms  0.716 ms
```

Given the restrictions on altering the flow of IPv4 packets within the test bed, resolving this problem by reconfiguring the OSPF area boundaries, say by making r5 function as an internal area 2 router, or by disabling OSPF on the link between r4 and r7, are not really viable options. Given these circumstances, your best bet is to simply enable MPLS forwarding and LSP signaling support on r7 so that an LDP signaled LSP can be established along the existing IGP route from r4 to 10.0.9.6. This solution results in no significant changes to your IGP or the manner in which IPv4 packets are being forwarded. The changes made to r7 are shown next with added highlights:

```
[edit]
lab@r7# show interfaces fe-0/3/1
unit 0 {
    family inet {
        address 10.0.8.10/30;
    }
    family mpls;
}
```

```
[edit]
lab@r7# show interfaces fe-0/3/3
unit 0 {
    family inet {
        address 10.0.2.17/30;
    }
    family mpls;
}

[edit]
lab@r7# show protocols ldp
interface fe-0/3/1.0;
interface fe-0/3/3.0;

[edit]
lab@r7# show protocols mpls
interface fe-0/3/1.0;
interface fe-0/3/3.0;
```

Do not forget to add the `mpls` family to the Fast Ethernet interfaces that connect `r4` and `r5` to `r7`; you also need to enable MPLS processing and LDP support on these interfaces. After committing the changes at `r4`, `r5`, and `r7`, the `inet.3` table is again displayed at `r4`:

```
[edit]
lab@r4# run show route table inet.3

inet.3: 3 destinations, 3 routes (3 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.3.5/32        *[LDP/9] 00:33:38, metric 1
                    > via so-0/1/1.0
10.0.9.6/32        *[LDP/9] 00:03:03, metric 1
                    > to 10.0.2.17 via fe-0/0/3.0, Push 100002
10.0.9.7/32        *[LDP/9] 00:03:35, metric 1
                    > to 10.0.2.17 via fe-0/0/3.0
```

The highlight calls out the presence of an LSP from `r4` to `r6`'s loopback address, which has been successfully established through `r7` (and `r5`). In this example, the specific nature of the underlying IGP results in asymmetric routing between the PE routers. This is not a problem, but is a behavior worth noting to avoid confusion and surprises down the road. With the MPLS and LDP infrastructure now in place, all that remains to complete the Layer 2 component of the VPN case study is to define the `l2circuit` between PE routers `r4` and `r6`. The changes made to `r4` are shown here:

```
[edit protocols l2circuit]
lab@r4# show
```

```
neighbor 10.0.9.6 {
    interface fe-0/0/0.700 {
        virtual-circuit-id 700;
    }
}
```

A similar configuration is also added to r6. The key aspects of the l2circuit configuration are the correct specification of the egress PE's loopback address, the listing of the VPN interface along with the correct logical unit, and a virtual circuit ID value that is identical at both ends. After committing the l2circuit configuration at both PE routers, LDP extended neighbor discovery is verified at r6:

```
[edit protocols l2circuit]
lab@r6# run show ldp neighbor
Address            Interface          Label space ID         Hold time
10.0.3.4           lo0.0              10.0.3.4:0               14
10.0.8.6           fe-0/1/0.0         10.0.3.5:0               13
```

The display confirms that extended neighbor discovery is operational between r4 and r6; the status of the l2circuit is now displayed, again at r6:

```
[edit protocols l2circuit]
lab@r6# run show l2circuit connections
Layer-2 Circuit Connections:

Legend for connection status (St)   Legend for interface status
EI -- encapsulation invalid         Up -- operational
MM -- mtu mismatch                  Dn -- down
EM -- encapsulation mismatch        NP -- no present
CM -- control-word mismatch         DS -- disabled
OL -- no outgoing label             WE -- wrong encapsulation
Dn -- down                          UN -- uninitialized
VC-Dn -- Virtual circuit Down
Up -- operational
XX -- unknown

Neighbor: 10.0.3.4
    Interface                 Type  St     Time last up          # Up trans
    fe-0/1/3.700 (vc 700)     rmt   Up     Jun 19 17:44:41 2003           1
      Local interface: fe-0/1/3.700, Status: Up, Encapsulation: VLAN
      Remote PE: 10.0.3.4, Negotiated control-word: Yes (Null)
      Incoming label: 100007, Outgoing label: 100007
```

The display confirms correct establishment of the l2circuit. The final confirmation comes with end-to-end testing and verification of RIP route exchange between C1 and C2:

```
[edit protocols l2circuit]
lab@r4# run telnet 172.16.0.6
```

```
Trying 172.16.0.6...
Connected to 172.16.0.6.
Escape character is '^]'.

c1 (ttyp0)

login: lab
Password:
Last login: Thu Jun 19 10:50:34 from 172.16.0.5

--- JUNOS 5.6R2.4 built 2003-02-14 23:22:39 UTC

lab@c1> show route protocol rip

inet.0: 10 destinations, 10 routes (10 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

220.220.0.0/16     *[RIP/100] 00:01:28, metric 2, tag 0
                    > to 192.168.32.2 via fe-0/0/0.700
224.0.0.9/32       *[RIP/100] 00:02:09, metric 1
                      MultiRecv
```

The presence of the 220.220/16 prefix as a RIP route on the C1 device is a very good sign that the Layer 2 VPN is operational. Traceroute testing conducted at C1 with packets sourced from the 200.200.0.1 address provides the final proof that the draft-Martini based Layer 2 VPN between C1 and C2 is fully operational:

```
lab@c1> traceroute 220.220.0.1 source 200.200.0.1
traceroute to 220.220.0.1 (220.220.0.1) from 200.200.0.1, 30 hops max, 40 byte
   packets
 1  220.220.0.1 (220.220.0.1)  0.443 ms  0.327 ms  0.316 ms
```

With the Layer 2 VPN aspects of the case study dealt with in a resoundingly successful fashion, you begin the Layer 3 aspects of the case study by addressing a subset of Layer 3 VPN criteria that functions to establish baseline connectivity between C3 and C4:

- For VPN B:
  - Ensure that C3 and C4 exchange their respective routes using EBGP.
  - You must have connectivity between C3 and C4.
  - You must support traffic that originates or terminates on the multi-access VRF interfaces.
  - Your VPN must not disrupt or alter the flow of IPv4 packets within your network.

As with the Layer 2 scenario, the wording "Layer 3 VPN" is nowhere to be found in the scenario's requirement listing. The indication that a Layer 3 VPN solution is required comes with the lack of a common IP subnet between the C3 and C4 devices, and by the details of

Figure 7.8 that show the CE device's BGP sessions terminating at the local PE routers. This scenario requires redundant VRF configuration at r1 and r2, and is made complex by the need to perform firewall-filtering functions on VPN traffic that egresses at r3.

You have the option of using either LDP or RSVP-based signaling, given that none of your restrictions preclude, or require, any particular signaling protocol. Because LDP signaling is already in effect in portions of the test bed, you decide to begin the Layer 3 VPN scenario by adding MPLS and LDP support to r1, r2, and r3. The changes made to r3 are shown next with added highlights:

```
[edit]
lab@r3# show interfaces fe-0/0/0
unit 0 {
    family inet {
        address 10.0.4.13/30;
    }
    family mpls;
}

[edit]
lab@r3# show interfaces fe-0/0/1
unit 0 {
    family inet {
        address 10.0.4.1/30;
    }
    family mpls;
}

[edit]
lab@r3# show protocols mpls
interface fe-0/0/0.0;
interface fe-0/0/1.0;

[edit]
lab@r3# show protocols ldp
interface fe-0/0/0.0;
interface fe-0/0/1.0;
```

Note that enabling LDP on the router's loopback interface (to support extended neighbor discovery) is not necessary because the Layer 3 VPN's signaling protocol is based on MP-BGP. Although not shown, r2 is configured to support MPLS and LDP signaling in a manner that is similar to the changes shown here for r1:

