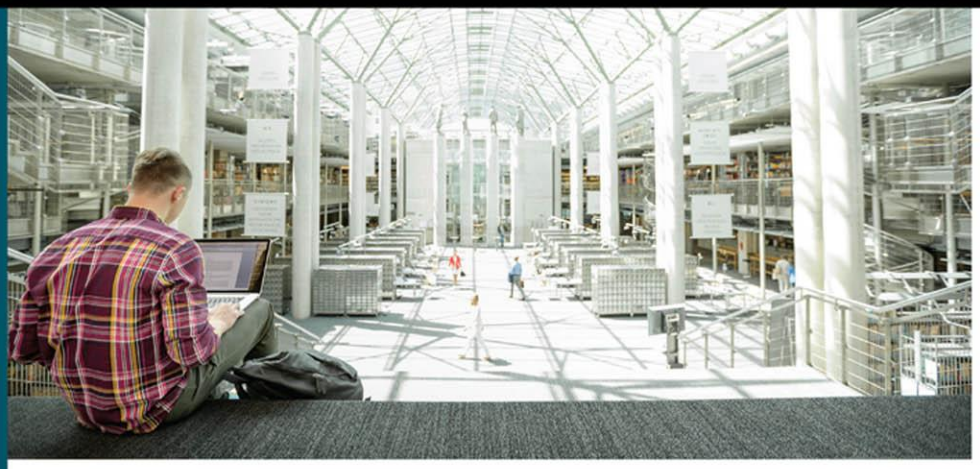

Troubleshooting BGP

A Practical Guide to Understanding and Troubleshooting BGP

Vinit Jain, CCIE No. 22854
Brad Edgeworth, CCIE No. 31574

ciscopress.com

Troubleshooting BGP

A Practical Guide to Understanding and Troubleshooting BGP

Vinit Jain, CCIE No. 22854

Brad Edgeworth, CCIE No. 31574

Cisco Press

800 East 96th Street

Indianapolis, Indiana 46240 USA

Troubleshooting BGP

Vinit Jain, Brad Edgeworth

Published by:

Cisco Press
800 East 96th Street
Indianapolis, IN 46240 USA

Printed in the United States of America

First Printing December 2016

Library of Congress Control Number: 2016958006

ISBN-13: 978-1-58714-464-6

ISBN-10: 1-58714-464-6

Warning and Disclaimer

This book is designed to provide information about troubleshooting BGP. Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied.

The information is provided on an “as is” basis. The authors, Cisco Press, and Cisco Systems, Inc. shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the discs or programs that may accompany it.

The opinions expressed in this book belong to the author and are not necessarily those of Cisco Systems, Inc.

Trademark Acknowledgments

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Cisco Press or Cisco Systems, Inc., cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Special Sales

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Feedback Information

At Cisco Press, our goal is to create in-depth technical books of the highest quality and value. Each book is crafted with care and precision, undergoing rigorous development that involves the unique expertise of members from the professional technical community.

Readers' feedback is a natural continuation of this process. If you have any comments regarding how we could improve the quality of this book, or otherwise alter it to better suit your needs, you can contact us through email at feedback@ciscopress.com. Please make sure to include the book title and ISBN in your message.

We greatly appreciate your assistance.

Editor-in-Chief: Mark Taub

Alliances Manager, Cisco Press: Ron Fligge

Product Line Manager: Brett Bartow

Managing Editor: Sandra Schroeder

Development Editor: Marianne Bartow

Senior Project Editor: Tonya Simpson

Copy Editor: Barbara Hacha

Technical Editors: Richard Furr, Ramiro Garza Rios

Editorial Assistant: Vanessa Evans

Cover Designer: Chuti Prasertsith

Composition: codeMantra

Indexer: Cheryl Lenser

Proofreader: Deepa Ramesh

Americas Headquarters
Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV
Amsterdam, The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at www.cisco.com/go/offices.

About the Authors

Vinit Jain, CCIE No. 22854 (R&S, SP, Security & DC), is a High Touch Technical Support (HTTS) engineer with Cisco providing support to premium customers of Cisco on complex routing technologies. Before joining Cisco, Vinit worked as a CCIE trainer and a network consultant. In addition to his expertise in networks, he has experience with software development, with which he began his career.

Vinit holds certifications for multiple vendors, such as Cisco, Microsoft, Sun Microsystems, VMware, and Oracle, and also is a Certified Ethical Hacker. Vinit is a speaker at Cisco Live and various other forums, including NANOG. Vinit pursued his graduation from Delhi University in Mathematics and earned his Masters in Information Technology from Kuvempu University in India. Vinit is married and is presently based out of RTP, North Carolina. Vinit can be found on Twitter @vinugenie.

Brad Edgeworth, CCIE No. 31574 (R&S & SP), has been with Cisco working as a systems engineer and a technical leader. Brad is a distinguished speaker at Cisco Live, where he has presented on multiple topics. Before joining Cisco, Brad worked as a network architect and consulted for various Fortune 500 companies. Brad's other certifications include Cisco Certified Design Professional (CCDP) and Microsoft Certified Systems Engineer (MCSE). Brad has been working in the IT field with an emphasis on enterprise and service provider environments from an architectural and operational perspective. Brad holds a Bachelor of Arts degree in Computer Systems Management from St. Edward's University in Austin, Texas. Brad can be found on Twitter @BradEdgeworth.

About the Technical Reviewers

Richard Furr, CCIE No. 9173 (R&S & SP), is a technical leader with the Cisco Technical Assistance Center (TAC). For the past 15 years, Richard has worked for Cisco TAC and high touch technical support (HTTS) organizations, supporting service providers and large enterprise environments with a focus on troubleshooting routing protocols, MPLS, IP Multicast, and QoS.

Ramiro Garza Rios, CCIE No. 15469 (R&S, SP, and Security), is a solutions integration architect with Cisco Advanced Services, where he plans, designs, implements, and optimizes IP NGN service provider networks. Before joining Cisco in 2005, he was a network consulting and presales engineer for a Cisco Gold Partner in Mexico, where he planned, designed, and implemented both enterprise and service provider networks.

Dedications

I would like to dedicate this book to my brother, Lalit, who is the inspiration and driving force behind everything I have achieved.

—Vinit

This book is dedicated to my family. Thank you both for letting me sleep in after a late-night writing session. To my wife, Tanya, "The Queen of Catan," thank you for bringing joy to my life. To my daughter, Teagan, listen to your mother. She is *almost* always right, and way better with her grammar than I am.

—Brad

Acknowledgments

Vinit Jain:

I would like to thank Russ White, Carlos Pignataro, Richard Furr, Pete Lumbis, Alejandro Eguiarte, and Brett Bartow for making this book possible.

I'd like to give special recognition to Alvaro Retana, Xander Thujis, and Steven Cheung for providing expert technical knowledge and advice on various topics, making this book more useful and close to real-life troubleshooting scenarios.

To our technical editors, Richard and Ramiro. In addition to your technical accuracy, your insight into the technologies needed versus and different perspective has kept the size of the book manageable.

Many people within Cisco have provided feedback and suggestions to make this a great book. Thanks to all who have helped in the process, especially to my managers, Ruwani Biggers and Chip Little, who have helped me with this adventurous and fun-filled project.

Brad Edgeworth:

A debt of gratitude goes toward my co-author, Vinit. Thank you for allowing me to work on this book with you, although we spent way too many nights on the phone at 1 a.m. Your knowledge and input made this a better book.

To our technical editors, Richard and Ramiro. Thank you for finding all of our mistakes. Not that we had many, but you still saved us a couple times. I won't tell if you won't.

A special thank you goes to Brett Bartow and the Cisco Press team. You are the "magicians" that make this book look as good as it does!

A special thanks goes to Craig Smith. "*You are so money, and you don't even know it*!" To my co-workers Rob, John, and Gregg. Yes, this means I probably will need to go on another "book signing tour." If anything breaks while I'm gone, order a queso and chips!

Contents at a Glance

Contents

Icons Used in This Book

Ethernet Circuit


Network


ASA Firewall


Layer 2 Switch


Multi-Layer Switch


Router


IOS XR


Nexus Device


Leaf Device


Spine Device


Workstation


Server


Security Server


DDOS Analyzer


Redistribution

Command Syntax Conventions

The conventions used to present command syntax in this book are the same conventions used in the IOS Command Reference. The Command Reference describes these conventions as follows:

- **Boldface** indicates commands and keywords that are entered literally as shown. In actual configuration examples and output (not general command syntax), boldface indicates commands that are manually input by the user (such as a **show** command).
- *Italic* indicates arguments for which you supply actual values.
- Vertical bars (|) separate alternative, mutually exclusive elements.
- Square brackets ([ ]) indicate an optional element.
- Braces ({ }) indicate a required choice.
- Braces within brackets ([{ }]) indicate a required choice within an optional element.

Foreword

The Internet has revolutionized the world by providing an unlimited supply of information to a user's fingertips in a matter of seconds, or connecting people halfway around the world with voice and video calls. More people are using the Internet in ways unimaginable when it was first conceived. The size of the Internet routing prohibits the use of almost any routing protocol except for BGP.

More and more organizations continue to deploy BGP across every vertical, segment, and corner of the Earth because there have been so many new features and technologies introduced to BGP. BGP is not only used by the service providers but has become a fundamental technology in enterprises and data centers.

As the leader of Cisco's technical services for more than 25 years, I have the benefit of working with the best network professionals in the industry. This book is written by Vinit and Brad, two "Network Rock Stars," who have been in my organization for years supporting multiple Cisco customers. Vinit continues to provide dedicated service to Cisco's premium customers, with an emphasis on network routing protocols.

With any network deployment, it becomes important to understand and learn how to troubleshoot the network and the technologies the network uses. Organizations strive to achieve five 9s (that is, 99.999%) availability of their network. This makes it more important that the network engineers attain the skills to troubleshoot such complex network environments. BGP has features that provide such a highly available network that some large hosting companies use only BGP. This book delivers a convenient reference for troubleshooting, deployment of best practices, and advanced protocol theory of BGP.

Joseph Pinto

SVP, Technical Services

Cisco, San Jose

Introduction

BGP is a standardized routing protocol that provides scalability, flexibility, and network stability for a variety of functions. Originally, BGP was developed to support large IP routing tables. It is the de facto protocol for routers connecting to the Internet, which provides connectivity to more than 600,000 networks and continues to grow.

Although BGP provides scalability and unique routing policy, the architecture can be intimidating or create complexity, too. Over the years, BGP has had significant increases in functionality and feature enhancements. BGP has expanded from being an Internet routing protocol to other aspects of the network, including the data center. BGP provides a scalable control plane for IPv6, MPLS VPNs (L2 and L3), Multicast, VPLS, and Ethernet VPN (EVPN).

Although most network engineers understand how to configure BGP, they lack the understanding to effectively troubleshoot BGP issues. This book is the single source for mastering techniques to troubleshoot all BGP issues for the following Cisco operating systems: Cisco IOS, IOS XR, and NX-OS. Bringing together content previously spread across multiple sources and Cisco Press titles, it covers updated various BGP design implementations found in blended service providers and enterprise environments and how to troubleshoot them.

Who Should Read This Book?

This book is for network engineers, architects, or consultants who want to learn more about BGP and learn how to troubleshoot all the various capabilities and features that it provides. Readers should have a fundamental understanding of IP routing.

How This Book Is Organized

Although this book could be read cover to cover, it is designed to be flexible and allow you to easily move between chapters and sections of chapters to cover just the material that you need more work with.

Part I, "BGP Fundamentals," provides an overview of BGP fundamentals—its various attributes and features.

- **Chapter 1, "BGP Fundamentals":** This chapter provides a brief overview of the BGP protocols, configuration, and some of the most commonly used features. Additional information is provided on how BGP's behavior is different between an internal and an external BGP neighbor.

Part II, "Common BGP Troubleshooting," provides the basic building blocks for troubleshooting BGP. These concepts are then carried over into other sections of the book.

- **Chapter 2, "Generic Troubleshooting Methodologies":** This chapter discusses the various basic troubleshooting methodologies and tools that are used for troubleshooting generic network problems. It also discusses how to approach a problem and how the problem can be replicated to identify the root cause.
- **Chapter 3, "Troubleshooting Peering Issues":** This chapter discusses the common issues seen with BGP peering. It provides detailed troubleshooting methods that can be used when investigating BGP peering issues, such as peer down and peer flapping. The chapter finally concludes by discussing dynamic BGP peering functionality.
- **Chapter 4, "Troubleshooting Route Advertisement and BGP Policies":** This chapter covers the BGP path selection mechanism and troubleshooting complex BGP path selection or missing route issues, which are commonly seen in BGP deployments.
- **Chapter 5, "Troubleshooting BGP Convergence":** This chapter examines various scenarios and conditions that could cause convergence issues. It provides a detailed explanation of how the BGP messages are formatted for the update and the complete update generation process on all the platforms.

Part III, "BGP Scalability Issues," explains how specific problems can arise in a scaled BGP network.

- **Chapter 6, "Troubleshooting Platform Issues Due to BGP":** This chapter examines various platform issues that are usually seen in a production environment caused by BGP. It examines conditions such as high CPU conditions, high memory utilization, and memory leak conditions caused by BGP.
- **Chapter 7, "Scaling BGP":** This chapter walks you through various features in BGP that can be implemented to scale the BGP environment. It explains in detail how to scale BGP using route reflectors and other advanced features, such as BGP diverse paths.
- **Chapter 8, "Troubleshooting BGP Edge Architectures":** This chapter discusses BGP multihoming, which is mostly deployed in enterprise networks. It also discusses problems faced with the multihomed deployments. This chapter also explains how to achieve load balancing with BGP and how to troubleshoot any problems faced with such deployments.

Part IV, "Securing BGP," discusses how BGP can be secured and how BGP can be used to prevent attacks in the network.

- **Chapter 9, "Securing BGP":** This chapter explains various features that help to secure Internet routing and thus prevent outages due to security breaches. It explains and differentiates between S-BGP and SO-BGP. The chapter then explains the SIDR solution using RPKI. Then we talk about DDoS attacks and mitigating them through RTBH and the BGP Flowspec feature.

Part V, "Multiprotocol BGP," discusses Multiprotocol BGP and how other address families provide connectivity outside traditional IP routing.

- **Chapter 10, "MPLS Layer 3 VPN (L3VPN)":** This chapter discusses and explains various BGP use cases of Multi-Protocol BGP deployment in Layer 3 MPLS VPN services and how to troubleshoot them. It also describes how to scale the network in the service provider environment for L3 VPN services.
- **Chapter 11, "BGP for MPLS L2VPN Services":** This chapter discusses and explains various BGP use cases of Multi-Protocol BGP deployment in Layer 2 MPLS VPN services and how to troubleshoot them. It talks about features such as BGP autodiscovery for VPLS and EVPN.
- **Chapter 12, "IPv6 BGP for Service Providers":** This chapter covers various IPv6 services for service providers, such as 6PE, 6VPE, and methods for how to troubleshoot the problems with such deployments.
- **Chapter 13, "VxLAN BGP EVPN":** This chapter covers implementation of BGP in data-center deployments by providing VxLAN Overlay using BGP. The chapter also explains how the VxLAN BGP EVPN control-plane learning mechanism works and how to troubleshoot various issues faced with the VxLAN EVPN feature.

Part VI, "High Availability," explains the techniques to increase the availability of BGP in the network.

- **Chapter 14, "BGP High Availability":** High availability is one of the primary concerns in almost all network deployments. This chapter discusses in detail the various high-availability features such as GR, NSR, BFD, and so on that can be implemented in BGP.

Part VII, "BGP: Looking Forward," provides an overview of the recent enhancements to BGP and insight into future applications of BGP.

- **Chapter 15, "Enhancements in BGP":** This chapter discusses new enhancements in BGP, such as BGP for Link-State distribution, BGP for tunnel setup, and EVPN.

Learning in a Lab Environment

This book may contain new features and functions that do not match your current environment. As with any new technology, it is best to test in advance of actual deployment of new features.

Cisco Virtual Internet Routing Lab (VIRL) provides a scalable, extensible network design and simulation environment. Many customers use VIRL for a variety of testing before deployment of features or verification of the techniques explained in this book. VIRL includes several Cisco Network Operating System virtual machines (IOSv, IOS-XRv, CSR1000v, NX-OSv, IOSvL2, and ASAv) and has the capability to integrate with third-party vendor virtual machines or external network devices. It includes many unique capabilities, such as live visualization, that provide the capability to create protocol diagrams in real-time from your running simulation. More information about VIRL can be found at http://virl.cisco.com.

Additional Reading

The authors tried to keep the size of the book manageable while providing only necessary information for the topics involved.

Some readers may require additional reference material around the design concepts using BGP and may find the following books a great supplementary resource for the topics in this book:

Edgeworth, Brad, Aaron Foss, and Ramiro Garza Rios. *IP Routing on Cisco IOS, IOS XE, and IOS XR*. Indianapolis: Cisco Press, 2014.

Halabi, Sam. *Internet Routing Architectures*. Indianapolis: Cisco Press, 2000.

White, Russ, Alvaro Retana, and Don Slice. *Optimal Routing Design*. Indianapolis: Cisco Press, 2005.

Doyle, Jeff. *Routing TCP/IP*, *Volume 2*, Second Edition. Indianapolis: Cisco Press, 2016.

Chapter 1

BGP Fundamentals

The following topics are covered in this chapter:

- BGP Messages and Inter-Router Communication
- Basic BGP Configuration for IOS, IOS XR, and NX-OS
- IBGP Rules
- EBGP Rules
- BGP Route Aggregation

A router's primary function is to move packets from one network to a different network. A router learns about unattached networks through static configuration or through dynamic routing protocols that distribute network topology information between routers. Routers try to select the best loop-free path in a network based on the destination network. Link flaps, router crashes, and other unexpected events could impact the best path, so the routers must exchange information with each other so that the network topology updates during these types of events.

Routing protocols are classified as either an Interior Gateway Protocol (IGP) or an Exterior Gateway Protocol (EGP), which indicates whether the protocol is designed for exchanging routes within an organization or between organizations. In IGP protocols, all routers use a common logic within the routing domain to find the shortest path to reach a destination. EGP protocols may require a unique routing policy for every external organization that it exchanges routes.

Border Gateway Protocol

RFC 1654 defines Border Gateway Protocol (BGP) as an EGP standardized path-vector routing protocol that provides scalability, flexibility, and network stability. When BGP was created, the primary design consideration was for IPv4 inter-organization

connectivity on public networks, such as the Internet, or private dedicated networks. BGP is the only protocol used to exchange networks on the Internet, which has more than 600,000 IPv4 routes and continues to grow. BGP does not advertise incremental updates or refresh network advertisements like OSPF or ISIS. BGP prefers stability within the network, because a link flap could result in route computation for thousands of routes.

From the perspective of BGP, an autonomous system (AS) is a collection of routers under a single organization's control, using one or more IGPs, and common metrics to route packets within the AS. If multiple IGPs or metrics are used within an AS, the AS must appear consistent to external ASs in routing policy. An IGP is not required within an AS, and could use BGP as the only routing protocol in it, too.

Autonomous System Numbers

Organizations requiring connectivity to the Internet must obtain an Autonomous System Number (ASN). ASNs were originally 2 bytes (16 bit) providing 65,535 ASNs. Due to exhaustion, RFC 4893 expands the ASN field to accommodate 4 bytes (32 bit). This allows for 4,294,967,295 unique ASNs, providing quite a leap from the original 65,535 ASNs.

Two blocks of private ASNs are available for any organization to use as long as they are never exchanged publicly on the Internet. ASNs 64,512–65,535 are private ASNs within the 16-bit ASN range, and 4,200,000,000–4,294,967,294 are private ASNs within the extended 32-bit range.

The Internet Assigned Numbers Authority (IANA) is responsible for assigning all public ASNs to ensure that they are globally unique. IANA requires the following items when requesting a public ASN:

- Proof of a publicly allocated network range
- Proof that Internet connectivity is provided through multiple connections
- Need for a unique route policy from your providers

In the event that an organization does not meet those guidelines, it should use the ASN provided by its service provider.

Note It is imperative that you use only the ASN assigned by IANA, the ASN assigned by your service provider, or private ASNs. Using another organization's ASN without permission could result in traffic loss and cause havoc on the Internet.

Path Attributes

BGP attaches path attributes (PA) associated with each network path. The PAs provide BGP with granularity and control of routing policies within BGP. The BGP prefix PAs are classified as follows:

- Well-known mandatory
- Well-known discretionary
- Optional transitive
- Optional nontransitive

Per RFC 4271, well-known attributes must be recognized by all BGP implementations. Well-known mandatory attributes must be included with every prefix advertisement, whereas well-known discretionary attributes may or may not be included with the prefix advertisement.

Optional attributes do not have to be recognized by all BGP implementations. Optional attributes can be set so that they are *transitive* and stay with the route advertisement from AS to AS. Other PAs are nontransitive and cannot be shared from AS to AS. In BGP, the Network Layer Reachability Information (NLRI) is the routing update that consists of the network prefix, prefix length, and any BGP PAs for that specific route.

Loop Prevention

BGP is a path vector routing protocol and does not contain a complete topology of the network-like link state routing protocols. BGP behaves similar to distance vector protocols to ensure a path is loop free.

The BGP attribute AS_PATH is a well-known mandatory attribute and includes a complete listing of all the ASNs that the prefix advertisement has traversed from its source AS. The AS_PATH is used as a loop prevention mechanism in the BGP protocol. If a BGP router receives a prefix advertisement with its AS listed in the AS_PATH, it discards the prefix because the router thinks the advertisement forms a loop.

Address Families

Originally, BGP was intended for routing of IPv4 prefixes between organizations, but RFC 2858 added Multi-Protocol BGP (MP-BGP) capability by adding extensions called address-family identifier (AFI). An address-family correlates to a specific network protocol, such as IPv4, IPv6, and the like, and additional granularity through a subsequent address-family identifier (SAFI), such as unicast and multicast. MBGP achieves this separation by using the BGP path attributes (PAs) MP_REACH_NLRI and MP_UNREACH_NLRI. These attributes are carried inside BGP update messages and are used to carry network reachability information for different address families.

Note Some network engineers refer to Multi-Protocol BGP as MP-BGP, and other network engineers use the term MBGP. Both terms are the same thing.

Network engineers and vendors continue to add functionality and feature enhancements to BGP. BGP now provides a scalable control plane for signaling for overlay technologies like MPLS VPNs, IPsec Security Associations, and Virtual Extensible LAN (VXLAN). These overlays can provide Layer 3 connectivity via MPLS L3VPNs, or Layer 2 connectivity via MPLS L2VPNs (L2VPN), such as Virtual Private LAN Service (VPLS) or Ethernet VPNs (EVPNs).

Every address-family maintains a separate database and configuration for each protocol (address-family + subaddress family) in BGP. This allows for a routing policy in one address-family to be different from a routing policy in a different address family even though the router uses the same BGP session to the other router. BGP includes an AFI and a SAFI with every route advertisement to differentiate between the AFI and SAFI databases. Table 1-1 provides a small list of common AFI and SAFIs.

Table 1-1 *Common BGP Address Families and Subaddress Families*

AFI	SAFI	Network Layer Information
1	1	IPv4 Unicast
1	2	IPv4 Multicast
1	4	IPv4 Unicast with MPLS Label
1	128	MPLS L3VPN IPv4
2	1	IPv6 Unicast
2	4	IPv6 Unicast with MPLS Label
2	128	MPLS L3VPN IPv6
25	65	Virtual Private LAN Service (VPLS) Virtual Private Wire Service (VPWS)
25	70	Ethernet VPN (EVPN)

BGP Sessions

A BGP session refers to the established adjacency between two BGP routers. BGP sessions are always point-to-point and are categorized into two types:

- **Internal BGP (IBGP):** Sessions established with an IBGP router that are in the same AS or participate in the same BGP confederation. IBGP sessions are considered more secure, and some of BGP's security measures are lowered in comparison to EBGP

sessions. IBGP prefixes are assigned an administrative distance (AD) of 200 upon installing into the router's routing information base (RIB).

- **External BPG (EBGP):** Sessions established with a BGP router that are in a different AS. EBGP prefixes are assigned an AD of 20 upon installing into the router's RIB.

Note Administrative distance (AD) is a rating of the trustworthiness of a routing information source. If a router learns about a route to a destination from more than one routing protocol, and they all have the same prefix length, AD is compared. The preference is given to the route with the lower AD.

Inter-Router Communication

BGP does not use hello packets to discover neighbors like IGP protocols and cannot discover neighbors dynamically. BGP was designed as an interautonomous routing protocol, implying that neighbor adjacencies should not change frequently and are coordinated. BGP neighbors are defined by an IP address.

BGP uses TCP port 179 to communicate with other routers. TCP allows for handling of fragmentation, sequencing, and reliability (acknowledgement and retransmission) of communication packets.

IGP protocols follow the physical topology because the sessions are formed with hellos that cannot cross network boundaries (that is, single hop only). BGP uses TCP, which is capable of crossing network boundaries (that is, multihop capable). While BGP can form neighbor adjacencies that are directly connected, it can also form adjacencies that are multiple hops away. Multihop sessions require that the router use an underlying route installed in the RIB (static or from any routing protocol) to establish the TCP session with the remote endpoint.

In Figure 1-1, R1 is able to establish a direct BGP session with R2. In addition, R2 is able to form a BGP session with R4, even though it passes through R3. R1 and R2 use a directly connected route to locate each other. R2 uses a static route to reach the 10.1.34.0/24 network, and R4 has a static route to reach the 10.1.23.0/24 network. R3 is unaware that R2 and R4 have established a BGP session, even though the packets flow through R3.

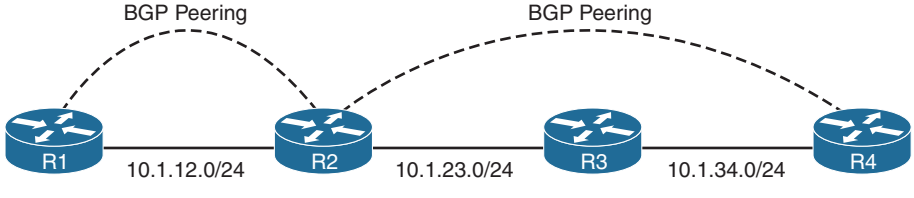

Figure 1-1 *BGP Direct and Multihop Sessions*

Note BGP neighbors connected via the same network use the ARP table to locate the Layer 2 address of the peer. Multihop BGP sessions require route table information for finding the IP address of the peer. It is common to have a static route or IGP running between IBGP neighbors for providing the topology path information for establishing the BGP TCP session. A default route is not sufficient to form a multihop BGP session.

BGP can be thought of as a control plane routing protocol or as an application, because it allows for the exchanging of routes with peers multiple hops away. BGP routers do not have to be in the data plane (path) to exchange prefixes, but all routers in the data path need to know all the routes that will be forwarded through them.

BGP Messages

BGP communication uses four message types, as shown in Table 1-2.

Table 1-2 *BGP Packet Types*

Type	Name	Functional Overview
1	OPEN	Sets up and establishes BGP adjacency
2	UPDATE	Advertises, updates, or withdraws routes
3	NOTIFICATION	Indicates an error condition to a BGP neighbor
4	KEEPALIVE	Ensures that BGP neighbors are still alive

OPEN

The OPEN message is used to establish a BGP adjacency. Both sides negotiate session capabilities before a BGP peering establishes. The OPEN message contains the BGP version number, ASN of the originating router, Hold Time, BGP Identifier, and other optional parameters that establish the session capabilities.

Hold Time

The Hold Time attribute sets the Hold Timer in seconds for each BGP neighbor. Upon receipt of an UPDATE or KEEPALIVE, the Hold Timer resets to the initial value. If the Hold Timer reaches zero, the BGP session is torn down, routes from that neighbor are removed, and an appropriate update route withdraw message is sent to other BGP neighbors for the impacted prefixes. The Hold Time is a heartbeat mechanism for BGP neighbors to ensure that the neighbor is healthy and alive.

When establishing a BGP session, the routers use the smaller Hold Time value contained in the two router's OPEN messages. The Hold Time value must be at least three seconds, or zero. For Cisco routers the default hold timer is 180 seconds.

BGP Identifier

The BGP Router-ID (RID) is a 32-bit unique number that identifies the BGP router in the advertised prefixes as the BGP Identifier. The RID can be used as a loop prevention mechanism for routers advertised within an autonomous system. The RID can be set manually or dynamically for BGP. A nonzero value must be set for routers to become neighbors. The dynamic RID allocation logic varies between the following operating systems.

- **IOS:** IOS nodes use the highest IP address of the any *up* loopback interfaces. If there is not an *up* loopback interface, then the highest IP address of any active *up* interfaces becomes the RID when the BGP process initializes.
- **IOS XR:** IOS XR nodes use the IP address of the lowest *up* loopback interface. If there is not any *up* loopback interfaces, then a value of zero (0.0.0.0) is used and prevents any BGP adjacencies from forming.
- **NX-OS:** NX-OS nodes use the IP address of the lowest *up* loopback interface. If there is not any *up* loopback interfaces, then the IP address of the lowest active *up* interface becomes the RID when the BGP process initializes.

Router-IDs typically represent an IPv4 address that resides on the router, such as a loopback address. Any IPv4 address can be used, including IP addresses not configured on the router. For IOS and IOS XR, the command **bgp router-id** *router-id* is used, and NX-OS uses the command **router-id** *router-id* under the BGP router configuration to statically assign the BGP RID. Upon changing the router-id, all BGP sessions reset and need to be reestablished.

Note Setting a static BGP RID is a best practice.

KEEPALIVE

BGP does not rely on the TCP connection state to ensure that the neighbors are still alive. Keepalive messages are exchanged every one-third of the Hold Timer agreed upon between the two BGP routers. Cisco devices have a default Hold Time of 180 seconds, so the default Keepalive interval is 60 seconds. If the Hold Time is set for zero, no Keepalive messages are sent between the BGP neighbors.

UPDATE

The Update message advertises any feasible routes, withdraws previously advertised routes, or can do both. The Update message includes the Network Layer Reachability Information (NLRI) that includes the prefix and associated BGP PAs when advertising prefixes. Withdrawn NLRIs include only the prefix. An UPDATE message can act as a Keepalive to reduce unnecessary traffic.

NOTIFICATION Message

A Notification message is sent when an error is detected with the BGP session, such as a hold timer expiring, neighbor capabilities change, or a BGP session reset is requested. This causes the BGP connection to close.

BGP Neighbor States

BGP forms a TCP session with neighbor routers called peers. BGP uses the Finite State Machine (FSM) to maintain a table of all BGP peers and their operational status. The BGP session may report in the following states:

- Idle
- Connect
- Active
- OpenSent
- OpenConfirm
- Established

Figure 1-2 displays the BGP FSM and the states in order of establishing a BGP session.

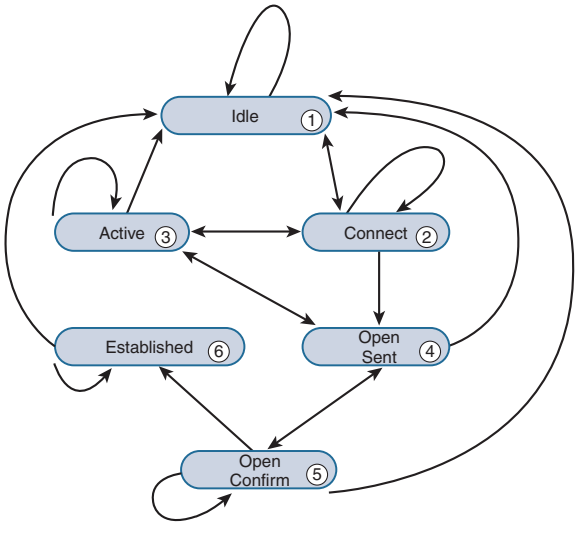

Figure 1-2 *BGP Finite State Machine*

Idle

This is the first stage of the BGP FSM. BGP detects a start event, tries to initiate a TCP connection to the BGP peer, and also listens for a new connect from a peer router.

If an error causes BGP to go back to the Idle state for a second time, the ConnectRetryTimer is set to 60 seconds and must decrement to zero before the connection is initiated again. Further failures to leave the Idle state result in the ConnectRetryTimer doubling in length from the previous time.

Connect

In this state, BGP initiates the TCP connection. If the 3-way TCP handshake completes, the established BGP Session BGP process resets the ConnectRetryTimer and sends the Open message to the neighbor, and then changes to the OpenSent State.

If the ConnectRetry timer depletes before this stage is complete, a new TCP connection is attempted, the ConnectRetry timer is reset, and the state is moved to Active. If any other input is received, the state is changed to Idle.

During this stage, the neighbor with the higher IP address manages the connection. The router initiating the request uses a dynamic source port, but the destination port is always 179.

Example 1-1 shows an established BGP session using the command **show tcp brief** to display the active TCP sessions between routers. Notice that the TCP source port is 179 and the destination port is 59884 on R1, and the ports are opposite on R2.

Example 1-1 *Established BGP Session*

```
RP/0/0/CPU0:R1# show tcp brief | exc "LISTEN|CLOSED"
   PCB      VRF-ID     Recv-Q Send-Q Local Address    Foreign Address     State
0x088bcbb8 0x60000000      0      0  10.1.12.1:179    10.1.12.2:59884     ESTAB
```

```
R2# show tcp brief
TCB        Local Address               Foreign Address             (state)
EF153B88  10.1.12.2.59884              10.1.12.1.179                ESTAB
```

Note Service providers consistently assign their customers the higher or lower IP address for their networks. This helps the service provider create proper instructions for access control lists (ACL) or firewall rules, or for troubleshooting them.

Active

In this state, BGP starts a new 3-way TCP handshake. If a connection is established, an Open message is sent, the Hold Timer is set to 4 minutes, and the state moves to OpenSent. If this attempt for TCP connection fails, the state moves back to the Connect state and resets the ConnectRetryTimer.

OpenSent

In this state, an Open message has been sent from the originating router and is awaiting an Open message from the other router. After the originating router receives the OPEN message from the other router, both OPEN messages are checked for errors. The following items are being compared:

- BGP Versions must match.
- The source IP address of the OPEN message must match the IP address that is configured for the neighbor.
- The AS number in the OPEN message must match what is configured for the neighbor.
- BGP Identifiers (RID) must be unique. If a RID does not exist, this condition is not met.
- Security Parameters (Password, TTL, and the like).

If the Open messages do not have any errors, the Hold Time is negotiated (using the lower value), and a KEEPALIVE message is sent (assuming the value is not set to zero). The connection state is then moved to OpenConfirm. If an error is found in the OPEN message, a Notification message is sent, and the state is moved back to Idle.

If TCP receives a disconnect message, BGP closes the connection, resets the ConnectRetryTimer, and sets the state to Active. Any other input in this process results in the state moving to Idle.

OpenConfirm

In this state, BGP waits for a Keepalive or Notification message. Upon receipt of a neighbor's Keepalive, the state is moved to Established. If the hold timer expires, a stop event occurs, or a Notification message is received, and the state is moved to Idle.

Established

In this state, the BGP session is established. BGP neighbors exchange routes via Update messages. As Update and Keepalive messages are received, the Hold Timer is reset. If the Hold Timer expires, an error is detected and BGP moves the neighbor back to the Idle state.

Basic BGP Configuration

When configuring BGP, it is best to think of the configuration from a modular perspective. BGP router configuration requires the following components:

- **BGP Session Parameters:** BGP session parameters provide settings that involve establishing communication to the remote BGP neighbor. Session settings include the ASN of the BGP peer, authentication, and keepalive timers.
- **Address-Family Initialization:** The address-family is initialized under the BGP router configuration mode. Networks advertisement and summarization occur within the address-family.
- **Activate the Address-Family on the BGP Peer:** Activate the address-family on the BGP peer. For a session to initiate, one address-family for that neighbor must be activated. The router's IP address is added to the neighbor table, and BGP attempts to establish a BGP session or accepts a BGP session initiated from the peer router.

For the remainder of this chapter, the BGP context is directed toward IPv4 routing. Other address families are throughout the book.

IOS

The steps for configuring BGP on an IOS router are as follows:

Step 1. Create the BGP Routing Process. Initialize the BGP process with the global command **router bgp** *as-number*.

Step 2. Identify the BGP Neighbor's IP address and Autonomous System Number. Identify the BGP neighbor's IP address and autonomous system number with the BGP router configuration command **neighbor** *ip-address* **remote-as** *as-number*.

Note IOS activates the IPv4 address-family by default. This can simplify the configuration in an IPv4 environment because Steps 3 and 4 are optional, but may cause confusion when working with other address families. The BGP router configuration command **no bgp default ip4-unicast** disables the automatic activation of the IPv4 AFI so that Steps 3 and 4 are required.

Step 3. Initialize the address-family with the BGP router configuration command **address-family** *afi safi*.

Step 4. Activate the address-family for the BGP neighbor with the BGP address-family configuration command **neighbor** *ip-address* **activate**.

Note On IOS routers, the default address-family modifier for the IPv4 and IPv6 address families is unicast and is optional. The address-family modifier is required on IOS XR nodes.

Example 1-2 demonstrates how to configure R1 and R2 using the IOS default and optional IPv4 AFI modifier CLI syntax. R1 is configured using the default IPv4 address-family enabled, and R2 disables IOS's default IPv4 address-family and manually activates it for the specific neighbor 10.1.12.1.

Example 1-2 *IOS Basic BGP Configuration*

```
R1 (Default IPv4 Address-Family Enabled)
router bgp 65100
 neighbor 10.1.12.2 remote-as 65100
```

```
R2 (Default IPv4 Address-Family Disabled)
router bgp 65100
 no bgp default ipv4-unicast
 neighbor 10.1.12.1 remote-as 65100
 !
 address-family ipv4
  neighbor 10.1.12.1 activate
 exit-address-family
```

IOS XR

The steps for configuring BGP on an IOS XR router are as follows:

Step 1. Create the BGP routing process. Initialize the BGP process with the global configuration command **router bgp** *as-number*.

Step 2. Initialize the address-family with the BGP router configuration command **address-family** *afi safi* so it can be associated to a BGP neighbor.

Step 3. Identify the BGP neighbor's IP address with the BGP router configuration command **neighbor** *ip-address*.

Step 4. Identify the BGP neighbor's autonomous system number with the BGP neighbor configuration command **remote-as** *as-number*.

Step 5. Activate the address-family for the BGP neighbor with the BGP neighbor configuration command **address-family** *afi safi*.

Step 6. Associate a route policy for EBGP Peers. IOS XR requires a routing policy to be associated to an EBGP peer as a security measure to ensure that routes are not accidentally accepted or advertised. If a route policy is not configured in

the appropriate address-family, then NLRIs are discarded upon receipt and no NLRIs are advertised to EBGP peers.

An inbound and outbound route policy is configured with the command **route-policy** *policy-name* {**in** | **out**} under the BGP neighbor address-family configuration.

Note IOS XR nodes do not establish a BGP session if the RID is set to zero, because the dynamic RID allocation did not find any *up* loopback interfaces. The RID needs to be set manually with the BGP router configuration command **bgp router-id**.

Example 1-3 displays the BGP configuration for R1 if it was running IOS XR. The RID is set on R1 because that router does not have any loopback interfaces.

Example 1-3 *IOS XR BGP Configuration*

```
IOS XR
router bgp 65100
 bgp router-id 192.168.1.1
 address-family ipv4 unicast
 !
 neighbor 10.1.12.2
  remote-as 65100
  address-family ipv4 unicast
```

NX-OS

The steps for configuring BGP on an NX-OS device are as follows:

Step 1. Create the BGP routing process. Initialize the BGP process with the global configuration command **router bgp** *as-number*.

Step 2. Initialize the address-family with the BGP router configuration command **address-family** *afi safi* so it can be associated to a BGP neighbor.

Step 3. Identify the BGP neighbor's IP address and autonomous system number with the BGP router configuration command **neighbor** *ip-address* **remote-as** *as-number*.

Step 4. Activate the address-family for the BGP neighbor with the BGP neighbor configuration command **address-family** *afi safi*.

Example 1-4 displays the BGP configuration for R1 if it was running NX-OS.

Example 1-4 *NX-OS BGP Configuration*

```
NX-OS
router bgp 65100
 address-family ipv4 unicast
 neighbor 10.1.12.2 remote-as 65100
  address-family ipv4 unicast
```

Verification of BGP Sessions

The BGP session is verified with the command **show bgp** *afi safi* **summary** on IOS, IOS XR, and NX-OS devices. Example 1-5 displays the IPv4 BGP unicast summary. Notice that the BGP RID and table versions are the first components shown. The Up/Down column reflects that the BGP session is up for over 5 minutes.

Example 1-5 *BGP IPv4 Session Summary Verification*

```
R1-IOS# show bgp ipv4 unicast summary
BGP router identifier 192.168.2.2, local AS number 65100
BGP table version is 1, main routing table version 1

Neighbor        V     AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
10.1.12.2       4  65100       8       9        1    0    0 00:05:23        0
```

```
RP/0/0/CPU0:R1-XR# show bgp ipv4 unicast summary
! Output omitted for brevity
BGP router identifier 192.168.1.1, local AS number 65100
BGP main routing table version 4

Process      RcvTblVer   bRIB/RIB   LabelVer  ImportVer  SendTblVer  StandbyVer
Speaker              4          4          4          4           4           4

Neighbor        Spk    AS MsgRcvd MsgSent   TblVer  InQ OutQ  Up/Down  St/PfxRcd
10.1.12.2         0 65100       8       7        4    0    0 00:05:23          0
```

```
R1-NXOS# show bgp ipv4 unicast summary
! Output omitted for brevity
BGP router identifier 192.168.1.1, local AS number 65100
BGP table version is 5, IPv4 Unicast config peers 2, capable peers 1
Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ  Up/Down    State/PfxRcd
10.1.12.2       4 65100      32      37        5    0    0  00:05:24   0
```

Table 1-3 explains the fields of output when displaying the BGP Table.

Table 1-3 *BGP Summary Fields*

Field	Description
Neighbor	IP address of the BGP peer
V	BGP Version spoken by BGP peer (IOS and NX-OS only)
AS	Autonomous system number of BGP peer
MsgRcvd	Count of messages received from the BGP peer
MsgSent	Count of messages sent to the BGP peer
TblVer	Last version of the BGP database sent to the peer
InQ	Number of messages queued to be processed from the peer
OutQ	Number of messages queued to be sent to the peer
Up/Down	Length of time the BGP session is established, or the current status if the session is not in established state
State/PfxRcd	Current state of BGP peer or the number of prefixes received from the peer

Note Earlier commands like **show ip bgp summary** came out before MBGP and do not provide a structure for the current multiprotocol capabilities within BGP. Using the AFI and SAFI syntax ensures consistency for the commands regardless of information exchanged by BGP.

BGP neighbor session state, timers, and other essential peering information is shown with the command **show bgp** *afi safi* **neighbors** *ip-address*, as shown in Example 1-6.

Example 1-6 *BGP IPv4 Neighbor Output*

```
R2# show bgp ipv4 unicast neighbors 10.1.12.1
! Output ommitted for brevity

! The first section provides the neighbor's IP address, remote-as, indicates if
! the neighbor is 'internal' or 'external', the neighbor's BGP version, RID,
! session state, and timers.
BGP neighbor is 10.1.12.1, remote AS100, internal link
  BGP version 4, remote router ID 192.168.1.1
  BGP state = Established, up for 00:01:04
  Last read 00:00:10, last write 00:00:09, hold is 180, keepalive is 60 seconds
  Neighbor sessions:
    1 active, is not multisession capable (disabled)
```

```
! This second section indicates the capabilities of the BGP neighbor and
! address-families configured on the neighbor.
  Neighbor capabilities:
    Route refresh: advertised and received(new)
    Four-octets ASN Capability: advertised and received
    Address family IPv4 Unicast: advertised and received
    Enhanced Refresh Capability: advertised
    Multisession Capability:
    Stateful switchover support enabled: NO for session 1
  Message statistics:
    InQ depth is 0
    OutQ depth is 0
! This section provides a list of the BGP packet types that have been received
! or sent to the neighbor router.
                         Sent       Rcvd
    Opens:                  1          1
    Notifications:          0          0
    Updates:                0          0
    Keepalives:             2          2
    Route Refresh:          0          0
    Total:                  4          3
  Default minimum time between advertisement runs is 0 seconds
! This section provides the BGP table version of the IPv4 Unicast address-
! family. The table version is not a 1-to-1 correlation with routes as multiple
! route change can occur during a revision change. Notice the Prefix Activity
! columns in this section.
For address family: IPv4 Unicast
  Session: 10.1.12.1
  BGP table version 1, neighbor version 1/0
  Output queue size : 0
  Index 1, Advertise bit 0
                                Sent       Rcvd
  Prefix activity:              ----       ----
    Prefixes Current:              0          0
    Prefixes Total:                0          0
    Implicit Withdraw:             0          0
    Explicit Withdraw:             0          0
    Used as bestpath:            n/a          0
    Used as multipath:           n/a          0

                                  Outbound    Inbound
  Local Policy Denied Prefixes:   --------    -------
    Total:                               0          0
  Number of NLRIs in the update sent: max 0, min 0
```

```
! This section indicates that a valid route exists in the RIB to the BGP peer IP
! address, provides the number of times that the connection has established and
! time dropped, since the last reset, the reason for the reset, if path-mtu-
! discovery is enabled, and ports used for the BGP session.
  Address tracking is enabled, the RIB does have a route to 10.1.12.1
  Connections established 2; dropped 1
  Last reset 00:01:40, due to Peer closed the session
  Transport(tcp) path-mtu-discovery is enabled
Connection state is ESTAB, I/O status: 1, unread input bytes: 0
Mininum incoming TTL 0, Outgoing TTL 255
Local host: 10.1.12.2, Local port: 179
Foreign host: 10.1.12.1, Foreign port: 56824
```

Prefix Advertisement

BGP uses three tables for maintaining the network prefix and path attributes (PA) for a route. The BGP tables are as follows:

- **Adj-RIB-in:** Contains the NLRIs in original form before inbound route policies are processed. The table is purged after all route policies are processed to save memory.
- **Loc-RIB:** Contains all the NLRIs that originated locally or were received from other BGP peers. After NLRIs pass the validity and next-hop reachability check, the BGP best path algorithm selects the best NLRI for a specific prefix. The Loc-RIB table is the table used for presenting routes to the ip routing table.
- **Adj-RIB-out:** Contains the NLRIs after outbound route policies have processed.

BGP **network** statements do not enable BGP for a specific interface. Instead they identify a specific network prefix to be installed into the BGP table, known as the *Loc-RIB table*.

After configuring a BGP network statement, the BGP process searches the global RIB for an exact network prefix match. The network prefix can be a connected network, secondary connected network, or any route from a routing protocol. After verifying that the network statement matches a prefix in the global RIB, the prefix installs into the BGP Loc-RIB table. As the BGP prefix installs into the Loc-RIB, the following BGP PA are set depending on the RIB prefix type:

- **Connected Network:** The next-hop BGP attribute is set to 0.0.0.0, the origin attribute is set to *i* (IGP), and the BGP weight is set to 32,768.
- **Static Route or Routing Protocol:** The next-hop BGP attribute is set to the next-hop IP address in the RIB, the origin attribute is set to *i* (IGP), the BGP weight is set to 32,768; and the MED is set to the IGP metric.

The network statement resides under the appropriate address-family within the BGP router configuration. The command **network** *network* **mask** *subnet-mask* [**route-map** *route-map-name*] is used for advertising IPv4 networks on IOS and NX-OS devices.

NX-OS devices also support prefix-length notation with the command **network** *network /prefix-length* [**route-map** *route-map-name*]. IOS XR routers use the command **network** *network/prefix-length* [**route-policy** *route-policy-name*] for installing routes into the BGP table. The optional **route-map** or **route-policy** parameter provides a method to set specific BGP PAs when the prefix installs into the Loc-RIB.

The command **show bgp** *afi safi* displays the contents of the BGP database (Loc-RIB) on IOS, IOS XR, and NX-OS devices. Every entry in the BGP Loc-RIB table contains at least one route, but could contain multiple routes for the same network prefix.

Note By default, BGP advertises only the best path to other BGP peers regardless of the number of routes (NLRIs) in the BGP Loc-RIB. The BGP best path executes individually per address-family. The best path selection of one address-family cannot impact the best path calculation on a different address-family.

Example 1-7 displays the BGP table for IOS, IOS XR, and NX-OS. The BGP table contains received routes and locally generated routes.

Example 1-7 *Display of BGP Table*

```
R1-IOS# show bgp ipv4 unicast
BGP table version is 5, local router ID is 192.168.1.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale, m multipath, b backup-path, f RT-Filter,
              x best-external, a additional-path, c RIB-compressed,
Origin codes: i - IGP, e - EGP, ? - incomplete
RPKI validation codes: V valid, I invalid, N Not found

     Network          Next Hop            Metric LocPrf Weight Path
 *>  192.168.1.1/32   0.0.0.0                  0         32768 i
 *   192.168.2.2/32   10.1.13.3                             0 65300 65200 i
 *>                   10.1.12.2                0             0 65200 i
 *>  192.168.3.3/32   10.1.13.3                             0 65300 i
 *                    10.1.12.2                             0 65200 65300 i
```

```
RP/0/0/CPU0:R2-XR# show bgp ipv4 unicast
! Output omitted for brevity
BGP router identifier 192.168.2.2, local AS number 65200
Status codes: s suppressed, d damped, h history, * valid, > best
              i - internal, r RIB-failure, S stale, N Nexthop-discard
Origin codes: i - IGP, e - EGP, ? - incomplete
   Network            Next Hop            Metric LocPrf Weight Path
*> 192.168.1.1/32     10.1.12.1                0             0 65100 i
*                     10.1.23.3                             0 65300 65100 i
```

```
*> 192.168.2.2/32     0.0.0.0                  0         32768 i
*  192.168.3.3/32     10.1.12.1                              0 65100 65300 i
*>                    10.1.23.3                              0 65300 i
Processed 3 prefixes, 5 paths
```

```
R3-NXOS# show bgp ipv4 unicast
! Output omitted for brevity
Status: s-suppressed, x-deleted, S-stale, d-dampened, h-history, *-valid, >-best
Path type: i-internal, e-external, c-confed, l-local, a-aggregate, r-redist,
  I-injected
Origin codes: i - IGP, e - EGP, ? - incomplete, | - multipath, & - backup

   Network            Next Hop          Metric     LocPrf     Weight Path
*>e192.168.1.1/32     10.1.13.1              0                     0 65100 i
* e                   10.1.23.2                                    0 65200 65100 i
*>e192.168.2.2/32     10.1.23.2              0                     0 65200 i
* e                   10.1.13.1                                    0 65100 65200 i
*>l192.168.3.3/32     0.0.0.0                           100      32768 i
```

Note NX-OS devices place *e* beside external learned BGP routes and *l* beside locally advertised BGP routes. IOS and IOS XR devices do not have this behavior.

Table 1-4 explains the fields of output when displaying the BGP table.

Table 1-4 *BGP Table Fields*

Field	Description
Network	List of the network prefixes installed in BGP. If multiple NLRIs exist for the same prefix, only the first prefix is identified, and others leave a blank space. Valid NLRIs are indicated by the *. The NLRI selected as the best path is indicated by an angle bracket (>).
Next Hop	*Next Hop*: A well-known mandatory BGP path attribute that defines the IP address for the next-hop for that specific NLRI.
Metric	*Multiple-Exit Discriminator (MED)*: An optional nontransitive BGP path attribute used in BGP algorithm for that specific NLRI.
LocPrf	*Local Preference*: A well-known discretionary BGP path attribute used in the BGP best path algorithm for that specific NLRI.
Weight	Locally significant Cisco defined attribute used in the BGP best path algorithm for that specific NLRI.

Field	Description
Path and Origin	*AS_PATH*: A well-known mandatory BGP path attribute used for loop prevention and in the BGP best path algorithm for that specific NLRI. *Origin*: A well-known mandatory BGP path attribute used in the BGP best path algorithm. A value of *i* represents an IGP, *e* for EGP, and *?* for a route that was redistributed into BGP.

BGP Best-Path Calculation

In BGP, route advertisements consist of the Network Layer Reachability Information (NLRI) and the path attributes (PAs). The NLRI composes the network prefix and prefix-length, and the BGP attributes such as AS-Path, Origin, and the like are stored in the path attributes. A BGP route may contain multiple paths to the same destination network. Every path's attributes impact the desirability of the route when a router selects the best path. A BGP router advertises only the best path to the neighboring routers.

Inside the BGP Loc-RIB table, all the routes and their path attributes are maintained with the best path calculated. The best path is then installed in the RIB of the router. In the event the best path is no longer available, the router can use the existing paths to quickly identify a new best path. BGP recalculates the best path for a prefix upon four possible events:

- BGP next-hop reachability change
- Failure of an interface connected to an EBGP peer
- Redistribution change
- Reception of new paths for a route

The BGP best path selection algorithm influences how traffic enters or leaves an autonomous system (AS). BGP does not use metrics to identify the best path in a network. BGP uses path attributes to identify its best path.

Some router configurations modify the BGP attributes to influence inbound traffic, outbound traffic, or inbound and outbound traffic depending on the network design requirements. BGP path attributes can be modified upon receipt or advertisement to influence routing in the local AS or neighboring AS. A basic rule for traffic engineering with BGP is that modifications in outbound routing policies influence inbound traffic, and modifications to inbound routing policies influence outbound traffic.

BGP installs the first received path as the best path automatically. When additional paths are received, the newer paths are compared against the current best path. If there is a tie, then processing continues onto the next step, until a best path winner is identified.

The following list provides the attributes that the BGP best path algorithm uses for the best route selection process. These attributes are processed in the order listed:

1. Weight
2. Local Preference
3. Local originated (network statement, redistribution, aggregation)
4. AIGP
5. Shortest-AS Path
6. Origin Type
7. Lowest MED
8. EBGP over IBGP
9. Lowest IGP Next-Hop
10. If both paths are external (EBGP), prefer the first (oldest)
11. Prefer the route that comes from the BGP peer with the lower RID
12. Prefer the route with the minimum cluster list length
13. Prefer the path that comes from the lowest neighbor address

The best path algorithm can be used to manipulate network traffic patterns for a specific route by modifying various path attributes on BGP routers. Changing of BGP PA can influence traffic flow into, out of, and around an AS.

BGP supports three types of equal cost multipath (ECMP): EBGP multipath, IBGP multipath, or eIBGP multipath. EBGP multipath requires that the weight, local preference, AS-Path length, AS-Path content, Origin, and MED match for a second route to install into the RIB. Chapter 8, "Troubleshooting BGP Edge Architectures," explains BGP ECMP in more detail.

Route Filtering and Manipulation

Route filtering is a method for selectively identifying routes that are advertised or received from neighbor routers. Route filtering may be used to manipulate traffic flows, reduce memory utilization, or to improve security. For example, it is common for ISPs to deploy route filters on BGP peerings to customers. Ensuring that only the customer routes are allowed over the peering link prevents the customer from accidentally becoming a transit AS on the Internet.

Filtering of routes within BGP is accomplished with filter-lists, prefix-lists, or route-maps on IOS and NX-OS devices. IOS XR uses route policies for filtering of routes. Route-filtering is explained in more detail in Chapter 4, "Troubleshooting Route Advertisement and BGP Policies."

Depending on the change to the BGP route manipulation technique, the BGP session may need to be refreshed to take effect. BGP supports two methods of clearing a BGP session: The first method is a hard reset, which tears down the BGP session, removes BGP routes from the peer, and is the most disruptive. The second method is a soft reset, which invalidates the BGP cache and requests a full advertisement from its BGP peer.

IOS and NX-OS devices initiate a hard reset with the command **clear ip bgp** *ip-address* [**soft**], and the command **clear bgp** *ip-address* [**graceful**] is used on IOS XR nodes. Soft reset on IOS and NX-OS devices use the optional **soft** keyword, whereas IOS XR nodes use the optional **graceful** keyword. Sessions can be cleared with all BGP neighbors by using an asterisk * in lieu of the peer's IP address.

When a BGP policy changes, the BGP table must be processed again so that the neighbors can be notified accordingly. Routes received by a BGP peer must be processed again. If the BGP session supports route refresh capability, then the peer readvertises (refreshes) the prefixes to the requesting router, allowing for the inbound policy to process using the new policy changes. The route refresh capability is negotiated for each address-family when the session is established.

Performing a soft reset on sessions that support route refresh capability actually initiates a route refresh. Soft resets can be performed for a specific address-family with the command **clear bgp** *address-family address-family modifier ip-address* **soft** [**in | out**]. Soft resets reduce the amount of routes that must be exchanged if multiple address families are configured with a single BGP peer. Changes to the outbound routing policies use the optional **out** keyword, and changes to inbound routing policies use the optional **in** keyword.

Older IOS versions that do not support route refresh capability require the usage of inbound soft reconfiguration so that updates to inbound route policies can be applied without performing a hard reset. Inbound soft reconfiguration does not purge the Adj-RIB-In table after routes process into the Loc-RIB table. The Adj-RIB-In maintains only the raw unedited routes (NLRIs) that were received from the neighbors and thereby allows the inbound route policies to be processed again.

Enabling this feature can consume a significant amount of memory because the Adj-RIB-In table stays in memory. Inbound soft reconfiguration uses the address-family command **neighbor** *ip-address* **soft-reconfiguration inbound** for IOS nodes. IOS XR and NX-OS devices use the neighbor specific address-family command **soft-reconfiguration inbound**.

IBGP

The need for BGP within an AS typically occurs when the multiple routing policies exist, or when transit connectivity is provided between autonomous systems. In Figure 1-3, AS65200 provides transit connectivity to AS65100 and AS65300. AS65100 connects at R2, and AS65300 connects at R4.

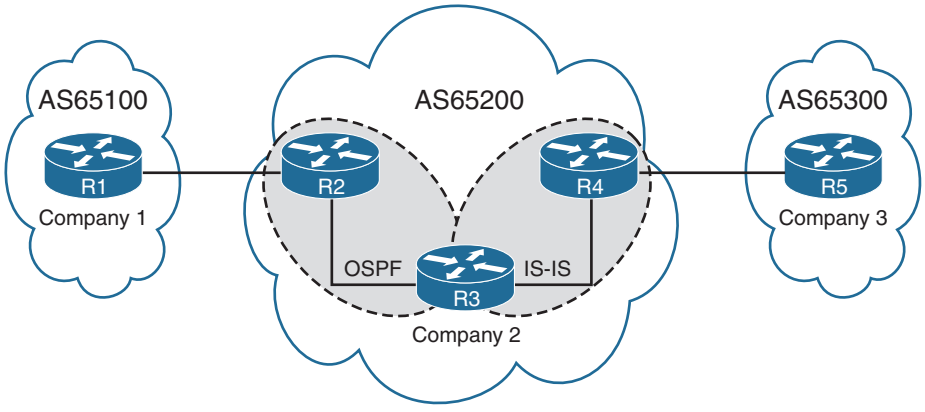

Figure 1-3 *AS65200 Provides Transit Connectivity*

R2 could form a BGP session directly with R4, but R3 would not know where to route traffic from AS65100 or AS65300 when traffic from either AS reaches R3, as shown in Figure 1-4, because R3 would not have the appropriate route forwarding information for the destination traffic.

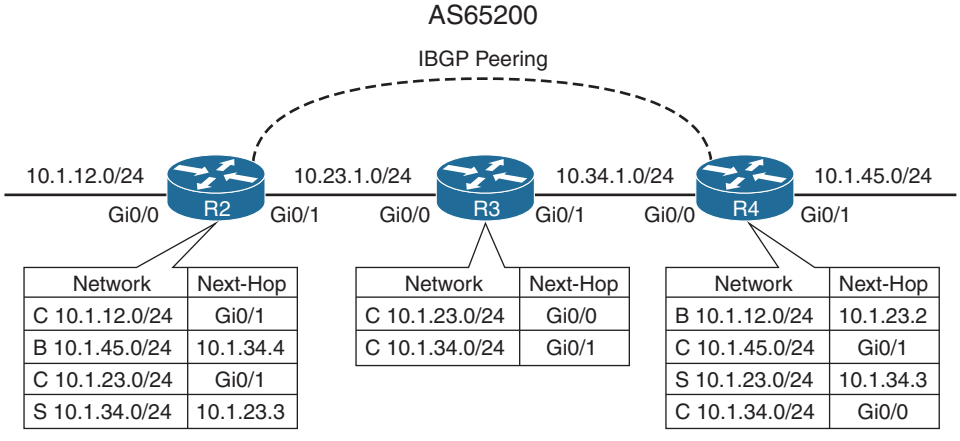

Figure 1-4 *Transit Devices Need Full Routing Table*

Advertising the full BGP table into an IGP is not a viable solution for the following reasons:

- **Scalability:** The Internet at the time of this writing has 600,000+ IPv4 networks and continues to increase in size. IGPs cannot scale to that level of routes.
- **Custom Routing:** Link state protocols and distance vector routing protocols use metric as the primary method for route selection. IGP protocols always use this routing pattern for path selection. BGP uses multiple steps to identify the best path and allows for BGP path attributes to manipulate the path for a specific prefix (NLRI). The path could be longer, which would normally be deemed suboptimal from an IGP protocol's perspective.

- **Path Attributes:** All the BGP path attributes cannot be maintained within IGP protocols. Only BGP is capable of maintaining the path attribute as the prefix is advertised from one edge of the AS to the other edge.

IBGP Full Mesh Requirement

It was explained earlier in this chapter how BGP uses the AS_PATH as a loop detection and prevention mechanism because the ASN is prepended when advertising to an EBGP neighbor. IBGP peers do not prepend their ASN to the AS_PATH, because the NLRIs would fail the validity check and would not install the prefix into the IP routing table.

No other method exists to detect loops with IBGP sessions, and RFC 4271 prohibits the advertisement of a NLRI received from an IBGP peer to another IBGP peer. RFC 4271 states that all BGP routers within a single AS must be fully meshed to provide a complete loop-free routing table and prevent traffic blackholing.

In Figure 1-5, R1, R2, and R3 are all within AS65100. R1 has an IBGP session with R2, and R2 has an IBGP session with R3. R1 advertises the 10.1.1.0/24 prefix to R2, which is processed and inserted into R2's BGP table. R2 does not advertise the 10.1.1.0/24 NLRI to R3 because it received the prefix from an IBGP peer. To resolve this issue, R1 must form a multihop IBGP session so that R3 can receive the 10.1.1.0/24 prefix directly from R1. R1 connects to R3's 10.1.23.3 IP address, and R3 connects to R1's 10.1.12.1 IP address. R1 and R3 need a static route to the remote peering link, or R2 must advertise the 10.1.12.0/24 and 10.1.23.0/24 network into BGP.

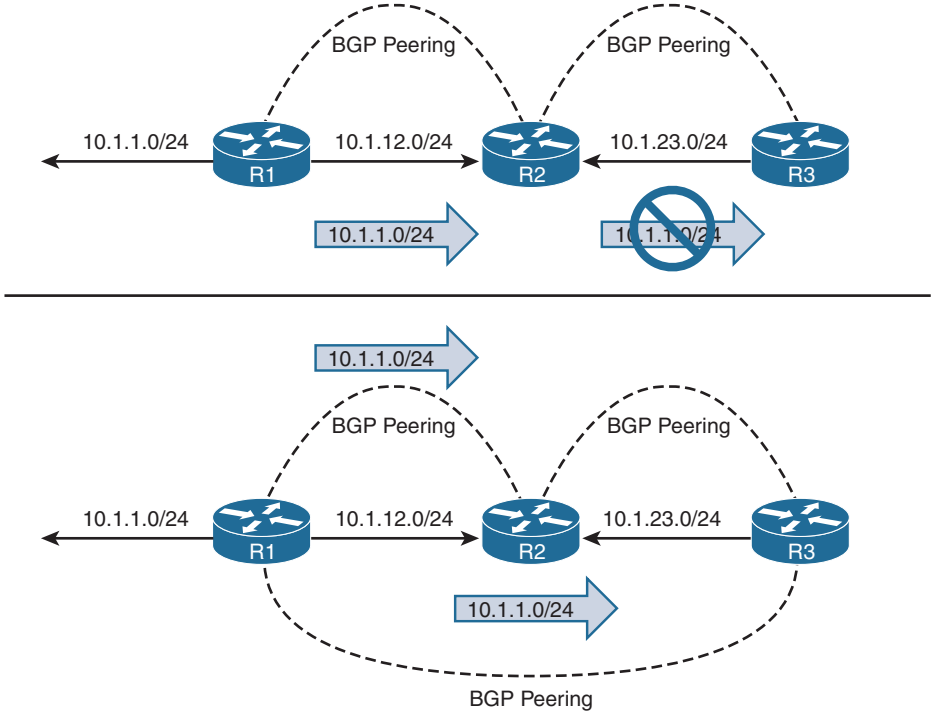

Figure 1-5 *IBGP Prefix Advertisement Behavior*

Peering via Loopback Addresses

BGP sessions are sourced by the outbound interface toward the BGP peers IP address by default. Imagine three routers connected via a full mesh. In the event of a link failure on the R1-R3 link, R3's BGP session with R1 times out and terminates. R3 loses connectivity to R1's networks even though R1 and R3 could communicate through R2 (multihop path). The loss of connectivity occurs because IBGP does not advertise routes learned from another IBGP peer as in the previous section.

Two solutions exist to overcome the link failure:

- Add a second link between all routers (3 links will become 6 links) and establish two BGP sessions between each router.
- Configure an IGP protocol on the routers' transit links, advertise loopback interfaces into the IGP, and then configure the BGP neighbors to establish a session to the remote router's loopback address.

Of the two methods, the second is more efficient and preferable.

The loopback interface is virtual and always stays up. In the event of link failure, the session remains intact while the IGP finds another path to the loopback address and, in essence, turns a single-hop IBGP session into a multihop IBGP session.

Updating the BGP configuration to set the destination of the BGP session to the remote router's loopback IP address is not enough. The source IP address of the BGP packets will still reflect the IP address of the outbound interface. When a BGP packet is received, the router correlates the source IP address of the packet to the BGP neighbor table. If the BGP packet source does not match an entry in the neighbor table, the packet cannot be associated to a neighbor and is discarded.

The source of BGP packets can be set statically to an interface's primary IP address with the BGP session configuration command **neighbor** *ip-address* **update-source** *interface-type interface-number* on IOS nodes. IOS XR and NX-OS devices use the command **update-source** *interface-type interface-number* under the neighbor session within the BGP router configuration.

Figure 1-6 illustrates the concept of peering using loopback addresses after the 10.1.13.0/24 network link fails. R1 and R3 still maintain BGP session connectivity while routes learned from OSPF allow BGP communication traffic between the loopbacks via R2. R1 can still forward packets to R3 via R2 because R1 performs a recursive lookup to identify R2 as the next-hop address.

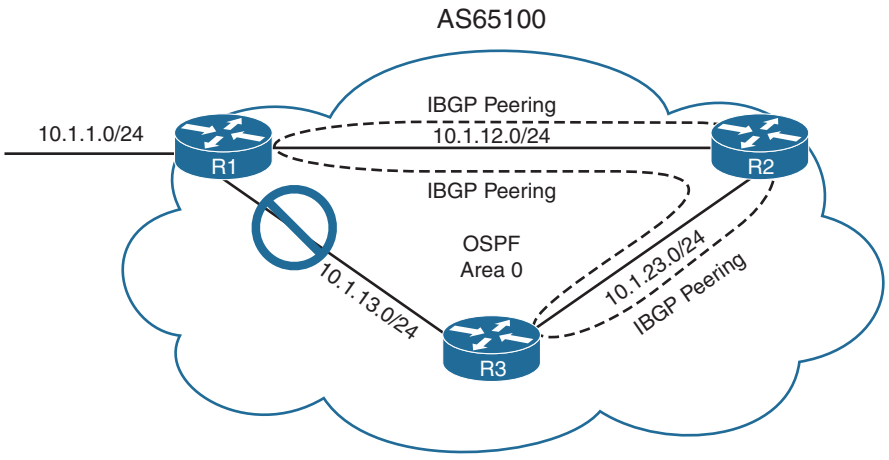

Figure 1-6 *Link Failure with IBGP Sessions on Loopback Interfaces*

Note Sourcing BGP sessions from loopback interfaces eliminates the need to recompute the BGP best path algorithm if a peering link fails as shown in Figure 1-6. It also provides automatic load balancing if there are multiple equal cost paths via IGP to the loopback address.

EBGP

EBGP peerings are the core component of the BGP protocol on the Internet. EBGP is the exchange of network prefixes between autonomous systems. The following behaviors are different on EBGP sessions when compared to IBGP sessions:

- Time to Live (TTL) on BGP packets is set to one. BGP packets drop in transit if a multihop BGP session is attempted (TTL on IBGP packets is set to 255, which allows for multihop sessions).
- The advertising router modifies the BGP next-hop to the IP address sourcing the BGP connection.
- The advertising router prepends its ASN to the existing AS_PATH.
- The receiving router verifies that the AS_PATH does not contain an ASN that matches the local routers. BGP discards the NLRI if it fails the AS_PATH loop prevention check.

The configuration for EBGP and IBGP sessions are fundamentally the same on IOS, IOS XR, and NX-OS devices, except that the ASN in the **remote-as** statement is different from the ASN defined in the BGP process.

Note Different outbound (or inbound) route policies may be different from neighbor-to-neighbor, which allows for a dynamic routing-policy within an AS.

EBGP learned paths always have at least one ASN in the AS_PATH. If multiple ASs are listed in the AS_PATH, the most recent AS is always prepended (the furthest to the left). The BGP attributes for all paths to a specific network prefix can be shown with the command **show bgp ipv4 unicast** *network* on IOS, IOS XR, and NX-OS devices.

Example 1-8 displays the BGP path attributes for the remote prefix (192.168.3.3/32).

Example 1-8 *BGP Prefix Attributes for Remote Prefix*

```
R1-IOS# show bgp ipv4 unicast 192.168.3.3
BGP routing table entry for 192.168.3.3/32, version 11
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  65200 65300
    10.1.12.2 from 10.1.12.2 (192.168.2.2)
      Origin IGP, localpref 100, valid, external, best
```

Table 1-5 explains the output provided in Example 1-8 and its correlation to BGP. Some of the BGP path attributes may change depending on the BGP features used.

Table 1-5 *BGP Prefix Attributes*

Output	Description
Paths: (1 available, best #1)	Provides a count of BGP paths in the BGP Loc-RIB and identifies the path selected as the BGP best path. All the paths and BGP attributes are listed after this.
Not advertised to any peer	Identifies whether the prefix was advertised to a BGP peer or not. BGP neighbors are consolidated into BGP update-groups. Explicit neighbors can be seen with the command **show bgp ipv4 unicast update-group** on IOS or IOS XR nodes.
65200 65300	This is the AS_PATH for the NLRI as it was received.

Output	Description
10.1.12.2 from 10.1.12.2 (192.168.2.2)	The first entry lists the IP address of the EBGP edge peer. The *from* field lists the IP address of the IBGP router that received this route from the EBGP edge peer. (In this case, the route was learned from an EBGP edge peer, so the address will be the EBGP edge peer.) Expect this field to change when an external route is learned from an IBGP peer. The number in parentheses is the BGP Identifier (RID) for that node.
Origin IGP	The Origin is the BGP well-known mandatory attribute that states the mechanism for advertising this route. In this instance, it is an Internal route
metric 0	Displays the optional nontransitive BGP attribute *Multiple-Exit Discriminator (MED)*, also known as BGP metric.
localpref 100	Displays the well-known discretionary BGP attribute Local Preference.
valid	Displays the validity of this path.
External	Displays how the route was learned. It will be internal, external, or local.

EBGP and IBGP Topologies

Combining EBGP sessions with IBGP sessions can cause confusion in terminology and concepts. Figure 1-6 provides a reference topology for clarification of concepts. R1 and R2 form an EBGP session, R3 and R4 form an EBGP session as well, and R2 and R3 form an IBGP session. R2 and R3 are IBGP peers and follow the rules of IBGP advertisement, even if the routes are learned from an EBGP peer.

As an EBGP prefix is advertised to an IBGP neighbor, issues may arise with the NLRI passing the validity check and the next-hop reachability check preventing advertisements to other BGP peers. The most common issue involves the failure of the next-hop accessibility. IBGP peers do not modify the next-hop address if the NLRI has a next-hop address other than 0.0.0.0. The next-hop address must be resolvable in the global RIB for it to be valid and advertised to other BGP peers.

To demonstrate this concept, only R1 and R4 have advertised their loopback interfaces into BGP, 192.168.1.1/32, and 192.168.4.4/32. Figure 1-7, displays the BGP table for all four routers. Notice that the BGP best path symbol (>) is missing for the 192.168.4.4/32 prefix on R2, and for the 192.168.1.1/32 on R3.

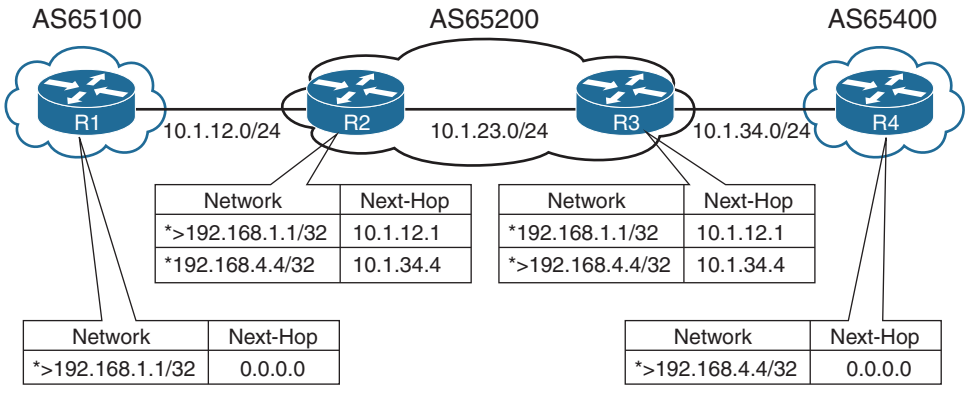

Figure 1-7 *EBGP and IBGP Topology*

R1's BGP table is missing the 192.168.4.4/32 prefix because the prefix did not pass R2's next-hop accessibility check preventing the execution of the BGP best path algorithm. R4 advertised the prefix to R3 with the next-hop address of 10.1.34.4, and R3 advertised the prefix to R2 with a next-hop address of 10.1.34.4. R2 does not have a route for the 10.1.34.4 IP address and deems the next-hop inaccessible. The same logic applies to R1's 192.168.1.1/32 prefix when advertised toward R4.

Example 1-9 shows the BGP attributes on R3 for the 192.168.1.1/32 prefix. Notice that the prefix is not advertised to any peer because the next-hop is *inaccessible*.

Example 1-9 *BGP Path Attributes for 192.168.1.1/32*

```
R3-IOS# show bgp ipv4 unicast 192.168.1.1
BGP routing table entry for 192.168.1.1/32, version 2
Paths: (1 available, no best path)
  Not advertised to any peer
  Refresh Epoch 1
  65100
    10.1.12.1 (inaccessible) from 10.1.23.2 (192.168.2.2)
      Origin IGP, metric 0, localpref 100, valid, internal
```

To correct the issue, the peering links, 10.1.12.0/24 and 10.1.34.0/24, need to be in both R2's and R3's routing table via either technique:

- IGP advertisement. Remember to use the passive interface to prevent an accidental adjacency from forming. Most IGPs do not provide the filtering capability like BGP.
- Advertising the networks into BGP.

Both techniques allow the prefixes to pass the next-hop accessibility test.

Figure 1-8 displays the topology with both transit links advertised into BGP. Notice that this time all four prefixes are valid with a BGP best path selected.

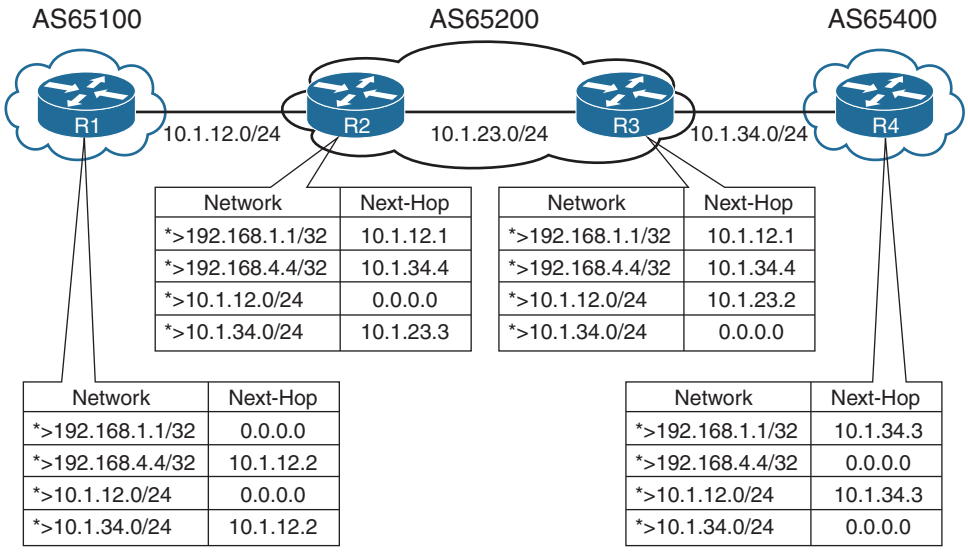

Figure 1-8 *EBGP and IBGP Topology After Advertising Peering Links*

Next-Hop Manipulation

Imagine a service provider network with 500 routers and every router containing 200 EBGP peering links. To ensure that the next-hop address is reachable to the IBGP peers requires the advertisement of 100,000 peering networks in BGP or an IGP consuming router resources.

Another technique to ensure that the next-hop address check passes without advertising peering networks into a routing protocol involves the modification of the next-hop address in the BGP advertisement. The next-hop IP address can be modified on inbound or outbound neighbor routing policies. Managing IP addresses in a route policy can be a complicated task. Configuring the **next-hop-self** address-family feature modifies the next-hop address in all external NLRIs using the IP address of the BGP neighbor.

The command **neighbor** *ip-address* **next-hop-self** [**all**] is used for each neighbor under the address-family configuration on IOS nodes, and the command **next-hop-self** is applied under the neighbor address-family configuration for IOS XR and NX-OS devices.

Figure 1-9 shows the topology and BGP routing table for all four routers. Notice that R2 and R3 advertised the EBGP routes to each other with the next-hop address as the BGP session IP address, allowing the NLRIs to pass the next-hop accessibility check.

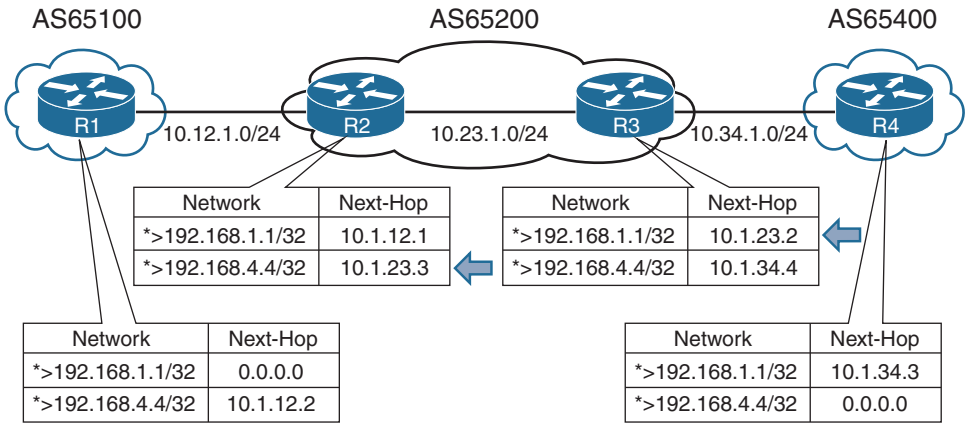

Figure 1-9 *EBGP and IBGP Topology with Next-Hop-Self*

Note The **next-hop-self** feature does not modify the next-hop address for IBGP prefixes by default. IOS nodes can append the optional **all** keyword, which modifies the next-hop address on IBGP prefixes, too. IOS XR provides the BGP configuration command **IBGP policy out enforce-modifications** that will modify IBGP NLRIs in the same manner as EBGP NLRIs. NX-OS devices need to modify the next-hop address in a route-map to overcome this behavior for IBGP routes.

IBGP Scalability

The inability for BGP to advertise a prefix learned from one IBGP peer to another IBGP peer can lead to scalability issues within an AS. The formula *n(n-1)/2* provides the number of sessions required where *n* represents the number of routers. A full mesh topology of 5 routers requires 10 sessions, and a topology of 10 routers requires 45 sessions. IBGP scalability becomes an issue for large networks.

Route Reflectors

RFC 1966 introduces the concept that an IBGP peering can be configured so that it reflects routes to another IBGP peer. The router reflecting routes is known as a *route reflector (RR)*, and the router receiving reflected routes is a *route reflector client*. Three basic rules involve route reflectors and route reflection:

Rule #1: If a RR receives a NLRI from a non-RR client, the RR advertises the NLRI to a RR client. It does not advertise the NLRI to a non-route-reflector client.

Rule #2: If a RR receives a NLRI from a RR client, it advertises the NLRI to RR client(s) and non-RR client(s). Even the RR client that sent the advertisement

receives a copy of the route, but it discards the NLRI because it sees itself as the route originator.

Rule #3: If a RR receives a route from an EBGP peer, it advertises the route to RR client(s) and non-RR client(s).

Figure 1-10 demonstrates the route reflector rules.

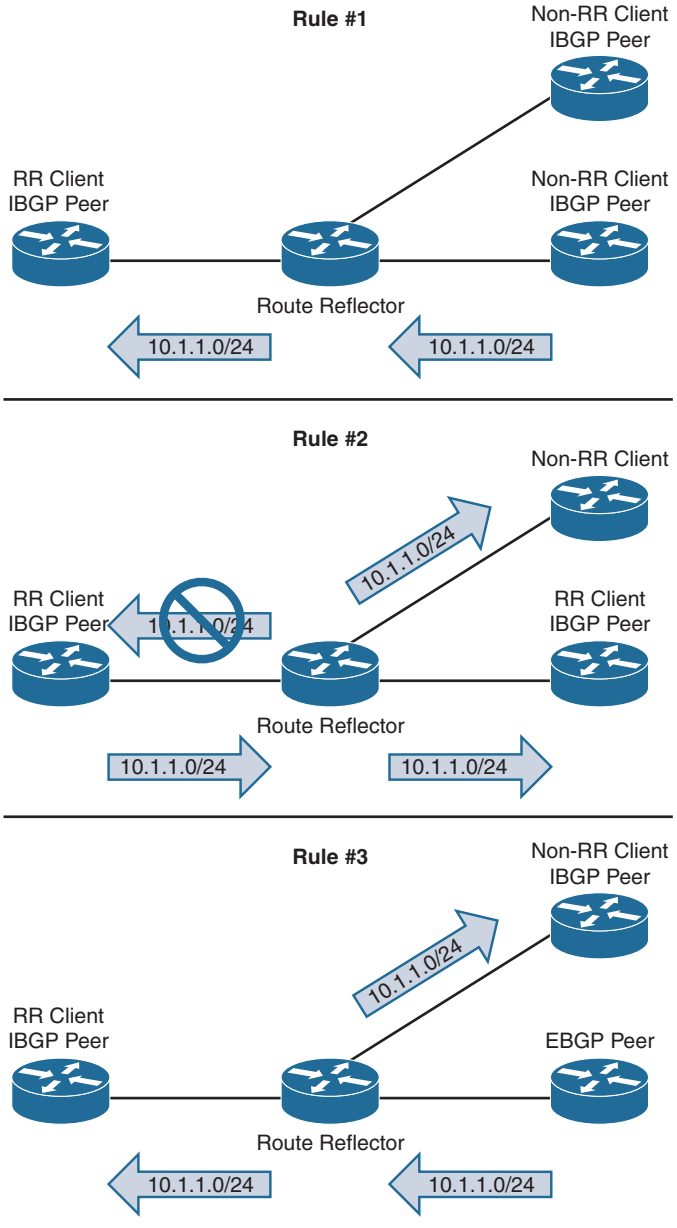

Figure 1-10 *Route Reflector Rules*

Only route reflectors are aware of this change in behavior because no additional BGP configuration is performed on route-reflector clients. BGP route reflection is specific to each address-family. The command **neighbor** *ip-address* **route-reflector-client** is used on IOS nodes, and the command **route-reflector-client** is used on IOS XR and NX-OS devices under the neighbor address-family configuration.

Loop Prevention in Route Reflectors

Removing the full mesh requirements in an IBGP topology introduces the potential for routing loops. When RFC 1966 was drafted, two other BGP route reflector specific attributes were added to prevent loops.

ORIGINATOR_ID, an optional nontransitive BGP attribute is created by the first route reflector and sets the value to the RID of the router that injected/advertised the route into the AS. If the ORIGINATOR_ID is already populated on an NLRI, it should not be overwritten.

If a router receives a NLRI with its RID in the Originator attribute, the NLRI is discarded.

CLUSTER_LIST, a nontransitive BGP attribute, is updated by the route reflector. This attribute is appended (not overwritten) by the route reflector with its cluster-id. By default this is the BGP identifier. The cluster-id can be set with the BGP configuration command **bgp cluster-id** *cluster-id* on IOS and IOS XR nodes. NX-OS devices use the command **cluster-id** *cluster-id*.

If a route reflector receives a NLRI with its cluster-id in the Cluster List attribute, the NLRI is discarded.

Example 1-10 provides sample output prefix output from a route that was reflected. Notice that the originator ID is the advertising router and that the cluster list contains two route-reflector IDs listed in the order of the last route reflector that advertised the route.

Example 1-10 *Route Reflector Originator ID and Cluster List Attributes*

```
RP/0/0/CPU0:R1-XR# show bgp ipv4 unicast 10.4.4.0/24
! Output omitted for brevity
Paths: (1 available, best #1)
  Local
    10.1.34.4 from 10.1.12.2 (192.168.4.4)
      Origin IGP, metric 0, localpref 100, valid, internal, best, group-best
      Received Path ID 0, Local Path ID 1, version 7
      Originator: 192.168.4.4, Cluster list: 192.168.2.2, 192.168.3.3
```

Out-of-Band Route Reflectors

As explained earlier, BGP can establish multihop BGP sessions and does not change the next-hop path attribute when routes are advertised to IBGP neighbors. Some large network topologies use dedicated BGP routers for route reflection that are outside of the data path.

These out-of-band route reflectors provide control plane programming for the BGP routers that are in the data path and only require sufficient memory and processing power for the BGP routing table. Out-of-band route reflectors should not use the **next-hop-self**, or it will place the route reflector into the data path. Organizations that use MPLS L2VPNs, L3VPNs, and so on will use multiple out-of-band route reflectors for exchanging BGP path information.

Confederations

RFC 3065 introduced the concept of BGP confederations as an alternative solution to IBGP full mesh scalability issues shown earlier. A confederation consists of sub-ASs known as a Member-AS that combine into a larger AS known as an AS Confederation. Member ASs normally use ASNs from the private ASN range (64512-65535). EBGP peers from the confederation have no knowledge that they are peering with a confederation, and they reference the confederation identifier in their configuration.

Figure 1-11 demonstrates a BGP confederation with the confederation identifier of AS200. The Member-ASs are AS65100 and AS65200. R3 provides route reflection in Member-AS 65100.

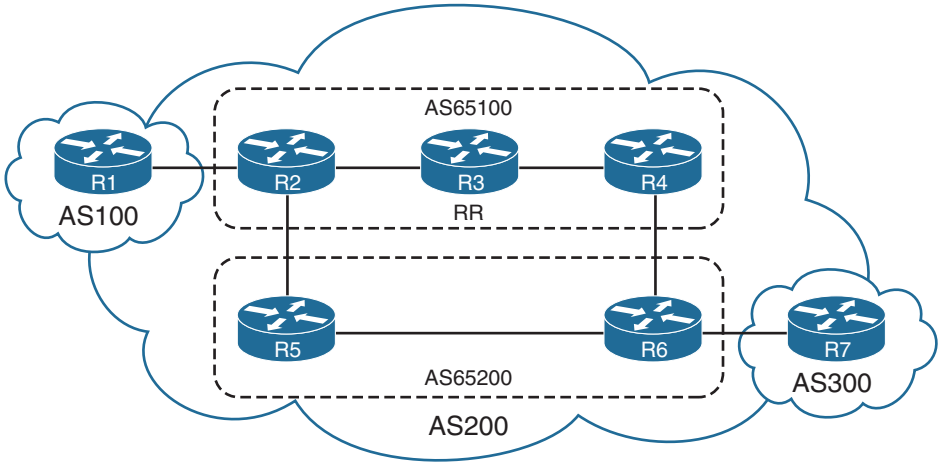

Figure 1-11 *Sample BGP Confederation Topology*

Confederations share behaviors from both IBGP sessions and EBGP sessions. The changes are as follows:

- The AS_PATH attribute contains a subfield called AS_CONFED_SEQUENCE. The AS_CONFED_SEQUENCE is displayed in parentheses before any external ASNs in the AS_PATH. As the route passes from Member-AS to Member-AS, the AS_CONFED_SEQUENCE is appended to contain the Member-AS ASNs. The

AS_CONFED_SEQUENCE attribute is used to prevent loops, but it is not used (counted) when choosing shortest AS_PATH.

- Route reflectors can be used within the Member-AS like normal IBGP peerings.
- The BGP MED attribute is transitive to all other Member-ASs, but does not leave the confederation.
- The LOCAL_PREF attribute is transitive to all other Member-ASs, but does not leave the confederation.
- IOS XR nodes do not require a route policy when peering with a different Member-AS, even though the **remote-as** is different.
- The next-hop address for external confederation routes does not change as the route is exchanged between Member-AS to Member-AS.
- The AS_CONFED_SEQUENCE is removed from the AS_PATH when the route is advertised outside of the confederation.

Configuring a BGP confederation is shown in the following steps:

Step 1. Create the BGP Routing Process. Initialize the BGP process with the global command **router bgp** *member-asn*.

Step 2. Set the BGP Confederation Identifier. Identify the BGP confederations with the command **bgp confederation identifier** *as-number*. The *as-number* is the BGP confederation ASN.

Step 3. Identify Peer Member-ASs. On routers that directly peer with another Member-AS, identify the peering Member-AS with the command **bgp confederation peers** *member-asn*.

Step 4. Configure BGP confederation members as normal; the remaining configuration follows normal BGP configuration guidelines.

Example 1-11 displays R1's and R2's BGP table. R1 resides in AS100 and does not see any of the BGP subconfederation information. R1 is not aware the AS200 is subdivided into a BGP confederation.

R2's BGP table participates in the Member-AS 65100. Notice the next-hop address is not modified for the 10.67.1.0/24 (Network between R6 and R7) even though a Member-AS. The AS_CONFED_SEQUENCE is listed in parentheses to indicate it passed through Sub-AS 65200 in the AS200 confederation.

Example 1-11 *R1's and R2's BGP Table*

```
R1-IOS# show bgp ipv4 unicast
! Output omitted for brevity
     Network          Next Hop            Metric LocPrf Weight Path
 r>  10.1.12.0/24     10.1.12.2                0             0 200 i
 *>  10.1.23.0/24     10.1.12.2                0             0 200 i
 *>  10.1.25.0/24     10.1.12.2                0             0 200 i
 *>  10.1.34.0/24     10.1.12.2                                0 200 i
 *>  10.1.46.0/24     10.1.12.2                                0 200 i
 *>  10.1.56.0/24     10.1.12.2                                0 200 i
 *>  10.1.67.0/24     10.1.12.2                                0 200 i
R2-IOS# show bgp ipv4 unicast

! Output omitted for brevity
     Network          Next Hop            Metric LocPrf Weight Path
 *>  10.1.12.0/24     0.0.0.0                  0         32768 i
 * i 10.1.23.0/24     10.1.23.3                0    100      0 i
 *>                   0.0.0.0                  0         32768 i
 *   10.1.25.0/24     10.1.25.5                0    100      0 (65200) i
 *>                   0.0.0.0                  0         32768 i
 *>i 10.1.34.0/24     10.1.23.3                0    100      0 i
 *>i 10.1.46.0/24     10.1.34.4                0    100      0 i
 *>  10.1.56.0/24     10.1.25.5                0    100      0 (65200) i
 *   10.1.67.0/24     10.1.56.6                0    100      0 (65200) i
 *>i                  10.1.46.6                0    100      0 (652000) i
```

Example 1-12 displays the NLRI information for 10.67.1.0/24 from the perspective of R2. Notice that the NLRI from within a confederation includes the option of *confed-internal* and *confed-external* for sources.