```
[edit]
lab@r1# show interfaces fe-0/0/1
```

```
unit 0 {
    family inet {
        address 10.0.4.14/30;
    }
    family mpls;
}

[edit]
lab@r1# show protocols ldp
interface fe-0/0/1.0;

[edit]
lab@r1# show protocols mpls
interface fe-0/0/1.0;
```

You are required to configure redundancy for the failure of either r1 or r2, not for the failure of individual links. Therefore there is no need to enable LDP and MPLS support on r1's Fast Ethernet links to r2 or r4. After committing the MPLS and LDP changes, LSP establishment is verified at r3:

```
[edit]
lab@r3# run show route table inet.3

inet.3: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.6.1/32         *[LDP/9] 00:05:24, metric 1
                     > to 10.0.4.14 via fe-0/0/0.0
10.0.6.2/32         *[LDP/9] 00:00:09, metric 1
                     > to 10.0.4.2 via fe-0/0/1.0
```

Although not shown, you may assume that both r1 and r2 confirm establishment of an LDP signaled LSP that egresses at r3's loopback address. With the MPLS forwarding and control infrastructure in place, you move on to the configuration of the Layer 3 VPN. You decide to initially concentrate on r1 and r3; once all other aspects of the VPN are confirmed, it will be easy to replicate the working configuration from r1 to r2 to meet the stated redundancy requirements. You begin actual Layer 3 VPN configuration at r1 by defining the *c4* VRF. The completed VRF is shown next:

```
[edit routing-instances c4]
lab@r1# show
instance-type vrf;
interface fe-0/0/0.0;
route-distinguisher 10.0.6.1:1;
vrf-target target:65412:100;
```

```
protocols {
    bgp {
        group c4 {
            type external;
            peer-as 65222;
            neighbor 10.0.5.254;
        }
    }
}
```

Because the `vrf-target` option is not prohibited in this example, its use is highly recommended because it greatly simplifies the VRF policy and route target-related aspects of the VRF's configuration. The `protocols` portion of the *c4* VRF correctly defines the EBGP peering session to C4, including its new AS number of *65222*. However, when you attempt to commit your changes, you receive the following error:

```
[edit routing-instances c4]
lab@r1# commit
[edit protocols ospf area 0.0.0.1 interface fe-0/0/0.0]
  interface fe-0/0/0.0
  duplicate intf or intf not configured in this instance
error: configuration check-out failed
```

This problem is easily rectified by removing the pre-existing reference to the fe-0/0/0 VRF interface in the main OSPF instance:

```
[edit]
lab@r1# delete protocols ospf area 1 interface fe-0/0/0

[edit]
lab@r1# commit
commit complete
```

With the VRF and PE-CE routing protocol configured, you add the `inet-vpn` protocol family to the IBGP session between `r1` and `r3` to support the exchange of labeled routes between the PE routers. The changes are displayed next with added highlights:

```
[edit]
lab@r1# show protocols bgp group int
type internal;
local-address 10.0.6.1;
neighbor 10.0.6.2;
neighbor 10.0.3.3 {
    family inet {
        unicast;
    }
```

```
    family inet-vpn {
        unicast;
    }
}
neighbor 10.0.3.4;
neighbor 10.0.3.5;
neighbor 10.0.9.6;
neighbor 10.0.9.7;
```

In this example, MP-BGP protocol family support is only modified on the peering session associated with r3. It does not cause any harm to apply this change to all IBGP peers, but it will result in the temporary tear-down of the established IBGP sessions. The inet family is also explicitly configured to prevent disruption to any IPv4 traffic that may rely on the IBGP session between r1 and r3. After committing the changes at r1, the EBGP peering session to C4 is quickly verified:

```
[edit]
lab@r1# run show bgp summary instance c4
Groups: 1 Peers: 1 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
C4.inet.0              1          1          0          0          0          0
Peer               AS       InPkt     OutPkt    OutQ  Flaps Last Up/Dwn
   State|#Active/Received/Damped...
10.0.5.254      65222         27         29       0       0       12:35 Establ
   c4.inet.0: 1/1/0

[edit]
lab@r1# run show route table c4

c4.inet.0: 3 destinations, 3 routes (3 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.5.0/24        *[Direct/0] 00:12:44
                    > via fe-0/0/0.0
10.0.5.1/32        *[Local/0] 00:12:44
                      Local via fe-0/0/0.0
120.120.0.0/16     *[BGP/170] 00:12:40, MED 0, localpref 100
                      AS path: 65222 I
                    > to 10.0.5.254 via fe-0/0/0.0
```

The EBGP session between r1 and C4 is established, and a display of the *c4* instance's VRF confirms the presence of the 120.120/16 prefix as learned though BGP. The displays indicate that r1 is correctly configured for Layer 3 VPN interaction with its attached CE device. Additional confirmation will have to wait until r3 has its VRF and MP-IBGP configuration in place. With

your attention now focused on r3, you begin modifying its configuration by adding inet-vpn family support to the IBGP peering sessions associated with r1 and r2:

```
[edit protocols bgp group int]
lab@r3# show
type internal;
local-address 10.0.3.3;
export nhs;
neighbor 10.0.6.1 {
    family inet {
        unicast;
    }
    family inet-vpn {
        unicast;
    }
}
neighbor 10.0.6.2 {
    family inet {
        unicast;
    }
    family inet-vpn {
        unicast;
    }
}
neighbor 10.0.3.4;
neighbor 10.0.3.5;
neighbor 10.0.9.6;
neighbor 10.0.9.7;
```

The VRF-related changes that are made to r3 to support initial Layer 3 VPN connectivity are shown next:

```
[edit routing-instances c3]
lab@r3# show
instance-type vrf;
interface fe-0/0/2.0;
route-distinguisher 10.0.3.3:1;
vrf-target target:65412:100;
protocols {
    bgp {
        group c3 {
            type external;
            peer-as 65222;
```

```
            neighbor 172.16.0.14;
        }
    }
}
```

Note that the *c3* VRF at r3 also makes use of the vrf-target option, and that a matching route target community has been configured. After committing the changes at r3, support for the inet and inet-vpn families is verified on the MP-IBGP session between r1 and r3:

```
[edit routing-instances c3]
lab@r3# run show bgp neighbor 10.0.6.1 | match NLRI
  NLRI advertised by peer: inet-unicast inet-vpn-unicast
  NLRI for this session: inet-unicast inet-vpn-unicast
```

The display confirms that both r3 and r1 are correctly configured to support the required address families. You move on to verify the BGP interaction between r3 and C3:

```
[edit routing-instances c3]
lab@r3# run show bgp summary instance c3
Groups: 1 Peers: 1 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
c3.inet.0              3          3          0          0          0          0
Peer               AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn
   State|#Active/Received/Damped...
172.16.0.14     65222         18         20       0       2        6:30 Establ
   c3.inet.0: 1/1/0
```

The summary display confirms EBGP session establishment between r3 and C3, and also shows that a single prefix has been received and installed in the *c3* VRF. The *c3* VRF is displayed to determine what routes have been received from local CE and the remote PE devices:

```
[edit routing-instances c3]
lab@r3# run show route table c3

c3.inet.0: 5 destinations, 5 routes (5 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.5.0/24             *[BGP/170] 00:11:17, localpref 100, from 10.0.6.1
                          AS path: I
                        > to 10.0.4.14 via fe-0/0/0.0, Push 100001
120.120.0.0/16          *[BGP/170] 00:11:17, MED 0, localpref 100, from 10.0.6.1
                          AS path: 65222 I
                        > to 10.0.4.14 via fe-0/0/0.0, Push 100001
130.130.0.0/16          *[BGP/170] 00:06:33, MED 0, localpref 100
                          AS path: 65222 I
                        > to 172.16.0.14 via fe-0/0/2.0
```