Example 1-12 *Confederation NLRI*

```
R2-IOS# show bgp ipv4 unicast 10.67.1.0/24
! Output omitted for brevity
BGP routing table entry for 10.1.67.0/24, version 8
Paths: (2 available, best #2, table default)
  Advertised to update-groups:
     1           3
  Refresh Epoch 1
  (65200)
    10.56.1.6 from 10.1.25.5 (10.1.56.5)
      Origin IGP, metric 0, localpref 100, valid, confed-external
      rx pathid: 0, tx pathid: 0
```

```
Refresh Epoch 1
(65200)
  10.46.1.6 from 10.1.23.3 (10.1.23.3)
    Origin IGP, metric 0, localpref 100, valid, confed-internal, best
    Originator: 10.1.34.4, Cluster list: 10.1.23.3
    rx pathid: 0, tx pathid: 0x0
```

BGP Communities

BGP communities provide additional capability for tagging routes and for modifying BGP routing policy on upstream and downstream routers. BGP communities can be appended, removed, or modified selectively on each attribute as the route travels from router to router.

BGP communities are an optional transitive BGP attribute that can traverse from *autonomous system* to *autonomous system*. A BGP community is a 32-bit number that can be included with a route. A BGP community can be displayed as a full 32-bit number (0-4,294,967,295) or as two 16-bit numbers (0-65535):(0-65535) commonly referred to as *new-format*.

Private BGP communities follow the convention that the first 16-bits represent the AS of the community origination, and the second 16-bits represent a pattern defined by the originating AS. The private BGP community pattern could vary from organization to organization, do not need to be registered, and could signify geographic locations for one AS while signifying a method of route advertisement in another AS. Some organizations publish their private BGP community patterns on websites, such as http://www.onesc.net/communities/.

In 2006, RFC 4360 expanded BGP communities' capabilities by providing an extended format. *Extended BGP communities* provide structure for various classes of information and are commonly used for VPN Services.

IOS XR and NX-OS devices display BGP communities in new-format by default, and IOS nodes display communities in decimal format by default. IOS nodes can display communities in new-format with the global configuration command **ip bgp-community new-format**.

Example 1-13 displays the BGP community in decimal format on top, and in new-format on bottom.

Example 1-13 *BGP Community Formats*

```
! DECIMAL FORMAT
R3# show bgp 192.168.1.1
! Output omitted for brevity
BGP routing table entry for 192.168.1.1/32, version 6
Community: 6553602 6577023
```

```
! New-Format
R3# show bgp 192.168.1.1
! Output omitted for brevity
BGP routing table entry for 192.168.1.1/32, version 6
Community: 100:2 100:23423
```

IOS and NX-OS devices do not advertise BGP communities to peers by default. Communities are enabled on a neighbor-by-neighbor basis with the BGP address-family configuration command **neighbor** *ip-address* **send-community** [**standard** | **extended** | **both**], and NX-OS devices use the command **send-community** [**standard** | **extended** | **both**] under the neighbor address-family configuration. Standard communities are sent by default, unless the optional **extended** or **both** keywords are used.

IOS XR advertises BGP communities to IBGP peers by default, but EBGP peers require the neighbor address-family configuration command **send-community-ebgp** for advertising standard BGP communities, and the command **send-extended-community-ebgp** to advertise extended BGP communities. Both commands are required if both community formats are to be sent to an EBGP peer.

Route Summarization

Summarizing prefixes conserves router resource(s) and accelerates best path calculation by reducing the size of the table. Summarization also provides the benefit(s) of stability by reducing routing churn by hiding route flaps from downstream routers. Although most ISPs do not accept prefixes larger than /24 for IPv4 (/25-/32), the Internet, at the time of this writing, still has more than 600,000 routes and continues to grow toward a million routes. Route summarization is required to reduce the size of the BGP table for Internet routers.

BGP route summarization on EBGP routers for nontransitive ASs reduce route computation on routers in the core of the nontransitive AS. In Figure 1-12, R3 summarizes all the EBGP routes received from AS65100 and AS65200 to reduce route computation on R4 during link flaps. In the event of a link flap on the 10.1.13.0/24 network, R3 removes all AS65100 routes learned directly from R1 and identifies the same networks via R2 with a different (longer AS_PATH). R4 processes the same changes that R3 processes and is a waste of CPU cycles because R4 receives connectivity only from R3. If R3 summarized the network range, instead of running the best-path algorithm against multiple routes, the best-path algorithm would execute only once.

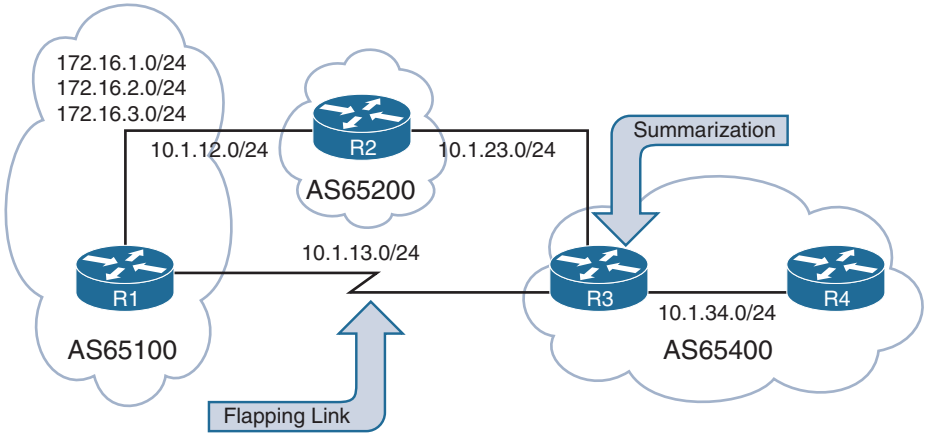

Figure 1-12 *BGP Route Summarization*

The two techniques for BGP summarization are the following:

- **Static:** Create a static route to Null 0 for the prefix, and then advertise the network via a network statement. The downfall to this technique is that the summary route will always be advertised even if the networks are not available.
- **Dynamic:** Configure an aggregation network range. When viable routes that match the network range enter the BGP table, an aggregate route is created. On the originating router, the aggregated prefix sets the next-hop to Null 0. The route to Null 0 is automatically created by BGP as a loop-prevention mechanism.

In both methods of route aggregation, a new network prefix with a shorter prefix length is advertised into BGP. Because the aggregated prefix is a new route, the summarizing router is the originator for the new aggregate route.

Aggregate-Address

Dynamic route summarization is accomplished with the BGP address-family configuration commands identified in Table 1-6.

Table 1-6 *BGP Route-Aggregation Commands*

OS	Command
IOS	**aggregate-address** *network subnet-mask* [**summary-only** | **suppress-map** *route-map-name*] [**as-set**] [**advertise-map** *route-map-name*]
IOS XR	**aggregate-address** *network/prefix-length* [**summary-only** | **route-policy** *route-policy-name*] [**as-set**] [**advertise-map** *route-policy-name*]
NX-OS	**aggregate-address** {*network subnet-mask network/prefix-length*}[**summary-only** | **suppress-map** *route-map-name*] [**as-set**] [**advertise-map** *route-map-name*]

The aggregate-address command advertises the aggregated route in addition to the original networks. Using the optional **no-summary** keyword suppresses the networks in the summarized network range. BGP considers aggregated addresses as local routes.

Note Aggregate addresses are local BGP routes when modifying BGP AD.

Flexible Route Suppression

Some traffic engineering designs require "leaking" routes, which is the advertisement of a subset of more specific routes in addition to performing the summary. Leaking routes can be done at the process by explicitly stating the prefixes to suppress, or on a neighbor level by indicating which prefixes should not be suppressed.

Selective Prefix Suppression

Selective prefix suppression explicitly lists the networks that should not be advertised along with the summary route to neighbor routers.

IOS and NX-OS uses a *suppress-map*, which uses the keyword **suppress-map** *route-map-name* instead of using the **no-summary** keyword. In the referenced route-map, only the prefixes that should be suppressed are permitted. IOS XR routers use the keyword **route-policy** *route-policy-name* in lieu of the **no-summary** keyword. In the route policy, the action command **suppress** is used after conditionally matching the prefixes that should be suppressed.

Leaking Suppressed Routes

The **summary-only** keyword suppresses all the more specific routes of an aggregate address from being advertised. After a route is suppressed, it is still possible to advertise the suppressed route to a specific neighbor.

IOS devices use an *unsuppress-map* with the BGP neighbor address-family configuration command **neighbor** *ip-address* **unsuppress-map** *route-map-name*. In the referenced route-map, only the prefixes that should be leaked are permitted. IOS XR routers use an outbound route policy with the action command **unsuppress** to indicate which prefixes should be leaked.

Atomic Aggregate

Aggregated routes act like new BGP routes with a shorter prefix length. When a BGP router summarizes a route, it does not advertise the AS path information from before the aggregation. BGP path attributes such as AS-Path, MED, and BGP communities are not included in the new BGP advertisement. The Atomic Aggregate attribute indicates that a loss of path information has occurred.

For example:

- R1 and R2 are advertising the 172.16.1.0/24 and 172.16.2.0/24 networks.
- R3 is aggregating the routes into the 172.16.0.0/22 network range, which is advertised to all of R3's peers

Example 1-14 displays R3's BGP table. R1's BGP prefix 172.16.1.0/24 advertised to R3. Notice the AS-Path of 65100 and BGP Community of 100:100.

Example 1-14 *172.16.1.0/24 BGP Path Information*

```
R3-IOS# show bgp ipv4 unicast 172.16.1.0
BGP routing table entry for 172.16.1.0/24, version 13
Paths: (1 available, best #1, table default, Advertisements suppressed by an aggre-
  gate.)
  Not advertised to any peer
  Refresh Epoch 1
 65100
    10.1.13.1 from 10.1.13.1 (192.168.1.1)
      Origin IGP, metric 0, localpref 100, valid, external, best
     Community: 100:100
```

R3's aggregate route (summary) does not include the BGP communities (including AS-Path history) for the routes in the summarization range. R3 advertises the aggregate route to R1 and R2, and those routers install the 172.16.0.0/22 summary route because their AS-Path is not listed in the AS-Path attribute and passes the AS-Path loop check.

Example 1-15 displays the BGP path information for the 172.16.0.0/22 summary network on R1. The AS-Path of the aggregated route displays only the aggregating router, but does not include the AS-Path of the routes being summarized (AS65100 or AS65200), nor is the BGP community included in the routes being summarized. The BGP path information indicates that this is an aggregated prefix and was aggregated by R3 (192.168.3.3). The *Atomic-Aggregate* in the route indicates a loss of information occurred during aggregation on the aggregating router.

Example 1-15 *172.16.0.0/22 BGP Path Information*

```
R1-IOS# show bgp ipv4 unicast 172.16.0.0
BGP routing table entry for 172.16.0.0/21, version 5
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  300, (aggregated by 300 192.168.3.3)
    10.1.13.3 from 10.1.13.3 (192.168.3.3)
      Origin IGP, metric 0, localpref 100, valid, external, atomic-aggregate, best
```

Route Aggregation with AS_SET

To keep the BGP path information history, the optional **as-set** keyword may be used with the **aggregate-address** command. As the router generates the aggregate route, BGP attributes from the summarized routes are copied over to it. The AS-Path settings from the original prefixes are stored in the AS_SET portion of the AS-Path. (The AS_SET is displayed within brackets, and counts only as one hop, even if multiple ASs are listed.)

Route Aggregation with Selective Advertisement of AS-SET

Using the AS-SET feature with network aggregation combines all the attributes of the original prefixes into the aggregated prefixes. This might cause issues with your routing policy. For example, if one of the prefixes contains the No-Export BGP community, the aggregate address will not be exported. To resolve these types of problems, selectively choose the routes that the path attributes will copy to the aggregate route. The use of the **advertise-map** option allows for conditionally matching and denying attributes that should be permitted or denied in the aggregated route.

Default Route Advertisement

Advertising a default route into the BGP table requires the default route to exist in the RIB and the BGP configuration command **default-information originate** to be used. The redistribution of a default route or use of a network 0.0.0.0/0 does not work without the **default-information originate** command.

Default Route Advertisement per Neighbor

Some network topologies restrict the size of the BGP advertisements to a neighbor because the remote router does not have enough processing power or memory for the full BGP routing table. Connectivity is still required, so the peering routers only advertise the default route to the remote router.

A default route is advertised to a BGP peer with the BGP address-family configuration command **neighbor** *ip-address* **default-originate** for IOS nodes or with the BGP neighbor address-family configuration command **default-originate** for IOS XR and NX-OS devices. Default route advertisement to a specific neighbor does not require a default route to be present in the RIB or BGP Loc-RIB table.

Note A behavior difference between IOS and IOS XR occurs when a default route is already present in the BGP table. IOS nodes advertise the route as if it was the originating router. (None of the existing attributes are passed to the peer.) IOS XR nodes advertise the network to the peer as it exists in the BGP table with the entire default route attributes (AS-Path, and so on).

Remove Private AS

Some organizations might not be able to meet the qualifications for obtaining their own ASN but still want to receive Internet routing tables from their service provider. In these situations, the service provider may assign the organization a private ASN for peering. Private ASNs should not be advertised by the service provider to other ISPs on the Internet.

The feature *remove private AS* removes the private AS of routes that are advertised to the configured peer. The router performs the following path analysis with the remove private AS feature:

- Removes only private ASNs on routes advertised to EBGP peers.
- If the AS-Path for the route has only private ASNs, the private ASNs are removed.
- If the AS-Path for the route has a private ASN between public ASNs, it is assumed that this is a design choice, and the private ASN is not removed
- If the AS-Path contains confederations (AS_CONFED_SEQ), BGP removes the private AS numbers only if they are included after the AS_CONFED_SEQ (Confederation AS-Path) of the path.

The remove private AS feature is configured on IOS nodes with the BGP address-family configuration command **neighbor** *ip-address* **remove-private-as**. IOS XR and NX-OS devices use the BGP neighbor address-family configuration command **remove-private-as**.

Allow AS

The *Allow AS* feature allows for routes to be received and processed even if the router detects its own ASN in the AS-Path. A router discards BGP network prefixes if it sees its ASN in the AS-Path as a loop prevention mechanism. Some network designs use a transit AS to provide connectivity to two different locations. BGP detects the network advertisements from the remote site as a loop and discards the route. The AS-Path loop check feature needs to be disabled to maintain connectivity in scenarios such as these.

On IOS nodes, the command **neighbor** *ip-address* **allowas-in** is placed under the address-family. IOS XR and NX-OS nodes use the BGP neighbor address-family configuration command **allowas-in**.

LocalAS

When two companies merge, one of the ASNs is usually returned to the regional Internet registry (RIR). During the migration, each company needs to maintain its own ASN while changes are made with its peering neighbors to update their configuration.

The *LocalAS* feature is configured on a per peer basis, and allows for BGP sessions to establish using an alternate ASN than the ASN that the BGP process is running on. The LocalAS feature works only with EBGP peerings.

IOS nodes use the BGP address-family neighbor configuration command **neighbor** *ip-address* **local-as** *alternate-as-number* [**no-prepend** [**replace-as** [**dual-as**]]]. IOS XR and NX-OS devices use the equivalent command **local-as** *alternate-as-number* [**no-prepend** [**replace-as** [**dual-as**]]] under the neighbor. By default, the alternate ASN is added to the AS-Path for routes that are sent and received between these two peers.

One problem with the alternate ASN being prepended when receiving the routes is that other IBGP peers drop the network prefixes as part of a routing loop detection.

- To stop the alternate ASN from being prepended when *receiving routes*, the optional keyword **no-prepend** is used.
- To stop the alternate ASN from being prepended when *sending routes*, the optional keywords **no-prepend replace-as** is used.
- If both **no-prepend replace-as** keywords are used, all routers see the BGP advertisements as if they were running the original AS in the BGP process.

After the remote peer changes the remote-as setting on the BGP configuration, the **local-as** commands should be removed. If the coordination of maintenance windows cannot occur during the same time, the **no-prepend replace-as dual-as** optional keywords allow the remote peer to user either ASN for the BGP session. The remote BGP router peers with the ASN in the router process statement, or the alternate ASN in the **local-as** configuration.

Summary

BGP is a powerful path vector routing protocol that provides scalability and flexibility that cannot be compared to any other routing protocol. BGP uses TCP port 179 for all BGP communication between peers, which allows BGP to establish sessions with directly attached routers or with routers that are multiple hops away.

Originally, BGP was intended for the routing of IPv4 prefixes between organizations, but over the years has had significant increase in functionality and feature enhancements. BGP has expanded from being an Internet routing protocol to other aspects of the network, including the data center.

BGP provides a scalable control-plane signaling for overlay topologies, including MPLS VPNs, IPsec SAs, and VXLAN. These overlays can provide Layer 3 services, such as L3VPNs, or Layer 2 services, such as eVPNs, across a widely used scalable control plane for everything from provider-based services to data center overlays. Every AFI / SAFI combination maintains an independent BGP table and routing policy, which makes BGP the perfect control plane application.

This chapter provided a fundamental overview of BGP from a session perspective, as well as route advertisement behaviors for IPv4 and IPv6 protocols. Networking vendors continue to use BGP for new features, and having the ability to effectively troubleshoot BGP is becoming more and more necessary.

This book provides emphasis on various BGP-related problems that are encountered in real-life deployments, which have caused major outages to the network over the years.

References

RFC 1654: *A Border Gateway Protocol 4 (BGP-4)*, Y. Rekhter, T. Li, http://tools.ietf.org/html/rfc1654, July 1994.

RFC 1966: *BGP Route Reflection, An alternative to full mesh IBGP*, T. Bates, R. Chandra, http://www.ietf.org/rfc/rfc1966.txt, June 1996.

RFC 3065: *Autonomous System Confederations for BGP*, P. Traina et al., http://www.ietf.org/rfc/rfc3065.txt, February 2001.

RFC 4271: *A Border Gateway Protocol 4 (BGP-4)*, Y. Rekhter et al., http://www.ietf.org/rfc/rfc4271.txt, January 2006.

RFC 4360: *BGP Extended Communities Attribute*, Srihari Sangli, Dan Tappan, Yakov Rekhter, IETF, http://www.ietf.org/rfc/rfc4360.txt, February 2006.

RFC 4451: *BGP MULTI_EXIST_DISC (MED) Considerations*, D. McPherson, V. Gill, http://www.ietf.org/rfc/rfc4451.txt, March 2006.

RFC 4893: *BGP Support for Four-octet AS Number Space*, Q. Vohra, E. Chen, http://www.ietf.org/rfc/rfc4893.txt, May 2007.

Edgeworth, Brad, Foss, Aaron, Garz Rios, Ramiro. *IP Routing on Cisco IOS, IOS XE, and IOS XR*. Indianapolis: Cisco Press: 2014.

Cisco. Cisco IOS Software Configuration Guides. http://www.cisco.com

Cisco. Cisco IOS XR Software Configuration Guides. http://www.cisco.com

Cisco. Cisco NX-OS Software Configuration Guides. http://www.cisco.com

This page intentionally left blank

Chapter 2

Generic Troubleshooting Methodologies

The following topics are covered in this chapter:

- Identifying problems
- Understanding variables
- Reproducing the problem
- Platform-specific packet capture tools
- Event monitoring/tracing

Finding and narrowing down a problem is not so easy. For this reason, troubleshooting is considered to be an art. Every issue can be quickly resolved when approached logically and examined thoroughly. Most network problems are not as complex as they look. Even simpler network problems can appear to be complex because either the issue is not defined clearly or is not properly understood. A few basic questions help clarify the problem and further help with troubleshooting:

1. What is the problem description?
2. What caused the problem?
3. Is the problem reproducible?

These questions are discussed in detail throughout this chapter.

Identifying the Problem

The most important information required during troubleshooting is defining the problem description, which should be done first. A vague or generic description can be misleading.

A common example for an Internet connection not working properly is a generic statement such as “the Internet is down” or “the Internet is broken.” From the initial reading

of the problem description, you might start thinking, How could the Internet break? Is the Internet down for everyone or just one user? The problem could be that users are unable to access certain websites, which can possibly indicate a problem with the DNS server, or that a company's Internet gateway could have problems. If the DNS server is not able to resolve the website name to an IP address, the websites are not accessible.

If the problem description is not clearly defined, a network engineer might start investigating the state of the network rather than focusing on the actual problem, which in the above stated example could be a DNS server. After the issue is defined, it should be documented. Documentation plays a vital role in every network deployment as it helps in forensic investigation, analysis of network outages, and mitigating future outages due to similar problems. It is rightly said that "unless it is documented, it never happened."

In most cases, the focus is on solving a problem instead of understanding it. Proper troubleshooting is crucial for a timely resolution. Therefore, defining, documenting, and understanding a problem is very important in minimizing the outage.

Understanding Variables

The famous Newton's law of physics says, "For every action, there is an equal and opposite reaction." In context of a computer network, "Every event (reaction) is the result of some action." The statement means that every network event (expected or unexpected) is the consequence of one or more triggers, such as configuration changes or software or hardware changes. This rule applies for any major or minor network outage. For every network incident, there has to be a trigger, and the trigger could be a manual trigger or due to a network event or any external tool-generated trigger. These triggers are obvious triggers. There are also other non-obvious triggers, such as inter-process communication (IPC) failure or Finite State Machine (FSM) errors. These non-obvious triggers may not have an obvious signature, such as syslog messages, and may be called defects, or bugs.

For example, a router in a network crashes and goes down. The crash could occur due to a hardware or software failure. Hardware failure like Dynamic Random Access Memory (DRAM) on the router might have gone bad, or the motherboard itself may have failed, causing the router to be completely down. Software failure could be due to a new configuration change or a software defect. Similarly, high CPU utilization on a router could be due to a flapping link on one of the remote end devices. Along with the trigger, there are other variables that require serious consideration. Some of the examples of such variables are traffic pattern, traffic load, number of paths, and so on. These variables are as important as the trigger of the problem. It may so happen that the problem occurs only if a certain type of traffic is passing through the router, or the problem might occur only during business hours when the traffic load on the device is high. For instance, a router experiences a crash and goes down when there is Transmission Control Protocol (TCP) stream coming to the router sourced from a particular IP address and for a defined destination port number and when such traffic hits an ACL entry. In this situation, the traffic

hitting the ACL entry is the trigger, but the variables are the TCP stream with a particular IP address and destination port number.

A problem can be identified and temporarily mitigated using workarounds, but those are not permanent fixes. If the exact trigger of the problem is not known, such as the event that primarily triggered the problem, the *root cause analysis (RCA)* cannot be done nor a proper fix be prepared. For example, a user reports management access to the router is lost after a recent configuration change. Unless the exact configuration change is known and verified, it is still possible that the user can repeat the same mistake, even if the problem was resolved after rebooting the router. The configuration change should be verified as to whether it is a valid configuration. An incorrect configuration, such as an incorrectly configured ACL, can block the legitimate traffic as well and cause a network outage.

As mentioned before, documentation plays a vital role during investigation of network outages. Documenting the trigger of the problem is as important as noting the description of the problem. The next time that a similar problem occurs, it will not take long to understand if it is a known problem or a new one.

Reproducing the Problem

It is now clear how important it is to document the detailed problem description, the variables attached to the problem, and the trigger that led to that problem. But if the documented event is just a one-time event that caused the problem, then RCA for that problem could not be validated and would merely be a hypothesis based on assumptions. For a successful documentation of a problem and its resolution, the problem should be reproducible all the time using the same trigger. Different triggers can exhibit same or different behaviors. For example, two different triggers can cause a same problem, or the same trigger can exhibit two different problems. Therefore, consistency is required for a problem to be successfully documented and fixed. If there are different behaviors, the problems may not be the same.

Simulating a problem is not an easy task either. It takes a lot of effort to set up the lab, put up the configuration, and simulate multiple triggers. And yes, sometimes it includes a bit of luck. The first three points are crucial for reproducing a problem. The problem might get reproduced in single attempt. It might take multiple tries.

If the problem is not replicated in a lab environment, sometimes it is worth taking a downtime in the production environment to investigate the problem. When scheduling such windows, it is advised to move the traffic to backup devices to minimize any outage for end users; or if the problem is related to traffic or occurs only when the traffic is present on the box, then downtime is the way to go so that users are aware of the outage.

Setting Up the Lab

Lab environments have fewer resources as compared to the production environments. The production environment has hundreds of routers and switches and other devices that

cannot be accommodated in a lab environment. Based on the problem, only a relevant part of the topology should be focused on, and the lab environment should be built and set up using the same.

The relevant part of the topology is based on the assumption and understanding of the problem. This does not imply that the minimum setup will always help replicate the problem. Sometimes the lab environment has to be scaled up to make it closer to the production deployment.

Figure 2-1 shows a topology for Multiprotocol Label Switching (MPLS) Virtual Private Networks (VPN) deployment of a service provider for Customer A and Customer B. The service provider is using MPLS and MPLS Traffic Engineering (TE) in its core network. In this topology, Customer A faces reachability issues between two sites after the MPLS TE tunnel flapped between R1 and R14 caused by a link failure between R7 and R14.

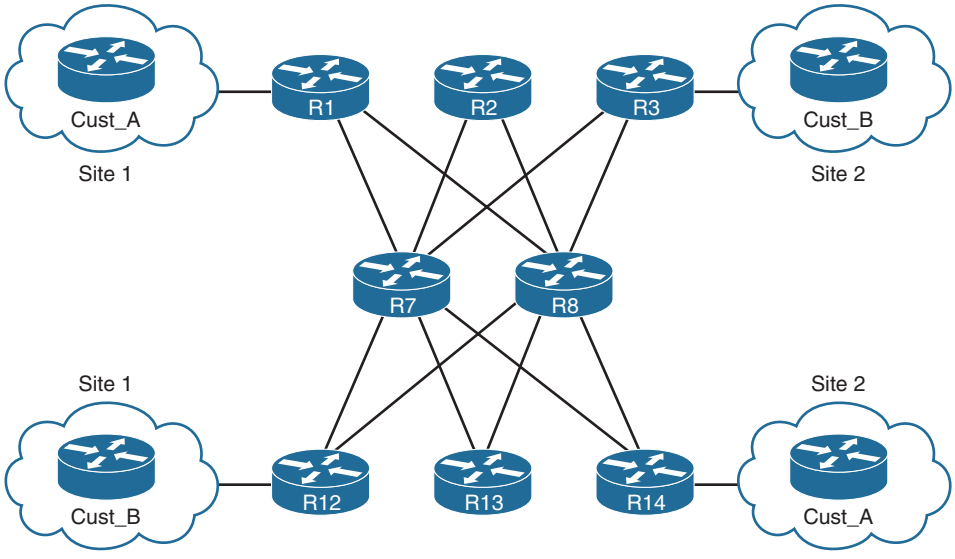

Figure 2-1 *MPLS VPN Topology for Customer A and Customer B*

Though the topology in Figure 2-1 is not large, it is hard to allocate this many routers in the lab to simulate this problem. So how can the lab be setup? Before setting up the lab, the most crucial step is to understand and list the requirements for the lab topology. There are two Provider Edge (PE) routers and two Customer Edge (CE) routers required at minimum. Because TE is the variable in this problem, it should be provisioned as well. Now the TE tunnel can be configured directly between the two PE routers, but in the preceding topology there are multiple paths from R1 to reach R14. Thus, a minimum of two distinct paths should be set up. Therefore, two more routers should be added in the core to simulate two distinct paths. This concludes the topology requirements to replicate the problem. Figure 2-2 shows the topology that can be used to set up the lab to replicate this problem.

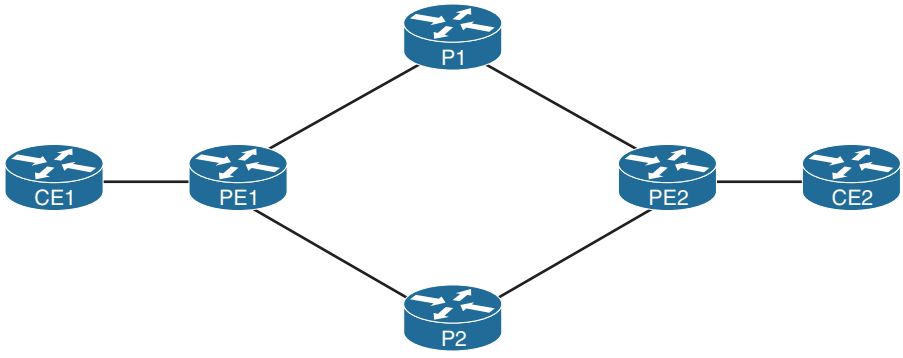

Figure 2-2 *MPLS VPN Lab Topology for Customer A*

After the lab topology is finalized, the next task is to determine the hardware and software requirements to replicate the problem. It is necessary to use the exact software version, because the similar problem may not exist in another software version or might have been fixed.

Choosing the hardware depends on various factors. For example, the problem might exist on a particular kind of hardware, but the same feature and configuration might run smoothly on a different hardware. In modern world network technologies, lots of features get programmed in hardware based on the instructions from software. This is because hardware-based packet switching is faster than software-based switching. Those processed at a hardware level are called Platform Dependent (PD) features and those processed at the software level are called the Platform Independent (PI) features. For instance, most of the control-plane functions are PI features, whereas most data plane functions are PD features on platforms such as Cisco ASR1000 or ASR9000 or even Nexus 7000 / 9000 series.

If the problem is related to a PI feature, real hardware equipment is not required. PI problems can be simulated in virtual environment using tools such as GNS3 or Cisco Virtual Internet Routing Labs (VIRL) by using the same version of software.

Note There are multiple simulators available, but Cisco VIRL provides a scalable, extensible network design and simulation environment. VIRL includes several Cisco Network Operating System virtual machines (IOSv, IOS-XRv, CSR1000v, NX-OSv, IOSvL2, and ASAv) and has the capability to integrate with third-party vendor virtual machines. It includes many unique capabilities, such as "live visualization," that provide the capability to create protocol diagrams in real-time from a running simulation. More information about VIRL can be found at http://virl.cisco.com.

For simulating PD problems, the exact hardware and software is required. The reason is that different line cards on a router have different architecture and different registers and asics, which are used to program the hardware. So based on the problem and the components involved in a feature, a choice has to be made between a physical hardware or a virtual environment.

Configuring Lab Devices

After the lab is set up with relevant software and hardware components, the next step is to configure the lab devices. It should be a close match to what is present in the production environment from the perspective of features being used.

In the topology shown in Figure 2-1, the ISP has Open Shortest Path First (OSPF) as its IGP, MPLS Label Distribution Protocol (LDP), MPLS Traffic Engineering (TE), and BGP vpnv4 address-family, along with Virtual Routing and Forwarding (VRF); all these features should be configured on the lab devices to match as closely as possible with the production devices. Whenever possible, using the exact configuration is preferred. Though you may need to change interface numbering to apply the configuration to the specific lab, but using the exact configuration sometimes catches problems associated with specific network addressing, especially for features such as BGP route-policy, access-lists, NAT, and so on. This also saves time by not having to reengineer the entire configuration in the lab.

These are the minimum features required to set up the lab. But sometimes just having the minimum configuration to bring up the lab devices is not enough. There may be other features configured on the router globally or in interface configuration mode that could add to the trigger of the problem. For example, a QoS policy configuration, though having no correlation with MPLS functionality, might add to the trigger of the problem. Whenever possible, using the exact configuration as that of the production is recommended.

It is also possible that in order to trigger the problem, the box needs to be loaded with configuration. Configuring various features on the device consumes more system resources, which can also play a vital role in triggering the problem. Another important factor that sometimes helps replicate the problem is by simulating traffic using traffic generator devices or software, such as Cisco's TREX, which is used for application simulation or from a third-party vendor like IXIA and Spirent that are highly capable of generating traffic. In addition, these devices help scale the environment because they can also simulate Layer 2 and Layer 3 protocols.

Note You can find more details about TREX Traffic Generator at http://trex-tgn.cisco.com.

Not everyone can afford IXIA or Spirent type of devices in their testing environment. Other alternative tools and applications are available online that can be used to generate traffic. One such application is Iperf. Iperf is commonly used to measure throughput of a network and is capable of creating TCP and User Datagram Protocol (UDP) streams. Some Cisco devices have a built-in tool, Test TCP (TTCP) utility, that helps simulate TCP streams in a client/server (Transmit/Receive) mode. Example 2-1 demonstrates how to use TTCP to simulate a TCP stream from router R1 to router R6. R6 is configured as the receive side, and R1 is configured as the transmit side.

Example 2-1 *TTCP on Cisco 7600 / ASR1000 Router*

```
! TTCP on Receive side
R6# ttcp
transmit or receive [receive]:
receive packets asynchronously [n]:
perform tcp half close [n]:
receive buflen [32768]:
bufalign [16384]:
bufoffset [0]:
port [5001]:
sinkmode [y]:
rcvwndsize [32768]:
ack frequency [0]:
delayed ACK [y]:
show tcp information at end [n]: y

ttcp-r: buflen=32768, align=16384/0, port=5001
rcvwndsize=32768, delayedack=yes  tcp
ttcp-r: accept from 12.12.12.1
ttcp-r: 23658496 bytes in 155593 ms (155.593 real seconds) (~148 kB/s) +++
ttcp-r: 7197 I/O calls
ttcp-r: 0 sleeps (0 ms total) (0 ms average)
Connection state is CLOSEWAIT, I/O status: 1, unread input bytes: 1
Connection is ECN Disabled
Mininum incoming TTL 0, Outgoing TTL 255
Local host: 6.6.6.6, Local port: 5001
Foreign host: 12.12.12.1, Foreign port: 58747
Connection tableid (VRF): 0

Enqueued packets for retransmit: 0, input: 0  mis-ordered: 0 (0 bytes)

Event Timers (current time is 0x31E05897):
Timer          Starts    Wakeups            Next
Retrans             1          0             0x0
TimeWait            0          0             0x0
AckHold          7198         27             0x0
SendWnd             0          0             0x0
KeepAlive        8273          0      0x31E142E8
GiveUp              0          0             0x0
PmtuAger            0          0             0x0
DeadWait            0          0             0x0
Linger              0          0             0x0
```

```
iss: 3992454486  snduna: 3992454487  sndnxt: 3992454487     sndwnd:   4128
irs: 1163586071  rcvnxt: 1187244569  rcvwnd:      32696  delrcvwnd:     72

SRTT: 37 ms, RTTO: 1837 ms, RTV: 1800 ms, KRTT: 0 ms
minRTT: 1 ms, maxRTT: 300 ms, ACK hold: 200 ms
Status Flags: passive open, retransmission timeout, gen tcbs
Option Flags: none

Datagrams (max data segment is 536 bytes):
Rcvd: 44795 (out of order: 7177), with data: 44764, total data bytes: 23658496
Sent: 51276 (retransmit: 0 fastretransmit: 0),with data: 0, total data bytes: 0
```

```
! TTCP on Transmit side
R1# ttcp
transmit or receive [receive]: transmit
Target IP address: 6.6.6.6
calculate checksum during buffer write [y]:
perform tcp half close [n]:
send buflen [32768]:
send nbuf [2048]:
bufalign [16384]:
bufoffset [0]:
port [5001]:
sinkmode [y]:
buffering on writes [y]:
show tcp information at end [n]: y

ttcp-t: buflen=32768, nbuf=2048, align=16384/0, port=5001  tcp  -> 6.6.6.6
ttcp-t: connect
ttcp-t: 23625728 bytes in 155584 ms (155.584 real seconds) (~147 kB/s) +++
ttcp-t: 722 I/O calls
ttcp-t: 0 sleeps (0 ms total) (0 ms average)
Connection state is ESTAB, I/O status: 1, unread input bytes: 0
Connection is ECN Disabled
Mininum incoming TTL 0, Outgoing TTL 255
Local host: 12.12.12.1, Local port: 58747
Foreign host: 6.6.6.6, Foreign port: 5001
Connection tableid (VRF): 0

Enqueued packets for retransmit: 24, input: 0  mis-ordered: 0 (0 bytes)

Event Timers (current time is 0x31E05892):
Timer          Starts    Wakeups            Next
Retrans          9470       1748      0x31E059BA
```

```
TimeWait            0           0           0x0
AckHold             0           0           0x0
SendWnd             0           0           0x0
KeepAlive           0           0           0x0
GiveUp              0           0           0x0
PmtuAger            0           0           0x0
DeadWait            0           0           0x0
Linger              0           0           0x0

iss: 1163586071  snduna: 1187232168  sndnxt: 1187244568     sndwnd:  32768
irs: 3992454486  rcvnxt: 3992454487  rcvwnd:       4128  delrcvwnd:      0

SRTT: 300 ms, RTTO: 303 ms, RTV: 3 ms, KRTT: 0 ms
minRTT: 0 ms, maxRTT: 843 ms, ACK hold: 200 ms
Status Flags: retransmission timeout
Option Flags: higher precendence

Datagrams (max data segment is 536 bytes):
Rcvd: 51252 (out of order: 0), with data: 0, total data bytes: 0
Sent: 45270 (retransmit: 1769 fastretransmit: 525),with data: 45268,
      total data bytes: 23926320
```

Note The receive node should always be configured first so that when the transmit direction is set up, the destination device is ready and listening on the port specified (in this case, port number 5001—the default port for ttcp).

This feature is also available on IOS XR but not on NX-OS. Example 2-2 demonstrates the use of **ttcp** cli on the IOS XR platform.

Example 2-2 *TTCP on IOS XR*

```
RP/0/0/CPU0:XR_R1# ttcp ?
  align         Align the start of buffers to this modulus (default 16384)
  buflen        Length of bufs read from or written to network (default 8192)
  debug         Enable socket debug mode(cisco-support)
  format        Format for rate: k,K = kilo{bit,byte}; m,M = mega; g,G = giga
  fullblocks    (Receiver) Only output full blocks as specified by buflen
  host          Host name or Ip address(cisco-support)
  multi         Number of connections(cisco-support)
  nbufs         (Transmitter) Number of source bufs written to network
                (default 2048)
  nobuffering   (Tranmitter) Don't buffer TCP writes (sets TCP_NODELAY socket
                option)(cisco-support)
```

```
  nofilter     Don't filter icmp errors(cisco-support)
  nonblock     Use non-blocking sockets(cisco-support)
  offset       Start buffers at this offset from the modulus (default 0)
  password     MD5 Password to be used for tcp connection(cisco-support)
  port         Port number to send to or listen at (default 5001)
  receive      Receive mode(cisco-support)
  sockbuf      Socket buffer size(cisco-support)
  source       Source/Sink a pattern to/from network(cisco-support)
  timeout      Stop listening after timeout seconds(cisco-support)
  touch        (Receiver) "touch": access each byte as it's read
  transmit     Transmit mode(cisco-support)
  udp          Use UDP instead of TCP(cisco-support)
  verbose      Verbose: print more statistics(cisco-support)
  vrfid        Use this VRF to connect(cisco-support)

RP/0/0/CPU0:XR_R1# ttcp receive
ttcp-r: thread = 1, buflen=8192, nbuf=2048, align=16384/0, port=5001  tcp
 ttcp-r: socket
! Output omitted for brevity
```

```
RP/0/0/CPU0:XR_R6# ttcp transmit verbose host 1.1.1.1
ttcp-t: thread = 1, buflen=8192, nbuf=2048, align=16384/0, port=5001  tcp  ->
  1.1.1.1
ttcp-t: socket
! Output omitted for brevity
```

The difference between the IOS and IOS XR command-line interface (CLI) is that in IOS XR there is no step-by-step method, but options are part of the command itself. If no options are specified, default values are taken. The TTCP cli on IOS XR has more options. For example, there is an option to specify the format of the transmit or received rate, MD5 password for the TCP connection, Source of TCP stream, and so on. In IOS XR, TTCP can be used to send a UDP stream as well, which is not possible with Cisco IOS.

All the previously discussed factors increase the chances of successful reproduction of the problem.

Triggering Events

The fun part actually begins after the lab is set up and configured. Based on the documentation of the series of events that occurred before the problem started, various triggers are tried in lab.

In Figure 2-1, a link flap appears to be the trigger of the problem. In any network, a link flap can occur every now and then. Therefore, it should not be assumed that triggering just one link flap would reproduce the problem in a lab environment. The problem can be due to a timing condition, and the trigger has to be tried multiple times before the device runs into a problem state.

For example, one of the most complex problems to replicate in a lab could be simulating a memory leak problem. The triggers (if known) have to be tried multiple times to replicate the problem in a lab. Repetitive attempts of the same trigger or a combination of different triggers have to be tried, which is a tedious task. For such situations, it is better to make use of automation, which could trigger the event continuously or at a certain time interval. Various scripting languages can be used for automation purposes—for example Perl, Expect, TCL, and the like.

If the trigger is required to be instantiated based on an event, it is a better option to use Cisco's Embedded Event Manager (EEM). With EEM, certain actions are configured, which get triggered based on an event. For example, an EEM script can be configured to capture a set of commands and save it in a file on bootflash wherever the router experiences a BGP session flap or a high CPU condition.

Note EEM is explained in Chapter 6, "Troubleshooting Platform Issues due to BGP."

Sniffer-Packet Capture

Troubleshooting at the packet level is complex, and it becomes more difficult when it is unknown what kind of packet is being received or transmitted. It is important to know if the packet is actually reaching the device. There are three perspectives:

- Transmitting router
- Receiving router
- Transmission media

This is where the concept of *sniffing* comes in. Sniffing is the technique of intercepting the traffic passing over the transmission media for protocol or packet analysis. A sniffing technique is not used just for network troubleshooting purposes but also by network security experts for analysis for any security loopholes. In sniffing, a packet capture tool such as Wireshark is installed on a PC that can be attached to a network device such as a switch. The switch, in most cases, is capable of mirroring the packets coming in on

a switch interface and sending it to the PC connected to the switch. Figure 2-3 shows how a sniffer capture is set up on a switch.

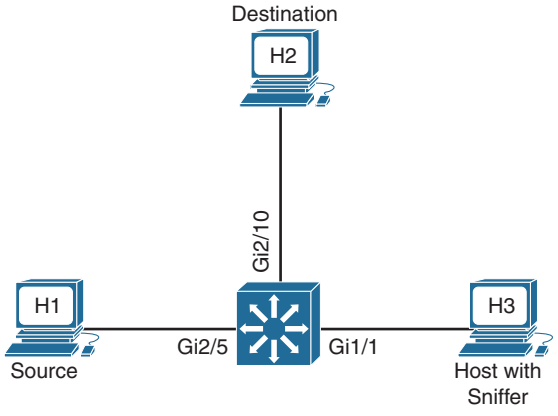

Figure 2-3 *Sniffer Setup on Switch*

On Cisco devices, the sniffing capability is called a Switched Port Analyzer (SPAN) feature. The source port is called a monitored port, and the destination port is called the monitoring port. SPAN functionality has variations in different platforms that are discussed in the following section.

SPAN on Cisco IOS

Almost all Cisco Catalyst switches, including multilayer switches/routers such as the Cisco 6500 or Cisco 7600 series, support the SPAN feature. Before configuring SPAN, the source and the destination interface should be identified. There could be one or more source interfaces from which the traffic can be spanned, but there can be only one destination interface. The SPAN can be configured using the command **monitor session** *session-number* [**source** | **destination**] **interface** *interface-id*.

Example 2-3 demonstrates setting up a SPAN session on Cisco 7600 series router. The same configuration applies to catalyst switches as well. The SPAN session 1 uses a source interface of GigabitEthernet2/5 and a destination interface as GigabitEthernet1/1. Thus, all the traffic coming on the interface GigabitEthernet2/5 is mirrored and sent to interface GigabitEthernet1/1, where a PC connected with the *Wireshark* software installed on it captures the packets received on the interfaces. Optionally, the direction of the packet that needs to be captured on the incoming interface can also be specified just after the interface name using option **[both | rx | tx]**. In this example, the source interface is a physical interface. The source interface can also be configured as a VLAN interface. Therefore, all traffic coming on the SVI is mirrored and sent to the configured destination interface.

Example 2-3 *SPAN Configuration*

```
R1# configure terminal
R1(config)# monitor session 1 source interface GigabitEthernet2/5 both
R1(config)# monitor session 1 destination interface GigabitEthernet1/1
```

The SPAN sessions can also be configured to capture the traffic based on the filter, such as VLAN or an access control list (ACL), by using the command **monitor session session-id filter [vlan** *vlan-id* **| ip access-group** *acl*]. This is useful in cases when it is required to capture specific traffic.

To verify the SPAN session, use the command **show monitor session** *session-number*. Example 2-4 displays the configured span session. Remember that after the destination interface is specified for the SPAN, no protocol will work on that port. The port will be working in promiscuous mode. Notice that the **show interface GigabitEthernet1/1** command's output displays that the port is in "monitoring" status.

Example 2-4 *Verifying SPAN Session*

```
R1# show monitor session 1
Session 1
---------
Type                   : Local Session
Source Ports           :
    Both               : Gi2/5
Destination Ports      : Gi1/1
MTU                    : 1464

Egress SPAN Replication State:
Operational mode       : Distributed
Configured mode        : Distributed
```

```
R1# show interface GigabitEthernet1/1
GigabitEthernet1/0/7 is up, line protocol is down (monitoring)
  Hardware is Gigabit Ethernet, address is 2c54.2d68.1207 (bia 2c54.2d68.1207)
  MTU 1998 bytes, BW 1000000 Kbit/sec, DLY 10 usec,
     reliability 255/255, txload 1/255, rxload 18/255
  Encapsulation ARPA, loopback not set
  Keepalive not set
  Full-duplex, 1000Mb/s, link type is auto, media type is 10/100/1000BaseTX SFP
! Output omitted for brevity
```

SPAN on Cisco IOS XR

The SPAN feature is not available on all IOS XR platforms. Traffic mirroring or SPAN capability is available only on ASR9000 and CRS routers. This feature was introduced on the Cisco Carrier Routing System (CRS) platform in XR 4.3.0 release and was introduced on the Aggregation Services Routers 9000 (ASR9000) series platform, starting with the XR 3.9.1 release, with some enhancements made in the 4.0.1 release. Cisco 12000 series—that is, the Gigabit Switch Router (GSR) router running IOS XR, does not have traffic mirroring capabilities. IOS XR supports two types of traffic mirroring methods:

- Layer 3 traffic mirroring
- ACL-based traffic mirroring

In Layer 3 traffic mirroring, the destination port is identified by an IP Address and not by an interface, because routing decides which interface the mirrored packets are actually sent over.

Note Layer 2 SPAN is not supported on Cisco Carrier Routing Services (CRS) routers.

Example 2-5 displays how to configure Layer 3 traffic mirroring configuration. In this example, a SPAN session named test is configured with the destination IP of 10.1.1.1. On the source port, which is interface GigabitEthernet0/1/3/1, the span session named TEST is applied with the direction set to capture ingress only packets; that is, **rx-only**. It is optional to specify how many bytes to mirror, and it is also optional to specify if the mirroring has to be done just for IPv4 or IPv6 packets.

Example 2-5 *Configuring Layer 3 Traffic Mirroring*

```
RP/0/RP0/CPU0:CRS(config)# monitor-session TEST ipv4
RP/0/RP0/CPU0:CRS(config-mon)# destination next-hop 10.1.1.1
RP/0/RP0/CPU0:CRS(config-mon)# exit
RP/0/RP0/CPU0:CRS(config)# interface gigabitEthernet0/1/3/1
RP/0/RP0/CPU0:CRS(config)# ip address 192.168.10.1 255.255.255.0
RP/0/RP0/CPU0:CRS(config-if)# monitor-session test ipv4 direction rx-only
RP/0/RP0/CPU0:CRS(config-if)# exit
RP/0/RP0/CPU0:CRS(config)# commit
```

To verify the status of the configured monitor-session, the **show monitor-session** *name* **status detail** command can be used. Example 2-6 displays the status of the monitor-session test with the command **show monitor-session** *name* **status detail errors**. The status should always be in Operational state, but if there is an error on the monitor-session, it is displayed in the command output.

Example 2-6 *Verifying Configured Traffic Mirroring Status*

```
RP/0/RP0/CPU0:CRS# show monitor-session test status detail
Monitor-session TEST (IPv4)
  Destination next-hop IPv4 address 10.1.1.1
  Source Interfaces
  -----------------
  GigabitEthernet0/1/3/0
    Direction: Rx-only
    ACL match: Enabled
    Portion:   Full packet
    Status:    Operational
```

ACL-based traffic mirroring adds an interesting flavor to IOS XR SPAN capabilities. The permit and deny statements determine the behavior of the regular traffic being forwarded or dropped. The **capture** keyword determines whether the packet is mirrored to the destination. But the main behavior lies on which direction the ACL is applied. If the ACL is on ingress direction, SPAN mirrors all traffic, including traffic dropped by an ACL. In other words, it always mirrors the traffic. If the ACL is on egress direction, it mirrors traffic if the regular traffic is forwarded via the **permit** statement and does not mirror if the regular traffic is dropped via the **deny** statement.

Example 2-7 displays how to configure ACL-based traffic mirroring. In this example, SPAN session TEST is created with the destination interface as GigabitEthernet0/1/0/2. Note that the destination interface can also be an ipv4 or ipv6 address. Next, an ACL named **SPAN_ACL** is created which allows traffic from any source, destined to host 1.1.1.10 address and denies the rest of the traffic.

Example 2-7 *Configuring Layer 3 ACL-Based Traffic Mirroring*

```
RP/0/RP0/CPU0:CRS(config)# monitor-session TEST
RP/0/RP0/CPU0:CRS(config-mon)# destination interface GigabitEthernet0/1/0/2
RP/0/RP0/CPU0:CRS(config-mon)# exit
RP/0/RP0/CPU0:CRS(config)# ipv4 access-list SPAN_ACL
RP/0/RP0/CPU0:CRS(config)# permit ipv4 any host 1.1.1.10 capture
RP/0/RP0/CPU0:CRS(config)# deny ipv4 any any
RP/0/RP0/CPU0:CRS(config)# interface gigabitEthernet0/1/3/0
RP/0/RP0/CPU0:CRS(config-if)# ip address 192.168.10.1 255.255.255.0
RP/0/RP0/CPU0:CRS(config-if)# monitor-session TEST
RP/0/RP0/CPU0:CRS(config-if)# acl
RP/0/RP0/CPU0:CRS(config-if)# ipv4 access-group SPAN_ACL ingress
RP/0/RP0/CPU0:CRS(config-if)# exit
RP/0/RP0/CPU0:CRS(config)# commit
```

Notice that there is a **capture** keyword at the end of the first ACL entry. It means that this traffic will be captured. Any other traffic mapped to the ACL entry that

does not have the **capture** keyword at the end is not sent to the sniffer. SPAN session **TEST** is then assigned under the source interface GigabitEthernet0/1/3/0 along with the **acl** subcommand. This command specifies that the mirrored traffic is according to the defined global interface ACL. Along with assigning the SPAN session TEST to the interface, ACL is also bound to the ingress direction of the interface.

ASR9000 also supports traffic mirroring for pseudowire and Layer 2–based traffic mirroring. Other XR platforms do not support this feature.

SPAN on Cisco NX-OS

The SPAN feature on Cisco Nexus OS (NX-OS) is not different from Cisco IOS. Depending on the hardware platform, NX-OS may run differently from IOS platforms for the number of interfaces per monitor session. For example, there can be 128 source interfaces and 32 destination interfaces configured per session. Two active sessions are supported for all virtual device contexts (VDC).

Note The number of active sessions, sources interfaces and destination interfaces supported per session can vary on different Nexus platforms. Platform documentation should be referenced before using the feature for any such limitations.

Example 2-8 shows the SPAN configuration on Cisco NX-OS. In Example 2-8, the **switchport monitor** command is configured on the destination interface. SPAN session is configured using the **monitor session** *session-number* command with the subcommands to configure source and destination interfaces.

Example 2-8 *SPAN Configuration on Cisco NX-OS*

```
N7k-1(config)# interface Ethernet4/3
N7k-1(config-if)# switchport
N7k-1(config-if)# switchport monitor
N7k-1(config-if)# no shut
N7k-1(config)# monitor session 1
N7k-1(config-monitor)# source interface Ethernet4/1
N7k-1(config-monitor)# destination interface Ethernet4/3
N7k-1(config-monitor)# no shut
N7k-1(config-monitor)# exit
```

Note The source interface can also be a range of interfaces or a range of VLANs.

Notice that under the monitor session configuration, the **no shutdown** command is configured. By default, the monitor session is in shutdown state and has to be manually unshut for SPAN session to function using the **no shut** command.

Example 2-9 displays the monitor session status. In this example, the rx, tx, and both fields are seen as interface Eth4/1. Direction of the source interface can be manually specified in configuration, but it will be set to both if not specified. There is also an option to filter VLANs under the monitor session using the **filter vlan** command.

Example 2-9 *Verifying SPAN Session*

```
N7k-1# show monitor session 1
   session 1
---------------
type              : local
state             : up
source intf       :
    rx            : Eth4/1
    tx            : Eth4/1
    both          : Eth4/1
source VLANs      :
    rx            :
    tx            :
    both          :
filter VLANs      : filter not specified
destination ports : Eth4/3
```

Note For more details on using SPAN on Cisco IOS, IOS XR, and NX-OS platforms, refer to the documentation at cisco.com.

Remote SPAN

Remote SPAN (RSPAN) is an extension of SPAN with a difference that the destination port where the host is attached to capture traffic is on a remote switch a few hops away. In Figure 2-4, the topology demonstrates a remote SPAN setup across multiple switches. Switch SW1 is the source switch, whereas SW3 is the destination switch.

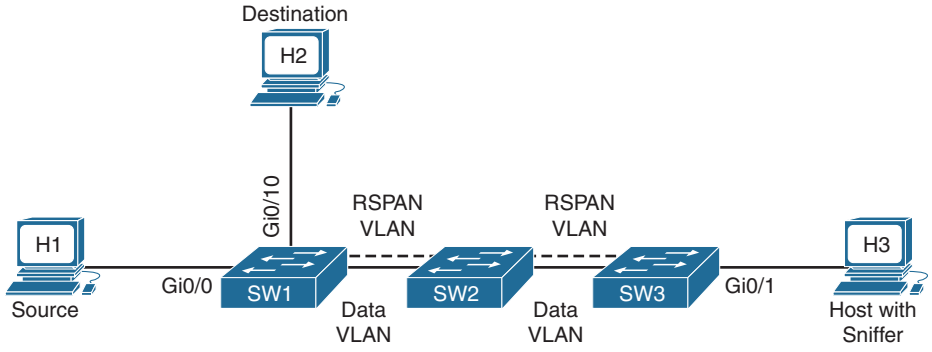

Figure 2-4 *Remote SPAN*

The VLAN used between the two switches to carry the SPAN traffic is called the RSPAN VLAN and is configured using the **remote-span** command under the vlan-config mode. The actual traffic flow is between the hosts connected on SW1, such as H1 and H2. The interface being spanned is GigabitEthernet0/0. On the source switch SW1, the destination interface is set to the RSPAN VLAN, whereas on the destination switch, the source interface is specified as the RSPAN VLAN. The reason is that when the packet arrives to the destination switch, the packet is being traversed across the RSPAN VLAN, and therefore the mirrored packets are received on that VLAN.

The RSPAN VLAN should be enabled on all the trunk ports on the switch in the middle. If the Layer 2 network has pruning enabled, then the RSPAN VLAN should be excluded from pruning using the command **switchport trunk pruning vlan remove** *rspan-vlan-id*.

Example 2-10 demonstrates the configuration of remote SPAN session on the source switch and the remote destination switch. In this example, the SW1 is attached to the monitored host on port GigabitEthernet0/0, with RSPAN VLAN 30, which spans across SW2 to reach SW3 where the host H3 is attached on port GigabitEthernet0/1 to capture the traffic.

Example 2-10 *RSPAN Configuration*

```
SW1# configure terminal
SW1(config)# vlan 30
SW1(config-vlan)# name RSPAN_Vlan
SW1(config-vlan)# remote-span
SW1(config-vlan)# exit
SW1(config)# monitor session 1 source interface GigabitEthernet0/0 rx
SW1(config)# monitor session 1 destination remote vlan 30
SW1(config)# end
```

```
SW3# configure terminal
SW3(config)# vlan 30
SW3(config-vlan)# name RSPAN_Vlan
SW3(config-vlan)# remote-span
SW3(config-vlan)# exit
SW3(config)# monitor session 1 destination interface GigabitEthernet0/1
SW3(config)# monitor session 1 source remote vlan 30
SW3(config)# end
```

The only problem that may arise with RSPAN is when the trunk between the two switches is not configured to allow the RSPAN VLAN. In that case, it will not function properly, and there could be a possible impact on the switch, because the CPU will keep dropping the spanned traffic.

Platform-Specific Packet Capture Tools

Most high-end routers and switches have in-built tools that are capable of capturing a packet at different ASIC levels within the router. These tools are capable of capturing either traffic destined to the devices or the transit traffic, which can be very useful during troubleshooting. With the use of the packet capture capability within the platform, it is easy to figure the actions being taken on a packet by each platform at various ASIC levels. These tools help figure out up to which hop the packet has reached in the network, and if a particular router/switch is dropping a packet, figure out where (on the ingress or egress line card or on the route processor) it is dropping it. These tools are helpful while troubleshooting complete packet loss, hardware programming issues, various routing protocol and QoS issues, as well as in situations where an external sniffer is not possible.

The following is a list of a few packet capture tools on various Cisco platforms:

- IOS/IOS XE
  - 7600/6500
    - ELAM
    - Mini Protocol Analyzer (MPA)
    - Netdr
  - ASR1K and other IOS and IOS XE platforms
    - Embedded Packet Capture
    - Packet Tracker (ASR1k)
- IOS XR
  - ASR9k
    - Network Processor Capture
  - CRS
    - Show captured packets
- NX-OS
  - Nexus 7K, 5K, 3K
    - Ethanalyzer
    - ELAM

Note Discussing all the packet capture tools is outside the scope of this book because many of these captures require deep platform architecture knowledge.

Netdr Capture

Netdr capture is a very handy tool to use on 6500 or 7600 Multilayer Switching (MLS) platforms. This tool is available starting with the 12.2(18)SXF release. Netdr capture can be run on the Route Processor (RP) and the Switch Processor (SP) of various supervisor cards, such as SUP720/RSP720/SUP2T, or on line cards with the Distributed Forwarding Card (DFC) installed on them, such as ES+, ES20, or WS-67XX cards with DFC module.

The DFC cards are similar to the supervisor card where all the Cisco Express Forwarding (CEF) and MLS information is downloaded to the line card. So all the lookups for the packets occur on the line card itself rather than punting the packet to the supervisor card for lookup on where to forward the packet.

The Netdr capture allows up to 4096 packets, after which the captured packets are overwritten. This tool is helpful in cases where there is a high CPU condition on the router due to traffic interrupts, and multiple packets are hitting the CPU. It can also be useful to capture the incoming or outgoing packets that are destined to the CPU, such as control plane packets. In other words, netdr is helpful only for capturing packets to or from Route Processor (RP) or Switch Processor (SP). For example, Netdr capture can help confirm that BGP packets are being received or sent out, and when received by the line card, whether they are being sent toward the Supervisor card.

Example 2-11 illustrates the use of netdr capture in rx direction on the RP to capture BGP packets coming from peer IP address 10.1.13.1. After waiting for few seconds (based on the packet rate hitting the CPU), use the **show netdr captured-packets** command to view the captured packets hitting the CPU.

In Example 2-11, the two captured packets are BGP packets based on the highlighted information. VLAN information is also valuable. On Cisco 6500 / 7600 platforms, every physical interface is allocated on an internal VLAN, which can be viewed using the **show vlan internal usage** command. This information locates the interface the packet is destined to. In this case, the destination index is *0x380*, which indicates that the packet is destined to CPU. The source VLAN is 1016, which resolves to interface TenGig1/4.

Example 2-11 *Netdr Capture on Cisco 7600 Platform*

```
7600_RTR# debug netdr capture ?
  acl                     (11) Capture packets matching an acl
  and-filter              (3) Apply filters in an and function: all must match
  continuous              (1) Capture packets continuously: cyclic overwrite
  destination-ip-address  (10) Capture all packets matching ip dst address
  dstindex                (7) Capture all packets matching destination index
  ethertype               (8) Capture all packets matching ethertype
  interface               (4) Capture packets related to this interface
  or-filter               (3) Apply filters in an or function: only one must
                          match
  rx                      (2) Capture incoming packets only
  source-ip-address       (9) Capture all packets matching ip src address
  srcindex                (6) Capture all packets matching source index
  tx                      (2) Capture outgoing packets only
```

```
  vlan                    (5) Capture packets matching this vlan number
  <cr>
7600_RTR# debug netdr capture source-ip-address 10.1.13.1
```

```
7600-RTR# show netdr captured-packets
A total of 2 packets have been captured
The capture buffer wrapped 0 times
Total capture capacity: 4096 packets
------- dump of incoming inband packet -------
interface Te1/4, routine process_rx_packet_inline, timestamp 15:20:07.111
dbus info: src_vlan 0x3F8(1016), src_indx 0x3(3), len 0x4F(79)
  bpdu 0, index_dir 0, flood 0, dont_lrn 0, dest_indx 0x380(896)
  48020400 03F80400 00030000 4F000000 00060408 0E000008 00000000 0380E753
destmac 00.1E.F7.F7.16.80, srcmac 84.78.AC.0F.76.C2, protocol 0800
protocol ip: version 0x04, hlen 0x05, tos 0xC0, totlen 61, identifier 7630
  df 1, mf 0, fo 0, ttl 1, src 10.1.13.1, dst 10.1.13.3
  tcp src 179, dst 11655, seq 788085885, ack 4134684341, win 17520 off 5 checksum
0x5F4E ack psh

------- dump of incoming inband packet -------
interface Te1/4, routine process_rx_packet_inline, timestamp 15:20:07.111
dbus info: src_vlan 0x3F8(1016), src_indx 0x3(3), len 0x40(64)
  bpdu 0, index_dir 0, flood 0, dont_lrn 0, dest_indx 0x380(896)
  50020400 03F80400 00030000 40000000 00060408 0E000008 00000000 0380639F
destmac 00.1E.F7.F7.16.80, srcmac 84.78.AC.0F.76.C2, protocol 0800
protocol ip: version 0x04, hlen 0x05, tos 0xC0, totlen 40, identifier 7631
  df 1, mf 0, fo 0, ttl 1, src 10.1.13.1, dst 10.1.13.3
  tcp src 179, dst 11655, seq 788085906, ack 4134684342, win 17520 off 5 checksum
0x6670 ack
```

```
7600-RTR# show vlan internal usage | in 1016
1016 TenGigabitEthernet1/4
```

To determine the packets hitting the CPU, use the command **show ibc | in rate** on Cisco 7600 platform which displays the ingress (rx) and egress (tx) packet rate. The rx packets are the ones punted to the CPU for processing, whereas the tx packets are processed or generated from the CPU.

Note Rx mode captures packets coming from ingress line card toward the supervisor card, and the tx mode captures packets leaving the supervisor card (from the supervisor card toward egress line card).

Embedded Packet Capture

SPAN is not supported on all the routers and switches. Even if supported, it may not be possible to run it in live production environments because of a company's security policies. It may take time to arrange for a sniffing device. Cisco created the Embedded Packet Capture (EPC) tool, which is capable of capturing packets on a router's buffer and later exporting the packets in PCAP format, which is suitable for analysis using Wireshark. Therefore, the packets captured can be later analyzed using Wireshark.

EPC allows for packet data to be captured at various points in the CEF packet-processing path—flowing through, to, and from a Cisco router. It is supported on various Cisco Routers, such as Cisco 800, 1900, 3900, 7200, and ASR 1000. The rate at which the packets are captured can be throttled by specifying a sampling interval, maximum packet capture rate, and even limiting the capture to interested packets by using an access control list (ACL). You can use EPC in five simple steps:

Step 1. Define capture buffer

Step 2. Define capture point

Step 3. Associate capture buffer and capture point

Step 4. Start and stop capture

Step 5. Display/export the data

Note In Step 3, associating the capture buffer and capture point depends on the platform and OS version. This step is not really required on the ASR1k platform.

Example 2-12 demonstrates EPC on the ASR1k platform and on 7200 series routers for capturing BGP packets. In this example, the buffer is set up to capture the packets matching the ACL named MYACL. This ACL is configured to capture BGP packets. If the size is not specified, the default value is taken by the base IOS.

Example 2-12 *Embedded Packet Capture*

```
ASR1k(config)# ip access-list extended MYACL
ASR1k(config-acl)# permit tcp any eq bgp any
ASR1k(config-acl)# permit tcp any any eq bgp
ASR1k# monitor capture CAP1 buffer circular packets 1000
ASR1k# monitor capture CAP1 buffer size 10
ASR1k# monitor capture CAP1 interface GigabitEthernet0/0/0 in
ASR1k# monitor capture CAP1 access-list MYACL
ASR1k# monitor capture CAP1 start
```

```
ASR1k# monitor capture CAP1 stop
ASR1k# monitor capture CAP1 export bootflash:cap1.pcap
```

```
7200_RTR(config)# ip access-list extended MYACL
7200_RTR(config-acl)# permit tcp any eq bgp any
7200_RTR(config-acl)# permit tcp any any eq bgp
7200_RTR# monitor capture buffer CAP1 circular
7200_RTR# monitor capture buffer CAP1 size 1024
7200_RTR# monitor capture buffer CAP1 filter access-list MYACL
7200_RTR# monitor capture point ip cef CPT1 GigabitEthernet0/0 in
7200_RTR# monitor capture associate CPT1 CAP1
7200_RTR# monitor capture point start CPT1
7200_RTR# monitor capture point stop CPT1
7200_RTR# monitor capture buffer CAP1 export tftp://10.1.1.1/cap1.pcap
```

The capture can be stopped either by setting the duration or the limit of the number of packets or by stopping the capture manually if the packet is captured. The buffer can then be exported to a bootflash: or an external tftp: / ftp: server. Saving the buffer capture to bootflash: is not available on all platforms and IOS versions.

The packet can be viewed on the terminal as well using the **show monitor capture buffer** *name* **dump** (**show monitor capture** *name* **buffer dump** for ASR1k) command, but it is in raw format and therefore needs to be decoded from a hex value to an understandable format. Example 2-13 displays the packet on the terminal session. In this example, **AABBCC00 0800** is the destination MAC, **AABBCC00 0700** is the source MAC, **0707 0707 (7.7.7.7)** in the third row is the source IP, and **0808 0808 (8.8.8.8)** is the destination IP. **0800** in the second row means that it is an IPv4 packet. **4A07** (18951) is the Source port, whereas **00B3** (179) is the destination port. From 9372 in the third row until 0000 in the fourth row are the various TCP fields, such as SEQ number, ACK number, Window size, and so on. **FFFF FFFFFFF FFFFFFFF FFFFFFFF** at the end is the BGP marker.

Example 2-13 *BGP Packet Captured Using EPC*

```
7200_RTR# show monitor capture buffer CAP1 dump
16:25:44.938 JST Aug 21 2015 : IPv4 LES CEF    : Gig0/0 None

F19495B0:                    AABBCC00 0800AABB          *;L...*;
F19495C0: CC000700 08004540 003B1C5D 4000FE06  L.....E@.;.]@.~.
F19495D0: 42020707 07070808 08084A07 00B39372  B.........J..3.r
F19495E0: FFE37CDC E3D35018 3D671161 0000FFFF  .c|\cSP.=g.a....
F19495F0: FFFFFFFF FFFFFFFF FFFFFFFF FD        ............}
```

Note The easiest method to analyze the EPC capture is by exporting it a remote server and reading the .pcap file using Wireshark.

Ethanalyzer

Ethanalyzer is a NX-OS implementation of TShark. TShark is a terminal version of Wireshark. It is capable of capturing inband and management traffic on all Nexus platforms. Ethanalyzer can be simply configured in three simple steps, as shown:

Step 1. Define Capture Interface.

Step 2. Define Filters. Set Capture Filter. Set Display Filter.

Step 3. Define Stop Criteria.

Starting with the capture interface, there are three kinds of capture interfaces:

- **Mgmt:** Captures traffic on Mgmt0 interface of the switch
- **Inbound-Hi:** Captures high-priority control packets on the inband, such as Spanning Tree Protocol (STP), Link Aggregation Control Protocol (LACP), Cisco Discovery Protocol (CDP), Data Center Bridging Exchange (DCBX), Fibre Channel, and Fibre Channel over Ethernet (FCOE)
- **Inbound-Lo:** Captures low-priority control packets on the inband, such as IGMP, TCP, UDP, IP, and ARP traffic.

The next step is setting the filters. With a working knowledge of Wireshark, it is fairly simple to configure filters for Ethanalyzer. There are two kinds of filters that can be set up for configuring Ethanalyzer: capture filter and display filter. As the name suggests, when capture filter is set, only frames matching the filter are captured. The display filter is used to display the packets matching the filter from the captured set of packets. That means Ethanalyzer captures other frames that are not matching the display filter but are not displayed in the output.

Ethanalyzer, by default, stops after capturing 10 frames. This value can be changed by setting the **limit-captured-frames** option where 0 means no limit. Example 2-14 illustrates how to configure Ethanalyzer for capturing BGP packets. In the Ethanalyzer output, various captured BGP packets can be seen, such as the BGP OPEN message, KEEPALIVE message, and UPDATE message.

Example 2-14 *Ethanalyzer on NX-OS*

```
N3K-1# ethanalyzer local interface inbound-hi display-filter "bgp"
limit-captured-frames 0

Capturing on 'eth4'
1 wireshark-cisco-mtc-dissector: ethertype=0xde09, devicetype=0x0
wireshark-broadcom-rcpu-dissector: ethertype=0xde08, devicetype=0x0
<snip>
2   81 2015-09-01 04:50:34.115833 192.168.10.2 -> 192.168.10.1 BGP 236 OPEN Message
5   86 2015-09-01 04:50:34.259108 192.168.10.1 -> 192.168.10.2 BGP 200 OPEN Message
 87 2015-09-01 04:50:34.259440 192.168.10.1 -> 192.168.10.2 BGP 149 KEEPALIVE Me
ssage
 88 2015-09-01 04:50:34.271319 192.168.10.2 -> 192.168.10.1 BGP 185 KEEPALIVE Me
ssage
6   92 2015-09-01 04:50:35.272488 192.168.10.1 -> 192.168.10.2 BGP 178 UPDATE Messa
ge, KEEPALIVE Message
8   93 2015-09-01 04:50:35.288438 192.168.10.2 -> 192.168.10.1 BGP 214 UPDATE Messa
ge, KEEPALIVE Message
 94 2015-09-01 04:50:35.288813 192.168.10.2 -> 192.168.10.1 BGP 214 UPDATE Messa
ge, KEEPALIVE Message
```

Example 2-14 displayed various packets with a brief description of the packet. To view the detailed view of the packet, use the **detail** keyword as shown in Example 2-15. The Ethanalyzer capture with the **detail** keyword has a similar view of Wireshark but on a terminal. The example shows the packet view, such as the source and destination mac and source and destination IP. Because this is a BGP control plane packet, the Differentiated Services Code Point (DSCP) value is seen as CS6, and at the end of the captured packet is a BGP KEEPALIVE Message.

Example 2-15 *Detailed Ethanalyzer Output*

```
N3K-1# ethanalyzer local interface inbound-hi display-filter "bgp"
limit-captured-frames 0 detail

Capturing on 'eth4'
wireshark-cisco-mtc-dissector: ethertype=0xde09, devicetype=0x0
wireshark-broadcom-rcpu-dissector: ethertype=0xde08, devicetype=0x0
Frame 8: 149 bytes on wire (1192 bits), 149 bytes captured (1192 bits) on interf
ace 0
    Interface id: 0
    Encapsulation type: Ethernet (1)
    Arrival Time: Sep  1, 2015 04:51:35.283124000 UTC
    [Time shift for this packet: 0.000000000 seconds]
```

```
    Epoch Time: 1441083095.283124000 seconds
    [Time delta from previous captured frame: 0.150360000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 1.440361000 seconds]
    Frame Number: 8
    Frame Length: 149 bytes (1192 bits)
    Capture Length: 149 bytes (1192 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:vlan:rcpu:eth:ip:tcp:bgp]
Ethernet II, Src: 02:10:18:97:3f:21 (02:10:18:97:3f:21), Dst: c0:8c:60:a8:ac:42
(c0:8c:60:a8:ac:42)
    Destination: c0:8c:60:a8:ac:42 (c0:8c:60:a8:ac:42)
        Address: c0:8c:60:a8:ac:42 (c0:8c:60:a8:ac:42)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory
 default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Source: 02:10:18:97:3f:21 (02:10:18:97:3f:21)
        Address: 02:10:18:97:3f:21 (02:10:18:97:3f:21)
        .... ..1. .... .... .... .... = LG bit: Locally administered address (th
is is NOT the factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: 802.1Q Virtual LAN (0x8100)
802.1Q Virtual LAN, PRI: 0, CFI: 0, ID: 4048
    000. .... .... .... = Priority: Best Effort (default) (0)
    ...0 .... .... .... = CFI: Canonical (0)
    .... 1111 1101 0000 = ID: 4048
    Type: Unknown (0xde08)
! Output omitted for brevity
Ethernet II, Src: c0:8c:60:a8:ac:41 (c0:8c:60:a8:ac:41), Dst: c0:8c:60:a8:68:01
(c0:8c:60:a8:68:01)
    Destination: c0:8c:60:a8:68:01 (c0:8c:60:a8:68:01)
        Address: c0:8c:60:a8:68:01 (c0:8c:60:a8:68:01)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory
 default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Source: c0:8c:60:a8:ac:41 (c0:8c:60:a8:ac:41)
        Address: c0:8c:60:a8:ac:41 (c0:8c:60:a8:ac:41)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory
 default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IP (0x0800)
Internet Protocol Version 4, Src: 192.168.10.1 (192.168.10.1), Dst: 192.168.10.2
 (192.168.10.2)
```

```
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0xc0 (DSCP 0x30: Class Selector 6; ECN: 0x00:
 Not-ECT (Not ECN-Capable Transport))
        1100 00.. = Differentiated Services Codepoint: Class Selector 6 (0x30)
        .... ..00 = Explicit Congestion Notification: Not-ECT (Not ECN-Capable T
ransport) (0x00)
    Total Length: 71
    Identification: 0x356e (13678)
    Flags: 0x00
        0... .... = Reserved bit: Not set
        .0.. .... = Don't fragment: Not set
        ..0. .... = More fragments: Not set
    Fragment offset: 0
    Time to live: 64
    Protocol: TCP (6)
    Header checksum: 0xaf2f [correct]
        [Good: True]
        [Bad: False]
    Source: 192.168.10.1 (192.168.10.1)
    Destination: 192.168.10.2 (192.168.10.2)
Transmission Control Protocol, Src Port: 64176 (64176), Dst Port: 179 (179), Seq
: 1, Ack: 1, Len: 19
    Source port: 64176 (64176)
    Destination port: 179 (179)
    [Stream index: 0]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 20    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    Header length: 32 bytes
    Flags: 0x018 (PSH, ACK)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 1... = Push: Set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
    Window size value: 17376
    [Calculated window size: 17376]
    [Window size scaling factor: -1 (unknown)]
```

```
    Checksum: 0x0139 [validation disabled]
        [Good Checksum: False]
        [Bad Checksum: False]
! Output omitted for brevity
            Kind: Timestamp (8)
            Length: 10
            Timestamp value: 10033933
            Timestamp echo reply: 13878630
    [SEQ/ACK analysis]
        [Bytes in flight: 19]
Border Gateway Protocol - KEEPALIVE Message
    Marker: ffffffffffffffffffffffffffffffff
    Length: 19
    Type: KEEPALIVE Message (4)
! Output omitted due to brevity
```

Logging

Network issues are hard to troubleshoot and investigate if there is not any information found on the device. For instance, if an OSPF adjacency goes down and there is not a correlating alert, it will be hard to find out what caused the problem and when it happened. For these reasons, logging is very important. All Cisco routers and switches support logging functionality. Logging capabilities are also available for specific features and protocols. For example, logging can be enabled for BGP neighborship state changes or OSPF neighborship state changes.

Table 2-1 lists the various logging levels that can be configured.

Table 2-1 *Logging Levels*

Level Number	Level Name
0	Emergency
1	Alert
2	Critical
3	Errors
4	Warnings
5	Notifications
6	Informational
7	Debugging

When the higher value is set; the lower logging level is enabled by default. That means if the logging level is set to 5, Notifications, it implies that all events falling under the category from 0 to 5—that is, from Emergency to Notifications—are logged. For troubleshooting purpose, it is always good to set the logging level to 7, which is debugging.

There are multiple logging options available on Cisco devices:

- Console logging
- Buffered logging
- Logging to syslog server

Console logging is important in conditions where the device is experiencing crashes or a high CPU condition. Access to the terminal session via Telnet or SSH is not available, but it is not a good practice to have it enabled when running debugs. Some debug outputs are chatty and can flood the device console. As a best practice, console logging should always be disabled when running debugs. Example 2-16 illustrates how to enable console logging on various Cisco platforms.

Example 2-16 *Configuring Console Logging*

```
IOS_R2(config)# logging console informational
```

```
RP/0/0/CPU0:XR_R3(config)# logging console informational
RP/0/0/CPU0:XR_R3(config)# commit
```

```
Nexus-R1(config)# logging console ?
   <CR>
   <0-7>  0-emerg;1-alert;2-crit;3-err;4-warn;5-notif;6-inform;7-debug
 Nexus-R1(config)# logging console 6
```

Buffered logging, on the other hand, is useful while running debugs and for more persistent logging as compared to console logging. The default buffer size is 4096 bytes, which gets filled very quickly, and the logs become overridden when the value is exceeded. The buffer size can be increased to a higher value, such as 1000000. Example 2-17 demonstrates how to configure buffered logging with the logging level as debugging.

Example 2-17 *Configuring Buffered Logging*

```
IOS_R2(config)# logging buffered debugging 1000000
```

```
RP/0/0/CPU0:XR_R3(config)# logging buffered 1000000 severity debugging
RP/0/0/CPU0:XR_R3(config)# commit
```

```
Nexus-R1(config)# logging buffered 7
Nexus-R1(config)# logging buffered 1000000
```

Although increasing the buffer size always helps, it is better to limit the number of messages to be logged in the buffer. This should be done when running chatty debugs such as **debug ip packet**. It is very important to understand the impact of a debug command and should be tested before any of the debug commands are run in a production environment. If chatty debugs are enabled on the router, they can lead to a high CPU condition on the router, which may in turn cause loss of management access to the router and other critical services getting impacted. In such situations, reloading the device is the only option to recover and gain back the management access. If there are dual route processors (RP) present on the router running in stateful switchover (SSO) mode, an RP switchover would help as well.

The rate-limit value can be set for all messages or for the messages sent to the console. This can be achieved using the **logging rate-limit** command. Setting the rate-limiter for the logs may cause some of the crucial messages to be missed but can be helpful when chatty debugs are enabled, which can flood the console or the terminal session (vty session). Example 2-18 demonstrates setting the logging rate limiter using the **logging rate-limit** command. In this example, the logging messages are rate limited to 100 messages, except for debug messages. Similarly, console messages can be rate limited using the **console** keyword with the command.

Example 2-18 *Logging Rate Limiter*

```
IOS_R2(config)# logging rate-limit ?
  <1-10000>  Messages per second
  all        Rate limit all messages, including debug messages
  console    Rate limit only console messages
IOS_R2(config)# logging rate-limit 100 except debugging
```

The most persistent form of logging is using a syslog server to log all the device logs. A syslog server can be anything from a text file to a custom application that actively stores device logging information in a database. Unless there is a packet loss that occurs in the network, or the device is continuously experiencing a high CPU condition, such as 99% or 100% for a long period of time, the device keeps logging the information to the syslog server. During the packet loss or high CPU conditions, there are chances of the syslog server logging or SNMP traps getting disrupted.

Example 2-19 illustrates syslog logging configuration. Before configuring syslog-based logging on Cisco IOS, enable the **service timestamps [log | debug] [datetime | uptime] [localtime] [show-timezone]** command for the logging messages. This ensures that all log messages have timestamps and helps in performing an investigation of the log messages. The timestamp can be set for a log message or a debug log. The log can either be viewed in Date-Time format or just-in-time format starting from system uptime. The CLI also provides an option to use the local time zone using the **localtime** keyword and also to display the time zone using the **show-timezone** keyword. If there is a separate interface for management traffic, enable the syslog server logging using that interface as the source interface, as shown in Example 2-19.

Example 2-19 *Syslog Logging Configuration*

```
IOS_R2(config)# service timestamps log datetime
IOS_R2(config)# logging host 10.1.1.1
IOS_R2(config)# logging trap 7
IOS_R2(config)# logging source-interface GiabitEthernet0/1
```

```
RP/0/0/CPU0:XR_R3(config)# logging 10.1.1.1
RP/0/0/CPU0:XR_R3(config)# logging trap
RP/0/0/CPU0:XR_R3(config)# logging hostnameprefix 100.1.1.1
RP/0/0/CPU0:XR_R3(config)# logging source-interface MgmtEth0/RSP0/CPU0/0
RP/0/0/CPU0:XR_R3(config)# commit
```

```
Nexus-R1(config)# logging server 10.1.1.1 7
Nexus-R1(config)# logging timestamp [microseconds | milliseconds | seconds]
```

Generally, the management interfaces are configured with a management **vrf**. In such cases, the syslog host has to be specified using the **logging host vrf** *vrf-name ip-address* command on IOS or **logging hostnameprefix** *ip-address* **use-vrf** *vrf-name* command on NX-OS, so that the router knows from which VRF routing table the server is reachable. If the vrf option is not specified, the system does a lookup in the default vrf; that is, the global routing table.

In Nexus platforms, there is another option to redirect debug output in a file, which is useful when running debugs and segregating the debug outputs with regular log messages. This feature can be used using the **debug logfile** *file-name* **size** *size* command. Example 2-20 demonstrates the use of the **debug logfile** command for capturing debugs in a log file. In this example, a debug log file named *bgp_dbg* is created with size of 10000 bytes. The size of the log file can range from 4096 bytes to 4194304 bytes. After the log file is created, all the debugs that are enabled are logged under the log file.

Example 2-20 *Capturing Debug in a Logfile on NX-OS*

```
Nexus-R1# debug logfile bgp_dbg size 100000
Nexus-R1# debug ip bgp update
```

The NX-OS software creates the logfile in the log: file system root directory and therefore all the created logfiles can be viewed using **dir log:**. After the debug logfile is created, the respective debugs can be enabled, and all the debug outputs are redirected to the debug logfile. To view the contents of the logfile, use the **show debug logfile** *file-name* command.

Event Monitoring/Tracing

Event tracing is an important feature that helps collect event-related information for various protocols. Event traces generate debug-like messages based on the events without enabling debugs and is stored in the memory of the router. There is no debug output

or syslog message generated for the event traces. These traces run in the background, therefore having the least impact on the router. There are no specific event traces or event history logs for BGP in IOS, but they are available in XR and NX-OS platforms.

Adjacency and CEF-related event traces can be collected on IOS devices, which are useful while troubleshooting forwarding issues or failed peering issues due to missing adjacency. Example 2-21 displays the various configuration options for event traces on Cisco IOS. In this example, adjacency related event trace is enabled.

Example 2-21 *Adjacency Event Trace Configuration*

```
IOS_R2(config)# monitor event-trace ?
  ac                AC traces
  adjacency         Adjacency Events
  all-traces        Configure merged event traces
  arp               ARP Events
  atom              AToM traces
  c3pl              Group traces
  cce               Group traces
  cef               CEF traces
! Output omitted for brevity
IOS_R2(config)# monitor event-trace adjacency
```

The event trace for adjacency can start logging an event whenever there is a change in any adjacency on the router. For example, bringing up a subinterface on the router, such as GigabitEthernet0/1.100, and then trying to form adjacency over that interface can trigger the logging for the adjacency in event traces. Example 2-22 demonstrates adjacency event traces generated for a subinterface. In this example, when the ping test is performed and the adjacency event traces show that router requests to add ARP, after the ARP is learned, it forms the adjacency for IP address 100.1.1.12 over the subinterface GigabitEthernet0/1.100.

Example 2-22 *Event Traces Showing Adjacency Formation*

```
IOS_R2# show run int GigabitEthernet0/1.100
interface GigabitEthernet0/1.100
  encapsulation dot1Q 100
  ip address 100.1.1.1 255.255.255.252
```

```
IOS_R2# ping 100.1.1.2
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 100.1.1.2, timeout is 2 seconds:
.!!!!
Success rate is 80 percent (4/5), round-trip min/avg/max = 7/10/14 ms
```

```
IOS_R2# show monitor event-trace adjacency all
! Output omitted for brevity
05:19:12.964: GLOBAL: adj mgr notified of fibidb state change int
                      GigabitEthernet0/1 to up [Ignr]
05:19:16.906: GLOBAL: adj mgr notified of fibidb state change int
                      GigabitEthernet0/1.100 to up [Ignr]
05:23:04.952: ADJ: IP 100.1.1.2 GigabitEthernet0/1.100: request to add ARP [OK]
05:23:04.952: ADJ: IP 100.1.1.2 GigabitEthernet0/1.100: update oce
                      bundle,  IPv4 incomplete adj oce [OK]
05:23:04.952: ADJ: IP 100.1.1.2 GigabitEthernet0/1.100: allocate [OK]
05:23:04.952: ADJ: IP 100.1.1.2 GigabitEthernet0/1.100: add source ARP [OK]
05:23:04.952: ADJ: IP 100.1.1.2 GigabitEthernet0/1.100: incomplete
                      behaviour change to drop [OK]
05:23:04.952: ADJ: IP 100.1.1.2 GigabitEthernet0/1.100: request to
                      update [OK]
05:23:04.952: ADJ: IP 100.1.1.2 GigabitEthernet0/1.100: update oce
                      bundle,  IPv4 no fixup, no redirect adj oce [OK]
05:23:04.953: ADJ: IP 100.1.1.2 GigabitEthernet0/1.100: update [OK]
```

If the adjacency is not formed or an incomplete ARP is seen in the **show ip arp** command output, no events for the adjacency getting completed will be seen in the event traces. This indicates that there is either a Layer 1 issue or a misconfiguration, such as VLAN Id not allowed on the switch in the middle. Similarly, event traces can be enabled for other features and protocols like EIGRP, MPLS, and ATOM, although it is not necessary to enable all the traces. The event traces for specific features and protocols can be turned on when a problem is seen on the device with respect to those features. In Cisco IOS, the size of the event traces is limited by default and thus, if the traces are captured after a certain time of the problem, they might not be of much use because the logs get rolled over. The size of the event traces can be increased, but the maximum number of entries supported is 1 million entries. Alternatively, if the problem is occurring frequently for features for which event traces are available, the event trace should be enabled for that feature just before the problem occurs and can be disabled after relevant logs have been captured. This will save system resources as well.

In IOS XR, traces are available for respective protocols and are very useful during troubleshooting. For example, in IOS XR, BGP traces can be verified when troubleshooting a peering issue for neighbor. Example 2-23 displays the BGP trace for a neighborship coming up. The **show bgp trace** command available for BGP captures all the activities taking place in BGP. At the beginning, the trace shows an OPEN message being sent out for peer IP 192.168.200.1. At the end, the trace log shows the neighborship being established.

Example 2-23 *BGP Neighborship Event Trace on IOS XR*

```
RP/0/0/CPU0:XR_R3# show bgp trace
! Output omitted for brevity
17:55:22.373 default-bgp/spkr-tr2-err 0/0/CPU0 t14 [ERR][GEN]:1300: OPEN from
  '192.168.200.1' has unrecognized cap code/len 70/0 - Ignored
17:55:22.373 default-bgp/spkr-tr2-upd 0/0/CPU0 t14 [UPD]:1820: Updgrp change
 scheduled after open processing: nbr=192.168.200.1, nbrfl=0x8310000
17:55:22.373 default-bgp/spkr-tr2-gen 0/0/CPU0 t14 [GEN]:546: nbr 192.168.200.1,
 old state 4, new state 5, fd type 1, fd 124
17:55:22.373 default-bgp/spkr-tr2-gen 0/0/CPU0 t14 [GEN]:2295: calling bgp_send_kee-
  palive, nbr 192.168.200.1, loc 1, data 0,0
17:55:22.373 default-bgp/spkr-tr2-gen 0/0/CPU0 t14 [GEN]:546: nbr 192.168.200.1,
 old state 5, new state 6, fd type 1, fd 124
17:55:22.373 default-bgp/spkr-tr2-gen 0/0/CPU0 t14 [GEN]:549: Nbr '192.168.200.1'
 established
```

The trace output can be filtered using the various keyword options available after the **show bgp trace** command, such as **show bgp trace error** or **show bgp trace update**, which helps in viewing the specific error and the BGP update, respectively.

Similarly in NX-OS, an **event-history** keyword option is available for various features under the respective feature command line. Example 2-24 displays the event-history cli option for BGP. In this example, the event history log displays an event with **clear_ip_bgp_cmd**, which tells that there was an execution of the **clear ip bgp** command on the router.

Example 2-24 *BGP Event History CLI*

```
Nexus_R1# show bgp event-history msgs
! Output omitted for brevity
46) Event:E_DEBUG, length:66, at 35995 usecs after Fri Aug 28 18:25:48 2015
     [100] [28046]: comp-mts-rx opc - from sap 9530 cmd clear_ip_bgp_cmd
47) Event:E_DEBUG, length:42, at 245222 usecs after Fri Aug 28 18:25:43 2015
     [100] [28046]: nvdb: terminate transaction
```

NX-OS not only maintains the event-history logs but also allows the user to define the size of the event-trace buffer per feature. Different sizes that can be set, such as small, medium, large, and so on.

Other event-history cli options are available for BGP, such as events, errors, logs, and so on. These are discussed and used in various chapters in this book.

Summary

This chapter explained various methodologies for troubleshooting network events. This chapter lays down a foundation of how to approach a network problem in few simple steps:

Step 1. Define the problem.

Step 2. Collect data.

Step 3. Refine the problem statement.

Step 4. Build a theory.

Step 5. Test the theory.

Step 6. Conclusion.

Step 7. Document the conclusion.

The chapter stressed the importance of documentation and reproducing the problem in a lab environment. Logging and event tracing options are essential tools for identifying what happened, when it happened, and where it happened. This information is vital for isolating a problem in the network.

Just as there are three sides to a story: yours, mine, and the truth, there are three sides to an event: sending router, receiving router, and the wire. Packet capture tools are available to verify what is happening from every perspective of the network.

Reference

BRKARC-2011, *Overview of Packet Capturing Tools*, Cisco Live.

This page intentionally left blank

Chapter 3

Troubleshooting Peering Issues

The following topics are covered in this chapter:

- BGP peering down issues
- BGP peer flapping issues
- Dynamic BGP peering

Border Gateway Protocol (BGP) is a complex protocol that requires proficient troubleshooting skills. To troubleshoot BGP issues, you should have a deep understanding of BGP protocol, its attributes, and its features. Some of these have already been discussed in Chapter 1, "BGP Fundamentals" and are discussed further in upcoming chapters. The most common problem with BGP is establishing and maintaining a BGP session that can occur when a new BGP neighbor is configured or can occur for an existing peer after a network event.

This chapter explains various techniques and commands that are helpful during the investigation of many BGP peering issues.

BGP Peering Down Issues

When a configured BGP neighbor is not in an established state, network engineers refer to this scenario as *BGP peering down*. The peering down issue occurs because of one of the following circumstances:

- During establishment of BGP sessions because of misconfiguration
- Triggered by network migration or event, or software or hardware upgrades
- Failure to maintain BGP keepalives due to transmission problems
- High CPU

- Blocked or stuck processes
- Firewall or ACL misconfiguration
- Software defects

A down BGP peer state is in either an Idle or Active state. From the peer state standpoint, these states would mean following possible problems:

- Idle state
  - No connected route to peer
- Active state
  - No route to peer address (IP connectivity not present)
  - Configuration error, such as update-source missing or wrongly configured
- Idle/Active state
  - Transmission Control Protocol (TCP) establishes but BGP negotiation fails; for example, misconfigured AS
  - Router did not agree on the peering parameters

Figure 3-1 provides a topology to demonstrate the logic and steps followed for troubleshooting BGP peering issues. In this topology, R1 and R2 are internal BGP (IBGP) neighbors, and R2 and R3 are external BGP (EBGP) neighbors. R1 is running NX-OS, R2 is running Cisco IOS, and R3 is running IOS XR.