```
172.16.0.12/30     *[Direct/0] 00:06:43
                    > via fe-0/0/2.0
172.16.0.13/32     *[Local/0] 00:11:28
                      Local via fe-0/0/2.0
```

The output confirms the presence of the 130.130/16 prefix, which is learned from the EBGP session to C3. Also present in the *c3* VRF are the 120.120/16 and the 10.0.5/24 prefixes, as advertised by PE router r1. The presence of these routes in the *c3* VRF, and their association with LSP-based next hops, indicates that labeled VPN route exchange between the PE routers is working, and that MPLS LSPs are available to accommodate the forwarding of VPN traffic. Because the routes learned from the remote PE are BGP routes, and because the default BGP policy is to advertise active BGP routes to EBGP peers, you expect to find that r3 is advertising C4's routes to C3 with no policy additions or modifications required. This behavior is in contrast to that seen in the PE-CE OSPF routing example in the chapter body, where policy was required to effect the redistribution of BGP into OSPF.

However, before you can issue a successful `show route advertising-protocol bgp 172.16.0.14` command to confirm the expected behavior, you must remove the pre-existing (and now duplicate) EBGP peering definition from the main routing instance of r3:

```
[edit]
lab@r3# delete protocols bgp group ext

[edit]
lab@r3# commit
commit complete

[edit]
lab@r3# run show route advertising-protocol bgp 172.16.0.14

c3.inet.0: 5 destinations, 5 routes (5 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
* 10.0.5.0/24             Self                                    I
* 120.120.0.0/16          Self                                    65222 I
* 130.130.0.0/16          172.16.0.14                             65222 I
```

As predicted, the routes associated with the C4 router are being correctly advertised to the C3. Similar results are observed at r1 after removing the redundant EBGP peering definition from its main routing instance:

```
[edit]
lab@r1# delete protocols bgp group p1

[edit]
lab@r1# commit
commit complete
```

```
[edit]
lab@r1# run show route advertising-protocol bgp 10.0.5.254

c4.inet.0: 5 destinations, 5 routes (5 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
* 120.120.0.0/16          10.0.5.254                              65222 I
* 130.130.0.0/16          Self                                    65222 I
* 172.16.0.12/30          Self                                    I
```

The 130.130/16 and VRF interface routes associated with C3 are correctly being advertised to C4. This is starting to seem too easy, so you decide to conduct some quick end-to-end testing before moving on to the remaining criteria:

```
[edit]
lab@r1# run telnet routing-instance c4 10.0.5.254
Trying 10.0.5.254...
Connected to 10.0.5.254.
Escape character is '^]'.

c4 (ttyp1)

login: lab
Password:
Last login: Sun Jun 29 19:07:59 from 10.0.5.1

--- JUNOS 5.6R2.4 built 2003-02-14 23:22:39 UTC

lab@c4> show route protocol bgp

inet.0: 7 destinations, 7 routes (7 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

172.16.0.12/30     *[BGP/170] 00:22:51, localpref 100
                      AS path: 65412 I
                    > to 10.0.5.1 via fe-0/0/0.0
```

Hmm, the display is puzzling because a previous command confirmed that `r1` is advertising both the 130.130/16 and the 172.16.0.12/30 prefixes to C4. Yet for some reason, C4 is not displaying the 130.130/16 route. Also of note is the indication that no hidden routes exist at C4. Seeing that one of the routes is present, and that the other is not, you start to wonder "what is different about these routes?"

Upon re-examination of the contents of the *c4* VRF on `r1` (as shown previously), you notice that the 130.130/16 route has an AS path of 65222 while the 172.16.0.12/30 route has a null

AS path. Seeing this, the true nature of the problem dawns upon you; C3 and C4 have the same AS number, and JUNOS software immediately discards (not hides) any route that fails AS path sanity checks! You can test this theory by setting the `keep-all` option in the CE device's BGP stanza, because this option causes routes with AS path sanity problems to be retained in the Adj-RIB-in, albeit as a hidden route.

You can not resolve this problem by configuring support for AS path loops under `[edit routing-options autonomous-system loops]` because your restrictions prevent configuration changes in the peripheral routers. The only viable solution for resolving the AS loop problem is to deploy the `as-override` feature at both PE routers. This option tells the PE to replace the last AS number in the AS path with an extra copy of the PE's AS number when the route is sent to the attached CE device. You configure `r3` to perform `as-override` and `commit` the change; a similar change is also made at `r1` (not shown):

```
[edit routing-instances c3]
lab@r3# set protocols bgp group c3 as-override
```

To confirm the fix, you telnet to a CE device and inspect its routing table for BGP routes:

```
[edit routing-instances c3]
lab@r3# run telnet routing-instance c3 172.16.0.14
Trying 172.16.0.14...
Connected to 172.16.0.14.
Escape character is '^]'.

C3 (ttyp1)

login: lab
Password:
Last login: Tue Apr  4 14:48:11 from 172.16.0.13

--- JUNOS 5.6R2.4 built 2003-02-14 23:22:39 UTC

lab@C3> show route protocol bgp

inet.0: 9 destinations, 10 routes (9 active, 0 holddown, 1 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.5.0/24             *[BGP/170] 00:02:13, localpref 100
                          AS path: 65412 I
                        > to 172.16.0.13 via fe-0/0/0.0
120.120.0.0/16          *[BGP/170] 00:02:13, localpref 100
                          AS path: 65412 65412 I
                        > to 172.16.0.13 via fe-0/0/0.0
```

The 120.120/16 route associated with C4 is now present, and the AS path clearly shows the effects of the `as-override` knob. While at C3 you decide to conduct some

end-to-end connectivity testing:

```
lab@C3> traceroute 120.120.0.1
traceroute to 120.120.0.1 (120.120.0.1), 30 hops max, 40 byte packets
 1  172.16.0.13 (172.16.0.13)  0.396 ms  0.296 ms  0.273 ms
 2  10.0.4.14 (10.0.4.14)  0.239 ms  0.211 ms  0.208 ms
     MPLS Label=100002 CoS=0 TTL=1 S=1
 3  120.120.0.1 (120.120.0.1)  0.296 ms  0.277 ms  0.276 ms

lab@C3> traceroute 120.120.0.1 source 130.130.0.1
traceroute to 120.120.0.1 (120.120.0.1) from 130.130.0.1, 30 hops max, 40 byte
   packets
 1  172.16.0.13 (172.16.0.13)  0.384 ms  0.290 ms  0.275 ms
 2  10.0.4.14 (10.0.4.14)  0.220 ms  0.211 ms  0.207 ms
     MPLS Label=100002 CoS=0 TTL=1 S=1
 3  120.120.0.1 (120.120.0.1)  0.293 ms  0.275 ms  0.272 ms
```

Both traceroute tests succeed, which confirms that you have established basic end-to-end connectivity for VPN B. The ability to support traffic originating on a multi-access VRF interface is confirmed with the first trace route test. With basic Layer 3 VPN functionality confirmed, you move on to address the next case study requirement:

- For VPN B
  - The failure of either `r1` or `r2` can not disable VPN B.

You need to configure `r2` with similar VRF and MP-IBGP settings to achieve the redundancy required by this criterion; this is a good time for a `load merge terminal` operation after you edit the route distinguisher value for use at `r2`. The initial changes made to `r2` are shown next using the CLI's `compare` function:

```
[edit]
lab@r2# show | compare rollback 1
[edit protocols bgp group int neighbor 10.0.3.3]
+      family inet {
+          unicast;
+      }
+      family inet-vpn {
+          unicast;
+      }
 [edit protocols bgp]
-    group p1 {
-        type external;
-        export ebgp-out;
-        neighbor 10.0.5.254 {
-            peer-as 65050;
-        }
-    }
```

The changes indicate that the required address families have been added to r2, and that the pre-existing *p1* peering definition has been removed from the main routing instance. This portion of the display shows that r2's fe-0/0/0 interface has been removed from the main OSPF routing instance:

```
[edit protocols ospf area 0.0.0.1]
-     interface fe-0/0/0.0 {
-         passive;
-     }
```

And the final portion of the display confirms the addition of a *c4* VRF to r2; note that the RD has been uniquely set based on r2's router ID while the RT is set to the same value in use at r1 and r3:

```
[edit]
+  routing-instances {
+      c4 {
+          instance-type vrf;
+          interface fe-0/0/0.0;
+          route-distinguisher 10.0.6.2:1;
+          vrf-target target:65412:100;
+          protocols {
+              bgp {
+                  group c4 {
+                      type external;
+                      peer-as 65222;
+                      as-override;
+                      neighbor 10.0.5.254;
+                  }
+              }
+          }
+      }
+  }
```

After the changes are committed, the presence of C3 and C4 routes in r2's *c4* VRF provides good indication that r2 is configured properly:

```
lab@r2> show route table c4

c4.inet.0: 5 destinations, 5 routes (5 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.5.0/24        *[Direct/0] 00:32:36
                    > via fe-0/0/0.0
10.0.5.2/32        *[Local/0] 00:32:36
                      Local via fe-0/0/0.0
```

```
120.120.0.0/16     *[BGP/170] 00:32:32, MED 0, localpref 100
                      AS path: 65222 I
                    > to 10.0.5.254 via fe-0/0/0.0
130.130.0.0/16     *[BGP/170] 00:02:15, MED 0, localpref 100, from 10.0.3.3
                      AS path: 65222 I
                    > to 10.0.4.1 via fe-0/0/2.0, Push 100012
172.16.0.12/30     *[BGP/170] 00:02:15, localpref 100, from 10.0.3.3
                      AS path: I
                    > to 10.0.4.1 via fe-0/0/2.0, Push 100012
```

A quick traceroute or two confirms that MPLS forwarding from r2 to C3 is functional:

```
lab@r2> traceroute routing-instance c4 172.16.0.14
traceroute to 172.16.0.14 (172.16.0.14), 30 hops max, 40 byte packets
 1  10.0.4.1 (10.0.4.1)  0.678 ms  0.504 ms  0.474 ms
     MPLS Label=100012 CoS=0 TTL=1 S=1
 2  172.16.0.14 (172.16.0.14)  0.246 ms  0.229 ms  0.214 ms

lab@r2> traceroute routing-instance c4 130.130.0.1
traceroute to 130.130.0.1 (130.130.0.1), 30 hops max, 40 byte packets
 1  10.0.4.1 (10.0.4.1)  0.708 ms  0.507 ms  0.461 ms
     MPLS Label=100012 CoS=0 TTL=1 S=1
 2  130.130.0.1 (130.130.0.1)  0.239 ms  0.231 ms  0.213 ms
```

The final redundancy test verifies that VPN connectivity is not permanently impacted by the failure of r1 or r2. You start with a traceroute from C4 to determine the current forwarding path for traffic flowing from C4 to C3:

```
[edit routing-instances c4]
lab@r2# run telnet routing-instance c4 10.0.5.254
Trying 10.0.5.254...
Connected to 10.0.5.254.
Escape character is '^]'.

C4 (ttyp1)

login: lab
Password:
Last login: Tue Apr  4 14:48:11 from 172.16.0.13

--- JUNOS 5.6R2.4 built 2003-02-14 23:22:39 UTC

lab@c4> traceroute 130.130.0.1
traceroute to 130.130.0.1 (130.130.0.1), 30 hops max, 40 byte packets
```

```
 1  10.0.5.1 (10.0.5.1)  0.385 ms  0.205 ms  0.237 ms
 2  10.0.4.13 (10.0.4.13)  0.632 ms  0.515 ms  0.507 ms
     MPLS Label=100003 CoS=0 TTL=1 S=1
 3  130.130.0.1 (130.130.0.1)  0.331 ms  0.287 ms  0.277 ms
```

Noting that `r1` is currently the first hop in the traceroute, you temporarily deactivate `r1`'s `protocols` stanza:

```
[edit]
lab@r1# deactivate protocols

[edit]
lab@r1# commit
commit complete
```

After a few moments, the traceroute test is repeated:

```
lab@c4> traceroute 130.130.0.1
traceroute to 130.130.0.1 (130.130.0.1), 30 hops max, 40 byte packets
 1  10.0.5.2 (10.0.5.2)  0.270 ms  0.263 ms  0.152 ms
 2  10.0.4.1 (10.0.4.1)  0.621 ms  0.515 ms  0.767 ms
     MPLS Label=100003 CoS=0 TTL=1 S=1
 3  130.130.0.1 (130.130.0.1)  0.320 ms  0.283 ms  0.279 ms
```

The presence of `r2` in the first hop, coupled with the successful competition of the traceroute, confirms that you have met the stated redundancy requirements. Do not forget to activate the `protocols` stanza on `r1` before proceeding! With Layer 3 VPN redundancy verified, you move on to the next case study requirement:

- For VPN B:
  - Ensure that the loopback addresses of the PE routers are reachable from the customer sites, and from within the VRF instances, without altering loopback address reachability for P routers.

This requirement can not be accomplished with a non-VRF interface that is used to provide a CE with "Internet" access because the requirements stipulate that the PE router's loopback address must also be reachable from within the VRF. Simply placing the router's loopback interface into the VRF instance makes the corresponding address unreachable for P routers. While it may be possible in some JUNOS software releases to achieve your goal with RIB group configurations and/or static routes with `receive` next hops, the most expedient solution is to assign a new logical unit to the PE's loopback interface and include the new logical interface in the VRF. The changes made to `r3` are shown next with highlights:

```
[edit]
lab@r3# show interfaces lo0
unit 0 {
    family inet {
        address 10.0.3.3/32;
```

```
    }
}
unit 1 {
    family inet {
        address 10.0.3.3/32;
    }
}

[edit]
lab@r3# show routing-instances
c3 {
    instance-type vrf;
    interface fe-0/0/2.0;
    interface lo0.1;
    route-distinguisher 10.0.3.3:1;
    vrf-target target:65412:100;
    protocols {
        bgp {
            group c3 {
                type external;
                peer-as 65222;
                as-override;
                neighbor 172.16.0.14;
            }
        }
    }
}
```

After committing the change, you will find that the loopback addresses of the local and remote PE routers are present in the VRF at r1 and r2:

```
[edit]
lab@r1# run show route table c4 10.0.3.3

c4.inet.0: 7 destinations, 7 routes (7 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.3.3/32        *[BGP/170] 00:08:55, localpref 100, from 10.0.3.3
                      AS path: I
                    > to 10.0.4.13 via fe-0/0/1.0, Push 100000

[edit]
lab@r1# run show route table c4 10.0.6.1
```

```
c4.inet.0: 7 destinations, 7 routes (7 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.6.1/32         *[Direct/0] 00:09:00
                     > via lo0.1
```

However, the loopback address of r2 is missing from r1's VRF:

```
[edit]
lab@r1# run show route table c4 10.0.6.2

[edit]
lab@r1#
```

This condition can be corrected by adding inet-vpn family support to the IBGP session between r1 and r2, or by making r3 a route reflector. The latter approach is taken here to demonstrate VPN route reflection. The highlighted changes are made to r3's configuration; note that route reflection is enabled at the neighbor level to minimize the impact on the other routers in the test bed:

```
[edit protocols bgp group int]
lab@r3# show
type internal;
local-address 10.0.3.3;
export nhs;
neighbor 10.0.6.1 {
    family inet {
        unicast;
    }
    family inet-vpn {
        unicast;
    }
    cluster 10.0.3.3;
}
neighbor 10.0.6.2 {
    family inet {
        unicast;
    }
    family inet-vpn {
        unicast;
    }
    cluster 10.0.3.3;
}
neighbor 10.0.3.4;
neighbor 10.0.3.5;
```

```
neighbor 10.0.9.6;
neighbor 10.0.9.7;
```

However, the lack of LSP forwarding capability between r1 and r2 results in r2's loopback address being hidden at r1:

```
[edit]
lab@r1# run show route table c4 10.0.6.2 hidden detail

c4.inet.0: 8 destinations, 10 routes (7 active, 0 holddown, 3 hidden)
10.0.6.2/32 (1 entry, 0 announced)
         BGP    Preference: 170/-101
                Route Distinguisher: 10.0.6.2:1
                Next hop type: Unusable
                State: <Secondary Hidden Int Ext>
                Local AS: 65412 Peer AS: 65412
                Age: 4:20
                Task: BGP_65412.10.0.3.3+1365
                AS path: I (Originator) Cluster list:  10.0.3.3
                AS path:  Originator ID: 10.0.6.2
                Communities: target:65412:100
                VPN Label: 100003
                Localpref: 100
                Router ID: 10.0.3.3
```

The route is hidden because it can not be resolved through an LSP in the inet.3 routing table. Adding LDP and MPLS support to the Fast Ethernet link connecting r1 and r2 resolves the issue. Modifications similar to those shown here for r1 are also needed at r2:

```
[edit]
lab@r1# set interfaces fe-0/0/2 unit 0 family mpls

[edit]
lab@r1# set protocols ldp interface fe-0/0/2

[edit]
lab@r1# set protocols mpls interface fe-0/0/2
```

With the changes committed, the loopback addresses of all three PE routers are confirmed in the VRF tables of all PE routers:

```
[edit]
lab@r2# run show route table c4 10.0.3.3

c4.inet.0: 8 destinations, 10 routes (8 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

```
10.0.3.3/32        *[BGP/170] 00:01:28, localpref 100, from 10.0.3.3
                      AS path: I
                    > to 10.0.4.1 via fe-0/0/2.0, Push 100000

[edit]
lab@r2# run show route table c4 10.0.6/24

c4.inet.0: 8 destinations, 10 routes (8 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.6.1/32        *[BGP/170] 00:01:18, localpref 100, from 10.0.3.3
                      AS path: I
                    > to 10.0.4.5 via fe-0/0/3.0, Push 100003
10.0.6.2/32        *[Direct/0] 00:20:51
                    > via lo0.1
```

Although the loopback addresses are present in the VRFs, you need to create and apply a `routing-instance` export policy to effect the advertisement of the direct routes to the attached CE routers; without such a policy, only loopback addresses learned through BGP are advertised:

```
[edit]
lab@r1# run show route advertising-protocol bgp 10.0.5.254 10.0.6/24

c4.inet.0: 8 destinations, 10 routes (8 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
* 10.0.6.2/32             Self                                    I
```

The changes shown here are for `r3`. Similar changes are required on `r1` and `r2`.

```
[edit]
lab@r3# show policy-options policy-statement send-lo0
term 1 {
    from {
        protocol direct;
        route-filter 10.0.3.3/32 exact;
    }
    then accept;
}

[edit]
lab@r3# show routing-instances c3 protocols bgp
group c3 {
    type external;
    export send-lo0;
```

```
    peer-as 65222;
    as-override;
    neighbor 172.16.0.14;
}
```

Proper operation is confirmed when all three loopback addresses are present at both CE devices, which is now the case for C4 and C3 (not shown):

```
lab@c4> show route protocol bgp 10.0.3.3

inet.0: 11 destinations, 16 routes (11 active, 0 holddown, 2 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.3.3/32        *[BGP/170] 00:32:35, localpref 100
                      AS path: 65412 I
                    > to 10.0.5.1 via fe-0/0/0.0
                    [BGP/170] 00:32:31, localpref 100
                      AS path: 65412 I
                    > to 10.0.5.2 via fe-0/0/0.0

lab@c4> show route protocol bgp 10.0.6/24

inet.0: 11 destinations, 16 routes (11 active, 0 holddown, 2 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.6.1/32        *[BGP/170] 00:25:03, localpref 100
                      AS path: 65412 I
                    > to 10.0.5.2 via fe-0/0/0.0
10.0.6.2/32        *[BGP/170] 00:25:14, localpref 100
                      AS path: 65412 I
                    > to 10.0.5.1 via fe-0/0/0.0
```

Traceroute testing at `r1` from the main routing instance, and from the *c4* instance, confirms loopback address reachability from within the VRF and also confirms that loopback reachability remains unchanged for P routers, which rely on the main instance for loopback reachability:

```
[edit]
lab@r1# run traceroute 10.0.3.3
traceroute to 10.0.3.3 (10.0.3.3), 30 hops max, 40 byte packets
 1  10.0.3.3 (10.0.3.3)  0.482 ms  0.396 ms  0.346 ms

[edit]
lab@r1# run traceroute 10.0.3.3 routing-instance c4
traceroute to 10.0.3.3 (10.0.3.3), 30 hops max, 40 byte packets
```

```
 1  10.0.3.3 (10.0.3.3)  0.679 ms  0.508 ms  0.462 ms
     MPLS Label=100000 CoS=0 TTL=1 S=1
 2  10.0.3.3 (10.0.3.3)  0.467 ms  0.453 ms  0.424 ms
```

The results shown thus far indicate that you have met all behavior requirements and configuration restrictions, save one. This brings you face to face with the final case study requirement for VPN B:

- For VPN B:
  - You must count all ICMP traffic that egresses `r3`'s fe-0/0/2 interface.

The specified behavior requires that you make IP II functionality available at `r3` for egress VPN traffic. Both the `vrf-table-label` and `vt-interface` options provide IP II functionality at the egress of a Layer 3 VPN, and both options have restrictions as to when they can be used. The JUNOS software release 5.6 deployed in the test bed supports `vrf-table-label` only when the PE router's core-facing interfaces are point-to-point. The presence of core-facing Ethernet interfaces at `r3` therefore eliminates the `vrf-table-label` option. The use of a `vt-interface` requires that the PE routers have a Tunnel Services (TS) PIC installed, which as luck would have it, happens to be the case with `r3`:

```
[edit]
lab@r3# run show chassis fpc pic-status
Slot 0 Online
  PIC 0    4x F/E, 100 BASE-TX
  PIC 1    2x OC-3 ATM, MM
  PIC 2    4x OC-3 SONET, MM
  PIC 3    1x Tunnel
```

You begin by adding the `vt-interface` to the *c3* VRF table at `r3`:

```
[edit routing-instances c3]
lab@r3# set interface vt-0/3/0
```

In this example, the `vt-interface` defaults to logical unit 0 because no unit number was specified. Use care to ensure that each additional VRF uses a unique `vt-interface` unit number for proper operation. The modified VRF table is displayed next with added highlights:

```
[edit routing-instances c3]
lab@r3# show
instance-type vrf;
interface fe-0/0/2.0;
interface vt-0/3/0.0;
route-distinguisher 10.0.3.3:1;
vrf-target target:65412:100;
protocols {
    bgp {
        group c3 {
            type external;
```

```
            peer-as 65222;
            as-override;
            neighbor 172.16.0.14;
        }
    }
}
```

Before the `vt-interface` can operate within the VRF, you must configure the `inet` family on the corresponding logical unit, as shown here:

```
[edit interfaces vt-0/3/0]
lab@r3# show
unit 0 {
    family inet;
}
```

After committing the changes, you will see that `vt-interface` status is displayed:

```
lab@r3> show interfaces vt-0/3/0
Physical interface: vt-0/3/0, Enabled, Physical link is Up
  Interface index: 26, SNMP ifIndex: 37
  Type: Loopback, Link-level type: Virtual-loopback-tunnel, MTU: Unlimited,
     Speed: 800mbps
  Device flags   : Present Running
  Interface flags: SNMP-Traps
  Input rate     : 0 bps (0 pps)
  Output rate    : 0 bps (0 pps)

  Logical interface vt-0/3/0.0 (Index 13) (SNMP ifIndex 39)
    Flags: Point-To-Point SNMP-Traps Encapsulation: Virtual-loopback-tunnel
    Bandwidth: 0
    Protocol inet, MTU: Unlimited
      Flags: None
```

The output indicates the `vt-interface` is operational. You move forward on the final task by defining a simple firewall filter that counts ICMP packets:

```
[edit]
lab@r3# show firewall
filter c3 {
    term 1 {
        from {
            protocol icmp;
        }
        then count vpnb-icmp;
    }
```

```
    term 2 {
        then accept;
    }
}
```

The *c3* filter is then applied in the output direction of r3's VRF interface:

```
[edit]
lab@r3# show interfaces fe-0/0/2
unit 0 {
    family inet {
        filter {
            output c3;
        }
        address 172.16.0.13/30;
    }
}
```

After committing the changes, clear the firewall counters and display the *vpnb-icmp* counter:

```
lab@r3> clear firewall all

lab@r3> show firewall
Filter: c3
Counters:
Name                                                    Bytes              Packets
vpnb-icmp                                                   0                    0
```

The display confirms that the current *vpnb-icmp* counter value is zero. You now generate 100 test packets from C4:

```
lab@c4> ping 130.130.0.1 rapid count 100
PING 130.130.0.1 (130.130.0.1): 56 data bytes
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !!!!!!!!!!!!!!!!!!!!
--- 130.130.0.1 ping statistics ---
100 packets transmitted, 100 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.279/0.286/0.463/0.024 ms

lab@c4>
```

And the results are verified at r3:

```
lab@r3> show firewall
Filter: c3
Counters:
Name                                                    Bytes              Packets
vpnb-icmp                                                8400                  100
```

The *vpnb-icmp* counter displays the exact number of ICMP packets generated during the test. This confirms that your `vt-interface` and firewall-related configuration is working as designed. Congratulations are now in order, because you have met all requirements posed in the VPN case study!

VPN Case Study Configurations

The changes needed in the OSPF baseline network topology to complete the VPN case study are listed in Listings 7.1 through 7.7 for all routers in the test bed, with highlights added.

Listing 7.1: VPN Case Study Configuration for *r1*

```
[edit]
lab@r1# show interfaces fe-0/0/1
unit 0 {
    family inet {
        address 10.0.4.14/30;
    }
    family mpls;
}
[edit]
lab@r1# show interfaces fe-0/0/2
unit 0 {
    family inet {
        address 10.0.4.5/30;
    }
    family mpls;
}
[edit]
lab@r1# show interfaces lo0
unit 0 {
    family inet {
        address 10.0.6.1/32;
    }
}
unit 1 {
    family inet {
        address 10.0.6.1/32;
    }
}
[edit]
lab@r1# show protocols
```

```
mpls {
    interface fe-0/0/1.0;
    interface fe-0/0/2.0;
}
bgp {
    group int {
        type internal;
        local-address 10.0.6.1;
        neighbor 10.0.6.2;
        neighbor 10.0.3.3 {
            family inet {
                unicast;
            }
            family inet-vpn {
                unicast;
            }
        }
        neighbor 10.0.3.4;
        neighbor 10.0.3.5;
        neighbor 10.0.9.6;
        neighbor 10.0.9.7;
    }
}
ospf {
    area 0.0.0.1 {
        stub;
        interface fe-0/0/1.0;
        interface fe-0/0/2.0;
        interface fe-0/0/3.0;
    }
}
ldp {
    interface fe-0/0/1.0;
    interface fe-0/0/2.0;
}
[edit]
lab@r1# show routing-instances
c4 {
    instance-type vrf;
    interface fe-0/0/1.0;
    interface lo0.1;
    route-distinguisher 10.0.6.1:1;
```

```
    vrf-target target:65412:100;
    protocols {
        bgp {
            group c4 {
                type external;
                peer-as 65222;
                as-override;
                neighbor 10.0.5.254;
            }
        }
    }
}
```

The following items were deleted from r1's OSPF baseline configuration to complete the VPN case study:

```
[edit protocols bgp]
-    group p1 {
-        type external;
-        export ebgp-out;
-        neighbor 10.0.5.254 {
-            peer-as 65050;
-        }
-    }
[edit protocols ospf area 0.0.0.1]
-     interface fe-0/0/0.0 {
-         passive;
-     }
```

Listing 7.2: VPN Case Study Configuration for *r2*

```
[edit]
lab@r2# show interfaces fe-0/0/2
unit 0 {
    family inet {
        address 10.0.4.2/30;
    }
    family mpls;
}
[edit]
lab@r2# show interfaces fe-0/0/3
unit 0 {
    family inet {
        address 10.0.4.6/30;
    }
```

```
    family mpls;
}
[edit]
lab@r2# show interfaces lo0
unit 0 {
    family inet {
        address 10.0.6.2/32;
    }
}
unit 1 {
    family inet {
        address 10.0.6.2/32;
    }
}
[edit]
lab@r2# show protocols
mpls {
    interface fe-0/0/2.0;
    interface fe-0/0/3.0;
}
bgp {
    group int {
        type internal;
        local-address 10.0.6.2;
        neighbor 10.0.6.1;
        neighbor 10.0.3.3 {
            family inet {
                unicast;
            }
            family inet-vpn {
                unicast;
            }
        }
        neighbor 10.0.3.4;
        neighbor 10.0.3.5;
        neighbor 10.0.9.6;
        neighbor 10.0.9.7;
    }
}
ospf {
```

```
    area 0.0.0.1 {
        stub;
        interface fe-0/0/1.0;
        interface fe-0/0/2.0;
        interface fe-0/0/3.0;
    }
}
ldp {
    interface fe-0/0/2.0;
    interface fe-0/0/3.0;
}
[edit]
lab@r2# show routing-instances
c4 {
    instance-type vrf;
    interface fe-0/0/0.0;
    interface lo0.1;
    route-distinguisher 10.0.6.2:1;
    vrf-target target:65412:100;
    protocols {
        bgp {
            group c4 {
                type external;
                peer-as 65222;
                as-override;
                neighbor 10.0.5.254;
            }
        }
    }
}
```

The following items were deleted from r2's OSPF baseline configuration to complete the VPN case study:

```
[edit protocols bgp]
-    group p1 {
-        type external;
-        export ebgp-out;
-        neighbor 10.0.5.254 {
-            peer-as 65050;
-        }
-    }
```

```
[edit protocols ospf area 0.0.0.1]
-     interface fe-0/0/0.0 {
-         passive;
-     }
```

Listing 7.3: VPN Case Study Configuration for *r3*

```
[edit]
lab@r3# show interfaces fe-0/0/0
unit 0 {
    family inet {
        address 10.0.4.13/30;
    }
    family mpls;
}
[edit]
lab@r3# show interfaces fe-0/0/1
unit 0 {
    family inet {
        address 10.0.4.1/30;
    }
    family mpls;
}
[edit]
lab@r3# show interfaces fe-0/0/2
unit 0 {
    family inet {
        filter {
            output c3;
        }
        address 172.16.0.13/30;
    }
}
[edit]
lab@r3# show interfaces vt-0/3/0
unit 0 {
    family inet;
}
[edit]
lab@r3# show interfaces lo0
unit 0 {
    family inet {
```

```
        address 10.0.3.3/32;
    }
}
unit 1 {
    family inet {
        address 10.0.3.3/32;
    }
}

[edit]
lab@r3# show protocols
mpls {
    interface fe-0/0/0.0;
    interface fe-0/0/1.0;
}
bgp {
    advertise-inactive;
    group int {
        type internal;
        local-address 10.0.3.3;
        export nhs;
        neighbor 10.0.6.1 {
            family inet {
                unicast;
            }
            family inet-vpn {
                unicast;
            }
            cluster 10.0.3.3;
        }
        neighbor 10.0.6.2 {
            family inet {
                unicast;
            }
            family inet-vpn {
                unicast;
            }
            cluster 10.0.3.3;
        }
        neighbor 10.0.3.4;
```

```
        neighbor 10.0.3.5;
        neighbor 10.0.9.6;
        neighbor 10.0.9.7;
    }
}
ospf {
    area 0.0.0.1 {
        stub default-metric 10;
        interface fe-0/0/0.0;
        interface fe-0/0/1.0;
    }
    area 0.0.0.0 {
        interface so-0/2/0.100;
        interface at-0/1/0.0;
    }
    area 0.0.0.2 {
        nssa {
            default-lsa default-metric 10;
        }
        interface fe-0/0/3.0;
    }
}
ldp {
    interface fe-0/0/0.0;
    interface fe-0/0/1.0;
}
[edit]
lab@r3# show policy-options policy-statement send-lo0
term 1 {
    from {
        protocol direct;
        route-filter 10.0.3.3/32 exact;
    }
    then accept;
}
[edit]
lab@r3# show firewall
filter c3 {
    term 1 {
        from {
            protocol icmp;
```

```
        }
        then count vpnb-icmp;
    }
    term 2 {
        then accept;
    }
}
[edit]
lab@r3# show routing-instances
c3 {
    instance-type vrf;
    interface fe-0/0/2.0;
    interface vt-0/3/0.0;
    interface lo0.1;
    route-distinguisher 10.0.3.3:1;
    vrf-target target:65412:100;
    protocols {
        bgp {
            group c3 {
                type external;
                peer-as 65222;
                as-override;
                neighbor 172.16.0.14;
            }
        }
    }
}
```

The following items were deleted from r3's OSPF baseline configuration to complete the VPN case study:

```
[edit protocols bgp]
-    group ext {
-        import ebgp-in;
-        export ebgp-out;
-        neighbor 172.16.0.14 {
-            peer-as 65222;
-        }
-    }
```

Listing 7.4: VPN Case Study Configuration for *r4*

```
[edit]
lab@r4# show interfaces fe-0/0/0
```

```
vlan-tagging;
encapsulation vlan-ccc;
unit 0 {
    vlan-id 1;
    family inet {
        address 172.16.0.5/30;
    }
}
unit 700 {
    encapsulation vlan-ccc;
    vlan-id 700;
    family ccc;
}
[edit]
lab@r4# show interfaces fe-0/0/3
unit 0 {
    family inet {
        address 10.0.2.18/30;
    }
    family mpls;
}
[edit]
lab@r4# show interfaces so-0/1/1
encapsulation ppp;
unit 0 {
    family inet {
        address 10.0.2.10/30;
    }
    family mpls;
}
[edit]
lab@r4# show protocols mpls
interface so-0/1/1.0;
interface fe-0/0/3.0;

[edit]
lab@r4# show protocols ldp
interface fe-0/0/3.0;
interface so-0/1/1.0;
interface lo0.0;
```

```
[edit]
lab@r4# show protocols l2circuit
neighbor 10.0.9.6 {
    interface fe-0/0/0.700 {
        virtual-circuit-id 700;
    }
}
```

Note that the *c1* EBGP peer group definition from the OSPF baseline configuration is no longer needed at r4. It was not deleted because it caused no operational impact.

Listing 7.5: VPN Case Study Configuration for *r5*

```
[edit]
lab@r5# show interfaces fe-0/0/0
unit 0 {
    family inet {
        address 10.0.8.6/30;
    }
    family mpls;
}
[edit]
lab@r5# show interfaces fe-0/0/1
unit 0 {
    family inet {
        address 10.0.8.9/30;
    }
    family mpls;
}
[edit]
lab@r5# show interfaces so-0/1/0
encapsulation ppp;
unit 0 {
    family inet {
        address 10.0.2.9/30;
    }
    family mpls;
}
[edit]
lab@r5# show protocols mpls
interface fe-0/0/0.0;
```

```
interface so-0/1/0.0;
interface fe-0/0/1.0;

[edit]
lab@r5# show protocols ldp
interface fe-0/0/0.0;
interface fe-0/0/1.0;
interface so-0/1/0.0;
```

Listing 7.6: VPN Case Study Configuration for *r6*

```
[edit]
lab@r6# show interfaces fe-0/1/0
unit 0 {
    family inet {
        address 10.0.8.5/30;
    }
    family mpls;
}
[edit]
lab@r6# show interfaces fe-0/1/3
vlan-tagging;
encapsulation vlan-ccc;
unit 0 {
    vlan-id 1;
    family inet {
        address 172.16.0.9/30;
    }
}
unit 700 {
    encapsulation vlan-ccc;
    vlan-id 700;
    family ccc;
}
[edit]
lab@r6# show protocols mpls
interface fe-0/1/0.0;
[edit]
lab@r6# show protocols ldp
interface fe-0/1/0.0;
interface lo0.0;
```

```
[edit]
lab@r6# show protocols l2circuit
neighbor 10.0.3.4 {
    interface fe-0/1/3.700 {
        virtual-circuit-id 700;
    }
}
```

Note that the *c2* EBGP peer group definition from the OSPF baseline configuration is no longer needed at r6. It was not deleted because it resulted in no operational impact.

Listing 7.7: VPN Case Study Configuration for *r7*

```
[edit]
lab@r7# show interfaces fe-0/3/1
unit 0 {
    family inet {
        address 10.0.8.10/30;
    }
    family mpls;
}

[edit]
lab@r7# show interfaces fe-0/3/3
unit 0 {
    family inet {
        address 10.0.2.17/30;
    }
    family mpls;
}
[edit]
lab@r7# show protocols mpls
interface fe-0/3/1.0;
interface fe-0/3/3.0;

[edit]
lab@r7# show protocols ldp
interface fe-0/3/1.0;
interface fe-0/3/3.0;
```

Note that the *c1* EBGP peer group definition from the OSPF baseline configuration is no longer needed at r7. It was left in place because it caused no operational impact.

Spot the Issues: Review Questions

1. Using the Layer 2 VPN topology from the case study, you are finding that sometimes telnet sessions between C1 and C2 seem to "hang," as shown below. Do you have any idea what might be causing this problem?

```
lab@c1> telnet 220.220.0.1
Trying 220.220.0.1...
Connected to 220.220.0.1.
Escape character is '^]'.

c2 (ttyp1)

login: lab
Password:
Last login: Fri Jun 20 23:05:38 from 172.16.0.9

--- JUNOS 5.2R2.3 built 2002-03-23 02:44:36 UTC

lab@c2> show route 200.200/16

inet.0: 10 destinations, 10 routes (10 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

200.200.0.0/16     *[RIP/100] 00:04:51, metric 2
                    > to 192.168.32.1 via fe-0/0/0.700
200.200.1.0/24     *[RIP/100] 00:04:51, metric 2
                    > to 192.168.32.1 via fe-0/0/0.700

lab@c2> show configuration | no-more
<session hangs>
Ctrl-d
telnet> quit
Connection closed.
```

2. Can you spot the problem in the case study configuration of `r3`? The *`c3`* VRF contains all the expected routes, but VRF pings and traceroutes initiated at `r3` to C4 destinations fail.

```
[edit]
lab@r3# show routing-instances
c3 {
    instance-type vrf;
```

```
    interface fe-0/0/2.0;
    interface lo0.1;
    route-distinguisher 10.0.3.3:1;
    vrf-target target:65412:100;
    vrf-table-label;
    protocols {
        bgp {
            group c3 {
                type external;
                peer-as 65222;
                as-override;
                neighbor 172.16.0.14;
            }
        }
    }
}
[edit]
lab@r3# show interfaces fe-0/0/3
unit 0 {
    family inet {
        address 10.0.2.14/30;
    }
}

[edit]
lab@r3# show firewall
filter c3 {
    term 1 {
        from {
            protocol icmp;
        }
        then {
            count vpnb-icmp;
            next term;
        }
    }
    term 2 {
        then accept;
    }
}
```

3. In the case study topology, you observe that r2 is advertising C4 routes to r3, but r3 does not display the receipt of these routes, even when the all switch is used. Any ideas on what might cause the symptoms shown here?

```
lab@r2> show route advertising-protocol bgp 10.0.3.3 120.120/16 detail

c4.inet.0: 4 destinations, 4 routes (4 active, 0 holddown, 0 hidden)
* 120.120.0.0/16 (1 entry, 1 announced)
 BGP group int type Internal
     Route Distinguisher: 10.0.6.2:1
     VPN Label: 100004
     Nexthop: Self
     MED: 0
     Localpref: 100
     AS path: 65222 I
 Communities: target:64512:100

[edit]
lab@r3# run show route receive-protocol bgp 10.0.6.2 all

inet.0: 29 destinations, 31 routes (29 active, 0 holddown, 0 hidden)

inet.3: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)

c3.inet.0: 7 destinations, 7 routes (7 active, 0 holddown, 0 hidden)

mpls.0: 8 destinations, 8 routes (8 active, 0 holddown, 0 hidden)

bgp.l3vpn.0: 3 destinations, 3 routes (3 active, 0 holddown, 0 hidden)

[edit]
lab@r3#
```

4. r4 is configured for a Layer 3 VPN with OSPF-based PE-CE routing as in the topology shown earlier in Figure 7.5. You notice that r4 is not sending C2's routes, as learned from r6, to C1. Can you spot the problem in its configuration?

```
[edit]
lab@r4# show routing-instances
c1-ospf {
    instance-type vrf;
    interface fe-0/0/0.0;
```

```
    route-distinguisher 65412:1;
    vrf-import c1-import;
    vrf-export c1-export;
    protocols {
        ospf {
            domain-id 10.0.3.4;
            export bgp-ospf;
            area 0.0.0.0 {
                interface all;
            }
        }
    }
}

[edit]
lab@r4# show policy-options policy-statement bgp-ospf
term 1 {
    from protocol ospf;
    then accept;
}

[edit]
lab@r4# show policy-options policy-statement c1-import
term 1 {
    from {
        protocol bgp;
        community c1-c2-vpn;
    }
    then accept;
}
[edit]
lab@r4# show policy-options policy-statement c1-export
term 1 {
    from protocol ospf;
    then {
        community add c1-c2-vpn;
        community add domain;
        accept;
    }
}
```

```
term 2 {
    from {
        protocol direct;
        route-filter 172.16.0.4/30 exact;
    }
    then {
        community add c1-c2-vpn;
        accept;
    }
}
```

5. What changes are required to r5's configuration to make it function as a route reflector for the Layer 3 VPN deployed in the case study?

Spot the Issues: Answers to Review Questions

1. The issue here relates to the default MTU on the Fast Ethernet core interfaces in the network's core, and the fact that this MTU is not large enough to accommodate the overhead that results when the PE encapsulates the customer's VLAN tagged Ethernet frame inside of another Ethernet frame while also adding two MPLS labels and a 4-byte Martini control word.

   The default Fast Ethernet MPLS MTU setting is 1488, which is designed to accommodate the addition of up to three MPLS labels (12 bytes) without producing jumbo frames; the largest IP packet that can be generated by the CE device is therefore 1462 bytes, which yields 1442 bytes of transport and application layer data when a default IP header length is in effect. When the CE adds the 14 bytes of Ethernet encapsulation and the 4-byte VLAN tag, the total frame length becomes 1480 bytes. When these 1480 byte frames are received by the PE router, the addition of the 4-byte Martini control word, and the 4-byte VC label brings the total MPLS family protocol data unit size to the 1488-byte MTU limit. Unlike a Layer 3 VPN, Layer 2 VPNs can not perform fragmentation. In this example, the TCP-based telnet session appears to hang when the application generates IP packets that exceed the 1462-byte limit described earlier. This MTU problem did not occur in the chapter's body, or in the case study, because the CE devices for VPN A were configured with an IP MTU of 1462 bytes on their Layer 2 VPN interfaces. Another workaround is to increase the MTU on your Fast Ethernet core interfaces to enable "Jumbo" frames. By default, SONET interfaces support a device MTU of 4474, which is plenty large enough to support Layer 2 VPN customers that are Fast Ethernet attached with default MTUs in effect.

2. The problem relates to the use of the `vrf-table-label` option on a router whose core-facing interfaces are not point-to-point. The 5.6 release of JUNOS software does not support the `vrf-table-label` option when the PE's core interfaces are multi-point, such as in the case of Fast Ethernet. Given the specifics of the current test bed, you must use the `vt-interface` option (in conjunction with a TS PIC) to obtain IP II functionality at the egress PE.

3. The most likely cause for a control plane problem such as this is mismatched route targets. Note that routes with at least one matching RT are installed in the `l3vpn.bgp` table, as well as in any matching VRFs. When the received RT does not match at least one VRF, the route is not retained in the Adj-RIB-in, and therefore the `all` switch has no effect on the output of the `show route receive-protocol` command. When you suspect that mismatched RTs are the problem, you might want to temporarily enable the `keep-all` option (which is *on* by default for a route reflector), because this results in the router retaining routes that do not match any locally configured RTs:

```
[edit]
lab@r3# set protocols bgp keep all

[edit]
lab@r3# commit
commit complete
```

```
[edit]
lab@r3# run clear bgp neighbor 10.0.6.2 soft-inbound

[edit]
lab@r3# run show route receive-protocol bgp 10.0.6.2

inet.0: 29 destinations, 31 routes (29 active, 0 holddown, 0 hidden)

inet.3: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)

c3.inet.0: 7 destinations, 7 routes (7 active, 0 holddown, 0 hidden)

mpls.0: 8 destinations, 8 routes (8 active, 0 holddown, 0 hidden)

bgp.l3vpn.0: 6 destinations, 6 routes (6 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
  10.0.6.2:1:10.0.5.0/24
*                         10.0.6.2                     100        I
  10.0.6.2:1:10.0.6.2/32
*                         10.0.6.2                     100        I
  10.0.6.2:1:120.120.0.0/16
*                         10.0.6.2             0       100        65222 I
```

In this example, the problem is caused by the configuration of an erroneous RT at `r2`. The actual RT community can be viewed by including the `detail` switch.

4. The problem lies in the OSPF-based match condition in the first term of the *`bgp-ospf`* policy. Recall that the routes received from `r6` are learned though BGP, and that the default OSPF export policy does not redistribute BGP routes into OSPF. This type of routing instance export policy is not needed when the PE-CE link runs BGP because the default BGP export policy accepts active BGP routes. To correct the problem, change the firm term to match on the BGP protocol as shown next:

```
[edit]
lab@r4# show policy-options policy-statement bgp-ospf
term 1 {
    from protocol bgp;
    then accept;
}
```

5. You need to add a cluster ID to its *int* peer group and add support for the `inet-vpn` and `l2vpn` families:

```
[edit protocols bgp group int]
lab@r5# show
type internal;
local-address 10.0.3.5;
family inet {
    unicast;
}
family inet-vpn {
    unicast;
}
family l2vpn {
    unicast;
}
cluster 10.0.3.5;
neighbor 10.0.6.1;
neighbor 10.0.6.2;
neighbor 10.0.3.3;
neighbor 10.0.3.4;
neighbor 10.0.9.6;
neighbor 10.0.9.7;
```

A VPN route reflector does not require any target community or explicit VRF policy configuration because the `keep-all` option is enabled automatically when acting as a route reflector. Even though the route reflector is not actually in the VPN's forwarding path, it hides routes that cannot be resolved through the `inet.3` routing table. You therefore also need to add LDP support to `r5`'s at-0/2/1 interface, and ensure that `r1` thorough `r4` are appropriately configured with LDP support to allow the establishment of LDP signaled LSPs from `r5` to the loopback address of all PE routers `r1`, `r2`, and `r3`. Once so configured, you can remove the IBGP peering statements that currently provide a full IBGP mesh between `r1` through `r3`.