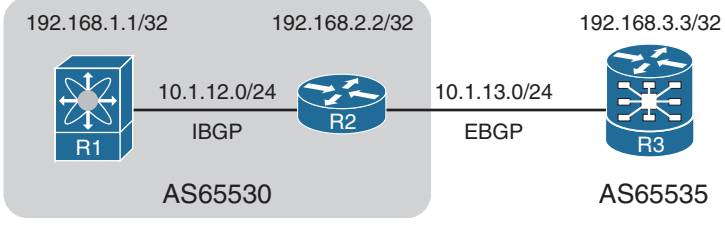

Figure 3-1 *IBGP and EBGP Topology*

Verifying Configuration

The first step for troubleshooting should be verifying the configuration and understanding the design. A lot of times, a basic configuration mistake can cause a BGP session not to come up. Example 3-1 displays working configurations of IBGP and EBGP sessions for the topology shown in Figure 3-1.

Note Nexus platforms require the BGP feature to be enabled using the command **feature bgp** to configure BGP. The **configuration** and **show** commands for BGP are not available on NX-OS until it is enabled.

Example 3-1 *BGP Configuration on All Routers*

```
R1# show running-config bgp
feature bgp

 router bgp 65530
   router-id 192.168.1.1
   address-family ipv4 unicast
     network 192.168.1.1/32
   neighbor 192.168.2.2
     remote-as 65530
     update-source loopback0
     address-family ipv4 unicast
```

```
R2# show running-config | section router bgp
 router bgp 65530
  bgp router-id 192.168.2.2
  bgp log-neighbor-changes
  no bgp default ipv4-unicast
  neighbor 10.1.13.2 remote-as 65535
  neighbor 192.168.1.1 remote-as 65530
  neighbor 192.168.1.1 update-source Loopback0
  !
  address-family ipv4
   network 192.168.2.2 mask 255.255.255.255
   neighbor 10.1.13.2 activate
   neighbor 192.168.1.1 activate
   neighbor 192.168.1.1 next-hop-self
```

```
RP/0/0/CPU0:R3# show running-config router bgp
router bgp 65535
  bgp router-id 192.168.3.3
  address-family ipv4 unicast
   network 192.168.3.3/32
  !
  neighbor 10.1.13.1
   remote-as 65530
   address-family ipv4 unicast
```

The following items should be checked when a new BGP neighbor is configured:

- Local autonomous system (AS) number
- Remote AS number
- Reachability between the peering addresses
- Verifying the network topology and other documentations
- Route policy planning, though not required for BGP neighborship to establish.

It is important to understand the traffic flow of BGP packets between peers. The source IP address of the BGP packets still reflects the IP address of the outbound interface. When a BGP packet is received, the router correlates the source IP address of the packet to the BGP neighbor table. If the BGP packet source does not match an entry in the neighbor table, the packet cannot be associated to a neighbor and is discarded.

In most of the deployments, the IBGP peering is configured over the loopback interface of the router. For the IBGP peering session to come up, the loopback interface should be explicitly defined as the source. The explicit sourcing of BGP packets from an interface can be verified by ensuring that the **neighbor** *ip-address* **update-source** *interface-id* command is correctly configured for the peer.

If there are multiple hops between the EBGP peers, then a proper hop count is required. Ensure the **neighbor** *ip-address* **ebgp-multihop** [*hop-count*] is configured with the correct hop count. If the *hop-count* is not specified, the default TTL value for IBGP sessions is 255, and the default TTL value for EBGP sessions is 1. The time to live (TTL) value can be viewed only on IOS routers, as shown in Example 3-2. Notice that the TTL value changes between IBGP and EBGP sessions. The TTL information is not displayed on IOS XR or NX-OS command output.

Example 3-2 *TTL Values for IBGP and EBGP Sessions*

```
R2# show bgp ipv4 unicast neighbor 192.168.1.1 | in TTL
 Connection is ECN Disabled, Mininum incoming TTL 0, Outgoing TTL 255
```

```
R2# show bgp ipv4 unicast 10.1.13.2 | in TTL
 Connection is ECN Disabled, Mininum incoming TTL 0, Outgoing TTL 1
```

Note With the **neighbor** *ip-address* **ttl-security** command, the **ebgp-multihop** command is not required. TTL security is discussed in more detail in Chapter 9, "Securing BGP."

Use the **neighbor** *ip-address* **disable-connected-check** command for an EBGP peer session that is single-hop away but peering over loopback interface. This command disables the connection verification mechanism, which by default prevents the session from getting established when the EBGP peer is not in the directly connected segment.

Another configuration that is important, although optional, for successful establishment of the BGP session is peer authentication. Misconfiguration or typo errors in authentication passwords can cause the BGP session to fail.

Verifying Reachability

After the configuration has been verified, the connectivity between the peering IPs must be confirmed as well. If the peering is being established between loopback interfaces, a loopback-to-loopback ping test should be performed. If a ping test is performed without specifying loopback as the source interface, the outgoing physical interface IP address will be used as the packet's source IP address instead of the router's loopback IP address.

Example 3-3 displays a loopback-to-loopback ping test between R1 and R2

Example 3-3 *Ping with Source Interface as Loopback*

```
NX-OS
R1# ping 192.168.2.2 source 192.168.1.1
PING 192.168.2.2 (192.168.2.2) from 192.168.1.1: 56 data bytes
64 bytes from 192.168.2.2: icmp_seq=0 ttl=254 time=4.809 ms
64 bytes from 192.168.2.2: icmp_seq=1 ttl=254 time=5.524 ms
64 bytes from 192.168.2.2: icmp_seq=2 ttl=254 time=8.882 ms
64 bytes from 192.168.2.2: icmp_seq=3 ttl=254 time=4.643 ms
64 bytes from 192.168.2.2: icmp_seq=4 ttl=254 time=6.091 ms
--- 192.168.2.2 ping statistics ---
5 packets transmitted, 5 packets received, 0.00% packet loss
round-trip min/avg/max = 4.643/5.989/8.882 ms
```

```
IOS
R2# ping 192.168.1.1 source loopback0
Sending 5, 100-byte ICMP Echos to 192.168.1.1, timeout is 2 seconds:
Packet sent with a source address of 192.168.2.2
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 3/5/8 ms
```

Reachability has to be verified for all IBGP and EBGP sessions. Example 3-3 shows that connectivity exists between the peering IP addresses. But if connectivity cannot be verified, refer to the upcoming subsections that explain how to isolate the issue in order to better understand if the connectivity problem is in the forwarding path or the return path. The following conditions are explained in more detail:

- Find the location and direction of packet loss.
- Verify whether packets are being transmitted.
- Use access control lists (ACL) to verify that packets are received.

- Check ACLs and firewalls in path.
- Verify TCP sessions.
- Simulate a BGP session.

Find the Location and Direction of Packet Loss

The **show ip traffic** command output is helpful to understand the direction of the packet loss. This command's output has a section for Internet Control Message Protocol (ICMP) packets, which shows the received and sent counters for both Echo-Request and Echo-Reply. First, ensure that the sent and receive counters are stable on both the source and the destination devices. If the counters keep incrementing, it will be difficult to correlate if the ICMP echo packet actually made it to the other end or if the other end sent an echo reply or not. In such cases, ACLs or packet capture tools can be used to verify whether the packets have been received and if the ICMP echo reply has been sent. If the counters are not incrementing, by using the source IP, such as loopback interface, perform a ping test to the destination IP. After the ping is completed (with or without success), verify the output of **show ip traffic | include echo**. The value by which echo (echo request) sent counters incremented on the source should match the value of echo received counters incremented on the destination device. The same value should be incremented for echo reply sent on the destination device and echo reply received on the source device.

Example 3-4 demonstrates the use of the **show ip traffic** command to understand the direction of the packet loss from R2 to R1. The echo reply sent counter is not incrementing where the echo received counter has incremented on R2.

Example 3-4 show ip traffic *Command Output*

```
NX-OS
R1# show ip traffic | include echo
   Redirect: 0, unreachable: 0, echo request: 10, echo reply: 5,
   Redirect: 0, unreachable: 0, echo request: 5, echo reply: 5,
! This is the Source Router
R1# ping 192.168.2.2 source 192.168.1.1
PING 192.168.2.2 (192.168.2.2) from 192.168.1.1: 56 data bytes
Request timed out
Request timed out
Request timed out
Request timed out
Request timed out
--- 192.168.2.2 ping statistics ---
5 packets transmitted, 0 packets received, 100.00% packet loss
round-trip min/avg/max = 3.482/3.696/3.904 ms
```

```
R1# show ip traffic | include echo
   Redirect: 0, unreachable: 0, echo request: 15, echo reply: 5,
   Redirect: 0, unreachable: 0, echo request: 5, echo reply: 5,
```

```
IOS
R2# show ip traffic | include echo
         10 echo, 14 echo reply, 0 mask requests, 0 mask replies, 0 quench
   Sent: 0 redirects, 0 unreachable, 55 echo, 10 echo reply
! Notice the below echo reply counter is not incrementing in the Sent section.
R2# show ip traffic | include echo
         15 echo, 14 echo reply, 0 mask requests, 0 mask replies, 0 quench
   Sent: 0 redirects, 0 unreachable, 55 echo, 10 echo reply
```

Note On IOS XR, use the command **show ipv4 traffic** to verify the ICMP packet counters.

To understand why R2 is not sending ICMP echo replies to R1, routing and the CEF table on R2 can be checked. Interface configurations on R2 should also be checked for ACLs that could be dropping the ICMP echo replies. If you are still unable to figure out the cause, one of the packet capture tools, as shown in Chapter 2, "Generic Troubleshooting Methodologies," can be used to further troubleshoot the problem.

Verify Whether Packets Are Being Transmitted

If there is complete packet loss on the link, perform a ping connectivity test with the timeout set to 0 to confirm if the packet is actually leaving the router or if the other side is receiving the packets. The purpose of this test is to send the packets out and ensure that the counters are incrementing and not really waiting for a reply from the other endpoint.

When the ping with timeout value 0 is performed, the output packets counter should be incremented by the amount of ping packets sent in the **show interface** command output. The router sends out ping packets without waiting for any response.

Example 3-5 displays the use of ping with timeout 0. It can be seen from the output that R2's Gi0/1 interface output counter is incrementing while the input counter is not. Notice that on the Nexus platform, **show ip interface** *interface-id* displays the packets sent and received counters, rather than the **show interface** *interface-id* command.

Example 3-5 *Ping with Timeout 0*

```
IOS
R2# show interface Gi0/1 | in packets
   5 minute input rate 0 bits/sec, 0 packets/sec
   5 minute output rate 0 bits/sec, 0 packets/sec
      26540 packets input, 2129461 bytes, 0 no buffer
      37292 packets output, 2870820 bytes, 0 underruns
R2# ping 10.1.12.1 timeout 0 repeat 10
Type escape sequence to abort.
Sending 10, 100-byte ICMP Echos to 10.1.12.1, timeout is 0 seconds:
..........
Success rate is 0 percent (0/10)
R2# show interface Gi0/1 | in packets
   5 minute input rate 1000 bits/sec, 0 packets/sec
   5 minute output rate 1000 bits/sec, 1 packets/sec
      26540 packets input, 2129461 bytes, 0 no buffer
      37306 packets output, 2872281 bytes, 0 underruns
```

```
NX-OS
R1# show ip interface E2/1 | include packets
     Unicast packets    : 6602/6611/0/6602/7753
     Multicast packets  : 19846/15636/0/19846/31272
     Broadcast packets  : 0/2/0/0/2
     Labeled packets    : 0/0/0/0/0
! Sent and Received packet counts after the ping test on R1
R1# show ip interface E2/1 | include packets
     Unicast packets    : 6602/6611/0/6602/7753
     Multicast packets  : 19846/15637/0/19846/31274
     Broadcast packets  : 0/2/0/0/2
     Labeled packets    : 0/0/0/0/0
```

Use Access Control Lists to Verify Whether Packets Are Received

ACLs prove to be really useful when troubleshooting packet loss or reachability issues. Configuring an ACL matching the source and the destination IP can help confirm whether the packet has reached the destination router. As a best practice, specific protocol packets with specific source and destination addresses should be used as part of the ACL. For example, if the troubleshooting is being performed for ICMP traffic and BGP packets between routers R1 and R2, the ACL entry should match for just the ICMP traffic and TCP traffic on port 179 with the source and destination of R1 and R2 address. The only caution that needs to be taken is that while configuring ACL, **permit ip any any** should be configured at the end, or else it will trigger all the other packets to get dropped and cause a service impact.

Check ACLs and Firewalls in Path

In most of the deployments, the edge routers or Internet Gateway (IGW) routers are configured with ACLs to limit the traffic allowed in the network. If the BGP session is being established across those links where the ACL is configured, ensure that BGP packets (TCP port 179) are not getting dropped because of those ACLs.

Example 3-6 shows how the ACL configuration should look if BGP is passing through that link. The example shows the configuration for both IPv4 as well as ipv6 access-list in case of IPv6 BGP sessions. For applying IPv4 ACL on interface, the **ip access-group** *access-list-name* {**in**|**out**} command is used on all platforms. For IPv6 ACL, the **ipv6 traffic-filter** *access-list-name* {**in**|**out**} interface command is used on NX-OS and IOS, whereas **ipv6 access-group** *access-list-name* {**in**|**out**} interface config is used on IOS XR.

Example 3-6 *ACL for Permitting BGP Traffic*

```
NX-OS
R1(config)# ip access-list v4_BGP_ACL
R1(config-acl)# permit tcp any eq bgp any
R1(config-acl)# permit tcp any any eq bgp
! Output omitted for brevity
R1(config)# ipv6 access-list v6_BGP_ACL
R1(config-ipv6-acl)# permit tcp any eq bgp any
R1(config-ipv6-acl)# permit tcp any any eq bgp
! Output omitted for brevity
R1(config)# interface Ethernet2/1
R1(config-if)# ip access-group v4_BGP_ACL in
R1(config-if)# ipv6 traffic-filter v6_BGP_ACL in
```

```
IOS
R2(config)# ip access-list v4_BGP_ACL
R2(config-acl)# permit tcp any eq bgp any
R2(config-acl)# permit tcp any any eq bgp
! Output omitted for brevity
R2(config)# ipv6 access-list v6_BGP_ACL
R2(config-ipv6-acl)# permit tcp any eq bgp any
R2(config-ipv6-acl)# permit tcp any any eq bgp
! Output omitted for brevity
R2(config)# interface Gi0/1
R2(config-if)# ip access-group v4_BGP_ACL in
R1(config-if)# ipv6 traffic-filter v6_BGP_ACL in
```

```
IOS XR
RP/0/0/CPU0:R3(config)# ipv4 access-list v4_BGP_ACL
RP/0/0/CPU0:R3(config-ipv4-acl)# permit tcp any eq bgp any
RP/0/0/CPU0:R3(config-ipv4-acl)# permit tcp any any eq bgp
! Output omitted for brevity
```

```
RP/0/0/CPU0:R3(config)# ipv6 access-list v6_BGP_ACL
RP/0/0/CPU0:R3(config-ipv4-acl)# permit tcp any eq bgp any
RP/0/0/CPU0:R3(config-ipv4-acl)# permit tcp any any eq bgp
! Output omitted for brevity
RP/0/0/CPU0:R3(config)# interface Gi0/0/0/0
RP/0/0/CPU0:R3(config-if)# ipv4 access-group v4_BGP_ACL in
RP/0/0/CPU0:R3(config-if)# ipv6 access-group v6_BGP_ACL in
RP/0/0/CPU0:R3(config-if)# commit
```

Having two ACL entries for BGP ensures that the BGP packet is not dropped if the TCP session is initiated from either the local or the remote router or if the ACL is on a device that is in transit.

Other than having ACLs configured on the edge devices, many deployments have firewalls to protect the network from unwanted and malicious traffic. It is a better option to have a firewall installed than to have a huge ACL configured on the routers and switches. Firewalls can be configured in two modes:

- Routed mode
- Transparent mode

In Routed mode, the firewall has routing capabilities and is considered to be a routed hop in the network. In Transparent mode, the firewall is not considered as a router hop to the connected device but merely acts like a "bump in the wire." Therefore, if an EBGP session is being established across a transparent firewall, then **ebgp-multihop** might not be required. Even if it is required to configure **ebgp-multihop** because of multiple devices in the path, the firewall is not counted as another routed hop.

In some deployments, network operators add NAT on the routed firewalls. In cases where NAT is configured on the router or on the firewall, the BGP peering should be configured with the translated IP address rather than the remote IP. If the peering is configured with the remote IP address and there is a NAT device in transit, the packet on the remote router will be received with the source address of the translated IP and not sourcing the IP address of the source router. This will cause the remote router to reset the TCP connection and send a reset (RST) and acknowledgment (ACK) in response to the synchronize (SYN) packet received for the TCP connection. This is because the receiving router is configured for BGP connection with the IP address of the source router but is receiving a different IP address than the configured peer.

In Figure 3-2, a transparent firewall is added between router R2 and R3 of our topology. There is no need to configure the **neighbor** *ip-address* **ebgp-multihop** [*hop-count*] command for the EBGP neighborship.

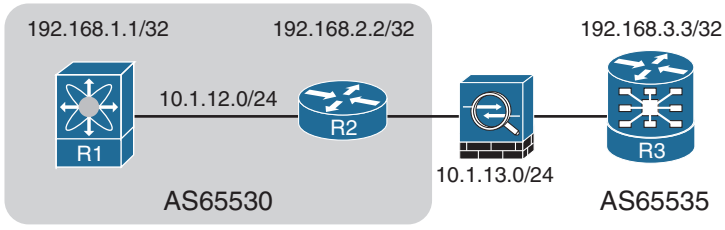

Figure 3-2 *IBGP and EBGP Topology with Firewall*

Firewalls implement various security levels for the interfaces. For example, on Cisco's Adaptive Security Appliance (ASA) firewall, the inside interface is assigned a security level of 100 (highest security level) and the outside interface is assigned a security level of 0 (lowest security level). An ACL needs to be configured to permit the relevant traffic from the least secure interface going toward a higher security interface. This rule applies for both Routed and Transparent mode firewalls.

Bridge groups are configured in Transparent mode firewall for each network to help minimize the overhead on security contexts. The interfaces become part of a bridge group and a bridge virtual interface (BVI) interface is configured with a management IP address.

Example 3-7 displays an ASA ACL configuration that allows ICMP as well as BGP packets to traverse across the firewall. It also demonstrates how to assign the ACL to the interface. Any traffic that is not part of the ACL gets dropped.

Example 3-7 *ASA Transparent Mode Firewall Configuration for Permitting BGP*

```
interface GigabitEthernet0/0
  nameif Inside
  bridge-group 200
  security-level 100
!
interface GigabitEthernet0/1
  nameif Outside
  bridge-group 200
  security-level 0
!
! Creating BVI with Management IP and should be the same subnet
! as the connected interface subnet
interface BVI200
  ip address 10.1.13.10 255.255.255.0
!
access-list Out extended permit icmp any any
access-list Out extended permit tcp any eq bgp any
access-list Out extended permit tcp any any eq bgp
!
access-group Out in interface Outside
```

In the ACL named Out, both ACL entries permitting BGP are not required, although it is good practice to have both. It is important to note that while troubleshooting flapping BGP peers, firewall MTUs could also play a part while sending BGP updates across the firewall. This concept is explained in detail for troubleshooting BGP flapping issues later in this chapter.

Verify TCP Sessions

A BGP session is a TCP session. Therefore, it is very important to verify if the TCP session is getting established to ensure successful BGP session establishment. Example 3-8 shows TCP sessions on port 179 on all the routers. Notice that the routers are listening on port 179. On R3 (IOS XR), there are two VRF-IDs. VRF-ID 0x60000000 indicates default VRF, whereas 0x00000000 indicates that it is a Multi Virtual Routing and Forwarding (VRF) TCP protocol control block (pcb) under which specific VRF pcbs are created by applications using TCP. The application can be BGP, Label Discovery Protocol (LDP), and so on.

Note The TCP session is allowed to be established only for a specific VRF pcb.

Example 3-8 *Output Showing TCP in Listening State*

```
R1# show sockets connection tcp
! Output omitted for brevity
 Total number of tcp sockets: 6
 Active connections (including servers)
 Protocol State/       Recv-Q/   Local Address(port)/
          Context      Send-Q    Remote Address(port)
 tcp      LISTEN       0         *(179)
          Wildcard     0         *(*)

 tcp6     LISTEN       0         *(179)
          Wildcard     0         *(*)

 tcp      ESTABLISHED  0         192.168.1.1(179)
          default      0         192.168.2.2(15529)
```

```
R2# show tcp brief all
 TCB        Local Address               Foreign Address             (state)
 0DFE19B0   192.168.2.2.15529           192.168.1.1.179             ESTAB
 0F7D44F8   10.1.13.1.179               10.1.13.2.52005             ESTAB
 0BB8FC40   0.0.0.0.179                 10.1.13.2.*                 LISTEN
 0D911608   0.0.0.0.179                 192.168.1.1.*               LISTEN
```

```
RP/0/0/CPU0:R3# show tcp brief
! Output omitted for brevity
```

```
   PCB      VRF-ID      Recv-Q Send-Q Local Address      Foreign Address      State
0x101ad27c 0x60000000     0      0     :::179             :::0                 LISTEN
0x101a521c 0x00000000     0      0     :::179             :::0                 LISTEN
0x101ad62c 0x60000000     0      0     10.1.13.2:52005    10.1.13.1:179        ESTAB
0x10158cd0 0x60000000     0      0     0.0.0.0:179        0.0.0.0:0            LISTEN
0x101a5e30 0x00000000     0      0     0.0.0.0:179        0.0.0.0:0            LISTEN
```

If BGP is not getting established and there is a stale entry noticed in the TCP table in established state, clear the TCP session using the TCP control block (tcb) entry by issuing the **clear tcp tcb** *value* on IOS or **clear tcp pcb** *value* command on IOS XR. Such stale entry problems would cause stopping of BGP keepalives, and TCP would move to a closing state.

Note The **clear** command is not available on NX-OS for TCP sessions. The **clear** command is available only to clear statistics.

Simulate a BGP Session

A good troubleshooting technique for BGP peers that are down is using Telnet on TCP port 179 toward the destination peer IP and implementing local peering IP as the source. This technique helps ensure that the TCP is not getting blocked or dropped between the two BGP peering devices. This test is useful for verifying any TCP issues on the destination router and also helps define any ACL or firewall that could possibly block the BGP packets.

Example 3-9 shows the use of Telnet on port 179 to verify the BGP session. When this test is performed, the BGP TCP session is established, but the BGP remains in *OpenSent* state.

Example 3-9 *Using Telnet to Port 179*

```
NX-OS
R1# telnet 192.168.2.2 179 source loopback 0
 Trying 192.168.2.2...
 Connected to 192.168.2.2.
 Escape character is '^]'.

 ÿÿÿÿ9
 Connection closed by foreign host.
```

```
IOS
R2# show bgp ipv4 unicast summary
! Output omitted for brevity
  Neighbor        AS    MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
 10.1.13.2     4   65535    1409    1329        5    0    0 20:00:13        0
 192.168.1.1   4   65530       0       1        1    0    0 00:01:40 OpenSent
```

```
R2# show tcp brief
 TCB        Local Address               Foreign Address             (state)
 0FE84118   192.168.2.2.179             192.168.1.1.5282            ESTAB
 0FEE0558   10.1.13.1.62644             10.1.13.2.179               ESTAB
```

Note The preceding Telnet method will not work if BGP authentication has been configured. This test is primarily useful when there is any ACL configured in the path or a firewall is involved.

If the Telnet is not sourced from the interface or IP that the remote device is configured to form a BGP neighborship with, the Telnet request will be refused. This is another way to confirm that the peering device is configured as dictated by the network requirements.

Demystifying BGP Notifications

BGP notifications play a crucial role in understanding and troubleshooting failed BGP peering or flapping peer issues. A BGP notification is sent from a BGP speaker to a peer when an error is detected. The notification can be sent either before the BGP session has been established or afterward based on the type of error. Each message has a fixed-size header. There may or may not be a data portion following the header, depending on the message type. The layout of these fields is as follows:

```
0                   1                   2                   3

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

|                           Marker                              |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

|          Length               |      Type     |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

In addition to the fixed-size BGP message header, a notification contains the following fields:

```
0                   1                   2                   3

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Error code    | Error subcode |   Data (variable)             |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Error code and Error-Subcode values are defined in RFC 4271. Table 3-1 shows all the Error codes, Error-Subcodes and their interpretation.

Table 3-1 *BGP Notification Error and Error-Subcode*

Error Code	Subcode	Description
01	00	Message Header Error
01	01	Message Header Error—Connection Not Synchronized
01	02	Message Header Error—Bad Message Length
01	03	Message Header Error—Bad Message Type
02	00	OPEN Message Error
02	01	OPEN Message Error—Unsupported Version Number
02	02	OPEN Message Error—Bad Peer AS
02	03	OPEN Message Error—Bad BGP Identifier
02	04	OPEN Message Error—Unsupported Optional Parameter
02	05	OPEN Message Error—Deprecated
02	06	OPEN Message Error—Unacceptable Hold Time
03	00	Update Message Error
03	01	Update Message Error—Malformed Attribute List
03	02	Update Message Error—Unrecognized Well-known Attribute
03	03	Update Message Error—Missing Well-Known Attribute
03	04	Update Message Error—Attribute Flags Error
03	05	Update Message Error—Attribute Length Error
03	06	Update Message Error—Invalid Origin Attribute
03	07	(Deprecated)
03	08	Update Message Error—Invalid NEXT_HOP Attribute
03	09	Update Message Error—Optional Attribute Error
03	0A	Update Message Error—Invalid Network Field
03	0B	Update Message Error—Malformed AS_PATH
04	00	Hold Timer Expired
05	00	Finite State Machine Error
06	00	Cease
06	01	Cease—Maximum Number of Prefixes Reached
06	02	Cease—Administrative Shutdown

Error Code	Subcode	Description
06	03	Cease—Peer Deconfigured
06	04	Cease—Administrative Reset
06	05	Cease—Connection Rejected
06	06	Cease—Other Configuration Change
06	07	Cease—Connection Collision Resolution
06	08	Cease—Out of Resources

Whenever a notification is generated, the error code and the subcode are always printed in the message. These notification messages are helpful when troubleshooting down peering issues or flapping peer issues.

Example 3-10 demonstrates a BGP notification being sent to the peer device when the peer is in the wrong autonomous system. The notification received by R2 displays the neighbor from which the notification was received, and the error code '2/2'—which is the Error/Error-Subcode format. From the preceding table, the error code 2 represents an OPEN error message, and the subcode 2 represents that the peer is in the wrong AS. The other important thing is the notification data, which is seen at the end (4 bytes 0000FFFA). The hex value in the data section helps to understand the problem and also gives the correct value in this case. The data section tells that the AS number should be 65530 rather than 65531.

Example 3-10 *BGP Notification*

```
IOS XR
RP/0/0/CPU0:R3# configure terminal
RP/0/0/CPU0:R3(config)# router bgp 65535
RP/0/0/CPU0:R3(config-bgp)# neighbor 10.1.13.1
RP/0/0/CPU0:R3(config-bgp-nbr)# remote-as 65531
RP/0/0/CPU0:R3(config-bgp-nbr)# commit
Sep 19 15:57:52.153 : bgp[1046]: %ROUTING-BGP-5-ADJCHANGE : neighbor 10.1.13.1
    Down - Remote AS configuration changed (VRF: default) (AS: 65531)
RP/0/0/CPU0:Sep 19 15:57:52.153 : bgp[1046]: %ROUTING-BGP-5-NSR_STATE_CHANGE :
    Changed state to NSR-Ready
```

```
IOS
R2#
*Sep 19 15:55:34.859: %BGP-3-NOTIFICATION: received from neighbor 10.1.13.2 active
  2/2 (peer in wrong AS) 4 bytes 0000FFFA
15:55:34.860: %BGP-5-NBR_RESET: Neighbor 10.1.13.2 active reset (BGP Notification
  received)
15:55:34.866: %BGP-5-ADJCHANGE: neighbor 10.1.13.2 active Down BGP Notification
received
15:55:34.867: %BGP_SESSION-5-ADJCHANGE: neighbor 10.1.13.2 IPv4 Unicast topology
base removed from session  BGP Notification received
```

Note that in Example 3-10, the notification message has a 4-byte data field with a value of 0000FFFA, which resolves to AS number 65530. But if the similar testing is done from the Cisco IOS router R2 toward R3, which is an IOS XR router, the notification data sent out from R2 with the 2-byte data field displays the AS number of 65535 (FFFF). Therefore, it is important to remember that for a wrong AS notification message, an IOS XR router will send a 4-byte data field, whereas the Cisco IOS router sends a 2-byte data field.

Sometimes, the BGP notifications can be confusing on the IOS devices. For example, the notification might contain the Error/Subcode as 2/8, but as per RFC 4271, Subcode 8 is not defined. IOS has some newer Error-Subcodes, as defined in Table 3-2.

Table 3-2 *Special BGP Error-Subcode in IOS*

Error Code	Subcode	Description
02	07	OPEN Message Error—No Supported Capability Value
02	08	OPEN Message Error—No Supported AFI/SAFI
02	09	OPEN Message Error—Grouping Conflict
02	0A	OPEN Message Error—Grouping Required

These special error codes and subcodes may not be applicable on other platforms and other vendors. They are specific to Cisco IOS platforms.

Decode BGP Messages

Various BGP notifications contain a huge data section that has a lot of information regarding the notification itself. This information is not understood from the notification Error/Subcode. If a peer sends any message, and the receiver of the message detects an error in that message or does not recognizes the message, BGP generates a hex dump of the message. These hex dumps can then be analyzed to understand why the BGP router was unable to process the message.

Because all the data is in hexadecimal format, it becomes an extremely difficult and time-consuming process to decode those messages. Yet without doing so, it is not possible to get to root of the problem and fix it.

Examine the notification message shown in Example 3-11. The example shows a notification message generated in the logs along with huge notification data. The notification sent before the message dump was generated shows that the peer is in the wrong AS and also shows that the AS number is 65535 (FFFF). Therefore, the main value that needs to be checked in the message dump is the AS value, which is 0xFFFF as highlighted.

Example 3-11 *BGP Message Dump*

```
BGP-3-NOTIFICATION: sent to neighbor 10.1.13.2 active 2/2 (peer in wrong AS)
     2 bytes FFFF
%BGP-4-MSGDUMP: unsupported or mal-formatted message received from 10.1.13.2:
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF 003B 0104 FFFF 00B4 C0A8 0303 1E02 0601
 0400 0100 0102 0280 0002 0202 0002 0641 0400 00FF FB02 0440 0280 78
BGP Message Decode:

BGP Marker:    0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
BGP Length:    0x003B    - 59 bytes
BGP Type:    0x01    - OPEN Message
OPEN Message
VERSION:    0x04    - 4
AS:         0xFFFF    - 65535
HOLD TIME:    0x00B4    - 180
ROUTER ID:    0xC0A80303    - 192.168.3.3
OPTIONAL PARAMETERS LENGTH:    0x1E    - 30 bytes
Opt Length:    28
Param Type:    2 - Optional Parameter
Param Length:    6
Cap Type:    0x01    - 1
          Multi Protocol Capability
Cap Length:    0x04    - 4 bytes
    AFI:    0x0001    - 1
     Reserved Bits:    0x00    - 0
     SAFI:    0x01    - 1
Opt Length:    20
Param Type:    2 - Optional Parameter
Param Length:    2
Cap Type:    0x80    - 128
        Route Refresh Capability (old)
Cap Length:    0x00    - 0 bytes
Opt Length:    16
Param Type:    2 - Optional Parameter
Param Length:    2
Cap Type:    0x02    - 2
        Route Refresh Capability (new)
Cap Length:    0x00    - 0 bytes
Opt Length:    12
Param Type:    2 - Optional Parameter
Param Length:    6
Cap Type:    0x41    - 65
         4-byte AS Capability
```

```
Cap Length:    0x04    - 4 bytes
AS Number:    0.65531 (0x0000.0xFFFB)
Opt Length:    4
Param Type:    2 - Optional Parameter
Param Length:    4

Cap Type:    0x40    - 64
        Graceful Restart Capability
Cap Length:    0x02    - 2 bytes
Restart flag:        0x8078    - 32888
```

Note There are external websites that help decode BGP messages; for example, http://bgpaste.convergence.cx.

IOS XR makes it easy to understand what value to look for and where to locate the problem in the message data. The command **show bgp update in error neighbor** *ip-address* **detail** provides more information about the malformed update.

Example 3-12 demonstrates a malformed update received on R3 and the use of the **show bgp update in error neighbor** *ip-address* **detail** command to further understand where the problem is present in the BGP message. Notice that the problem is in the Attribute 2 field of the update and also a missing attribute in Attribute 2 of the update. The process of decoding the BGP message remains the same, as shown in Example 3-11.

Example 3-12 *Malformed BGP Update on IOS XR*

```
RP/0/0/CPU0:R3# show logging | in MALFORM
bgp[1046]: %ROUTING-BGP-3-MALFORM_UPDATE : Malformed UPDATE message received
from neighbor 10.1.13.1 - message length 126 bytes, error flags 0x00000a00,
action taken "TreatAsWithdraw". Error details: "Error 0x00000800,
Field "Attr-missing", Attribute 2 (Flags 0x00, Length 0), Data []"
```

```
RP/0/0/CPU0:R3# show bgp update in error neighbor 10.1.13.1 detail
! Output omitted for brevity
VRF "default"

 Neighbor 10.1.13.1
   Update error-handling: Allow session reset

   Malformed messages stored: 5 (current index: 0)
     Malformed message #1
      Received: Sep 20 02:53:02.269
```

```
      Error flags: 0x00000a00
      Discarded attributes: 0
      Final action: TreatAsWithdraw
      Error elements: 2
      [1] Error 0x00000200, Field "Attr-data", Attribute 2 (Flags 0x40, Length 10)
          Error data&colon; [40020a02020000272a00005d7a] (13 bytes)
            Action: TreatAsWithdraw
      [2] Error 0x00000800, Field "Attr-missing", Attribute 2 (Flags 0x00, Length 0)
          Error data&colon; [] (0 bytes)
            Action: TreatAsWithdraw
       Reset/notification information:
        Reason "None", Postit type "Update malformed"
        Notification code 3, sub-code 3
          Notification data [02] (1 bytes)
      Message data&colon; 81 bytes
        FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
        00510200 00003640 01010040 020A0202
        0000272A 00005D7A 400304CE 7EEC28C0
          07080000 5D7A797F 0040C008 10272A10
        68272A7E 58272A9C A65E3327 2A18745D
        30
```

On NX-OS, the BGP messages are not dumped on logs but have to be collected using debugs. The command **debug bgp packets** helps view what BGP messages are being received and can be decoded. Example 3-13 demonstrates how to capture a BGP packet on NX-OS using the **debug bgp packet** command. The BGP packet shown in the output is a BGP OPEN message between R1 and R2. The initial hexdump of the data contains the BGP marker. The second line begins with 00460104. 0046 is the BGP Length field followed by 01, which tells that the message is a BGP OPEN message.

Example 3-13 debug bgp packet *on NX-OS*

```
R1# debug logfile bgp
R1# debug bgp packets
! Below debug output shows the Hexdump of the BGP packet received.
R1# show debug logfile bgp
! Output omitted for brevity
18:24:04.560109 bgp: 65530 [8480] Hexdump at 0x84bc64c, 70 bytes:
18:24:04.560631 bgp: 65530 [8480]       FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
18:24:04.560743 bgp: 65530 [8480]       00460104 FFFA00B4 C0A80101 29022742
 18:24:04.560853 bgp: 65530 [8480]       00430302 01408000 02000104 00010001
18:24:04.560953 bgp: 65530 [8480]       40060078 00010100 41040000 FFFA0506
18:24:04.561051 bgp: 65530 [8480]       00010001 0002
! Output omitted for brevity
R1# undebug all
```

If too many BGP updates and messages are being exchanged on the NX-OS devices, it is better to perform an Ethanalyzer or Switched Port Analyzer (SPAN) to capture a malformed BGP update packet to further analyze it.

Troubleshoot Blocked Process in IOS XR

IOS XR is a distributed operating system, and every component (feature) runs as a separate process with its own set of threads that manages various tasks of the component. Unlike traditional IOS, if a process crashes on the router, IOS XR does not reboot the whole system. Rather, the process is restarted without affecting other running processes. Because it's a multithreaded environment, threads keep changing their states. Some are for a short span of time and some for longer. An essential low-level process such as a kernel thread may block a high-level process thread so it can perform the tasks at a higher priority. Blocked processes can cause various features to malfunction or not function at all on the system.

In IOS XR, the BGP Process Manager (BPM) and BGP processes create the BGP protocol functionality. The primary tasks of a BPM process is to spawn the speaker process, such as the BGP process and verify and apply all BGP-related configurations. When BGP runs in multi-instance mode, it works with a placed process to distribute the different BGP speaker processes to appropriate nodes.

The BPM process also has the responsibility to calculate the router-id if one is not explicitly configured. It interacts with NETIO, TCP, and a few other processes internally to perform the necessary tasks in the system and finally installs the routes in the Routing Information Base (RIB). Figure 3-3 displays the interaction between these processes on a route processor (RP).

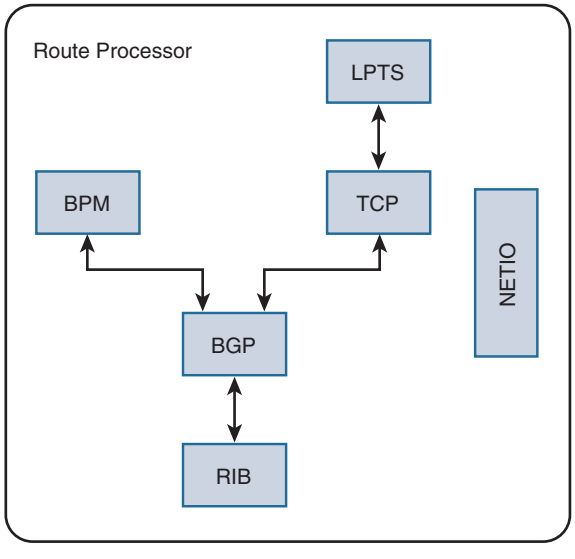

Figure 3-3 *Interaction of Processes with BGP in IOS XR*

Note There are other processes that are not shown in Figure 3-3. The discussion of those processes is outside the scope of this book.

Therefore, if any these processes are blocked on the RP or line card (LC) for a longer period of time, it could severely impact BGP services on the router. The following sections show how to troubleshoot the BGP processes on IOS XR platforms.

Verify BGP and BPM Process State

Verify that the BGP and BPM processes are in Run state by using the command **show process** *process-name* [**detail** | **location** {**LC/RP** *location* | **all**}] where *process-name* can be either *bgp* or *bpm* or any other process. The location option is helpful when the process state and information has to be checked on the line card or standby route processor card. Example 3-14 shows the use of the **show process** command for verifying the state of the *bgp* process. In this output, the most important fields to look at are the Job Id, PID, Respawn count, Process state, and the state of the threads under the process. In Example 3-14, the Respawn count is seen as 6. This number can be incremented if the *bgp* process keeps crashing or is unconfigured and reconfigured.

Example 3-14 **show process bgp** *Command Output*

```
RP/0/0/CPU0:R3# show process bgp
                Job Id: 1046
                   PID: 827666
          Executable path: /disk0/iosxr-routing-5.3.0/bin/bgp
               Instance #: 1
               Version ID: 00.00.0000
                  Respawn: ON
            Respawn count: 6
             Last started: Sun Sep 13 05:18:54 2015
            Process state: Run
            Package state: Normal
        Started on config: default
             Feature name: ON
                      Tag : default
            Process group: v4-routing
                     core: MAINMEM
                Max. core: 0
                Placement: Placeable
             startup_path: /pkg/startup/bgp.startup
                    Ready: 1.229s
                Available: 7.849s
         Process cpu time: 1.470 user, 0.610 kernel, 2.080 total
```

```
 JID    TID  Stack pri state         TimeInState       HR:MM:SS:MSEC NAME
 1046   1     392K  10 Receive        0:00:00:0099    0:00:00:0136 bgp
 1046   2     392K  10 Receive       50:56:47:0715    0:00:00:0019 bgp
 1046   3     392K  10 Receive       84:33:28:0736    0:00:00:0000 bgp
 1046   4     392K  10 Receive       84:33:27:0646    0:00:00:0000 bgp
 1046   5     392K  10 Receive       84:33:23:0086    0:00:00:0000 bgp
 1046   6     392K  10 Sigwaitinfo   84:33:28:0606    0:00:00:0000 bgp
 1046   7     392K  10 Receive       83:59:49:0874    0:00:00:0009 bgp
 1046   8     392K  10 Receive        0:00:01:0529    0:00:00:0000 bgp
 1046   9     392K  10 Receive       36:49:17:0309    0:00:00:0000 bgp
 1046   10    392K  10 Receive        0:00:01:0689    0:00:00:0010 bgp
 1046   11    392K  10 Receive       12:37:51:0414    0:00:00:0000 bgp
 1046   12    392K  10 Nanosleep      0:00:00:0899    0:00:00:0030 bgp
 1046   13    392K  10 Receive       84:21:24:0516    0:00:00:0000 bgp
! Output omitted for brevity
```

If the BGP processes are crashing, it is important to understand what really triggered them. Verifying the last event or change before the crash will help stabilize the *bgp* process.

Verify Blocked Processes

Execute **show process blocked** [**location** {**RP/LC**}] to verify whether there are any blocked processes, which could cause an impact on the BGP process. Primarily the *bgp*, *bpm*, *tcp*, and *netio* processes are the ones that are critical. Example 3-15 demonstrates how to verify any blocked processes and the amount of time that they have been in that state. There are always some processes, which will be in blocked state like *ksh*, or *lpts_fm* process in the output. You can ignore those processes.

Example 3-15 show process blocked *Command Output*

```
RP/0/0/CPU0:R3# show processes blocked
 Thu Sep 17 06:25:11.639 UTC
   Jid       Pid Tid             Name State   TimeInState    Blocked-on
 65542    200710   1              ksh Reply  121:14:38:0413       2  devc-ser8250
 65555    217107   1              ksh Reply  121:14:28:0649  217106  devc-conaux
 65692    680092   1             exec Reply    0:00:00:0079       1  kernel
   289    643248   2          lpts_fm Reply   11:30:09:0743  331870  lpts_pa
 65734    975046   1  show_processes Reply    0:00:00:0000       1  kernel
```

Restarting a Process

If for some reason a process is in blocked state for a long period of time, restart the process using **process restart** [*job-id* | *process-name*]. However, there can be processes that might be blocked for very long time and might not be related, such as *devc-conaux* process. This process is for the console access. Based on what problem is being troubleshot, related processes to the problem, and when the problem actually started, the time of the blocked process could be analyzed and the process could then be restarted. For example, if the TCP process got stuck 10 hours ago, which was about the same time BGP peering went down and never came back up, the blocked TCP process would make more sense to have restarted.

BGP Traces in IOS XR

The IOS XR BGP tracing facility has been implemented to help track BGP issues. BGP traces allow the user to see some level of history concerning what BGP has been doing when the problem occurred. In general, BGP trace messages fall into two categories:

- Nonfatal error conditions
- Informational messages

Nonfatal error conditions can recover automatically. Informational messages are used for logging data regarding the events that occurred in context to the error message. But they do not provide a complete log on what BGP is doing. Fatal error always logs an unsolicited message in the system log and therefore is not included in the trace output.

Because logging trace messages is a slow process and the trace buffer is stored in memory, tracing of successful events or detailed tracing about particular events is not supported. Traces stored in memory have limited size. So when the buffer memory is full, the trace logs are overridden. For this reason, error traces are written in different buffer than informational traces.

Each BGP trace message used for debugging BGP issues belongs to one of the categories shown in Table 3-3.

Table 3-3 *BGP Trace Categories*

Category	Debugging Problem
bgp	Router-id, neighbor resets, and state changes, OPEN messages, and the like
update	Inbound and outbound update messages
event	Process startup/shutdown and mode, general failures, and so on
io	TCP socket level failures
rib	RIB install and redistribution
brib	bRIB events and bRIB/speaker communication
policy	Routing Policy Language (RPL)

An easy way to find all the BGP-related trace commands is by using the command **show parser dump | include (.*bgp.*trace)**. This command prints all the **show** commands available for BGP trace.

Example 3-16 shows BGP trace using **show bgp trace** command for a BGP session coming up. Each trace entry starts with a timestamp, followed by the trace buffer name. The buffer name identifies the process that recorded the trace and whether it is an informational or an error trace. Distributed speaker processes and BGP RIB (bRIB) processes also include the process ID in the name. For error traces, the name ends with err. Example 3-16 shows both the error and informational messages.

Example 3-16 show bgp trace *Output*

```
RP/0/0/CPU0:R3# show bgp trace
! Output omitted for brevity
04:58:19.822 default-bgp/spkr-tr2-gen 0/0/CPU0 t21 [GEN]:546: nbr 10.1.13.1,
     old state 7, new state 1, fd type 0, fd 0
04:58:33.871 default-bgp/spkr-tr2-sync 0/0/CPU0 t1  [SYNC]:8104: Nbr '10.1.13.1'
: Loc 2: Determined lport/fport 179/38338
04:58:33.871 default-bgp/spkr-tr2-gen 0/0/CPU0 t1  [GEN]:546: nbr 10.1.13.1,
old state 1, new state 2, fd type 1, fd 125
04:58:33.901 default-bgp/spkr-tr2-err 0/0/CPU0 t14 [ERR][GEN]:1300: OPEN
from '10.1.13.1' has unrecognized cap code/len 70/0 - Ignored
04:58:33.901 default-bgp/spkr-tr2-upd 0/0/CPU0 t14 [UPD]:1818: Updgrp change not
  scheduled after open processing: nbr=10.1.13.1, nbrfl=0x8010000
04:58:33.901 default-bgp/spkr-tr2-gen 0/0/CPU0 t14 [GEN]:2222: calling
bgp_send_open, nbr 10.1.13.1, loc 4, data 0,0
04:58:33.901 default-bgp/spkr-tr2-gen 0/0/CPU0 t14 [GEN]:546: nbr 10.1.13.1,
old state 2, new state 4, fd type 1, fd 125
04:58:33.901 default-bgp/spkr-tr2-gen 0/0/CPU0 t14 [GEN]:546: nbr 10.1.13.1,
old state 4, new state 5, fd type 1, fd 125
04:58:33.901 default-bgp/spkr-tr2-gen 0/0/CPU0 t14 [GEN]:2295: calling bgp_send_
  keepalive, nbr 10.1.13.1, loc 1, data 0,0
04:58:33.961 default-bgp/spkr-tr2-sync 0/0/CPU0 t19 [SYNC]:5610: Sent
NSR-Not-Ready notif to Rmf
04:58:33.961 default-bgp/spkr-tr2-upd 0/0/CPU0 t18 [UPD]:1376: Filter-group op
(Alloc) Tbl/Nbr(TBL:default (1/1)) fgrp idx 0.2 subgrp idx 0.1 updgrp 0.0 rtset 0
04:58:33.961 default-bgp/spkr-tr2-upd 0/0/CPU0 t18 [UPD]:2640: Filter-group op
(Filter-group Add Nbr new) Tbl/Nbr(Afi:IPv4 Unicast:Nbr:10.1.13.1)
   fgrp idx 0.2 subgrp idx 0.1 updgrp 0.2 rtset 0
04:58:33.961 default-bgp/spkr-tr2-upd 0/0/CPU0 t18 [UPD]:2662: Created filtergrp 2
for nbr 10.1.13.1, afi 0. Subgrp version 0
04:58:33.961 default-bgp/spkr-tr2-upd 0/0/CPU0 t18 [UPD]:4213: Created  subgrp 1
for nbr 10.1.13.1, afi 0 with version 0
```

```
04:58:33.961 default-bgp/spkr-tr2-ev 0/0/CPU0 t14 [EV]:1952: Inside scan_adjust_
  estab_outstanding for nbr 10.1.13.1, nbr-down 0, state 2,
evt-in-nbrwait 0
04:58:33.961 default-bgp/spkr-tr2-gen 0/0/CPU0 t14 [GEN]:546: nbr 10.1.13.1,
old state 5, new state 6, fd type 1, fd 125
04:58:33.961 default-bgp/spkr-tr2-gen 0/0/CPU0 t14 [GEN]:549: Nbr '10.1.13.1'
established
! Output omitted for brevity
```

To filter just the error messages or any other messages, such as update or event, use the **show bgp trace** *trace-option* command, where *trace-option* value can be error, event, or other supported BGP trace filters.

Example 3-17 demonstrates the use of the **show bgp trace error** command to view BGP errors. The error trace demonstrates that the router ignores the capability code value 70, which refers to Enhanced Route Refresh Capability.

Note The Enhanced Route Refresh Capability is explained in detail in Chapter 7, "Scaling BGP."

Example 3-17 show bgp trace *error Command*

```
RP/0/0/CPU0:R3# show bgp trace error
 6 wrapping entries (1826304 possible, 3840 allocated, 643 filtered, 649 total)
21:14:28.248 default-bgp/spkr-tr2-err 0/0/CPU0 t1  [ERR][SYNC]:284: pid 0 : Got my
  local_node as 0x0
21:14:35.638 default-bgp/spkr-tr2-err 0/0/CPU0 t14 [ERR][GEN]:1300: OPEN from
'10.1.13.1' has unrecognized cap code/len 70/0 - Ignored

! Output omitted for brevity
```

BGP Traces in NX-OS

In NX-OS, all the traces related to all the events and errors are stored as event-history. A few event-history traces are enabled by default, such as for BGP events or errors, but more detailed event-history logging must be enabled using the **event-history** [**cli** | **detail** | **events** | **periodic**] [**size** {*size_in_text* | *bytes*}] under **router bgp**. This command is also used to set the size of the buffer for the event-history. Example 3-18 shows the BGP event-history configuration under router bgp configuration, how to enable detailed event-history event logging and event-history logging generated for syslog.

Example 3-18 *BGP Event-History on NX-OS*

```
R1(config)# router bgp 65530
R1(config-router)# event-history ?
   cli       CLI event history buffer
   detail    Detailed event history buffer
   events    Events history buffer
   periodic  Periodic events history buffer

R1(config-router)# event-history detail size ?
   disable  Disabled the buffer
            *Default value is disble
   large    Large buffer
   medium   Medium buffer
   small    Small buffer

 R1(config-router)# event-history detail size medium
 R1(config-router)# end
```

```
R1# show bgp event-history logs
 bgp-65530 logs events
19:37:15.618509 bgp 65530 [8556]: [8566]: (default) neighbor 192.168.2.2 Up
19:37:07.425970 bgp 65530 [8556]: [8567]: (default) neighbor
192.168.2.2 Down - session cleared
```

Table 3-4 displays the use of various **show bgp event-history** command options.

Table 3-4 *BGP* **event-history** *Options*

Option	Description
cli	Event-history for cli commands executed
detail	Show detail event logs
errors	Show error logs for BGP
events	Show event logs for BGP
logs	Show messages logged by Syslog
msgs	Show various message logs of BGP
periodic	Show periodic event logs

When troubleshooting BGP peering issues, it is also important to check the event-history logs for a netstack process. Netstack is an implementation of a Layer 2 to Layer 4 stack on NX-OS. It is one of the critical components involved in the control plane on NX-OS. If there is a problem with establishing a TCP session on a Nexus device, it could

be an issue with the netstack process. The **show sockets internal event-history events** command helps determine the TCP state transitions that occurred for the BGP peer IP.

Example 3-19 demonstrates the use of the **show sockets internal event-history events** command to see the TCP session getting closed for BGP peer IP 192.168.2.2 but does not show any TCP request coming in for establishing a new session.

Example 3-19 **show sockets internal event-history events** *Command*

```
R1# show sockets internal event-history event
 1) Event:E_DEBUG, length:67, at 24439 usecs after Tue Sep 22 20:36:01 2015
    [138] [3767]: Marking desc 22 in mts_open for client 15195, sotype 2
! Output omitted for brevity
60) Event:E_DEBUG, length:91, at 471283 usecs after Tue Sep 22 19:37:12 2015
    [138] [3730]: PCB: Removing pcb from hash list L: 192.168.1.1.179, F:
  192.168.2.2.29051 C: 1
 61) Event:E_DEBUG, length:62, at 471234 usecs after Tue Sep 22 19:37:12 2015
    [138] [3730]: PCB: Detach L 192.168.1.1.179 F 192.168.2.2.29051
 62) Event:E_DEBUG, length:77, at 471174 usecs after Tue Sep 22 19:37:12 2015
    [138] [3730]: TCP: Closing connection L: 192.168.1.1.179, F: 192.168.2.2.29051
```

For more detailed troubleshooting on netstack, the **show tech netstack** [**detail**] command can be collected to investigate the problem.

Debugs for BGP

Running debugs should always be the last resort for troubleshooting any network problem. Debugs can sometimes cause an impact in the network if not used carefully. But sometimes they are the only options when other troubleshooting techniques cannot repair the problem.

Note A lot of debugs are chatty, which means that a huge amount of log messages are printed in the debug output. If console logging is enabled along with such debugs, it can result in high CPU condition on the router and may lead to the loss of management connectivity, traffic loss, or even router or process crash. In such situations, sometimes the only alternative to recover the service is reloading the router. But when run cautiously, the debugs can remedy the problem and achieve a faster resolution.

When the BGP peers are down, and all the other troubleshooting steps cannot resolve the problem or are unable to capture the BGP packets, debugs can be enabled. They detect if the router is generating and sending the necessary BGP packets and if it is receiving the relevant packets.

On IOS devices, use the **debug ip bgp** *ip-address* or **debug bgp ipv4 unicast** *ip-address* command to run the debugs for a particular neighbor that is down. **debug** is not required

on NX-OS because the traces in BGP have sufficient information to debug the problem. **debug bgp ipv4 unicast events** return almost the same output as **show bgp event-history periodic** output.

Example 3-20 displays that router R2 is going into an Active state because the remote host is not responding. The session goes from Idle to Active state but from Active it doesn't go to Established state.

Example 3-20 **debug bgp ipv4 unicast** *ip-address Command*

```
R2# debug bgp ipv4 unicast 192.168.1.1
BGP debugging is on for neighbor 192.168.1.1 for address family: IPv4 Unicast
19:44:10.859: BGP: 192.168.1.1 active went from Idle to Active
19:44:10.860: BGP: 192.168.1.1 open active, local address 192.168.2.2
19:44:21.831: BGP: topo global:IPv4 Unicast:base Scanning routing tables
19:44:40.866: BGP: 192.168.1.1 open failed: Connection timed out;
     remote host not responding
19:44:40.866: BGP: 192.168.1.1 Active open failed - tcb is not available,
 open active delayed 10240ms (35000ms max, 60% jitter)
19:44:40.867: BGP: ses global 192.168.1.1 (0xF63BD98:0) act Reset
     (Active open failed).
19:44:40.872: BGP: 192.168.1.1 active went from Active to Idle
```

The output also shows that the tcb is not available for the session. This indicates that the TCP session is not getting established. On an IOS device, if the TCP session is not getting established, the **debug ip tcp transaction** [**address** *src-or-dst-addr*] [**port** *port-num*] command can be enabled to check why the TCP session is not getting established.

On IOS XR devices, **debug tcp packet** [**v4-access-list** *acl-name*] can be used to limit the debugs to just the packets matching the ACL. On Nexus, **debug sockets tcp** can be employed. Although there is no option to filter the output in Cisco IOS or IOS XR, the outputs can be captured in a log file. Example 3-21 demonstrates the use of debugs on both R1 and R2 to check the TCP negotiations between them. The debug output shows that R2 is sending a SYN packet to R1, which is received at R1, but R2 does not receive any SYN-ACK in return for the SYN packet. This indicates that the TCP packet is being lost in the direction from R1 to R2.

Example 3-21 *TCP Debugs on IOS and NX-OS*

```
NX-OS
R1# debug sockets tcp
20:22:18.687624 netstack: syncache_insert: SYN added for
   L:192.168.2.2.42829 F:192.168.1.1.179, tp:0x8740cd4 inp:0x8740c0c
20:22:48.790993 netstack: tcp_connect: Originating Connections
   with ports (Src 60250, Dst 179)
20:22:48.793718 netstack: tcp_notify: TCP: Notify
    L: 192.168.1.1.60250, F: 192.168.2.2.179 of error Connection refused
```

```
20:22:51.813236 netstack: tcp_notify: TCP: Notify L: 192.168.1.1.60250, F:
  192.168.2.2.179 of error Connection refused
```

```
IOS
R2# debug ip tcp transaction 192.168.1.1
20:18:27.194: TCP0: RETRANS timeout timer expired
20:18:27.195: 192.168.2.2:15033 <---> 192.168.1.1:179   congestion window changes
20:18:27.195: cwnd from 1460 to 1460, ssthresh from 65535 to 2920
20:18:27.196: TCP0: timeout #1 - timeout is 4000 ms, seq 2704785302
20:18:27.196: TCP: (15033) -> 192.168.1.1(179)
20:18:31.194: TCP0: RETRANS timeout timer expired
20:18:31.196: TCP0: timeout #2 - timeout is 8000 ms, seq 2704785302
20:18:31.196: TCP: (15033) -> 192.168.1.1(179)
20:18:39.194: TCP0: RETRANS timeout timer expired
20:18:39.196: TCP0: timeout #3 - timeout is 15999 ms, seq 2704785302
! Output omitted for brevity
*Oct  4 20:18:55.194: TCP0: state was SYNSENT -> CLOSED [15033 -> 192.168.1.1(179)]
20:18:55.198: TCB 0xD770800 destroyed
```

Based on the debug logs, the path or interfaces between R1 and R2 can be checked for any blocking ACLs.

Troubleshooting IPv6 Peers

With the depletion of IPv4 routes, the IPv6 addresses have caught up pace. Most of the service providers have already upgraded or are planning to upgrade their infrastructure to dual stack for supporting both IPv4 and IPv6 traffic and are offering IPv6 ready services to their customers. Even the new applications are being developed with IPv6 compatibility or completely running on IPv6. With such a pace, there is also a need to have appropriate techniques for troubleshooting IPv6 BGP neighbors.

The methodology for troubleshooting IPv6 BGP peers is same as that of IPv4 BGP peers. The following steps can be used to troubleshoot BGP peering down issues for IPv6 BGP neighbors:

Step 1. Verify the configuration for correct peering IPv6 addresses, AS numbers, **update-source** *interface-id*, authentication passwords, **ebgp-multihop** configuration.

Step 2. The reachability is verified using **ping ipv6** *ipv6-neighbor-address* [**source** *interface-id* | *ipv6-address*].

Step 3. The TCP connections is verified using **show tcp brief** on IOS and IOS XR or **show socket connection tcp** on NX-OS. In the case of IPv6, check for TCP connections for source and destination IPv6 addresses and one of the ports as port 179.

Step 4. Like IPv4, the IPv6 ACLs in the path should permit for TCP connections on port 179 and ICMPv6 packets that help in verifying reachability.

Step 5. On IOS routers, use the **debug bgp ipv6 unicast** *ipv6-neighbor-address* command to capture IPv6 BGP packets. The debugs on NX-OS and IOS XR remain the same, with the exception that the ACLs when matched for debugs are IPv6 ACLs.

Case Study—Single Session Versus Multisession

Company A is having a merger with Company B. As part of the merger, both companies are exchanging their network information using EBGP at each location. Company A has a multivendor network environment, whereas Company B has all Cisco devices. At one location, the EBGP session between two routers is not establishing. Customer A opens a service request with Cisco Technical Assistance Center (TAC), and the following troubleshooting steps are performed:

Step 1. Customer verified that the configuration on both sides (A and B) are correct.

Step 2. Verified reachability.

Step 3. Performed TCP tests on port 179, which also worked fine.

Even after all the verifications, the BGP session was not establishing. On Company A's router, a BGP notification for unsupported/disjoint capability was noticed, as shown in Example 3-22.

Example 3-22 *BGP Notification on Company A Router*

```
*Oct  5 06:06:58.557: %BGP-3-NOTIFICATION: sent to neighbor 10.1.12.2
       active 2/7 (unsupported/disjoint capability) 0 bytes
*Oct  5 06:06:58.557: %BGP-4-MSGDUMP: unsupported or mal-formatted message
       received from 10.1.12.2:
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF 0039 0104 FDE9 00B4 0A01 0C02 1C02 0601
0400 0100 0102 0280 0002 0202 0002 0246 0002 0641 0400 00FD E9
```

The preceding notification clarified that there is a capability being exchanged by one of the peers that is not supported by the other. With the help of the **show bgp ipv4 unicast neighbor** *ip-address* command, it was noticed that one side is multisession capable, whereas the other side is not. Example 3-23 shows the difference between both BGP neighbors.

Example 3-23 **show bgp ipv4 unicast neighbor** *ip-address Command Output*

```
Comp_A_R1# show bgp ipv4 unicast neighbor 10.1.12.2
BGP neighbor is 10.1.12.2,  remote AS 65001, external link
  BGP version 4, remote router ID 10.1.12.2
  BGP state = Idle
  Neighbor sessions:
    0 active, is multisession capable
    Stateful switchover support enabled: NO for session 0
! Output omitted for brevity
```

```
Comp_B_R2# show bgp ipv4 unicast neighbor 10.1.12.1
BGP neighbor is 10.1.12.1,  remote AS 65000, external link
  BGP version 4, remote router ID 0.0.0.0
  BGP state = Idle
  Neighbor sessions:
    0 active, is not multisession capable (disabled)
    Stateful switchover support enabled: NO
! Output omitted for brevity
```

This means that one side is trying to establish a BGP session in single-session mode when the other side is trying to use multisession mode. This causes the BGP session not to come up. To resolve this problem, the session capabilities should match by either configuring the **transport multisession** command on side B or the **transport single-session** command on side A.

The following sections provide a brief description of BGP single-session and multisession capabilities.

Multisession Capability

Multisession capability is available on all the latest Cisco IOS software but is not enabled by default. With this capability, multiple sessions are established for a particular neighbor for each address-family. Multisession mode can be used for MP-BGP sessions where multiple BGP sessions are used between peers instead of a single session that exchanges updates for all address-families. For example, if a neighbor is establishing a session in IPv4 address-family and vpnv4 address-family, there are two BGP sessions established on the router. This capability provides more granularity for managing BGP sessions. One important benefit is that multiple address-families can be added in an incremental way without affecting neighborship in previously configured address-families. If multisession is disabled on the router, it can be enabled using the **neighbor** *ip-address* **transport multisession** configuration. This command forces the OPEN message to contain only one address-family while establishing the BGP neighbor relationship and therefore one session per address-family.

Single-Session Capability

In single-session mode, BGP adds multiple address-families together in a single OPEN message. In single-session mode, updates for all address-families are exchanged over a single BGP session. All the later IOS releases have this command deprecated, and therefore it is hidden. This is the default mode for forming neighbor relationships. If the router is running multisession in default mode, it can be changed into single-session mode using the **neighbor** *ip-address* **transport single-session** command.

BGP Peer Flapping Issues

A BGP peering down issue is when the BGP neighbor establishment keeps toggling between the Idle and Active states and never goes into the Established state. But when the BGP peer is flapping, it means it is changing state after the session is established. In this case the BGP state keeps flapping between Idle and Established states. Following are the two flapping states in BGP:

- Idle/Active: Discussed in the previous section.
- Idle/Established: Bad Update, TCP Problem (MSS size in multi-hop deployment).

Flapping BGP peers could be due to one of several reasons:

- Bad BGP update
- Hold timer expired
- MTU mismatch
- High CPU
- Interface and platform drops
- Improper control-plane policing

Bad BGP Update

A bad BGP update refers to a corrupted update packet received from a peer. This condition is not normal. It can be caused by one of the following reasons:

- Bad link carrying the update; bad hardware
- Problem with BGP update packaging
- Malicious update by an attacker (hacker)

Whenever a BGP update is corrupted, a BGP notification is generated with the error code of 3, as shown in Table 3-1.

Example 3-24 shows a sample malformed update that caused the BGP session to flap. It shows that one of the attribute length fields has a length higher than the update length itself and is therefore a corrupted update. The BGP message when decoded shows that the BGP update packet length is 93 bytes (0x005D) and the Attribute Type 17 (0x11) the Attribute Length equals to 53572 bytes (0xD144) which is not possible because an individual attribute length cannot exceed a total length of 93 bytes. Therefore the notification of 3/1 was generated, which points to Malformed Attribute List.

Example 3-24 *Malformed BGP Update*

```
Nov 17 09:36:06.990 CET: %BGP-3-NOTIFICATION: sent to neighbor 10.1.13.2 3/1 (update
  malformed) 47 bytes D011D144 02030000 32E60000 12830000 B0

Nov 17 09:36:06.990 CET: BGP: 10.1.13.2 Bad attributes FFFF FFFF FFFF FFFF FFFF FFFF
  FFFF FFFF 005D 0200 0000 4240 0101 0240 020C 0203 32E6 1283 B066 0101 5BA0 D011
  D144 0203 0000 32E6 0000 1283 0000 B066 0101 0001 0049 4003 04D5 9080 C580 0404
  0000 0001 C007 06FF 77AC 11F1 0815 781D F0

Nov 17 09:36:21.518 CET: %BGP-5-ADJCHANGE: neighbor 10.1.13.2 Up
```

Hold Timer Expired

Hold timer expiry is a very common cause for flapping BGP peers. It means that the router didn't receive or process a keepalive message or any update message before the hold timer expired. So it sends a notification message (4/0) and closes the session. On IOS, the keepalive messages are sent by the BGP I/O process, and the BGP Router process interprets the incoming keepalive messages. BGP flaps due to hold timer expiry can be caused by one of the following reasons:

- Interface issues
  - Physical connectivity
  - Physical interface
  - Input hold queue
- TCP receive queue and BGP InQ
- BGP InQ
- Mismatch MTU

Interface Issues

Various interface issues, such as a physical layer concern or drops on the interface, can lead to a BGP session flapping because of hold timer expiry. The sections that follow examine a few interface related problems and various ways to resolve them.

Physical Connectivity

If the problem is with the physical connectivity, packets are not transmitted correctly through the wire, or sometimes it affects only a given pattern of packet. In such a scenario, try to ping between the peering IPs and with different packet sizes and patterns. Setting a different value than 0 for the Type of Service (TOS) field or changing the data pattern itself by setting the value of 0xaabb rather than using the default value of 0xabcd. If the problem is with the physical media, then cyclic redundancy check (CRC) errors and reliability on the interface using the **show interface** command need to be checked. Typically, this problem can be solved by replacing the cable, small form-factor pluggable (SFP), line card, or the chassis, based on the location of the problem.

Physical Interface

Sometimes a packet received via an interface is dropped at the driver level. The most common reason could be that the interface was unable to process the packet because it was receiving traffic at an excessive rate. Check the **show interface** output for *overrun* or *ignore* counters. If there is excessive traffic, that needs to be controlled.

Input Hold Queue

Packets arrive to the router but are dropped in the input hold queue of the incoming interface. These packets are intended to be processed by the router's CPU or are being process switched (processed by software). The hold-queue size is a finite size. On most of Cisco's platforms, the default input hold queue size value is 75 packets and can be configured to a max value of 4096. A good value to be configured on the interface for input hold queue size is between 1500 and 2000. Check the input queue drops on the interface using the **show interface** command. The input queue size can be changed using the **hold-queue** *size* **in** command.

Typically, any router acting as a BGP speaker has the processing capability to process all BGP messages it receives, without having to drop them at the input hold queue. If the drops are seen, the problem is not usually with the BGP packets but with the other process switched traffic or traffic destined to the router received on the interface. Use Selective Packet Discard (SPD), which gives congestion precedence to high-priority traffic like BGP. Because BGP packets are sent with IP Precedence value of 6, they are considered inside the SPD headroom, such as an extra queuing space beyond input hold queue size. To configure SPD, use **spd enable**, and to set the SPD headroom size, use the **spd headroom** *headroom-size* global config commands. The SPD configuration is a hidden configuration, but the values can be checked using the **show ip spd** command output.

Note The headroom size should be enough to hold at least two packets per neighbor.

Example 3-25 shows the various fields to check in the show interface output and also how to configure the interface hold queue and SPD.

Example 3-25 **show interface** *and Configuring* **hold-queue** *and* **spd** *Commands*

```
R2# show interface gigabitEthernet 0/1
GigabitEthernet0/1 is up, line protocol is up
  Hardware is iGbE, address is fa16.3e86.6c2b (bia fa16.3e86.6c2b)
  Internet address is 10.1.12.2/30
  MTU 1500 bytes, BW 1000000 Kbit/sec, DLY 10 usec,
     reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA, loopback not set
  Keepalive set (10 sec)
  Auto Duplex, Auto Speed, link type is auto, media type is RJ45
  output flow-control is unsupported, input flow-control is unsupported
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input 00:00:00, output 00:00:04, output hang never
  Last clearing of "show interface" counters never
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: fifo
  Output queue: 0/40 (size/max)
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
     355 packets input, 32292 bytes, 0 no buffer
     Received 0 broadcasts (0 IP multicasts)
     0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
     0 watchdog, 0 multicast, 0 pause input
     876 packets output, 151705 bytes, 0 underruns
     0 output errors, 0 collisions, 2 interface resets
     0 unknown protocol drops
     0 babbles, 0 late collision, 0 deferred
     1 lost carrier, 0 no carrier, 0 pause output
     0 output buffer failures, 0 output buffers swapped out
```

```
! Configuring Input Hold Queue
R2(config)# interface GigabitEthernet 0/1
R2(config-if)# hold-queue 1500 in
```

```
R2# show ip spd
Current mode: disabled.
Queue min/max thresholds: 73/74, Headroom: 100, Extended Headroom: 75
IP normal queue: 0, priority queue: 0.
SPD special drop mode: none
```

```
! Configuring and verifying SPD

R2# configure terminal
R2(config)# spd enable
R2(config)# spd headroom 1000
R2(config)# end

R2# show ip spd
Current mode: init.
Queue min/max thresholds: 73/74, Headroom: 1000, Extended Headroom: 75
IP normal queue: 0, priority queue: 0.
SPD special drop mode: none
```

Note The SPD feature is available only on Cisco IOS/IOS XE platforms and not on IOS XR or NX-OS platforms.

TCP Receive Queue

Sometimes, BGP keepalives arrive at the TCP receiving queue but are not being processed and moved to the BGP InQ. When a non-zero value is seen for the BGP neighbor in the **show bgp** *afi safi* **summary** command, it indicates that the TCP messages are waiting in queue to be processed. The BGP InQ queues are empty when a BGP neighbor expires. It is possible that the BGP I/O process did not get a chance to run. BGP I/O process is in charge of putting messages from the TCP receiving queue into the BGP InQ. This often occurs if the BGP hold timer values are very low, there are many neighbors, and the CPU is running high.

Another possible reason for this issue could be that just one TCP packet was missed during a transient failure. Although there are some other packets in the TCP receiving queue, the router is still waiting for the first one (TCP has to deliver the packets in order or will not deliver them at all). The sending speaker waits for a transmission timeout and then again sends the first packet. If the BGP hold timer is small, the retransmission may not arrive on time. The only solution is to increase the BGP hold timer. Also Fast Retransmit can help in this scenario. The BGP Fast Retransmit feature resends the first packet after receiving three duplicate ACKs. Therefore, a better workaround is to increase the BGP hold timer and then enable BGP Fast Retransmit.

Finally, the TCP packets can also be dropped because of a CoPP policy. If the CoPP policy does have enough bandwidth allocated for the TCP packets, specifically for BGP, the packets can be dropped. CoPP policy issues are discussed later in this chapter.

MTU Mismatch Issues

Generally MTU is not a big concern when bringing up a BGP neighborship, but maximum transmission unit (MTU) mismatch issues cause BGP sessions to flap. MTU settings vary in different devices in the network because of various factors, such as the following:

- Improper planning and network design
- Device not supporting Jumbo MTU or certain MTU values
- Unknown transport circuits such as EoMPLS (may not support Jumbo MTU end to end)
- Change due to application requirement
- Change due to end customer requirement

BGP sends updates based on the maximum segment size (MSS) value calculated by TCP. If Path-MTU-Discovery (PMTUD) is not enabled and the destination is remote (not on same interface/subnet), the BGP MSS value defaults to 536 bytes as defined in RFC 879. So if there are a huge number of updates getting exchanged between the two routers at the MSS value of 536 bytes, convergence issues are detected, which cause inefficient use of the network. The reason is that the interface is capable of sending three times the MSS value, but it has to break down the updates in chunks of 536 bytes. If the TCP destination is on the same interface (case of non-multihop EBGP), the MSS value will be calculated based on the outgoing interface IP MTU settings.

Defined in RFC 1191, PMTUD is introduced to reduce the chances of IP packets getting fragmented along the path and to help with faster convergence. Using PMTUD, the source identifies the lowest MTU along the path to destination and then decides what packet size to send.

How does PMTUD work? When the source generates a packet, it sets the MTU size equal to the outgoing interface with a DF (Do-Not-Fragment) bit set. Any intermittent device that receives the packet and has an MTU value of its egress interface lower than the packet it received, the device drops the packet and sends an ICMP error message with Type 3 (Destination Unreachable) and Code 4 (Fragmentation needed and DF bit set) along with the MTU information of the outgoing interface in the Next-Hop MTU field back toward the source. When the source receives the ICMP unreachable error message, it modifies the MTU size of the outgoing packet to the value specified in the Next-Hop MTU field above. This process continues until the packet successfully reaches the final destination.

BGP also has support for PMTUD. PMTUD allows a BGP router to discover the best MTU size along the path to the neighbor to ensure efficient usage of exchanging packets. With Path MTU discovery enabled, the initial TCP negotiation between two neighbors has MSS value equal to (IP MTU – 20 byte IP Header – 20 byte TCP Header) and DF bit set. So the IP MTU value is 1500 (equal to the interface MTU) and the MSS value is

1460. If the device in the path has a lower MTU, or even if the destination router has a lower MTU, say 1400, then the MSS value is negotiated based on 1400-40 bytes = 1360 bytes. To derive MSS calculation, use one of the following formulas:

- **MSS without MPLS** = MTU – IP Header (20 bytes) – TCP Header (20 bytes)
- **MSS over MPLS** = MTU – IP Header – TCP Header – n*4 bytes (where n is the number of labels in the label stack)
- **MSS across GRE Tunnel** = MTU – IP Header (Inner) – TCP Header – [IP Header (Outer) + GRE Header (4 bytes)]

Note MPLS VPN Providers should increase the MPLS MTU to at least 1508 (assuming minimum of two labels) or MPLS MTU of 1516 (to accommodate up to four labels).

Example 3-26 demonstrates MSS negotiation with and without PMTUD enabled. Use the **ip tcp path-mtu-discovery** command to enable PMTUD. This command is applicable for IOS and NX-OS. For IOS XR, use the **tcp path-mtu-discovery** command.

Example 3-26 *TCP MSS Negotiation*

```
! Test without PMTUD enabled on Nexus R1 (192.168.1.1)
R2# debug ip tcp transaction
TCP special event debugging is on
R2# clear ip bgp 192.168.1.1
R2#
! Output omitted for brevity
*Sep 27 02:05:48.996: Reserved port 25222 in Transport Port Agent for TCP IP type 1
*Sep 27 02:05:48.996: TCB0F54AA18 getting property TCP_STRICT_ADDR_BIND (19)
*Sep 27 02:05:48.997: TCP: pmtu enabled,mss is now set to 1460
*Sep 27 02:05:48.998: TCP: sending SYN, seq 342586080, ack 0
*Sep 27 02:05:48.998: TCP0: Connection to 192.168.1.1:179, advertising MSS 1460
*Sep 27 02:05:48.999: TCP0: state was CLOSED -> SYNSENT [25222 -> 192.168.1.1(179)]
*Sep 27 02:05:49.002: TCP0: state was SYNSENT -> ESTAB [25222 -> 192.168.1.1(179)]
*Sep 27 02:05:49.002: TCP: tcb F54AA18 connection to 192.168.1.1:179, peer MSS 536,
     MSS is 536
! Output omitted for brevity
*Sep 27 02:05:57.952: %BGP-5-ADJCHANGE: neighbor 192.168.1.1 Up

R2# show bgp ipv4 unicast neighbor 192.168.1.1 | include max
  Number of NLRIs in the update sent: max 6, min 0
sndwnd:  16616  scale:      0  maxrcvwnd:  16384
minRTT: 8 ms, maxRTT: 1000 ms, ACK hold: 200 ms
Datagrams (max data segment is 536 bytes):
```

```
! Enabling PMTUD on Nexus R1 (192.168.1.1)
R1(config)# ip tcp path-mtu-discovery
Path-mtu-discovery enabled
```

```
! Test without PMTUD enabled on Nexus R1 (192.168.1.1)
R2# clear bgp ipv4 unicast 192.168.1.1
! Output omitted for brevity
02:11:56.653: TCP: pmtu enabled,mss is now set to 1460
02:11:56.653: TCP: sending SYN, seq 2163041542, ack 0
02:11:56.653: TCP0: Connection to 192.168.1.1:179, advertising MSS 1460
02:11:56.655: TCP0: state was CLOSED -> SYNSENT [50640 -> 192.168.1.1(179)]
02:11:56.659: TCP0: state was SYNSENT -> ESTAB [50640 -> 192.168.1.1(179)]
02:11:56.660: TCP: tcb FB9C638 connection to 192.168.1.1:179, peer MSS 1444,
     MSS is 1444
! Output omitted for brevity
*Sep 27 02:11:56.895: %BGP-5-ADJCHANGE: neighbor 192.168.1.1 Up
R2# show bgp ipv4 unicast neighbor 192.168.1.1 | include max
  Number of NLRIs in the update sent: max 6, min 0
sndwnd:  17328  scale:      0  maxrcvwnd:  16384
minRTT: 8 ms, maxRTT: 1000 ms, ACK hold: 200 ms
Datagrams (max data segment is 1444 bytes):
```

On IOS and IOS XR routers, PMTUD is enabled by default, whereas on Nexus, it is disabled by default.

Based on Example 3-26 and with an understanding of PMTUD, how can an MTU mismatch cause BGP flapping issues? If there are MTU discrepancies in the path, and if ICMP messages are blocked in the path from source to destination, PMTUD does not function and may result in session flap. After the configuration is done between the source and the destination router for BGP, TCP negotiation are successful, and BGP moves to an Established state. Now when BGP starts sending the update with the negotiated MSS value, it sends it with a DF bit set. If a device in the path or even the destination is not able to accept the packets with higher MTU, it sends an ICMP error message back to BGP speaker. The destination router either waits for the BGP keepalive or BGP update packet to update its hold down timer. After 180 seconds, the destination router sends a notification back to source with a *Hold time expired* error message.

Note When a BGP router sends an update to a BGP neighbor, it does not send a BGP keepalive; instead, the keepalive timer is restarted for that neighbor.

In Figure 3-4, R2 and R4 have a default MTU setting of 1500 where R1 has configured MTU of 1400 with ACL blocking ICMP packets. Now when the TCP session is being

instantiated for BGP, both sides send the initial MSS value of 1460. Because R1 has a lower MTU value for its outgoing interface, it tries to send an ICMP unreachable error message back toward both the routers. But because ICMP messages are blocked, none of the devices receive the ICMP error message. Therefore, both routers R2 and R4 have their TCP session established with the MSS value of 1460. When the router R2 exchanges its BGP table with the router R4, two things can happen:

1. If BGP is advertising only a few prefixes and the BGP Update packet is less than 1360 bytes, the session does not expire.
2. If the number of advertised prefixes is high and the BGP Update packet size is more than 1360 bytes, the BGP session keeps flapping around at the expiry time of the BGP Hold Timer.

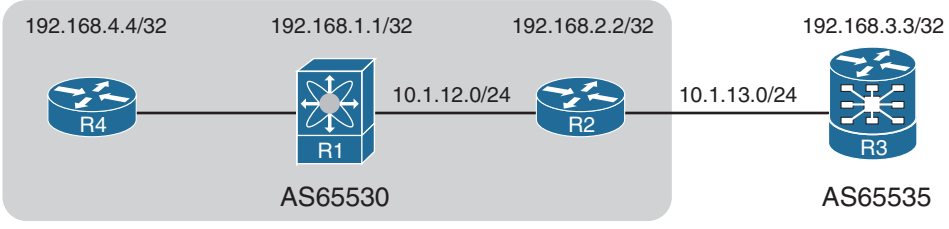

Figure 3-4 *Topology with MTU Mismatch Settings*

Example 3-27 demonstrates BGP flapping due to an MTU mismatch, as previously explained. R2 is having an IBGP neighborship with R4 and EBGP session with R3. R3 is advertising 10,000 prefixes toward R2. The BGP session starts flapping when R2 tries to send updates toward R4.

Example 3-27 *BGP Flapping due to MTU Mismatch*

```
NX-OS
! Blocking ICMP packets and setting MTU value of 1400
R1(config)# ip access-list BlockICMP
R1(config-acl)# deny icmp any any
R1(config-acl)# permit ip any any
R1(config-acl)# exit
R1(config)# interface Ethernet 2/1-2
R1(config-if-range)# mtu 1400
R1(config-if-range)# ip access-group BlockICMP in
R1(config-if-range)# ip access-group BlockICMP out
```

```
IOS
R2# show bgp ipv4 unicast summary
```

```
! Output omitted for brevity
Neighbor        V           AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
10.1.13.2       4        65535    1006       8    10005    0    0 00:02:21    10001
192.168.1.1     4        65530       5    1006    10005    0    0 00:00:41        1
192.168.4.4     4        65530       4      14    10005    0  992 00:00:15        0
```

IOS

```
R4# show bgp ipv4 unicast summary
BGP router identifier 192.168.4.4, local AS number 65530
BGP table version is 1, main routing table version 1

Neighbor        V           AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
192.168.2.2     4        65530       3       5        1    0    0 00:01:16        0
R4# show bgp ipv4 unicast summary
BGP router identifier 192.168.4.4, local AS number 65530
BGP table version is 1, main routing table version 1

Neighbor        V           AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
192.168.2.2     4        65530       3       7        1    0    0 00:02:59        0
R4#
07:35:26.950: %BGP-3-NOTIFICATION: sent to neighbor 192.168.2.2 4/0
(hold time expired) 0 bytes
*Sep 27 07:35:26.952: %BGP-5-NBR_RESET: Neighbor 192.168.2.2 reset
(BGP Notification sent)
07:35:26.955: %BGP-5-ADJCHANGE: neighbor 192.168.2.2 Down BGP Notification sent
07:35:26.955: %BGP_SESSION-5-ADJCHANGE: neighbor 192.168.2.2 IPv4 Unicast topology
base removed from session  BGP Notification sent
07:35:38.229: %BGP-5-ADJCHANGE: neighbor 192.168.2.2 Up
```

The following are a few possible causes of BGP session flapping due to an MTU mismatch:

- The interface MTU on both the peering routers do not match.
- The Layer 2 path between the two peering routers do not have consistent MTU settings.
- PMTUD didn't calculate correct MSS for the TCP BGP session.
- BGP PMTUD could be failing because of blocked ICMP messages by a router or a firewall in path.

To verify there are MTU mismatch issues in the path, perform extended **ping** tests by setting the size of the packet as the outgoing interface MTU value along with DF bit set. Also, ensure that ICMP messages are not being blocked in the path to have PMTUD function properly. An accurate review of the configuration should be done to ensure that the MTU values are consistent throughout the network.

High CPU Causing Control-Plane Flaps

The CPU on a router is continuously performing several tasks. Some tasks are low level, such as scheduling or other kernel-related tasks, and some are high level. But the CPU is capable of performing most of the tasks on a router. Problems arise when a process takes more CPU cycles than anticipated or the CPU has to process the packets, such as process switching of packets. High CPU conditions are caused by one of the following reasons:

- CPU process issues
- Interrupt (traffic processing)

Because all the control plane packets are processed by the CPU and are not switched in the hardware, a high CPU condition can cause control plane packets to get dropped or get delayed. This can be Bidirectional Failure Detection (BFD) or BGP or any other control plane packets. Note that the packets destined to the CPU are called the *for-us* packets.

Example 3-28 shows the output of **show process cpu sorted** to verify the CPU utilization on the router. The value 99% shows the total CPU utilization percentage, the value 7% indicates the CPU utilization due to interrupt, and the SNMP ENGINE process is consuming the most CPU cycles. The output also shows a 1-minute and 5-minute average CPU utilization.

Example 3-28 show process cpu sorted *Output*

```
Rtr# show process cpu sorted | exclude 0.0
CPU utilization for five seconds: 99%/7%; one minute: 84%; five minutes: 38%
PID Runtime(ms)    Invoked      uSecs    5Sec    1Min    5Min TTY Process
212   7157624761849980789        386 82.71% 70.16% 25.65%   0 SNMP ENGINE
  8   9190575241399458158        656  1.51%  1.47%  1.47%   0 ARP Input
318     76225596 348324353        218  1.11%  0.43%  0.16%   0 OSPF Router 220
! Output omitted for brevity
```

It is also good to check the CPU history using the **show processes cpu history** command, which shows the CPU utilization graphs for last 72 hours. This helps identify when the problem actually started, and events that occurred around that time can be investigated to troubleshoot the problem.

If the CPU is high due to process, it is important to understand the role of the process on the router. For example, the CPU is reportedly high in the output shown in Example 2-28 because of the SNMP Engine process. So it's clear that the SNMP process is busy doing something on the router. The **show snmp** command output shows how many packets the SNMP process is processing. If the SNMP process is receiving or sending too many packets, the SNMP process can consume lot of CPU cycles.

If the CPU is high due to interrupts, it could be due to one of the following problems:

- Excess process switched packets
- Packets with TTL value of 1
- Excess control plane packets

Excess process switching can happen because of Cisco Express Forwarding (CEF) not being enabled on the router or CEF not having forwarding information for the packet. If **ip cef** is configured globally but **no ip route-cache cef** is configured under the interface (some features require CEF to be disabled on the interface), this makes the traffic being received on the interface to be process switched. Also, insufficient memory on the router can lead to CEF/Forwarding Information Base (FIB) getting disabled on the router or line card. Therefore, it is important to verify CEF as well as memory on the router to troubleshoot excess process switching packets.

Every IP packet has a TTL value. The TTL value ensures that the packet doesn't go into a loop and create problems in a Layer 3 network. But packets with TTL size 1 can cause an impact on the CPU utilization of the router. Regular control plane packets like OSPF hello or BGP keepalive packets for EBGP session have a TTL value of 1, but those are expected packets. A normal IP packet that is not destined to any of the router interfaces but is destined for devices behind the router and has a TTL value of 1 gets punted to the router CPU to get processed. Because the router is not the final destination of the packet, the CPU drops those packets and consumes some CPU cycles. Another example is a configuration of a multicast application that has been set with a wrong TTL value, therefore causing the multicast packets to get punted to the CPU.

The CPU on the route processor processes the control plane packets, but if those packets are excessive and aggressive in nature, they can overwhelm the CPU. For example, 500 BGP neighbors configured with aggressive timers on the router can have a huge number of keepalive packets hitting the CPU to get processed or hundreds of IP service-level agreement (SLA) probes configured toward the router from various sources can send too many ICMP packets and spike up the CPU a bit. Most router CPUs are capable of handling that many packets. The challenge comes when an outside attacker sends thousands of BGP packets or a bad virtual host sends an enormous amount of ARP request packets toward the router. These situations can overwhelm the CPU on the router and impact services running on the router.

The following methods help mitigate the problems caused by packets hitting the CPU:

- Configuring an ACL to block the packets once identified
- Configuring rate limiters
- Using Control Plane Policing (CoPP)

After the packets hitting the CPU are identified and their source and destination are known, use the tools described in Chapter 2, "Generic Troubleshooting Methodologies," to block the packets by using an ACL on the interface.

Configuring rate limiters is another option to limit certain traffic from hitting the CPU. After the configured threshold value is reached, the packets are dropped by the hardware or software based on the platform. For example, 6500 and 7600 platform is Multi-Layer Switch (MLS)–based architecture. There are various MLS rate limiters available on this platform. For example an MLS rate limiter can be configured for ICMP redirect or multicast traffic using the **mls rate-limit** command to protect the CPU.

On NX-OS platforms, there are various platform-specific rate limiters that can be configured using the command **platform rate-limit** [**access-log-list** | **layer-2 port-security** | **layer-2 storm-control** | **layer-3 control** | **layer-3 glean** | **layer-3 mtu** | **layer-3 multicast** {**directly-connected** | **local-groups** | **rpf-leak**} | **layer-3 ttl** | **receive**] *packets*. The **show hardware rate-limit** can be used to display the configured rate limiters. XR platform takes care of rate limiting the packets using Local Packet Transport Services (LPTS). LPTS is not just used for rate limiting but also performs Control Plane Policing.

Control Plane Policing

Denial-of-Service (DoS) attacks can take on many forms, affecting both servers and infrastructure. Attacks targeted at infrastructure devices can generate IP traffic streams at very high data rates. These IP data streams contain packets that are destined for processing by the control plane of the Route Processor (RP). Based on the high rate of rogue packets presented to the RP, the control plane is forced to spend an inordinate amount of time, processing this DoS traffic. This scenario can result in one of the following issues:

- Loss of line protocol keepalives, which can cause a line to go down and lead to route flaps and major network transitions.
- Loss of routing protocol updates, which can lead to route flaps and major network transitions.
- Near 100% CPU utilization can lock up the router and prevent it from completing high-priority processing, resulting in other negative side effects.
- When the RP is near 100% utilization, the response time at the user command line interface (CLI) is very slow or the CLI is locked out. This prevents the user from taking corrective action to respond to the attack.
- Resources including memory, buffers, and data structures can be consumed causing negative side effects.
- Packet queues back up leading to indiscriminate drops of important packets.
- Router crashes.

The *Control Plane Policing (CoPP)* feature increases the device security by protecting its CPU (Route Processor) from unwanted and excess traffic or Denial of Service (DoS) attacks and gives priority to relevant traffic destined to the CPU. Different types of traffic can be organized in different classes and policed under the policy-maps. If a

violate-action is set to drop, the excess packets drop. If they are marked to forward, those packets are forwarded. The service-policy is then attached to the control plane using the **service-policy input** *policy-name* command under the **control-plane** configuration. Example 3-29 shows a sample CoPP policy on an IOS router to protect BGP packets. Any other traffic that is not classified in one of the classes defined under policy-map falls under the default class.

Example 3-29 *CoPP Configuration on IOS*

```
R2(config)# ip access-list extended CPP_BGP
R2(config-ext-nacl)# permit tcp any eq bgp any
R2(config-ext-nacl)# permit tcp any any eq bgp
R2(config-ext-nacl)# exit
R2(config)# class-map Protect_BGP
R2(config-cmap)# match access-group name CPP_BGP
! Define other class-maps for matching other traffic
R2(config)# policy-map CPP_Protection
R2(config-pmap)# class Protect_BGP
R2(config-pmap-c)# police cir 36000 3000 conform-action transmit violate action drop
! Configure Policing for other defined classes
R2(config-pmap-c)# control-plane
R2(config-cp)# service-policy input CPP_Protection

19:27:30.292: %CP-5-FEATURE: Control-plane Policing feature enabled on
      Control plane aggregate path
```

Once the CoPP policy has been defined, use the **show policy-map control-plane** command to see the traffic hitting various classes and verify any violated packets in those classes. Example 3-30 shows the output of show policy-map control-plane command.

Example 3-30 show policy-map control-plane *Output*

```
R2# show policy-map control-plane
 Control Plane

  Service-policy input: CPP_Protection

    Class-map: Protect_BGP (match-all)
     215 packets, 15983 bytes
      5 minute offered rate 0000 bps, drop rate 0000 bps
      Match: access-group name CPP_Protocols
      police:
          cir 36000 bps, bc 3000 bytes, be 3000 bytes
        conformed 215 packets, 15983 bytes; actions:
          transmit
```

```
        exceeded 0 packets, 0 bytes; actions:
          transmit
        violated 0 packets, 0 bytes; actions:
          drop
        conformed 0000 bps, exceeded 0000 bps, violated 0000 bps

! Output for other defined classes follows

! Output omitted for brevity

    Class-map: class-default (match-any)
      69 packets, 19610 bytes
      5 minute offered rate 0000 bps, drop rate 0000 bps
      Match: any
```

Example 3-30 demonstrates manual configuration of CoPP policy on Cisco IOS routers.

CoPP on NX-OS

On Nexus platforms, the NX-OS installs a default *copp-system-policy* policy to protect the device from DoS attacks and excessing packet processing. Various profiles that have different protection levels provided for NX-OS default CoPP policy include the following:

- **Strict:** This policy defines the BC value of 250 ms for regular classes, and for important classes it's set to 1000 ms.
- **Moderate:** This policy defines the BC value of 310 ms for regular classes, and for important classes it's set to 1250 ms.
- **Lenient:** This policy defines the BC value of 375 ms for regular classes, and for important classes it's set to 1500 ms.
- **Dense:** Introduced in 6.0(1) release.

If one of the policies is not selected during initial setup, NX-OS attaches the Strict profile to the control plane. The user can choose not to use one of these profiles and create a custom policy to be used for CoPP. The NX-OS default CoPP policy categorizes policy into various predefined classes, such as the following:

- **Critical:** Routing protocol packets with IP Precedence value 6
- **Important:** Redundancy protocols like GLBP, VRRP, HSRP, etc.
- **Management:** All management traffic, such as Telnet, SSH, FTP, NTP, Radius, and so on
- **Monitoring:** Ping and Traceroute traffic
- **Exception:** ICMP Unreachables and IP Options
- **Undesirable:** All unwanted traffic

Because the ICMP messages are also rate-limited, a few ICMP packets are dropped when performing a ping from and to the Nexus device. Example 3-31 shows a Sample Strict CoPP policy when the system comes up for the first time.

Example 3-31 *CoPP Strict Policy on Nexus*

```
class-map type control-plane match-any copp-system-p-class-critical
  match access-group name copp-system-p-acl-bgp
  match access-group name copp-system-p-acl-rip
  match access-group name copp-system-p-acl-vpc
  match access-group name copp-system-p-acl-bgp6
  match access-group name copp-system-p-acl-lisp
  match access-group name copp-system-p-acl-ospf
  ! Output omitted for brevity
class-map type control-plane match-any copp-system-p-class-exception
  match exception ip option
  match exception ip icmp unreachable
  match exception ipv6 option
  match exception ipv6 icmp unreachable
class-map type control-plane match-any copp-system-p-class-important
  match access-group name copp-system-p-acl-cts
  match access-group name copp-system-p-acl-glbp
  match access-group name copp-system-p-acl-hsrp
  match access-group name copp-system-p-acl-vrrp
  match access-group name copp-system-p-acl-wccp
! Output omitted for brevity
class-map type control-plane match-any copp-system-p-class-management
  match access-group name copp-system-p-acl-ftp
  match access-group name copp-system-p-acl-ntp
  match access-group name copp-system-p-acl-ssh
  match access-group name copp-system-p-acl-ntp6
  match access-group name copp-system-p-acl-sftp
  match access-group name copp-system-p-acl-snmp
  match access-group name copp-system-p-acl-ssh6
! Output omitted for brevity
class-map type control-plane match-any copp-system-p-class-monitoring
  match access-group name copp-system-p-acl-icmp
  match access-group name copp-system-p-acl-icmp6
  match access-group name copp-system-p-acl-mpls-oam
  match access-group name copp-system-p-acl-traceroute
  match access-group name copp-system-p-acl-http-response
! Output omitted for brevity
class-map type control-plane match-any copp-system-p-class-normal
  match access-group name copp-system-p-acl-mac-dot1x
```

```
  match exception ip multicast directly-connected-sources
  match exception ipv6 multicast directly-connected-sources
  match protocol arp
class-map type control-plane match-any copp-system-p-class-undesirable
  match access-group name copp-system-p-acl-undesirable
  match exception fcoe-fib-miss

policy-map type control-plane copp-system-p-policy-strict
  class copp-system-p-class-critical
    set cos 7
    police cir 36000 kbps bc 250 ms conform transmit violate drop
  class copp-system-p-class-important
    set cos 6
    police cir 1400 kbps bc 1500 ms conform transmit violate drop
  class copp-system-p-class-management
    set cos 2
    police cir 10000 kbps bc 250 ms conform transmit violate drop
  class copp-system-p-class-normal
    set cos 1
    police cir 680 kbps bc 250 ms conform transmit violate drop
  class copp-system-p-class-exception
    set cos 1
    police cir 360 kbps bc 250 ms conform transmit violate drop
  class copp-system-p-class-monitoring
    set cos 1
    police cir 130 kbps bc 1000 ms conform transmit violate drop
  class class-default
    set cos 0
    police cir 100 kbps bc 250 ms conform transmit violate drop
```

There are a few more classes added into the latest releases for traffic related to multicast and L2 traffic, but the configuration is all available automatically. Keep in mind that the CoPP policy should not be too aggressive and should be designed based on the network design and configuration. For example, if the rate at which routing protocol packets are hitting the CoPP policy is more than the policed rate, then even the legitimate sessions are dropped and protocol flaps are detected. If the predefined CoPP policies have to be modified, a custom CoPP policy can be created by copying a preclassified CoPP policy and then editing the new custom policy. Any of the predefined CoPP profiles cannot be edited. Also, the CoPP policies are hidden from the **show running-config** output. View the CoPP policies from the **show running-config all** or **show running-config copp all**

command. Example 3-32 shows how to view the CoPP policy configuration and create a custom strict policy.

Example 3-32 *Viewing CoPP Policy and Creating Custom CoPP Policy*

```
R1# show running-config copp
copp profile strict
R1# show running-config copp all
class-map type control-plane match-any copp-system-p-class-critical
  match access-group name copp-system-p-acl-bgp
  match access-group name copp-system-p-acl-rip
  match access-group name copp-system-p-acl-vpc
  match access-group name copp-system-p-acl-bgp6
! Output omitted for brevity
```

```
R1# copp copy profile strict ?
  prefix  Prefix for the copied policy
  suffix  Suffix for the copied policy
R1# copp copy profile strict prefix custom
R1# configure terminal
R1(config)# control-plane
R1(config-cp)# service-policy input custom-copp-policy-strict
```

The command **show policy-map interface control-plane** displays the counters of the CoPP policy. For an aggregated view, use this command with the **include "class|conform|violated"** filter to see how many packets have been conformed and how many have been violated and dropped, as shown in Example 3-33.

Example 3-33 **show policy-map interface control-plane** *Output*

```
R1# show policy-map interface control-plane | include "class|conform|violated"
    class-map custom-copp-class-critical (match-any)
        conformed 123126534 bytes; action: transmit
        violated 0 bytes; action: drop
        conformed 0 bytes; action: transmit

        violated 0 bytes; action: drop
        conformed 107272597 bytes; action: transmit
        violated 0 bytes; action: drop
        conformed 0 bytes; action: transmit
        violated 0 bytes; action: drop
    class-map custom-copp-class-important (match-any)
        conformed 0 bytes; action: transmit
        violated 0 bytes; action: drop
        conformed 0 bytes; action: transmit
        violated 0 bytes; action: drop
```

```
        conformed 0 bytes; action: transmit
        violated 0 bytes; action: drop
        conformed 0 bytes; action: transmit
        violated 0 bytes; action: drop
! Output omitted for brevity
```

As a best practice, the **copp profile strict** command should be used after each NX-OS upgrade, or at least after each major NX-OS upgrade. If a CoPP policy modification was previously done, it must be reapplied.

Starting with NX-OS release 5.1, the threshold value can be configured to generate a syslog message for the drops enforced by the CoPP policy on a particular class. The syslog messages are generated when the drops within a traffic class exceed the user-configured threshold value. Configure the threshold by using the logging **drop threshold** *dropped-bytes-count* [**level** *logging-level*] command. Example 3-34 demonstrates how to configure the logging threshold value to be set for 100 drops and logging level 7. It also shows how the syslog message is generated in the event of the drop threshold being exceeded.

Example 3-34 *Drop Threshold for Syslog Logging*

```
R1(config)# policy-map type control-plane custom-copp-policy-strict
R1(config-pmap)# class custom-copp-class-critical
R1(config-pmap-c)# logging drop threshold ?
  <1-80000000000>  Dropped byte count

R1(config-pmap-c)# logging drop threshold 100 ?
  <CR>
  level  Syslog level

R1(config-pmap-c)# logging drop threshold 100 level ?
  <1-7>  Specify the logging level between 1-7

R1(config-pmap-c)# logging drop threshold 100 level 7
%COPP-5-COPP_DROPS5: CoPP drops exceed threshold in class:
custom-copp-class-critical,
check show policy-map interface control-plane for more info.
```

Here are a few best practices for NX-OS CoPP policy configuration:

- Use strict CoPP mode by default.
- Dense CoPP profile is recommended when the chassis is fully loaded with F2 Series Modules or loaded with more F2 Series Modules than any other I/O modules.
- It is not recommended to disable CoPP. Tune the default CoPP, as needed.

- Monitor unintended drops, and add or modify the default CoPP policy in accordance to the expected traffic.
- The CoPP policy must be periodically evaluated and adjusted as network conditions change over time.

Local Packet Transport Services

Like IOS and NX-OS, IOS XR doesn't have CoPP. Instead, it uses a comprehensive and powerful feature called Local Packet Transport Services (LPTS). LPTS is an integral component of IOS-XR, which provides firewall and policing functionality. It also decides which of the for-us packets needs to be punted to the CPU or dropped. LPTS provides hardware policers on the line card that limits traffic sent to the RP.

LPTS is based on the concept of Reflexive ACLs and punt policers. It also has an Internal FIB (IFIB) that takes care of directing packets to certain nodes. Figure 3-5 shows a 1000-foot view of how LPTS functions on IOS XR.

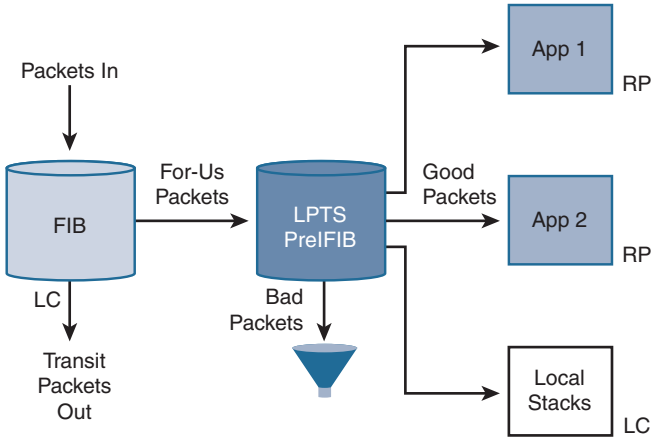

Figure 3-5 *LPTS on IOS XR*

LPTS performs a firewall function by allowing only *for-us* protocol packets into the router for only the configured application/server processes. LPTS also rate limits the *for-us* packets entering the line card (LC) so that nonconfirming packets are dropped on the ingress LC itself. LPTS installs dynamic flow entries for certain protocol packets on demand. For example, for an ICMP echo request sent out from the local router to peer router, LPTS creates flow entry in pre-IFIB (condensed version of IFIB) so that an ICMP echo reply received from the peer router will be matched against it.

Should any *for-us* packets arrive at the LC fragmented, packet lookup can be performed to determine where to deliver only upon reassembling the fragments. LPTS provides the reassembly function for fragmented *for-us* packets.

One of the most important functionalities of LPTS is dynamic ACL creation. This is configuration driven, based on the feature or service configured, so no user intervention is required. The stats indicated in Example 3-30 are from the line card in location 0/1/CPU0. If the location option is not specified, the stats shown are from the active RP.

There are various tables in LPTS at different locations maintained by hardware and software. This is very important to understand because the drops can happen in any of those tables based on the forwarding and punt path. The various tables are as follows:

- **Pre-IFIB (PIFIB):** Maintained at Ingress PSE (Packet Switching Engine) TCAM Hardware of the line card.
- **Software Pre-IFIB:** Maintained on line card as well as RP. The Software PIFIB takes care of more complex inspection and operations, such as fragmentation reassembly.
- **IFIB "Slices":** These are distributed across the RP. These act as secondary lookup tables when the PIFIB is incomplete. The IFIB table is distributed in slices, such as TCP Slice, UDP Slice, ISIS Slice, and so on.

The following steps illustrate how the bindings occur for various tables in LPTS:

Step 1. At boot, the IFIB is empty, the bindings, PIFIB software and hardware tables have default entries.

Step 2. As the box is configured, the LPTS IFIB populates, and the bindings and pifib are updated.

Step 3. The first message for a configured entity (for example, BGP), hits the *any.any* entry (Unknown State) in the PIFIB hardware.

Step 4. Subsequent matches hit more granular entries.

Step 5. When peers oscillate, entries are added and deleted.

LPTS also maintains different flow categories based on the protocol state. For instance, BGP has three states:

- **Unknown:** Traffic flow with TCP port 179 but no neighbor is configured for that source. This resolves to *BGP-default* flow type.
- **Configured:** Flow with source address of the peer but the session is not established yet. This resolves to *BGP-cfg-peer* flow type. The neighbor statement configured under the **router bgp** section drives these entries.
- **Established:** Flow with an established state with relevant L3 and L4 information. This flow is policed lightly and the flow resolves to *BGP-known* flow type.

Example 3-35 shows the output of the **show lpts pifib hardware police** [**location** *location-id*] and **show lpts pifib hardware entry brief** [**location** *location-id*] commands. All the preceding BGP states can be seen with the use of these commands.

Example 3-35 *BGP States in LPTS*

```
RP/0/0/CPU0:R3# show lpts pifib hardware police location 0/1/CPU0
--------------------------------------------------------------
                Node 0/1/CPU0:
--------------------------------------------------------------
 Burst = 100ms for all flow types
--------------------------------------------------------------
FlowType       Policer Type    Cur. Rate  Def. Rate  Accepted   Dropped  TOS Value
-------------  ------- ------- ---------- ---------- ---------  -------- ----------
BGP-known      106     Static   2500       2500       200          0      01234567
BGP-cfg-peer   107     Static   2000       2000       0            0      01234567
BGP-default    108     Static   1500       1500       5            0      01234567
! Output omitted for brevity
```

```
RP/0/0/CPU0:R3# show lpts pifib hardware entry brief location 0/1/CPU0
             Node: 0/1/CPU0:
--------------------------------------------------------------
Offset L3   VRF id       L4     Intf      Dest       laddr,Port raddr,Port
------ ---- ------------ ------ ------ ---------  ----------------------
! Below entry shows Configured state
22      IPV4  default    TCP     any    LU(30)     any,179 10.1.13.1,any
! Below entry show Established State
24      IPV4  default    TCP     any    LU(30)     10.1.13.2,57286 10.1.13.1,179
! Below entry shows Unknown State
142     IPV4 *           TCP     any    LU(30)     any,any any,179
! Output omitted for brevity
```

Example 3-36 shows the use of the **show lpts ifib all brief** command to display various entries under different slices. For example, the BGP entry for neighbor 10.1.13.1 is under BGP4 slice. Also, the use of the **statistics** keyword at the end of the command is used to verify packets that have been accepted and dropped counters to correlate the drops happening in Internal Forwarding Information Base (IFIB).

Example 3-36 **show lpts ifib all brief** *[statistics] Command Output*

```
RP/0/0/CPU0:R3# show lpts ifib all brief
Slice    VRF-ID   L4     Interface  Dlvr       laddr,Port raddr,Port
-------- -------- ------ --------- ---------- ----------------------------------
RAWIP4   default  L2TPV3 any       0/0/CPU0    any any
RAWIP6   default  ICMP6  any       0/0/CPU0    any,MLDLQUERY any
RAWIP6   default  ICMP6  any       0/0/CPU0    any,LSTNRREPORT any
RAWIP6   default  ICMP6  any       0/0/CPU0    any,MLDLSTNRDN any
RAWIP6   default  ICMP6  any       0/0/CPU0    any,LSTNRREPORT any
```

```
BGP4      default  TCP    any        0/0/CPU0    10.1.13.2,179 10.1.13.1
BGP4      default  TCP    any        0/0/CPU0    10.1.13.2,57286 10.1.13.1,179
```

```
RP/0/0/CPU0:R3# show lpts ifib all brief statistics
Slice  VRF-ID   L4     Interface  Accept/Drop   laddr,Port raddr,Port
------ ------- ------ ---------- -----------  ----------------------------------
RAWIP4 default L2TPV3 any          0/0           any any
RAWIP6 default ICMP6  any          0/0           any,MLDLQUERY any
RAWIP6 default ICMP6  any          0/0           any,LSTNRREPORT any
RAWIP6 default ICMP6  any          0/0           any,MLDLSTNRDN any
RAWIP6 default ICMP6  any          0/0           any,LSTNRREPORT any
BGP4   default TCP    any          0/0           10.1.13.2,179 10.1.13.1
BGP4   default TCP    any          6/0           10.1.13.2,57286 10.1.13.1,179

 Slice    Num. Entries Accepts/Drops
 -------- ------------ -------------
 RAWIP4   1            0/0
 RAWIP6   4            0/0
 BGP4     2            6/0
 Total    7            6/0
```

The Pre-Internal Forwarding Information Base (PIFIB) software table is viewed by using the command **show lpts pifib brief** [**statistics**], where the statistics displays the *Accept* and *Drop* counters. With the use of the hardware keyword, as shown in Example 3-35, the hardware-based PIFIB counters are seen. Example 3-37 demonstrates the software PIFIB table showing there were 100 packets accepted for the BGP session established between 10.1.13.1 and 10.1.13.2. Also, it shows cumulative statistics for various types of packets.

Example 3-37 *Software PIFIB Table*

```
RP/0/0/CPU0:R3# show lpts pifib brief statistics
* - Any VRF;
Type       VRF-ID    L4   Interface Accepts/Drops laddr,Port raddr,Port
---------- -------- ------ -------  ------------- ---------------------------------
! Output omitted for brevity
IPv4       default  TCP    any          100/0       10.1.13.2,57286 10.1.13.1,179
IPv4       default  TCP    any          0/0         10.1.13.2,179 10.1.13.1
IPv4       *        TCP    any          6/0         any any,179
IPv4       *        TCP    any         1/0          any,179 any
! Output omitted for brevity
------------------------
```

```
statistics:
Type            Num. Entries         Accepts/Drops
------          ------------         -------------
ISIS            1                    0/0
IPv4_frag       1                    0/0
IPv4_echo       1                    174/0
IPv4            15                   182/0
IPv6_frag       1                    0/0
IPv6_echo       1                    0/0
IPv6_ND         5                    0/0
IPv6            13                   0/0
BFD_any         0                    0/0
Total           38
Packets into Pre-IFIB: 361
Lookups: 361
Packets delivered locally: 361
Packets delivered remotely: 0
```

All these commands can be used with the location keyword, which allows you to look at the counters on a particular line card or RP.

Dynamic BGP Peering

Configuring BGP neighbors is not so complex. But manually configuring a large number of peers (100 BGP neighbors, for example) could be tedious. One way to minimize the configuration is by using BGP peer groups. If there are multiple neighbors that will share the same **remote-as** number or the same outbound policies, peer groups make it very easy to manage the configuration for those neighbors. But again, those 100 BGP peers have to be manually configured and added to the peer group. With the BGP dynamic neighbors feature, BGP peering can be established with a group of remote neighbors that are defined by an IPv4 address range. This feature is not available for IPv6 addresses.

Note At the time of writing, dynamic BGP neighbor feature is not available on IOS XR and NX-OS.

The BGP dynamic neighbor concept is helpful in a hub-spoke topology where only the spoke router needs to have the peering configuration toward the hub. The spoke routers can be part of the same subnet. The hub router only needs to know the subnet. It can

also be useful in topologies where RR is configured and there are huge numbers of RR clients. Similarly, a dynamic BGP peering concept can be used with confederations.

How does BGP dynamic neighbor work? For every peer group, an IP subnet range is configured, and the router starts listening in passive mode for incoming TCP sessions for that subnet. A BGP peer tries to establish a BGP neighborship with the passively listening router and initiates a TCP session. If the IP and the AS number matches with any of the configured IP subnets, a BGP session is dynamically created on the router.

Dynamic BGP Peer Configuration

The dynamic BGP neighbor configuration is a one-time configuration. A router can be configured to dynamically form neighborship in few simple steps, as shown:

Step 1. Define the peer group by using Rtr(config-router)# **neighbor** *peer-group-name* **peer-group**.

Step 2. Create a global limit of BGP dynamic subnet range neighbors. The value ranges from 1 to 5000. Rtr(config-router)# **bgp listen limit** *value*.

Step 3. Configure an IP Subnet Range and associate it with a peer group. Multiple subnets can be added to the same peer group. Rtr(config-router)# **bgp listen range** *subnet* **peer-group** *peer-group-name*.

Step 4. Define the remote-as for the peer group. Optionally, define the list of AS numbers that can be accepted to form neighborship with. The max limit of alternate-as numbers is 5. Rtr(config-router)# **neighbor** *peer-group-name* **remote-as** *asn* [**alternate-as** [*asn*] [*asn*] [*asn*] [*asn*] [*asn*]].

Step 5. Activate the peer group under ipv4 address-family by using Rtr(config-router-af)# **neighbor** *peer-group-name* **activate**.

A few things to note from the configuration steps are as follows:

- The maximum number of dynamic BGP neighbors that can be configured is 5000.
- The defined IP subnet range can form a BGP neighbor relationship with peers in six different AS numbers. The first is defined as part of **remote-as** configuration, and optional AS numbers are defined as part of **alternate-as** configuration.

Note The alternate-as option is not available when configuring IBGP sessions.

Figure 3-6 shows a sample topology for dynamic BGP peering. In this figure, R1 forms a dynamic neighborship with routers from R2, R3, and R4.

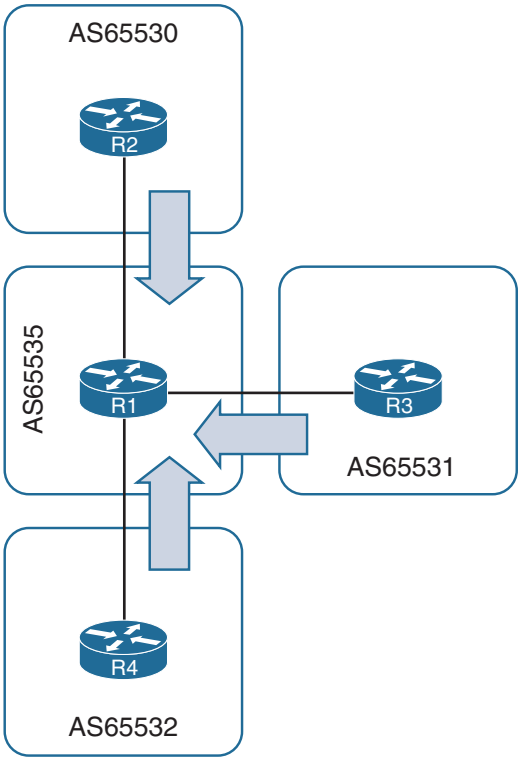

Figure 3-6 *Dynamic BGP Peering*

Example 3-38 demonstrates a dynamic BGP neighbor configuration. It shows the configuration from router R1 and R2. R3 and R4 have a similar configuration as R2.

Example 3-38 *Dynamic BGP Neighbor Configuration*

```
R1# show running-config | section router bgp
 router bgp 65535
  bgp log-neighbor-changes
  bgp listen range 10.1.0.0/16 peer-group DYNAMIC-BGP
  bgp listen limit 200
  neighbor DYNAMIC-BGP peer-group
  neighbor DYNAMIC-BGP remote-as 65530 alternate-as 65531 65532 65533
  !
  address-family ipv4
! Loopback Advertisement
   network 192.168.1.1 mask 255.255.255.255
! peer group activated
```

```
  neighbor DYNAMIC-BGP activate
 exit-address-family
```

```
R2# show running-config | section router bgp
 router bgp 65530
  bgp router-id 192.168.2.2
  bgp log-neighbor-changes
  no bgp default ipv4-unicast
  neighbor 10.1.12.1 remote-as 65535
  !
  address-family ipv4
   network 192.168.2.2 mask 255.255.255.255
   neighbor 10.1.12.1 activate
  exit-address-family
```

With dynamic BGP neighbor configuration, the **show tcp brief all** output illustrates that the router is listening on port 179 but with foreign address of *.*. Also, all the dynamically established BGP peerings have an * marked in the **show bgp ipv4 unicast summary** output, as shown in Example 3-39.

Example 3-39 *Verifying Dynamic BGP Peers*

```
R1# show tcp brief all
 TCB        Local Address               Foreign Address             (state)
 0C89CDD0   10.1.14.1.179              10.1.14.2.22059             ESTAB
 0E4CC190   10.1.12.1.179              10.1.12.2.62747             ESTAB
 0C88F3A0   10.1.13.1.179              10.1.13.2.58756             ESTAB
 0D8B3B08   0.0.0.0.179                *.*                         LISTEN
```

```
R1# show bgp ipv4 unicast summary
 BGP router identifier 10.1.12.1, local AS number 65535
 BGP table version is 7, main routing table version 7
 4 network entries using 576 bytes of memory
 4 path entries using 320 bytes of memory
 4/4 BGP path/bestpath attribute entries using 608 bytes of memory
 3 BGP AS-PATH entries using 72 bytes of memory
 0 BGP route-map cache entries using 0 bytes of memory
 0 BGP filter-list cache entries using 0 bytes of memory
 BGP using 1576 total bytes of memory
 BGP activity 5/1 prefixes, 5/1 paths, scan interval 60 secs
```

```
Neighbor        V      AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
*10.1.12.2      4   65530     377     377        7    0    0 05:38:22        1
*10.1.13.2      4   65531     376     378        7    0    0 05:37:37        1
*10.1.14.2      4   65532      80      84        7    0    0 01:08:03        1
* Dynamically created based on a listen range command
Dynamically created neighbors: 3, Subnet ranges: 1
BGP peergroup DYNCMIC-BGP listen range group members:
  10.1.0.0/16
Total dynamically created neighbors: 3/(200 max), Subnet ranges: 1
```

Dynamic BGP Challenges

With dynamic BGP features, additional challenges are present, such as

- Misconfigured MD5 password
- Resource issues in a scaled environment
- TCP starvation

Misconfigured MD5 Password

This problem is very common and is generally caused by human error due to typo mistakes. You have to be careful when configuring passwords on the router configured for dynamically establishing a BGP neighbor relationship.

Resource Issues in a Scaled Environment

Because up to 5000 neighbors can form peering dynamically, the IP subnet range in the **bgp listen** command must be configured carefully. If there are too many neighbors allowed to form peering than the router can handle, it could lead to a severe impact on the network. At some point the resource limit will be hit, and the router will not have any resources to serve any request coming to it. So proper planning must be done to determine how many neighbors can dynamically form BGP neighbor relationships on the router.

TCP Starvation

Because of the TCP slow start capability and non-availability of this for UDP traffic, a router can run into a TCP starvation condition. TCP tries to back off on bandwidth when there is traffic loss in the network, but UDP does not. As a result, UDP occupies all the queues and makes TCP starve for bandwidth. Therefore, it is good to limit the number of BGP neighbors and be cautious during removal/addition of new IP subnet ranges.

Summary

This chapter explained various techniques and commands that are used to troubleshoot BGP peering issues, such as completely down peers or flapping peers. This chapter provides multiple techniques to identify and overcome peering issues which results in a higher uptime for the BGP neighbors and make the network more stable. The chapter explains various factors like MTU issues, High CPU conditions, or misconfigured CoPP policies that could lead to unstable BGP sessions and cause them to continuously flap. At the end of the chapter, Dynamic BGP peering feature was introduced, which provides the capability to dynamically establish BGP neighbors without having to configure all of them.

References

RFC 4271, *A Border Gateway Protocol 4 (BGP-4)*, Y. Rekhter, T. Li, S. Hares, IETF, https://tools.ietf.org/html/rfc4271, January 2006.

RFC 879, *The TCP Maximum Segment Size and Related Topics*, J. Postel, IETF, https://tools.ietf.org/html/rfc879, November 1983.

RFC 1191, *Path MTU Discovery*, J. Mogul, S. Deering, IETF, https://tools.ietf.org/html/rfc1191, November 1990.

This page intentionally left blank

Chapter 4

Troubleshooting Route Advertisement and BGP Policies

The following topics are covered in this chapter:

- BGP Route Advertisement Process
- BGP Best Path Selection
- Troubleshooting Missing BGP Routes
- Conditional Matching Techniques
- Troubleshooting BGP Router Policies

Chapter 3, "Troubleshooting Peering Issues," focuses on the establishment of a Border Gateway Protocol (BGP) session. Now that the BGP session has been established, network prefixes and the path attributes can be exchanged between routers. Unlike Interior Gateway Protocols (IGP) routing protocols, BGP allows a routing policy to be different from router to router in an autonomous system (AS).

BGP routers can influence path selection by intentionally blocking longer matching (more specific) routes, or the modification of BGP attributes to influence best path selection. Those behaviors should be intentional, but this chapter focuses on troubleshooting unintentional problems with BGP route advertisement.

Troubleshooting BGP Route Advertisement

BGP route advertisement is often misunderstood and is the first concept that this chapter demonstrates how to troubleshoot.

Local Route Advertisement Issues

Imagine a single router that wants to install networks 10.1.0.0/16 and 10.0.0.0/8 into BGP so that those prefixes could be advertised to other routers. Example 4-1 provides a basic configuration.

Example 4-1 *Basic 10.1.0.0/16 Network Advertisement and 10.0.0.0/8 Aggregation*

```
R1# show run | s router bgp
router bgp 100
 no bgp log-neighbor-changes
 !
 address-family ipv4
  network 10.0.0.0
  network 10.1.0.0 mask 255.255.0.0
 exit-address-family
```

Upon examination of the BGP table in Example 4-2, neither route was installed into the BGP table.

Example 4-2 *Confirmation of Missing BGP Routes*

```
R1# show bgp ipv4 unicast
R1#
```

Reviewing the router's local routing table, note that the router has a directly connected route for the 10.1.1.0/24 network, as shown in Example 4-3. Performing an explicit route lookup in the router's global routing table, routing information base (RIB), for the two configured networks reveals that neither subnet exists in the table.

Example 4-3 *Verification That Routes Are in RIB*

```
R1# show ip route | b Gateway
Gateway of last resort is not set
      10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
C        10.1.1.0/24 is directly connected, Ethernet0/1
L        10.1.1.1/32 is directly connected, Ethernet0/1
      192.168.1.0/32 is subnetted, 1 subnets
C        192.168.1.1 is directly connected, Loopback0
```

```
R1# show ip route 10.1.0.0 255.255.0.0
% Subnet not in table
R1# show ip route 10.0.0.0 255.0.0.0
% Subnet not in table
```

An exact route must exist in the router's RIB (routing table) for the route to be installed into the BGP table so that it can be advertised to BGP neighbors. There are two solutions: modify the BGP configuration to match the local networks that already exist in the RIB or create a static route for the network in the BGP configuration. Example 4-4 demonstrates the new configuration.

Example 4-4 *R1's Correct Configuration for Route Advertisement and Aggregation*

```
R1# show run | section  route
router bgp 100
!
 address-family ipv4
  network 10.0.0.0
  network 10.1.1.0 mask 255.255.255.0
 exit-address-family
ip route 10.0.0.0 255.0.0.0 Null0
```

Note The static route uses the Null 0 interface as a safety mechanism to prevent routing loops. If R1 has a more explicit route (longer match), it can forward the packet to that direction. If it does not have a more explicit route, the packet is dropped.

After making the appropriate changes, Example 4-5 demonstrates that the 10.0.0.0/8 and 10.1.1.0/24 networks have installed into the BGP table.

Example 4-5 *Verification of BGP Routes on R1*

```
R1# show bgp ipv4 unicast
! Output omitted for brevity
     Network          Next Hop            Metric LocPrf Weight Path
 *>  10.0.0.0         0.0.0.0                  0         32768 i
 *>  10.1.1.0/24      0.0.0.0                  0         32768 i
```

Route Aggregation Issues

Another common problem occurs with BGP route aggregation. As shown in Example 4-6, R1 has multiple RIB routes in the 10.1.0.0/16 range that it would like to have aggregated to a single 10.1.0.0/16 address in BGP. Notice that the BGP configuration has been reconfigured, and only the 10.1.0.0/16 network is defined as the aggregate prefix.

Example 4-6 *R1's Global RIB Table*

```
R1# show ip route
! Output omitted for brevity
Gateway of last resort is not set
      10.0.0.0/8 is variably subnetted, 6 subnets, 2 masks
C        10.1.12.0/24 is directly connected, Ethernet0/0
```

```
C        10.1.13.0/24 is directly connected, Ethernet0/1
C        10.1.14.0/24 is directly connected, Ethernet0/2
```

```
R1# show run | s router bgp
router bgp 100
 bgp log-neighbor-changes
 !
 address-family ipv4
  aggregate-address 10.1.0.0 255.255.0.0
 exit-address-family
```

Example 4-7 displays the BGP table on R1. The 10.1.0.0/16 aggregate route is not present because there are not any prefixes within the summary aggregate prefix range in the BGP table.

Example 4-7 *BGP Table on R1*

```
R1# show bgp ipv4 unicast
R1#
```

By adding the smaller network prefixes (10.1.12.0/24, 10.1.13.0/24, and 10.1.14.0/24) into the BGP table, the 10.1.0.0/16 aggregate route can be created. Example 4-8 provides the new configuration and verifies that the BGP table has populated properly with all three smaller prefixes and the aggregate route. Notice that the smaller prefixes (10.1.12.0/24, 10.1.13.0/24, and 10.1.14.0/24) are not suppressed and are advertised with the aggregate route.

Example 4-8 *R1's Configuration for Smaller and Aggregate Prefixes*

```
R1# show run | section router bgp
router bgp 100
 bgp log-neighbor-changes
 !
 address-family ipv4
  network 10.1.12.0 mask 255.255.255.0
  network 10.1.13.0 mask 255.255.255.0
  network 10.1.14.0 mask 255.255.255.0
  aggregate-address 10.1.0.0 255.255.0.0
 exit-address-family
```

```
R1# show bgp ipv4 unicast
BGP table version is 31, local router ID is 192.168.1.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale, m multipath, b backup-path, f RT-Filter,
              x best-external, a additional-path, c RIB-compressed,
```

```
Origin codes: i - IGP, e - EGP, ? - incomplete
RPKI validation codes: V valid, I invalid, N Not found

     Network          Next Hop            Metric LocPrf Weight Path
 *>  10.1.0.0/16      0.0.0.0                             32768 i
 *>  10.1.12.0/24     0.0.0.0                  0          32768 i
 *>  10.1.13.0/24     0.0.0.0                  0          32768 i
 *>  10.1.14.0/24     0.0.0.0                  0          32768 i
```

To keep the smaller prefixes from being advertised, they can be filtered with the router's outbound BGP policy or through the suppression locally by appending the keyword **summary-only** to the **aggregate-address** command.

Example 4-9 demonstrates the new configuration and the BGP table. Notice that the smaller prefixes are suppressed, which is indicated by the *s* in the output.

Example 4-9 *R1's BGP Configuration for Suppression of Smaller Prefixes*

```
R1# show run | section router bgp
router bgp 100
 bgp log-neighbor-changes
 !
 address-family ipv4
  network 10.1.12.0 mask 255.255.255.0
  network 10.1.13.0 mask 255.255.255.0
  network 10.1.14.0 mask 255.255.255.0
  aggregate-address 10.1.0.0 255.255.0.0 summary-only
 exit-address-family
```

```
R1# show bgp ipv4 unicast
! Output omitted for brevity
BGP table version is 34, local router ID is 192.168.1.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale, m multipath, b backup-path, f RT-Filter,
              x best-external, a additional-path, c RIB-compressed,
Origin codes: i - IGP, e - EGP, ? - incomplete
RPKI validation codes: V valid, I invalid, N Not found

     Network          Next Hop            Metric LocPrf Weight Path
 *>  10.1.0.0/16      0.0.0.0                             32768 i
 s>  10.1.12.0/24     0.0.0.0                  0          32768 i
 s>  10.1.13.0/24     0.0.0.0                  0          32768 i
 s>  10.1.14.0/24     0.0.0.0                  0          32768 i
```

Route Redistribution Issues

Redistributing routes into BGP is a common method of populating the BGP table. Figure 4-1 provides a simple drawing where R1 connects to R2, R3, and R4.

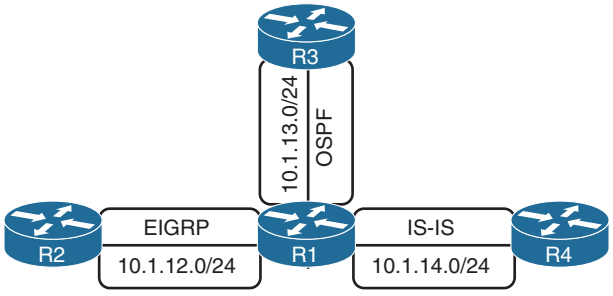

Figure 4-1 *Route Redistribution Topology*

R1 wants to redistribute all the prefixes that it has in the routing table, which includes prefixes learned via Enhanced Interior Gateway Routing Protocol (EIGRP), Open Shortest Path First (OSPF), and Intermediate System to Intermediate System (IS-IS). R1's BGP configuration and RIB has been displayed in Example 4-10. Notice that R1 has routes from all three routing protocols in the 192.168.0.0 range, and all three routing protocols should redistribute into BGP.

Example 4-10 *R1's Routing Table for Route Redistribution*

```
R1# show ip route | exclude subnetted
! Output omitted for brevity
Gateway of last resort is not set

C        10.1.12.0/24 is directly connected, Ethernet0/0
C        10.1.13.0/24 is directly connected, Ethernet0/1
C        10.1.14.0/24 is directly connected, Ethernet0/2
D EX     192.168.2.2 [170/409600] via 10.1.12.2, 00:06:30, Ethernet0/0
O E2     192.168.3.3 [110/20] via 10.1.13.3, 00:02:12, Ethernet0/1
i L2     192.168.4.4 [115/10] via 10.1.14.4, 00:06:30, Ethernet0/2
```

```
R1# show run | section router bgp
router bgp 100
!
 address-family ipv4
  redistribute isis level-2
  redistribute eigrp 1
  redistribute ospf 1
 exit-address-family
```

Example 4-11 displays R1's BGP table after redistribution. The 192.168.3.3/32 prefix learned from OSPF and the 10.1.14.0/24 network from the IS-IS routing table did not get redistributed into BGP. Notice that for the networks that are not directly connected, the BGP next-hop IP address matches the next-hop IP address from the routing protocol, as shown in Example 4-10.

Example 4-11 *R1's BGP Table After Redistribution*

```
R1# show bgp ipv4 unicast | begin Network
     Network          Next Hop            Metric LocPrf Weight Path
 *>  10.1.12.0/24     0.0.0.0                  0         32768 ?
 *>  10.1.13.0/24     0.0.0.0                  0         32768 ?
 *>  192.168.2.2/32   10.1.12.2           409600         32768 ?
 *>  192.168.4.4/32   10.1.14.4               10         32768 ?
```

Some of the OSPF and IS-IS routes were not redistributed into BGP for the following reasons:

- **OSPF:** When redistributing OSPF into BGP, the default behavior includes only routes that are internal to OSPF (O or O IA). The redistribution of external OSPF routes requires a conditional match in the redistribution statement and/or an optional redistribution route-map.
- **IS-IS:** IS-IS does not include directly connected subnets for any destination routing protocol. This behavior is overcome by redistributing the connected networks into BGP.

Note If the **internal** keyword is omitted on a conditional match, the internal OSPF routes are not redistributed.

Example 4-12 provides the proper configuration where the conditional match has been added to the OSPF redistribution statement and the connected networks are being redistributed to overcome the IS-IS limitation.

Example 4-12 *Corrected R1's Configuration to Redistribute All Networks into BGP*

```
R1# show run | section router bgp
router bgp 100
 !
 address-family ipv4
  redistribute connected
  redistribute isis level-2
```

```
 redistribute eigrp 1
 redistribute ospf 1 match internal external 1 external 2
exit-address-familyR1#
```

```
R1# show bgp ipv4 unicast | begin Network
     Network          Next Hop            Metric LocPrf Weight Path
 *>  10.1.12.0/24     0.0.0.0                  0         32768 ?
 *>  10.1.13.0/24     0.0.0.0                  0         32768 ?
 *>  10.1.14.0/24     0.0.0.0                  0         32768 ?
 *>  192.168.2.2/32   10.1.12.2           409600         32768 ?
 *>  192.168.3.3/32   10.1.13.3               20         32768 ?
 *>  192.168.4.4/32   10.1.14.4               10         32768 ?
```

Note Although not directly related to the advertisement of routes into BGP, issues can arise when redistributing routes from BGP to an IGP protocol. By default, BGP does not redistribute internal routes (routes learned via an IBGP peer) into an IGP protocol (that is, OSPF) as a safety mechanism. The command **bgp redistribute-internal** allows IBGP routes to be redistributed into an IGP routing protocol.

BGP Tables

BGP uses three tables for maintaining a network prefix and the path attributes (PA) associated with each path for a prefix. The BGP tables are as follows:

- **Adj-RIB-in:** Contains the Network Layer Reachability Information (NLRI) (network prefix and prefix-length) and BGP PAs in the original form that they are received from a BGP peer. The Adj-RIB-in table is purged after all route policies are processed to save memory.
- **Loc-RIB:** Contains all the NLRIs that originate locally or that came from the Adj-RIB-in table after passing any inbound route policy processing. Note that BGP PAs in the Loc-RIB might have been modified by the router's inbound route policies for a specific neighbor.

  After NLRIs pass the validity and next-hop reachability check, the BGP best path algorithm selects the best NLRI for a specific prefix. The Loc-RIB table is the table used for presenting routes to the IP routing table and is the BGP table commonly referred to as "The BGP table."
- **Adj-RIB-out:** Contains the NLRIs after outbound route policies have processed. Different outbound route policies could exist between neighbors, so the Adj-RIB-out keeps track of the neighbors and the NLRIs for each neighbor. Outbound route policies could modify PAs for a specific neighbor.

The BGP **network** statements do not enable BGP for a specific interface; instead, they identify a specific network prefix to be installed into the Loc-RIB table. After configuring a BGP network statement, the BGP process searches the global RIB for an exact network prefix match. The network prefix can be a connected network, secondary connected network, or any route from a routing protocol. After verifying that the network statement matches a prefix in the global RIB, the prefix installs into the BGP Loc-RIB table.

Not every route in the Loc-RIB is advertised to a BGP peer. All routes in the Loc-RIB use the following steps for advertisement to BGP peers:

Step 1. **Verify next-hop reachability.** Confirm that the next-hop address is resolvable in the global RIB. If the next-hop address is not resolvable in the RIB, the NLRI remains but does not process after Step 2. The next-hop address must be resolvable for the BGP best path process to occur in Step 3.

Step 2. **Set BGP path attributes.** The following BGP PAs are set dependent upon the location of the route in the local RIB:

- **Connected Network:** The next-hop BGP attribute is set to 0.0.0.0; the origin attribute is set to *i* (IGP); and the BGP weight is set to 32,768.
- **Static Route or Routing Protocol:** The next-hop BGP attribute is set to the next-hop IP address in the RIB; the origin attribute is set to *i* (IGP); the BGP weight is set to 32,768; and the Multiple Exit Discriminator (MED) is set to the IGP metric.
- **Redistribution:** The next-hop BGP attribute is set to the next-hop IP address in the RIB; the origin attribute is set to *?* (incomplete); the BGP weight is set to 32,768; and the MED is set to the IGP metric.

Step 3. **Identify the BGP best path.** In BGP, route advertisements consist of the NLRI and the path attributes (PAs). A BGP route may contain multiple paths to the same destination network. Every path's attributes impact the desirability of the route when a router selects the best path. A BGP router only advertises the best path to the neighboring routers.

Inside the BGP Loc-RIB table, all the routes and their path attributes are maintained with the best path calculated. The best path is then installed in the RIB of the router. In the event the best path is no longer available, the router can use the existing paths to quickly identify a new best path. BGP recalculates the best path for a prefix upon four possible events:

- BGP next-hop reachability change
- Failure of an interface connected to an External Border Gateway Protocol (EBGP) peer
- Redistribution change
- Reception of new paths for a route

The BGP best path selection algorithm influences how traffic enters or leaves an AS. Changing of BGP PA can influence traffic flow into, out of, and around an AS.

Step 4. **Process outbound neighbor route policies.** The NLRI is processed through any specific outbound neighbor route policies. After processing, if the route was not denied by the outbound policies, the route is stored in the Adj-RIB-Out table for later reference.

Step 5. **Advertise the NLRI to BGP peers.** The router advertises the NLRI to BGP peers. If the NLRI's next-hop BGP PA is 0.0.0.0, then the next-hop address is changed to the IP address of the BGP session.

Figure 4-2 illustrates the concept of installing the network prefix from localized BGP network advertisements into the BGP table.

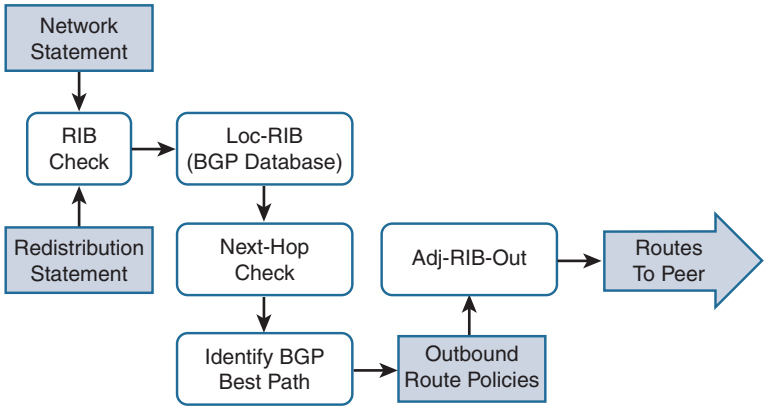

Figure 4-2 *Localized BGP Network Advertisement*

Receiving and Viewing Routes

The complete BGP route advertisement process must take into account routes that are advertised locally and routes that are received from other BGP neighbors. Not every prefix in the Loc-RIB is advertised to a BGP peer or installed into the Global RIB when received from a BGP peer. BGP performs the following route-processing steps:

Step 1. **Perform a quick validity check.** This is performed on the route to ensure that a routing loop is not occurring. If the router recognizes its autonomous system number (ASN) in the AS-Path or its router-ID (RID) in the Internal Border Gateway Protocol (IBGP) path attributes (Originator/Cluster-ID). If no duplicates are found, the NLRI passes the validity check and moves on to the next stage.

Step 2. **Store the route in Adj-RIB-In and process inbound route policies.** The NLRI is stored in the Adj-RIB-In table in its original state. The inbound route policy is applied based on the neighbor the route was received.

Step 3. **Update the Loc-RIB.** The BGP Loc-RIB database is updated with the NLRI after inbound route-policy processing has occurred. The Adj-RIB-in is cleared to save memory.

Step 4. **Verify next-hop reachability.** Confirm that the next-hop address is resolvable in the global RIB. If the next-hop address is not resolvable in the RIB, the NLRI remains but does not process further.

Step 5. **Compute the BGP best path.** Multiple NLRIs (paths) can exist for the same network prefix in the Loc-RIB table. BGP only advertises the best path to its neighbors. The router must identify the BGP best path and pass only the best path and its path attributes to Step 6. The BGP Best Path selection process was explained in Chapter 1, "BGP Fundamentals."

Step 6. **Install the BGP best path into the global RIB and advertise to peers.** Install the prefix into the Global RIB using the next-hop IP address from the BGP Loc-RIB table.

In some occurrences, the route cannot be installed into the Global RIB (a static route that has a lower administrative distance than BGP), which results in a RIB failure. A RIB failure does not prevent the advertisement of the NLRI to other BGP neighbors. The exact reason for a RIB failure is seen with the command **show ip bgp rib-failure**.

Step 7. **Process outbound neighbor route policies.** The NLRI is processed through any specific outbound neighbor route policies. After processing, if the route was not denied by the outbound policies, the route is stored in the Adj-RIB-Out table for later reference.

Step 8. **Advertise the NLRI to BGP peers.** Advertise the NLRI to BGP peers. If the NLRI's next-hop BGP PA is 0.0.0.0, then the next-hop address is changed to the IP address of the BGP session.

Figure 4-3 shows the complete BGP route-processing logic. It now includes the receipt of a route from a BGP peer and the BGP best path algorithm.

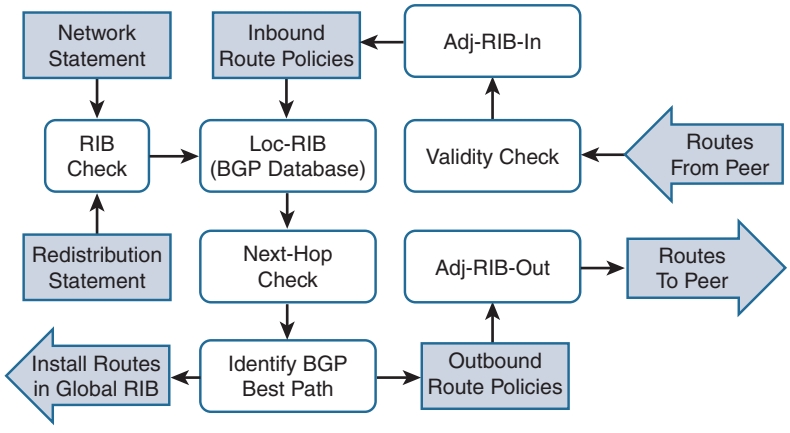

Figure 4-3 *Complete BGP Network Advertisement*

Troubleshooting Missing BGP Routes

The complete process for receiving routes from BGP peers and the advertisement to other peers has been fully explained. This knowledge provides a systematic process for troubleshooting the advertisement of routes between peers. Reasons that route advertisement fails between BGP peers are as follows:

- Next-Hop Check Failure
- Bad Network Design
- Validity Check Failure
- BGP Communities
- Mandatory EBGP Route Policy for IOS XR
- Route filtering

Most of these issues can be found by using the following:

- **BGP Loc-RIB:** Just because a route is missing from the Global RIB, it does not mean the route did not make it into the router's BGP table. Examine the BGP Loc-RIB to see if the prefix exists in the BGP table. It is possible that the route installed in the BGP table but did not install into the RIB. Viewing the local BGP table is the first step in troubleshooting any missing route.

  The BGP Loc-RIB is viewed with the command **show bgp** *afi safi* [*prefix/prefix-length*]. Examining a specific prefix provides the reason a route was not installed into the RIB.

- **BGP Adj-RIB-in:** The BGP Loc-RIB table contains only valid routes that passed the router's inbound route policies. Examining the BGP Adj-RIB-in table verifies whether the peer received the NLRI. If the peer received it, the local inbound route policy prevents the route from installing into the Loc-RIB table. Inbound Soft Configuration is required to view the BGP Adj-RIB-in table, because the table is purged by default after all inbound route-policy processing has occurred.

- **BGP Adj-RIB-out:** Viewing the BGP Adj-RIB-out table on the advertising router verifies that the route was advertised and provides a list of the BGP PAs that were included with the route. In the event that the route is not present in the advertising router's BGP Adj-RIB-out table, check the advertising router's BGP Loc-RIB table to verify the prefix exists there. Assuming the prefix is in the Loc-RIB table, but not in the Adj-RIB-out table, then the outbound route policies are preventing the advertisement of the route. Contents of the BGP Adj-RIB-out are viewed with the command **show bgp** *afi safi* **neighbor** *ip-address* [*prefix/prefix-length*] **advertised-routes**.

- **Viewing BGP Neighbor Sessions:** The information contained in the BGP neighbor session varies from platform to platform, but still provides a lot of useful information, such as the number of prefixes advertised, session and address-family options,

the route maps/route filters/route policy applied specifically for that neighbor. The BGP neighbor session is displayed with the command **show bgp** *afi safi* **neighbor** *ip-address*.

- **NX-OS Event History:** NX-OS contains a form of logging that runs in the background and is not as intensive as running a debug. Event history provides visibility as to what happened and when it happened. Knowing when certain events happen can correlate to other events that occur in the network at the same time.

  Increase the size of the event history with the command **event-history detail size large** under the BGP process on NX-OS devices. Event history is displayed with the command **show bgp event-history detail**.

- **Debug Commands:** Debug commands provide the most amount of information about BGP. On IOS nodes, BGP update debugs are enabled with the command **debug bgp** *afi safi* **updates** [**in** | **out**] [**detail**]. On IOS XR nodes, BGP update debugs are enabled with the command **debug bgp update**[**afi** *afi safi*] [**in** | **out**]. On NX-OS nodes, BGP update debugs are enabled with the command **debug bgp updates** [**in** | **out**].

Figure 4-4 provides a sample topology that connects five routers in a point-to-point line. R1 is in AS100, R5 is in AS300, and R2, R3, and R4 are in AS200. This topology is used to demonstrate how to troubleshoot the various reasons a route could be missing from the routing table. All BGP sessions are established by using the directly connected interfaces (that is, peer-link IP address) and do not contain multi-hop sessions. The following routes are being advertised:

- R1 is advertising the 10.0.0.0/8 aggregate prefix.
- R1 is advertising the 10.1.1.0/24 prefix.
- R2 is advertising the 10.2.2.0/24 prefix.

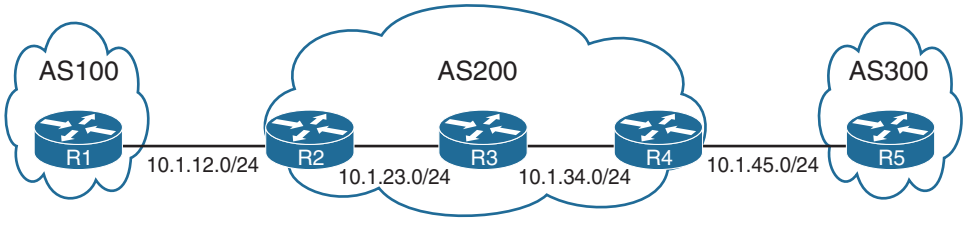

Figure 4-4 *BGP Troubleshooting Sample Topology*

Next-Hop Check Failures

R3 is missing the 10.0.0.0/8 network and the 10.1.1.0/24 network from the RIB, as shown in Example 4-13. Only the 10.2.2.0/24 network that is advertised by R2 is present.

Example 4-13 *Examination of R3's Global RIB*

```
R3# show ip route
! Output omitted for brevity
Gateway of last resort is not set
      10.0.0.0/8 is variably subnetted, 5 subnets, 2 masks
C        10.1.23.0/24 is directly connected, Ethernet0/2
C        10.1.34.0/24 is directly connected, Ethernet0/1
B        10.2.2.0/24 [200/0] via 10.1.23.2, 00:07:08
      192.168.3.0/32 is subnetted, 1 subnets
C        192.168.3.3 is directly connected, Loopback0
```

Both of the missing routes are advertised from R1. The first step is to check the R3's Loc-RIB BGP table, as shown in Example 4-14. The 10.0.0.0/8 network and the 10.1.1.0/24 network are present, but notice that both entries are missing the best path marker >.

Example 4-14 *Examination of R3's BGP Loc-RIB Table*

```
R3# show bgp ipv4 unicast
! Output omitted for brevity
BGP table version is 2, local router ID is 192.168.3.3
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale, m multipath, b backup-path, f RT-Filter,
              x best-external, a additional-path, c RIB-compressed,
Origin codes: i - IGP, e - EGP, ? - incomplete
RPKI validation codes: V valid, I invalid, N Not found

     Network          Next Hop            Metric LocPrf Weight Path
 * i 10.0.0.0         10.1.12.1                0    100      0 100 i
 * i 10.1.1.0/24      10.1.12.1                0    100      0 100 i
 *>i 10.2.2.0/24      10.1.23.2                0    100      0 ?
```

Displaying an explicit network prefix with the command **show bgp** *afi safi prefix/prefix-length*, as shown in Example 4-15, provides some clarity for why the NLRI was not selected as a best path.

Example 4-15 *Examination of NLRI Without Best Path*

```
R3# show bgp ipv4 unicast 10.1.1.0/24
BGP routing table entry for 10.1.1.0/24, version 0
Paths: (1 available, no best path)
  Not advertised to any peer
  Refresh Epoch 1
  100, (received & used)
    10.1.12.1 (inaccessible) from 10.1.23.2 (192.168.2.2)
      Origin IGP, metric 0, localpref 100, valid, internal
      rx pathid: 0, tx pathid: 0
```

In the output, the next-hop 10.1.12.1 is inaccessible. Let's verify that the next-hop exists on the router with the command **show ip route** *next-hop-IP-address*, as shown in Example 4-16.

Example 4-16 *Examination of RIB for Next-Hop IP Address*

```
R3# show ip route 10.1.12.1
% Subnet not in table
```

The next-hop IP address is not available in the RIB. There are multiple solutions to this issue that include the following:

- R2 advertises the peering link (10.1.12.0/24) into BGP. R3 is adjacent to R2 and receives the route with a next-hop of 10.1.23.2, which is in R3's RIB as a directly connected route. The 10.1.12.1 next-hop IP address would then be resolvable through a recursive lookup.
- Establish an IGP routing protocol within AS200 (R2, R3, and R4) and advertise the peering link (R1–R2) in OSPF, but make the peering link interface passive in OSPF.
- On R2 configure the **next-hop-self** feature in the address-family for the BGP peering with R3. All EBGP routes (that is, routes learned from R1) would then use R2 as their next-hop for any routes learned from R2.

Note Assuming that R3 is a route-reflector, the same scenario impacts R4 as well. As R4 learns the route from R3, the same fixes must be implemented on R3 because R4 will not know how to reach the 10.1.23.0/24 network. Most places use an IGP for advertising the internal networks among themselves. Then they choose to use either the **next-hop-self** feature or advertise the peering link's network (passive interface) into the IGP.

OSPF was enabled on R2, R3, and R4, and the **next-hop-self** feature was enabled on R2 toward R3 to address routes advertised from AS100 and on R4 toward R3 in case AS300 advertised any routes. Example 4-17 verifies that the BGP best path computation has completed and that 10.0.0.0/8 and 10.1.1.0/24 networks have been inserted into the RIB by BGP.

Example 4-17 *R3's BGP Table and Global RIB*

```
R3# show bgp ipv4 unicast
! Output omitted for brevity
     Network          Next Hop            Metric LocPrf Weight Path
 *>i 10.0.0.0         10.1.23.2                0    100      0 100 i
 *>i 10.1.1.0/24      10.1.23.2                0    100      0 100 i
 *>i 10.2.2.0/24      10.1.23.2                0    100      0 ?
```

```
R3# show ip route
! Output omitted for brevity
```

```
Gateway of last resort is not set
      10.0.0.0/8 is variably subnetted, 7 subnets, 3 masks
B        10.0.0.0/8 [200/0] via 10.1.23.2, 00:02:33
B        10.1.1.0/24 [200/0] via 10.1.23.2, 00:02:33
C        10.1.23.0/24 is directly connected, Ethernet0/2
C        10.1.34.0/24 is directly connected, Ethernet0/1
B        10.2.2.0/24 [200/0] via 10.1.23.2, 00:15:17
      192.168.3.0/32 is subnetted, 1 subnets
C        192.168.3.3 is directly connected, Loopback0
```

Bad Network Design

Networks that use BGP are more sensitive to design flaws than networks that use only IGP routing protocols. An improperly design BGP network can result in an inconsistent routing policy, missing routes, or worse. In Figure 4-4, R4 is missing all the routes 10.0.0.0/8, 10.1.1.0/24, and 10.2.2.0/24. OSPF has been deployed between R2, R3, and R4. R2 and R4 are now using next-hop-self for their BGP session toward R3.

The first step of troubleshooting is to check the Loc-RIB table, as shown in Example 4-18. This verifies that the NLRIs were never installed into the Loc-RIB table on R4. The next step is to see if any routes were received and exist in R4's Adj-RIB-in BGP table. (This assumes that soft-configuration inbound was configured for R4's session with R3.)

Example 4-18 *Display of R4's Adj-RIB-in Table*

```
R4# show bgp ipv4 unicast
R4#
```

```
R4# show bgp ipv4 unicast neighbors 10.1.34.3 received-routes

Total number of prefixes 0
R4#
```

After examining R4's BGP neighbor session details with R3, as shown in Example 4-19, no prefixes were received or denied from R3. Notice that the BGP session is an IBGP session.

Example 4-19 *Display of R3's BGP Neighborship with R4*

```
R4# show bgp ipv4 unicast neighbors 10.1.34.3
! Output omitted for brevity
BGP neighbor is 10.1.34.3,  remote AS 200, internal link
  BGP version 4, remote router ID 192.168.3.3
  BGP state = Established, up for 4d07h
```

```
For address family: IPv4 Unicast
  Session: 10.1.34.3
  BGP table version 1, neighbor version 1/0
  Inbound soft reconfiguration allowed
  NEXT_HOP is always this router for EBGP paths
  Slow-peer split-update-group dynamic is disabled
                                Sent       Rcvd
  Prefix activity:              ----       ----
    Prefixes Current:              0          0
    Prefixes Total:                0          0
    Implicit Withdraw:             0          0
    Explicit Withdraw:             0          0
    Used as bestpath:            n/a          0
    Used as multipath:           n/a          0

                                  Outbound    Inbound
  Local Policy Denied Prefixes:   --------    -------
    Total:                               0          0
```

The next step is to see whether the routes are present on R3's Loc-RIB table and that a best path has been selected. In Example 4-20, all three prefixes have a best path and were learned internally from the IBGP peer R2.

Example 4-20 *Display of R3's BGP Table*

```
R3# show bgp ipv4 unicast
BGP table version is 12, local router ID is 192.168.3.3
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale, m multipath, b backup-path, f RT-Filter,
              x best-external, a additional-path, c RIB-compressed,
Origin codes: i - IGP, e - EGP, ? - incomplete
RPKI validation codes: V valid, I invalid, N Not found
     Network          Next Hop            Metric LocPrf Weight Path
 *>i 10.0.0.0         10.1.23.2                0    100      0 100 i
 *>i 10.1.1.0/24      10.1.23.2                0    100      0 100 i
 *>i 10.2.2.0/24      10.1.23.2                0    100      0 ?
```

After reviewing the topology, R2, R3, and R4 are all in the same BGP autonomous system. An IBGP learned route is not advertised to another IBGP peer as part of a loop prevention mechanism. Full route advertisement for IBGP learned routes occurs if a full mesh of IBGP sessions between R2, R3, and R4 is formed, or if R3 becomes a route-reflector to R2 and R4.

After configuring R3 as a route-reflector to R2 and R4, the routes are seen on R4, as shown in Example 4-21.

Example 4-21 *Display of R4's BGP Table After R3 Becomes a Route-Reflector*

```
R4# show bgp ipv4 unicast
! Output omitted for brevity
     Network          Next Hop            Metric LocPrf Weight Path
 *>i 10.0.0.0         10.1.23.2                0    100      0 100 i
 *>i 10.1.1.0/24      10.1.23.2                0    100      0 100 i
 *>i 10.2.2.0/24      10.1.23.2                0    100      0 ?
```

Validity Check Failure

BGP performs a validity check upon receipt of prefixes. Specifically, BGP is looking for indicators of a loop, such as

- Identifying the router's ASN in the AS-Path
- Identifying the router's RID in as the Route-Originator ID
- Identifying the router's RID as the Cluster ID

AS-Path

The AS-Path (BGP attribute AS_PATH) is used as a loop prevention mechanism. The AS-Path is not prepended as a NLRI is advertised to other IBGP peers. Some common scenarios for a router to identify its ASN in an NLRI's AS-Path are as follows:

- **AS-Prepending:** Industry standards dictate that the AS being prepended should be owned by your organization. However, some organizations may prepend a route with an ASN that they do not own. This is done for malicious purposes or unintentionally.
- **Route Aggregation:** Default behavior for route aggregation is to not include any BGP attributes of the smaller routes that are being aggregated, which adds the atomic aggregate BGP attribute. The loss of path visibility could result in route feedback when an organization advertises an aggregate route that includes a smaller network that is advertised from your network. If the **as-set** keyword is used with the aggregation command, all the BGP attributes of the routes being summarized are included. This includes the AS-Path.

After configuring the **as-set** keyword on R1, R1 includes the PAs from the smaller aggregate routes. For example, the 10.2.2.0/24 network that is being learned on R1 from AS200 would be aggregated into the 10.0.0.0/8 aggregate with the AS200 as part of the AS-Path.

Detecting a router's ASN in a route that is received from a peer can be accomplished by the following:

- Viewing the BGP session on IOS routers.
- Viewing the network routes that are advertised to the router.
- Enabling debugging for BGP updates, which will indicate the AS-Path loop.

In Example 4-22, R2 displays the BGP neighbor session details for R1. After examining the IPv4 address-family, routes were denied for an AS_PATH loop. It is important to note that the count of routes is a cumulative count of route advertisements throughout the life of that BGP session.

Example 4-22 *Verification of Denied Routes from AS-Path Check*

```
R2# show bgp ipv4 unicast neighbors 10.1.12.1
! Output omitted for brevity
BGP neighbor is 10.1.12.1,  remote AS 100, external link
  BGP version 4, remote router ID 192.168.1.1
For address family: IPv4 Unicast
!
                                   Outbound    Inbound
  Local Policy Denied Prefixes:    --------    -------
    AS_PATH loop:                       n/a          1
    Bestpath from this peer:              3        n/a
    Total:                                3          1
```

NX-OS and IOS XR devices require additional techniques (explained later in the chapter) to identify the loss of routes. The second method is to list the routes on R1 that were advertised to R2 that include the ASN of R2 (200). As shown in Example 4-23, the 10.0.0.0/8 route includes the AS-Path of 200.

Example 4-23 *Verification of Route Advertisement of Route with AS-Path Loop*

```
R1# show bgp ipv4 unicast neighbors 10.1.12.2 advertised-routes | i Network|200
     Network          Next Hop            Metric LocPrf Weight Path
 *>  10.0.0.0         0.0.0.0                  100  32768 200 ?
```

The third method is to enable BGP debugging on R2 and initiate an inbound BGP soft-refresh as shown in Example 4-24.

Example 4-24 *Identification of AS-Path Loops with BGP Debugs*

```
R2# debug bgp ipv4 unicast updates in detail
R2# clear bgp ipv4 unicast 10.1.12.1
! Output omitted for brevity
4d07h: BGP(0): 10.1.12.1 rcv UPDATE w/ attr: nexthop 10.1.12.1, origin ?, metric 0,
  aggregated by 100 192.168.1.1, originator 0.0.0.0, merged path 100 200, AS_PATH ,
  community , extended community , SSA attribute
4d07h: BGPSSA ssacount is 0
4d07h: BGP(0): 10.1.12.1 rcv UPDATE about 10.0.0.0/8 -- DENIED due to: AS-PATH
  contains our own AS;
4d07h: BGP(0): no valid path for 10.0.0.0/8
4d07h: BGP: topo global:IPv4 Unicast:base Remove_fwdroute for 10.0.0.0/8
```

```
4d07h: BGP(0): (base) 10.1.23.3 send unreachable (format) 10.0.0.0/8
4d07h: BGP(0): 10.1.23.3 rcv UPDATE about 10.0.0.0/8 – withdrawn
```

```
RP/0/0/CPU0:XR2# debug bgp update in
RP/0/0/CPU0:XR2# clear bgp ipv4 unicast 10.1.12.1
! Output omitted for brevity
RP/0/0/CPU0: 04:55:57: bgp[1053]: (ip4u): UPDATE from 10.1.12.1, prefix 10.0.0.0/8
  (path ID: none) DENIED due to:
RP/0/0/CPU0: 04:55:57: bgp[1053]: (ip4u):  as-path contains our own AS, or 0;
RP/0/0/CPU0: 04:55:57: bgp[1053]: : Received UPDATE from 10.1.12.1 (length incl.
  header = 60)
RP/0/0/CPU0: 04:55:57: bgp[1053]: Receive message dump for 10.1.12.1:
```

```
NXOS2# debug bgp updates in
NXOS2# clear bgp ipv4 unicast 10.1.12.1
! Output omitted for brevity
04:47:06 bgp: 200 [8463]) UPD: Received UPDATE message from 10.1.12.3
04:47:06 bgp: 200 [8463] UPD: 10.1.12.3 parsed UPDATE message from peer, len 66 ,
  withdraw len 0, attr len 43, nlri len 0
04:47:06 bgp: 200 [8463] UPD: Attr code 1, length 1, Origin: IGP
04:47:06 bgp: 200 [8463] UPD: Attr code 7, length 8, Aggregator: (100,10.1.1.1)
04:47:06 bgp: 200 [8463] UPD: Peer 10.1.12.3 nexthop length in MP reach: 4
04:47:06 bgp: 200 [8463] UPD: Recvd NEXTHOP 10.1.12.3
04:47:06 bgp: 200 [8463] UPD: Attr code 14, length 11, Mp-reach
04:47:06 bgp: 200 [8463] UPD: 10.1.12.3 Received attr code 2, length 10, AS-Path:
  <100 200 >
04:47:06 bgp: 200 [8463] UPD: Received AS-Path attr with own ASN from 10.1.12.3
04:47:06 bgp: 200 [8463] UPD: [IPv4 Unicast] Received prefix 10.0.0.0/8 from peer
  10.1.12.3, origin 0, next hop 10.1.12.3, localpref 0, med 0
04:47:06 bgp: 200 [8463] UPD: [IPv4 Unicast] Dropping prefix 10.0.0.0/8 from peer
  10.1.12.3, due to attribute error
```

The 10.0.0.0/8 network prefix is accepted by asking the administrator of R1 to remove the **as-set** keyword on their route aggregation, or to selectively add BGP PAs from the network prefixes that are advertised only from their AS.

Another alternative is to add the **allow-as** functionality for that neighbor. This feature disables the ASN check on a neighbor-by-neighbor basis. Enabling the command may resolve the issues for R2, but the problem will still exist for other routers in that BGP AS (that is, R3 and R4).

Note IOS XR and NX-OS devices perform an ASN loop check as part of the advertisement process to avoid sending updates to peers that would fail the AS-Path loop check. This is done to reduce bandwidth and CPU processing on the downstream router.

This feature can be disabled globally on IOS XR routers with the BGP address-family configuration command **as-path-loopcheck out disable** and can be disabled per neighbor on NX-OS routers with the command **disable-peer-as-check**.

Originator-ID/Cluster-ID

Another potential reason a NLRI fails the validity check is if the Originator-ID or Cluster-ID matches the receiving router's RID. The Originator-ID is populated by a route-reflector (RR) with the advertising router's RID, and the Cluster-ID is populated by the RR. The default Cluster-ID setting is the RR's RID, unless it is specifically set, which is done for certain design scenarios. Checking the Originator-ID or Cluster-ID is considered a loop prevention mechanism.

Assume that in the sample topology, that R4's BGP RID was configured to 192.168.2.2, which unknowingly matches R2's BGP RID. The steps for troubleshooting are similar as before. On IOS routers, viewing the BGP neighbor session displays that some routes were dropped because of an Originator loop, as shown in Example 4-25.

Example 4-25 *Detection of Originator Loop with IOS BGP Neighbor Session*

```
R4# show bgp ipv4 unicast neighbors 10.1.34.3
! Output omitted for brevity
BGP neighbor is 10.1.34.3,  remote AS 200, internal link
  BGP version 4, remote router ID 192.168.3.3
  BGP state = Established, up for 00:01:24
  For address family: IPv4 Unicast
!
                                  Outbound    Inbound
  Local Policy Denied Prefixes:   --------    -------
    ORIGINATOR loop:                   n/a          3
    Total:                               0          3
  Number of NLRIs in the update sent: max
```

Just as before, IOS XR and NX-OS devices require BGP debugs to discover the reason a BGP update is denied. Example 4-26 displays the BGP updates for all three platforms with the appropriate message that indicates the Originator ID conflict.

Example 4-26 *Detection of Originator Loop via BGP Update Debugs*

```
R4# debug ip bgp updates in
BGP updates debugging is on (inbound) for address family: IPv4 Unicast
R4# clear bgp ipv4 unicast 10.1.34.3
! Output omitted for brevity
1w0d: BGP(0): 10.1.34.3 rcv UPDATE about 10.1.1.0/24 -- DENIED due to: ORIGINATOR
  is us;
1w0d: BGP: 10.1.34.3 Local router is the Originator; Discard update
1w0d: BGP(0): 10.1.34.3 rcv UPDATE w/ attr: nexthop 10.1.23.2, origin ?, localpref
  100, metric 0, originator 192.168.2.2, clusterlist 192.168.3.3, merged path ,
  AS_PATH , community , extended community , SSA attribute
1w0d: BGPSSA ssacount is 0
```

```
RP/0/0/CPU0:XR4# debug bgp update in
RP/0/0/CPU0:XR4# clear bgp 10.1.34.3
! Output omitted for brevity
RP/0/0/CPU0: 03:05:48: bgp[1053]: : UPDATE from 10.1.34.3 contains nh 10.1.23.2/32,
  gw_afi 0, flags 0x0, nlri_afi 0
RP/0/0/CPU0: 03:05:48: bgp[1053]: : NH-Validate-Create: addr=10.1.23.2/32, len=4,
  nlriafi=0, nbr=10.1.34.3, gwafi=0, gwlen=4, gwaddrlen=32::: nhout=0x10beae60,
  validity=1, attrwdrflags=0x00000000
RP/0/0/CPU0: 03:05:48: bgp[1053]: : UPDATE from 10.1.34.3 discarded: local router is
  the ORIGINATOR (192.168.2.2)
RP/0/0/CPU0: 03:05:48: bgp[1053]: : --bgp4_rcv_attributes--: END: nbr=10.1.34.3::
  msg=0x10037ddc/75, updlen=56, attrbl=0x10037df3/48, ipv4reachlen=4,
  msginpath=0x3df0bd0, asloopcheck=1, attrwdrfl=0x00002000:: samecluster=0, local_
  as_prepended=0, attr_wdr_flags 0x00002000, myascount=0:: rcvdata=0x10037e23/0,
  errptr=0x10037e1c/7
RP/0/0/CPU0: 03:05:48: bgp[1053]: (ip4u): Received unreachables from 10.1.34.3:
  attrcode=0, attrwdrflags=0x00002000
RP/0/0/CPU0: 03:05:48: bgp[1053]: (ip4u): UPDATE from 10.1.34.3 with attributes:
RP/0/0/CPU0: 03:05:48: bgp[1053]: (ip4u): nexthop 10.1.23.2/32, origin i, localpref
  100, metric 0, originator 192.168.2.2, clusterlist 3.3.168.192, path 100
RP/0/0/CPU0: 03:05:48: bgp[1053]: (ip4u): UPDATE from 10.1.34.3, prefix 10.1.1.0/24
   (path ID: none) DENIED due to:
RP/0/0/CPU0: 03:05:48: bgp[1053]: (ip4u):  originator is us;
```

```
NXOS4# debug bgp updates in
NXOS4# clear bgp ipv4 unicast 10.1.34.3
! Output omitted for brevity
04:55:38 bgp: 200 [8431]  UPD: Received UPDATE message from 10.1.34.3
04:55:38 bgp: 200 [8431]  UPD: 10.1.34.3 parsed UPDATE message from peer, len 68 ,
  withdraw len 0, attr len 45, nlri len 0
04:55:38 bgp: 200 [8431]  UPD: Attr code 1, length 1, Origin: IGP
04:55:38 bgp: 200 [8431]  UPD: Attr code 5, length 4, Local-pref: 100
04:55:38 bgp: 200 [8431]  UPD: Originator is us, dropping update from 10.1.34.3
04:55:38 bgp: 200 [8431]  UPD: Attr code 9, length 4, Originator: 192.168.2.2
04:55:38 bgp: 200 [8431]  UPD: Attr code 10, length 4, Cluster-list
04:55:38 bgp: 200 [8431]  UPD: Peer 10.1.34.3 nexthop length in MP reach: 4
04:55:38 bgp: 200 [8431]  UPD: Recvd NEXTHOP 10.1.23.2
04:55:38 bgp: 200 [8431]  UPD: Attr code 14, length 13, Mp-reach
04:55:38 bgp: 200 [8431]  UPD: [IPv4 Unicast] Received prefix 10.2.2.0/24 from peer
  10.1.34.3, origin 0, next hop 10.1.23.2, localpref 100, med 0
04:55:38 bgp: 200 [8431]  UPD: [IPv4 Unicast] Dropping prefix 10.2.2.0/24 from peer
  10.1.34.3, due to attribute error
```

Now that the issue has been isolated to the Originator-ID, it is time to identify the root cause. Example 4-27 demonstrates R3's BGP session with R2 and R4. Notice that R2 and R4 have the same RID of 192.168.2.2. By changing the RID on R4 to something unique, R4 can then process the routes.

Example 4-27 *Detection of Duplicate RID on R3*

```
R3# show bgp ipv4 unicast neighbors | i neighbor|ID
BGP neighbor is 10.1.23.2,  remote AS 200, internal link
  BGP version 4, remote router ID 192.168.2.2
  Do log neighbor state changes (via global configuration)
  BGP table version 35, neighbor version 35/0
BGP neighbor is 10.1.34.4,  remote AS 200, internal link
  BGP version 4, remote router ID 192.168.2.2
  Do log neighbor state changes (via global configuration)
  BGP table version 35, neighbor version 35/0
```

BGP Communities

BGP communities provide additional capability for tagging routes and are considered either *well-known* or *private* BGP communities. Private BGP communities are used for conditional matching for a router's route policy, which could influence routes during inbound or outbound route-policy processing. There are three well-known communities that affect only outbound route advertisement: No-Advertise, No-Export, and Local-As. Routes that contain these communities can be displayed with the command **show bgp** *afi safi* [**community** {**local-AS** | **no-advertise** | **no-export**}] on IOS, IOS XR, and NX-OS devices.

This section focuses on explaining the behavior of each of these communities and how to identify them. Conditionally matching private communities for filtering is demonstrated later in Example 4-66.

BGP Communities: No-Advertise

The No_Advertise community (0xFFFFFF02 or 4,294,967,042) specifies that routes with this community should not be advertised to any BGP peer. The No-Advertise BGP community can be advertised from an upstream BGP peer or locally with an inbound BGP policy. In either method, the No-Advertise community is set in the BGP Loc-RIB table that affects outbound route advertisement.

Figure 4-5 demonstrates that R1 is advertising the 10.1.1.0/24 network to R2, and R3 does not have the 10.1.1.0/24 network in its BGP table.

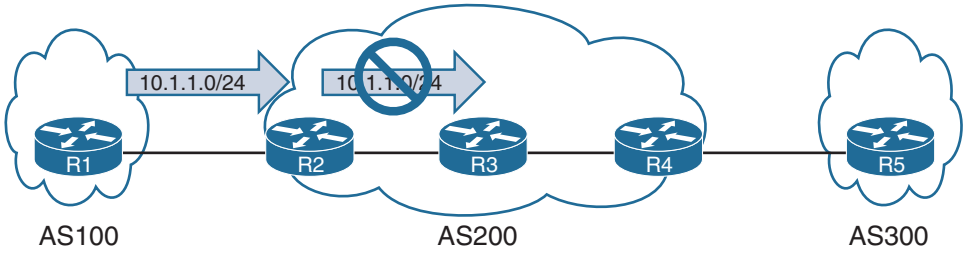

Figure 4-5 *BGP No-Advertise Community Topology*

R3 cannot see the route in its Adj-RIB-in table because the route was never advertised to it from R2. Example 4-28 displays the NLRI information for the 10.1.1.0/24 network prefix. Notice that the NLRI was "not advertised to any peer" and has the BGP community No-Advertise set.

Example 4-28 *BGP Attributes for No-Advertise Routes*

```
R2# show bgp 10.1.1.0/24
! Output omitted for brevity
BGP routing table entry for 10.1.1.0/24, version 18
Paths: (1 available, best #1, table default, not advertised to any peer)
  Not advertised to any peer
  Refresh Epoch 1
  100, (received & used)
    10.1.12.1 from 10.1.12.1 (192.168.1.1)
      Origin IGP, metric 0, localpref 100, valid, external, best
      Community: no-advertise
      rx pathid: 0, tx pathid: 0x0
```

BGP routes that are set with the No-Advertise community are quickly seen with the command **show bgp** *afi safi* **community no-advertise** on IOS, IOS XR, and NX-OS, as shown in Example 4-29.

Example 4-29 *Display of Prefixes with No-Advertise Community*

```
R2# show bgp ipv4 unicast community no-advertise
! Output omitted for brevity
     Network          Next Hop            Metric LocPrf Weight Path
 *>  10.1.1.0/24      10.1.12.1                0             0 100 i
```

On IOS routers, the count of prefixes that were denied due to any well-known BGP communities are seen by viewing the BGP neighbor sessions, as shown in Example 4-30.

Example 4-30 *Denied Prefix Advertisement Displayed on BGP Neighbor Session*

```
R2# show bgp ipv4 unicast neighbors 10.1.23.3
! Output omitted for brevity
For address family: IPv4 Unicast
!
                                  Outbound    Inbound
  Local Policy Denied Prefixes:   --------    -------
    Well-known Community:                1        n/a
    Total:                               1          0
```

BGP Communities: No-Export

The No_Export community (0xFFFFFF01 or 4,294,967,041) specifies that when a route is received with this community, the route is not advertised to any EBGP peer. If the router receiving the No-Export route is a confederation member, the route can be advertised to other sub-ASs in the confederation.

Figure 4-6 demonstrates that R1 is advertising the 10.1.1.0/24 network to R2, which advertises to R3, and then on to R4. R4 does not advertise the prefix on to R5, so R5 does not have the 10.1.1.0/24 network in its BGP table.

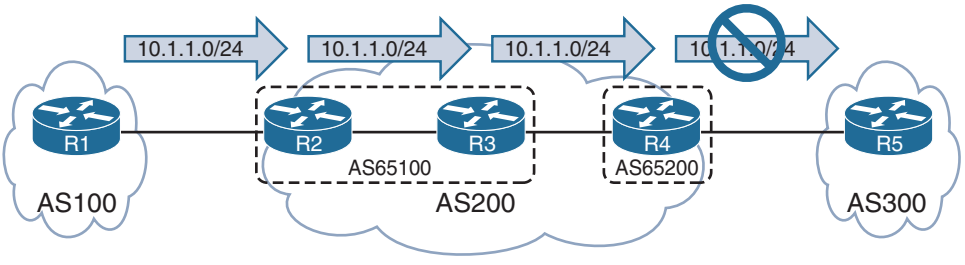

Figure 4-6 *BGP No-Export Community Topology*

Example 4-31 displays the BGP path attributes (PA) for the 10.1.1.10/24 network. Notice that the R3 and R4 displays "not advertised to EBGP peer." R3 is able to advertise the network to R4 (via Update Group #3) because they are members of the same confederation, even though their ASNs are different.

Example 4-31 *BGP Attributes for No-Export Routes*

```
R4# show bgp ipv4 unicast 10.1.1.0/24
! Output omitted for brevity
BGP routing table entry for 10.1.1.0/24, version 4
Paths: (1 available, best #1, table default, not advertised to EBGP peer)
  Not advertised to any peer
  Refresh Epoch 1
  (65100) 100, (received & used)
    10.1.23.2 (metric 20) from 10.1.34.3 (192.168.3.3)
      Origin IGP, metric 0, localpref 100, valid, confed-external, best
      Community: no-export
      rx pathid: 0, tx pathid: 0x0
```

```
R3# show bgp ipv4 unicast 10.1.1.0/24
BGP routing table entry for 10.1.1.0/24, version 6
Paths: (1 available, best #1, table default, not advertised to EBGP peer)
  Advertised to update-groups:
     3
```

```
  Refresh Epoch 1
  100, (Received from a RR-client), (received & used)
    10.1.23.2 from 10.1.23.2 (192.168.2.2)
      Origin IGP, metric 0, localpref 100, valid, confed-internal, best
      Community: no-export
      rx pathid: 0, tx pathid: 0x0

R3# show bgp ipv4 unicast update-group 3 | b member
  Has 1 member:
   10.1.34.4
```

Example 4-32 verifies that R2 and R4 both detect the No-Export community in the 10.1.1.0/24 network. This is the reason that R4 did not advertise the route to R5.

Example 4-32 *Viewing of BGP Routes with No-Export Community*

```
R4# show bgp ipv4 unicast community no-export | b Network
     Network          Next Hop            Metric LocPrf Weight Path
 *>  10.1.1.0/24      10.1.23.2                0    100      0 (65100) 100 i
```

```
R2# show bgp ipv4 unicast community no-export | b Network
     Network          Next Hop            Metric LocPrf Weight Path
 *>  10.1.1.0/24      10.1.12.1                0             0 100 i
```

Example 4-33 displays R4's neighbor session to R5. Notice that only the Well-known Community is shown for denied prefixes but does not differentiate between the No-Advertise and No-Export community.

Example 4-33 *Verification of Blocked Routes Due to Well-Known Communities*

```
R4# show bgp ipv4 unicast neighbors 10.1.45.5
! Output omitted for brevity
For address family: IPv4 Unicast
!
                                  Outbound    Inbound
  Local Policy Denied Prefixes:   --------    -------
    Well-known Community:                1        n/a
    Total:                               1          0
```

BGP Communities: Local-AS (No Export SubConfed)

The No_Export_SubConfed community (0xFFFFFF03 or 4,294,967,043) known as the Local-AS community specifies that a route with this community is not advertised outside of the local AS. If the router receiving a route with the Local-AS community is a

confederation member, the route can be advertised only within the sub-AS (Member-AS) and is not advertised between Member-ASs.

Figure 4-7 demonstrates that R1 is advertising the 10.1.1.0/24 network to R2, which advertises to R3. R3 does not advertise the prefix on to R4, so R4 does not have the 10.1.1.0/24 network in its BGP table.

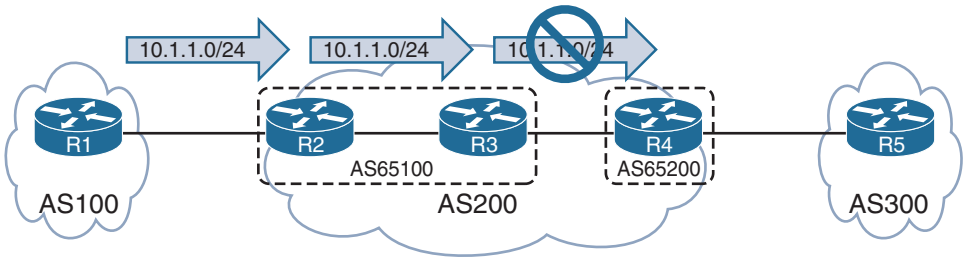

Figure 4-7 *BGP Local-AS Community Topology*

Example 4-34 confirms that the routes are set for "not advertised outside local AS" and that the routes are not advertised to any peer.

Example 4-34 *BGP Attributes for Local-AS Routes*

```
R3# show bgp ipv4 unicast 10.1.1.0/24
BGP routing table entry for 10.1.1.0/24, version 8
Paths: (1 available, best #1, table default, not advertised outside local AS)
  Not advertised to any peer
  Refresh Epoch 1
  100, (Received from a RR-client), (received & used)
    10.1.23.2 from 10.1.23.2 (192.168.2.2)
      Origin IGP, metric 0, localpref 100, valid, confed-internal, best
      Community: local-AS
      rx pathid: 0, tx pathid: 0x0
```

Example 4-35 displays how all of a router's prefixes with the Local-AS community are shown with the command **show bgp** *afi safi* **community local-as**.

Example 4-35 *Viewing of BGP Routes with Local-AS Community*

```
R3# show bgp ipv4 unicast community local-AS  | b Network
     Network          Next Hop            Metric LocPrf Weight Path
 *>i 10.1.1.0/24      10.1.23.2                0    100      0 100 i
```

```
R2# show bgp ipv4 unicast community local-AS  | b Network
     Network          Next Hop            Metric LocPrf Weight Path
 *>  10.1.1.0/24      10.1.12.1                0             0 100 i
```

Mandatory EBGP Route Policy for IOS XR

IOS XR does not advertise or receive prefixes from an EBGP peer by default. This is considered a safety mechanism to ensure that an organization is aware of the routes that it is receiving and advertising externally.

In Example 4-36, XR1 is configuring the initial BGP session for XR2 (10.1.12.2). Upon committing the configuration change, XR1 provides a warning message that explains that no prefixes will be accepted or advertised to the XR2.

Example 4-36 *No EBGP Route Policy for XR2*

```
RP/0/0/CPU0:XR1# config t
RP/0/0/CPU0:XR1(config)# router bgp 100
RP/0/0/CPU0:XR1(config-bgp)# neighbor 10.1.12.2
RP/0/0/CPU0:XR1(config-bgp-nbr)# remote-as 200
RP/0/0/CPU0:XR1(config-bgp-nbr)# address-family ipv4 unicast
RP/0/0/CPU0:XR1(config-bgp-nbr-af)# commit
RP/0/0/CPU0: 16:28:05.171 : config[67824]: %MGBL-CONFIG-6-DB_COMMIT : Configuration
  committed by user 'ChuckRobbins'. Use 'show configuration commit
    changes 1000000056' to view the changes.
RP/0/0/CPU0: 16:28:06.171 : bgp[1047]: %ROUTING-BGP-5-ADJCHANGE :
    neighbor 10.1.12.2 Up (VRF: default) (AS: 200)
RP/0/0/CPU0: 16:28:06.171 : bgp[1047]: %ROUTING-BGP-6-NBR_NOPOLICY :
    No inbound IPv4 Unicast policy is configured for EBGP neighbor 10.1.12.2
    No IPv4 Unicast prefixes will be accepted from the neighbor until inbound
    policy is configured.
RP/0/0/CPU0: 16:28:06.171 : bgp[1047]: %ROUTING-BGP-6-NBR_NOPOLICY :
    No outbound IPv4 Unicast policy is configured for EBGP neighbor 10.1.12..2
    No IPv4 Unicast prefixes will be sent to the neighbor until outbound policy
    is configured.
```

When you examine the BGP neighbor summary in Example 4-37, the ! is displayed next to the BGP State/Prefix Received column. In both the BGP neighbor summary and explicit BGP neighbor session output, a warning message is reiterated that routes will not be exchanged between XR1 and XR2.

Example 4-37 *IOS XR Summary with Indicator of Missing Route Policy*

```
RP/0/0/CPU0:XR1# show bgp ipv4 unicast summary
! Output omitted for brevity
BGP router identifier 192.168.1.1, local AS number 100

Some configured EBGP neighbors (under default or non-default vrfs)
do not have both inbound and outbound policies configured for IPv4 Unicast
address family. These neighbors will default to sending and/or
receiving no routes and are marked with '!' in the output below.
```

```
Use the 'show bgp neighbor <nbr_address>' command for details.

Neighbor        Spk    AS MsgRcvd MsgSent   TblVer  InQ OutQ  Up/Down  St/PfxRcd
10.1.12.2         0  1100       6       3        6    0    0 00:00:40          0!
```

```
RP/0/0/CPU0:XR1# show bgp ipv4 unicast neighbor 100.64.1.1
! Output omitted for brevity
BGP neighbor is 10.1.12.2

 For Address Family: IPv4 Unicast
  BGP neighbor version 6
  Update group: 0.3 Filter-group: 0.2  No Refresh request being processed
  EBGP neighbor with no inbound or outbound policy; defaults to 'drop'
  Route refresh request: received 0, sent 0
  0 accepted prefixes, 0 are bestpaths
```

Example 4-38 displays a simple PASS-ALL route policy that is associated to the EBGP peering between XR1 and XR2. This allows XR1 to advertise and receive all prefixes between the two routers.

Example 4-38 *Addition of Simple Route Policy for EBGP Neighbors*

```
XR1
route-policy PASS-ALL
  pass
end-policy

router bgp 100
  neighbor 10.1.12.2
    remote-as 200
    address-family ipv4 unicast
      route-policy PASS-ALL in
      route-policy PASS-ALL out
```

Filtering of Prefixes by Route Policy

The last component for finding missing BGP routes is through the examination of the BGP routing policies. As stated before, BGP route policies are applied before routes are inserted into the Loc-RIB table and as prefixes leave the Loc-RIB before they are advertised to a BGP peer.

IOS and NX-OS devices provide three methods of filtering routes inbound or outbound for a specific BGP peer. Each method could be used individually or simultaneously with other methods. The three methods are as follows: