

Junos® Networking Technologies

THIS WEEK: A PACKET WALKTHROUGH ON THE M, MX, AND T SERIES



How to get impressive results
using the power of ping.

By Antonio Sánchez-Monge

THIS WEEK: A PACKET WALKTHROUGH ON THE M, MX, AND T SERIES

One of the most exciting advantages of ping is its ability to uncover details of the internal architecture of M/MX/T-Series routers. In Antonio Sanchez Monge's new book, ping becomes a tourist guide that takes you on a packet walkthrough and provides you with a new perspective of the device architecture. The author shows you, in detail, that ping is a surprisingly flexible resource for feature testing, design, operation, and troubleshooting. Here is a book brimming with valuable information for troubleshooting the interactions between different components inside a router, with unique usage applications for both large and small network operators.

"It's not a trivial task to troubleshoot or monitor the healthiness of today's modern IP/MPLS network – yet here is a remedy using simple, basic tools. Once you go through this book you will be amazed at the impressive results you can get out of ping. *This Week: A Complete Packet Walkthrough on the M, MX, and T Series* should be essential reading for anyone who needs to quickly isolate and solve network issues."

Josef Buchsteiner, Distinguished Engineer, Juniper Networks

LEARN SOMETHING NEW ABOUT JUNOS THIS WEEK:

- Record the life of a packet as it walks through a network in a single capture (*the video camera*).
- Fully master the most universal and useful tools in the Internet: ping and traceroute.
- Track and influence the path followed by a packet not only in a network, but also inside a router.
- View the router as a set of functional components internally connected, and troubleshoot it very much like a real network.
- Understand the architecture of the control and forwarding planes in Junos devices.
- Better interpret link latency, reliability, and packet size tests.
- Generate gigabit packet streams with a simple ping.
- Perform all kinds of forwarding plane tests (including all Class of Service features) in a router with no network connections.
- Apply your skills to IPv4, IPv6, and MPLS networks.

Published by Juniper Networks Books
www.juniper.net/books

ISBN 978-1936779581



9 781936 779581

JUNIPER
NETWORKS



07100160

Junos® Networking Technologies

This Week: A Packet Walkthrough on the M, MX, and T Series

By Antonio Sánchez-Monge

<i>Chapter 1: Recording the Life of Ping and Traceroute</i>	<i>5</i>
<i>Chapter 2: Tracking and Influencing the Path of a Packet</i>	<i>37</i>
<i>Chapter 3: Spying on the Private Life of a Packet</i>	<i>61</i>
<i>Chapter 4: Classical Network Applications of Ping</i>	<i>85</i>
<i>Chapter 5: One Loop to Rule Them All</i>	<i>105</i>
<i>Appendix</i>	<i>157</i>

© 2013 by Juniper Networks, Inc. All rights reserved. Juniper Networks, the Juniper Networks logo, Junos, NetScreen, and ScreenOS are registered trademarks of Juniper Networks, Inc. in the United States and other countries. Junose is a trademark of Juniper Networks, Inc. All other trademarks, service marks, registered trademarks, or registered service marks are the property of their respective owners.

Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice. Products made or sold by Juniper Networks or components thereof might be covered by one or more of the following patents that are owned by or licensed to Juniper Networks: U.S. Patent Nos. 5,473,599, 5,905,725, 5,909,440, 6,192,051, 6,333,650, 6,359,479, 6,406,312, 6,429,706, 6,459,579, 6,493,347, 6,538,518, 6,538,899, 6,552,918, 6,567,902, 6,578,186, and 6,590,785.

Published by Juniper Networks Books

Author: Antonio Sanchez-Monge
 Technical Reviewers: Gonzalo Gomez Herrero, Anton Bernal, Josef Buchsteiner, Victor Rodriguez, David Dugal
 Editor in Chief: Patrick Ames
 Copyeditor and Proofer: Nancy Koerbel
 J-Net Community Manager: Julie Wider

This book is available in a variety of formats at:
www.juniper.net/dayone.

Send your suggestions, comments, and critiques by email to:
dayone@juniper.net.

Version History: First Edition, January 2013

ISBN: 978-1-936779-58-1 (print)
 Printed in the USA by Vervante Corporation.

ISBN: 978-1-936779-59-8 (ebook)

Juniper Networks Books are singularly focused on network productivity and efficiency. Peruse the complete library at <http://www.juniper.net/books>.

2 3 4 5 6 7 8 9 10 #7100160-en

About the Author

Antonio “Ato” Sanchez Monge (JNCIE-M #222 and CCIE #13098) holds a MS in Physics and a BA in Mathematics from the Universidad Autonoma de Madrid (UAM). He joined Juniper Networks in 2004, where he is currently working in the Advanced Services team. He has also authored another book in this series, *This Week: Deploying BGP Multicast VPNs*.

Author’s Acknowledgments

I would like to thank: Anton Bernal, my de facto mentor, for inspiring this book beginning-to-end; Patrick Ames, our great editor, for his continuous, high-quality, and timely help from the original idea through publication; Josef Buchsteiner, for finding some time in his busy schedule to provide expert-level technical feedback; Gonzalo Gomez and Victor Rodriguez for the very thorough and useful review; Jesus Angel Rojo and David Eduardo Martinez Fontano for setting the standards of the power of ping method; Lorenzo Murillo for teaching me the video camera trick; Oscar Pache and Javier Aviles for teaching me the snake testing basics back in 2007; David Dugal for assessing this book’s security issues; Oleg Karlashchuk and Erdem Sener for their infinite patience and help in the lab; Manuel Di Lenardo for sharing the Ethernet OAM loopback trick; Pablo Mosteiro, Efrain Gonzalez and Pablo Sagrera, for keeping the faith on this project; Dominique Cartella and Kisito Nguene-Ndoum for teaching me the value of accuracy in troubleshooting; Ramiro Cobo for sharing interesting failure scenarios. Last but not least, I would have never written this book without the support of my family and friends, especially Eva, Manuel, and Lucas.

Welcome to *This Week*

This Week books are an outgrowth of the extremely popular *Day One* book series published by Juniper Networks Books. *Day One* books focus on providing just the right amount of information that you can execute, or absorb, in a day. *This Week* books, on the other hand, explore networking technologies and practices that in a classroom setting might take several days to absorb or complete. Both libraries are available to readers in multiple formats:

- Download a free PDF edition at <http://www.juniper.net/dayone>.
- Get the ebook edition for iPhones and iPads at the iTunes Store>Books. Search for *Juniper Networks Books*.
- Get the ebook edition for any device that runs the Kindle app (Android, Kindle, iPad, PC, or Mac) by opening your device's Kindle app and going to the Kindle Store. Search for *Juniper Networks Books*.
- Purchase the paper edition at either Vervante Corporation (www.vervante.com) or Amazon (www.amazon.com) for prices between \$12-\$28 U.S., depending on page length.
- Note that Nook, iPad, and various Android apps can also view PDF files.
- If your device or ebook app uses .epub files, but isn't an Apple product, open iTunes and download the .epub file from the iTunes Store. You can now drag and drop the file out of iTunes onto your desktop and sync with your .epub device.

What You Need to Know Before Reading

Before reading this book, you should be familiar with the basic administrative functions of the Junos operating system, including the ability to work with operational commands and to read, understand, and change Junos configurations. There are several books in the *Day One* library on exploring and learning Junos, at <http://www.juniper.net/dayone>.

This book makes a few assumptions about you, the reader:

- You have a basic understanding of the Internet Protocol (IP) versions 4 and 6.
- You have access to a lab with at least the following components: one M/MX/T-Series router, one ethernet switch (with port mirroring capability), and one server or workstation.

After Reading This Book You'll Be Able To

- Record the life of a packet as it walks through a network in a single capture (the video camera).
- Fully master the most universal and useful tools in the Internet: ping and traceroute.
- Track and influence the path followed by a packet not only in a network, but also inside a router.
- View the router as a set of functional components internally connected, and troubleshoot it in much the same way as a real network.
- Understand the architecture of the control and forwarding planes in Junos devices.

- Better interpret link latency, reliability, and packet size tests.
- Generate gigabit packet streams with a simple ping.
- Perform all kinds of forwarding plane tests (including all Class of Service features) in a router with no network connections.
- Apply your skills to IPv4, IPv6, and MPLS networks.

Packet Captures and Configurations Source Files

All the packet captures and source configurations in this book are located on its landing page on J-Net: <http://www.juniper.net/dayone>.

Packet captures are available for download in libpcap format. These files are purified in the sense that all routing protocol noise is removed.

The Appendix contains the complete initial configurations of all the devices used for this book in the lab, and the Appendix is also available at the book's website in a .txt format.

A Book About Packet Walkthrough

This book started as a project called Power of Ping. If there is a command that literally every network professional or amateur has executed at least once, it is PING (Packet Inter Network Groper). Ping is not only the simplest networking tool, it is also the most powerful. Unfortunately, it is quite underrated. Someone who builds a network or troubleshoots an issue spending time and energy on ping tests is often considered a beginner. The results of ping tests are not always trivial to interpret, which is one of the reasons why ping is sometimes regarded as a suspect. This book acknowledges ping as an important tool, deserving of more respect. Properly used, ping is a surprisingly flexible resource for feature testing, design, operation, and troubleshooting.

As you read these pages, you will undertake a journey that reveals quite a few tricks the author has learned over years of lab testing and live network experiences. Ping is a topic unto itself, but discussing it is also a pretext to explain a variety of troubleshooting and lab techniques that can make your life easier.

One of the most exciting advantages of ping is its ability to uncover internal architecture details of M/MX/T-Series routers. In some ways, ping becomes a guide that takes you through a packet walkthrough and provides you with a new perspective on the device architecture.

Another killer application is the possibility of using ping as a surprisingly flexible traffic generator in places (like a production network or a remote lab) where it is either very difficult or impossible to get new ports connected within a reasonable timeframe, not to mention having an external traffic generator. Last but not least, ping can also behave as an independent auditor to provide valuable information while troubleshooting the interaction between different components inside a router; think of the router as a network on its own!

CHOOSE! There are several books for you to choose from inside of this one. If you are only interested in Junos router architecture, go straight to Chapter 3. If, on the other hand, you opened these pages just to learn new lab techniques, focus on Chapter 5 and the beginning of Chapter 1 (the video camera).

Roll up your sleeves and become a Lord (or Lady) of the Pings! :)

Chapter 1

Recording the Life of Ping and Traceroute

<i>Building a Flexible Lab Scenario.....</i>	<i>6</i>
<i>Life of an IPv4 Unicast Ping</i>	<i>9</i>
<i>Life of an IPv6 Unicast Ping</i>	<i>18</i>
<i>Life of an IPv4 Multicast Ping.....</i>	<i>22</i>
<i>Life of an IPv4 Traceroute</i>	<i>24</i>
<i>Life of a MPLS Ping</i>	<i>30</i>
<i>Answers to Try It Yourself Sections of Chapter 1.....</i>	<i>33</i>



This first chapter shows what happens every time you run ping or traceroute, although their varied applications are discussed later in the book. What you'll see here are ping and traceroute at work at the very beginning of the walkthrough. A selection of useful ping options are provided that become more elaborate as new scenarios are discussed, but don't expect an exhaustive coverage of all the ping options – this is not a command reference! Also, not all ICMP types and codes are covered; this is not a RFC!

This book is mainly (but not exclusively) about the Junos tools at your disposal.

Building a Flexible Lab Scenario

If you are working with a remote lab, cabling different topologies can be a challenge. Even if you are physically close to the lab, it is always nice to be able to create any logical topology just by using CLI commands. A flexible lab scenario can easily be achieved by connecting all your network devices to a switch, which is the sole physical requirement for all the labs in this book. With a switch, it all becomes much easier!

Physical Topology

In order to complete all the practical examples included in this book, you will need:

- Three MPLS-capable Junos OS routers with at least two ethernet (`fe-`, `ge-` or `xe-`) ports each. They don't need to be three different physical devices. For example, one single MX/M/T-series router supports `logical-systems` and is enough for the purposes of this book. You can leverage Junosphere to simulate the different devices!
- One ethernet switch with port mirroring capability. Any EX-series switch would be a perfect fit, but you can also use a `virtual-switch` or a `bridge-domain` in a MX-series or SRX device, or even a third-party switch.
- A host (PC, server, or workstation) with an ethernet port that can be put in promiscuous mode. This server is used to run a protocol decoder like `tcpdump` to spy on all the conversations held by the routers in the lab.

The proposed physical topology is depicted in Figure 1.1. The device models and port numbers are shown for reference, so you can interpret the output shown in the examples.

Three MX-Series routers with Packet Forwarding Engines (PFEs) based on the Trio chipset, running Junos OS 11.4R4.4, were used for this book. The Line Cards with Trio chipset contain the word *3D* in their name. Here we use the term *pre-Trio* for all the Line Cards whose PFEs were developed before the Trio chipset. At the time of this edition, non-Trio is equivalent to pre-Trio when we speak about M/MX/T-Series. But this may change in the future, hence the term pre-Trio.

Unless otherwise specified, you can safely assume that the results are independent of the PFE type. If available, use a multi-PFE router for P and choose ports from different PFEs to connect to switch X. Examples of multi-PFE routers are MX240, MX480, MX960, M120, M320, and all T-series models.

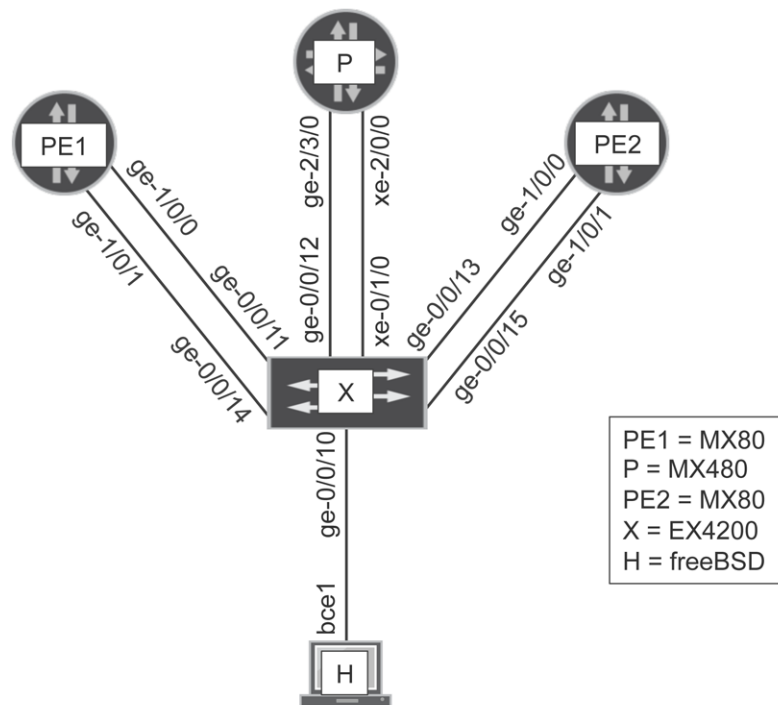


Figure 1.1 Physical Devices and Topology

TIP In order to map interfaces to PFEs in multi-PFE routers, combine the output of `show chassis fabric fpcs | match "fpc|pfe"` with `show chassis fpc pic-status`. Take into account empty PIC slots, too. See Chapter 3 for more details.

Logical Topology

The proposed logical topology and IP addressing scheme is shown in Figure 1.2. The VLAN IDs match the number of the logical interfaces. For example, `ge-2/3/0.113` has `vlan-id 113`.

PE1, P, and PE2 make up an IP/MPLS core running IS-IS and LDP. PE1 and PE2 are PE-routers, while P is a pure P-router. Finally, the CEs are virtual routers instantiated in the PEs. From a routing perspective, they behave like completely independent devices.

TIP The Appendix contains the complete initial configuration of all the devices.

NOTE Why a BGP/MPLS VPN setup? Frankly, it's more interesting for the tasks coming up in this book, because in this way you will see many of the case studies from both the perspective of IP and of MPLS.

MORE? An excellent source of information about MPLS is *This Week: Deploying MPLS*, available in the *Day One* library at <http://www.juniper.net/dayone>.

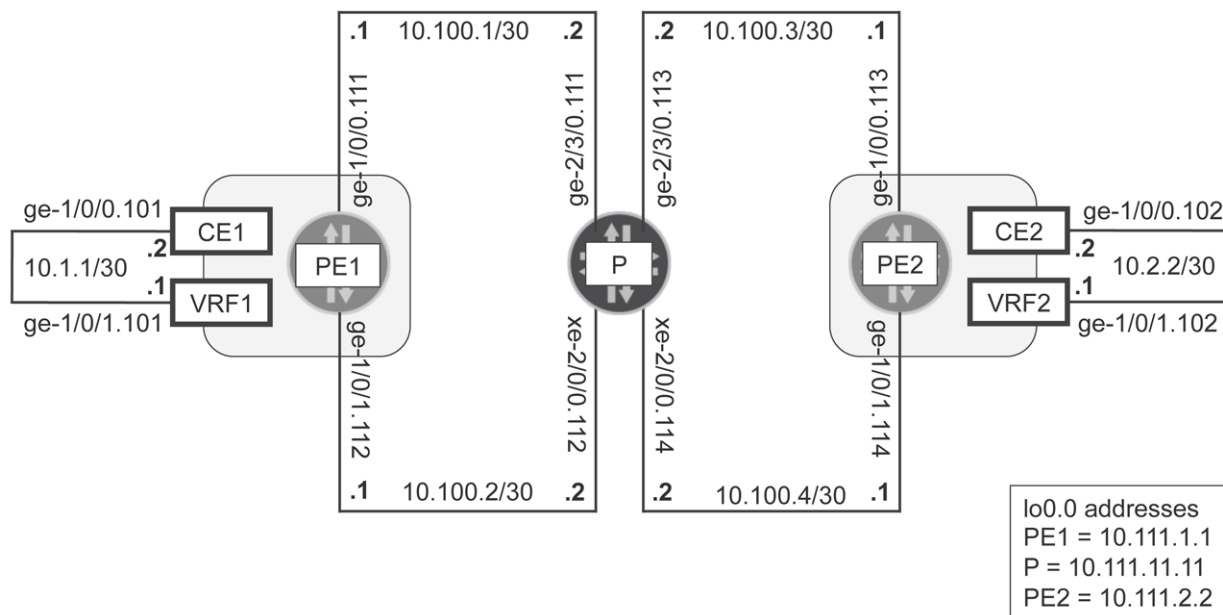


Figure 1.2 VLANs and Logical Connections Between the Routers

Inline Packet Captures

Capturing traffic in the wire, by mirroring packets at switch X towards server H, is a cornerstone of this chapter.

Let's start the packet capture at H:

```
[root@H ~]# tcpdump -nv -s 1500 -i bce1
tcpdump: listening on bce1, link-type EN10MB (Ethernet), capture size 1500 bytes
```

You can see this capture shows all traffic in the monitored VLANs, including routing protocol exchanges and transit data traffic. There are several ways to successfully find the packets involved in your ping:

- Decode the capture in real-time with a graphical utility like Wireshark (<http://www.wireshark.org/>) and filter the view. Alternatively, you can save the capture to a file and view it offline.
- Just let the capture run on the terminal and do a pattern search in the scroll buffer. You may have to clear the buffer often so that you can easily find the last flow. This can either be very easy, or a nightmare, depending on the terminal software.
- Remove the -v option so that only one line is displayed per packet. You will find the packets more easily, but obviously lose some details.
- Use tcpdump filtering expressions. This is not guaranteed to work, as the mirrored packets have a VLAN tag that doesn't match H's interface.

MORE? All the packet captures taken at H are available for download in libpcap format at the Juniper *Day One* website (<http://www.juniper.net/dayone>). These files are *purified* in the sense that all the routing protocol noise is removed.

TIP Every time that Capture x.y is referenced in the text, the corresponding file to open with Wireshark is *capture_x-y.pcap*.

Life of an IPv4 Unicast Ping

Once you have an operational test topology, it's time to start the game. One of the most frequent tasks for a networker is to perform a packet capture by sniffing the traffic that leaves or enters an interface. Let's go one step further. Let's prepare to take several pictures of a packet as it goes from CE1 to CE2: one picture of its trip from CE1 to PE1; another one from PE1 to P; then from P to PE2; and, finally, from PE2 to CE2. As you know, several pictures make up a movie, and that is what you are about to watch!

Now, let's send a one-packet ping from CE1 to CE2:

```
user@PE1> ping 10.2.2.2 routing-instance CE1 count 1
PING 10.2.2.2 (10.2.2.2): 56 data bytes
ping: sendto: No route to host

--- 10.2.2.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

TIP If you want to speed up the appearance of the statistics, use the `wait 1` option.

The packet capture shows no ICMP (Internet Control Message Protocol) echo request packets yet because CE1 does not have the route to the destination. Let's add it!

```
user@PE1> configure
user@PE1# set routing-instances CE1 routing-options static route 10/8 next-hop 10.1.1.1
user@PE1# commit and-quit
```

Troubleshooting One-Hop IPv4 Connectivity

Let's send the ping again:

```
user@PE1> ping 10.2.2.2 routing-instance CE1 count 1
PING 10.2.2.2 (10.2.2.2): 56 data bytes

--- 10.2.2.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

NOTE If the link was down or the IPv4 subnet was incorrectly configured, the message `No route to host` would still be displayed.

The `No route to host` message disappeared, but ping still fails. Let's have a look at the capture at H. Is there any ICMP packet? Not at all! Is CE1 actually generating any packet? Let's execute the `monitor traffic interface` command at PE1, which launches `tcpdump` in the background, and let it run for a minute:

```
user@PE1> monitor traffic interface ge-1/0/0.101 no-resolve size 2000
```

verbose output suppressed, use <detail> or <extensive> for full protocol decode
 Address resolution is OFF.
 Listening on ge-1/0/0.101, capture size 2000 bytes

```
<timestamp> Out arp who-has 10.1.1.1 tell 10.1.1.2
<timestamp> Out arp who-has 10.1.1.1 tell 10.1.1.2
[...]
```

So, CE1 is generating ARP requests, but it never receives an ARP reply back. The result is that CE1 cannot resolve the MAC address of 10.1.1.1, the next hop for IPv4 packets destined to 10.2.2.2.

Hence, even though you successfully configured a static route at CE1 towards 10/8, it remains in a hold state at the forwarding table:

```
user@PE1> show route 10.2.2.2 table CE1

CE1.inet.0: 3 destinations, 3 routes (3 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.0.0/8          *[Static/5] 00:09:39
                    > to 10.1.1.1 via ge-1/0/0.101

user@PE1> show route forwarding-table destination 10.2.2.2 table CE1
Routing table: CE1.inet
Internet:
Destination      Type RtRef Next hop          Type Index NhRef Netif
10.0.0.0/8       user   0 10.1.1.1      hold   532    3 ge-1/0/0.101

user@PE1> show arp vpn CE1
/* Starting with Junos 11.1, you can use the interface option too */

user@PE1>
```

For this same reason, the local IP address of VRF1 is also not reachable from CE1:

```
user@PE1> ping 10.1.1.1 routing-instance CE1 count 1
PING 10.1.1.1 (10.1.1.1): 56 data bytes

--- 10.1.1.1 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss

user@PE1> show route forwarding-table destination 10.1.1.1 table CE1
Routing table: CE1.inet
Internet:
Destination      Type RtRef Next hop          Type Index NhRef Netif
10.1.1.1/32       dest   0 10.1.1.1      hold   582    3 ge-1/0/0.101
```

Before turning off the ARP filtering at the switch X, let's try to see what can be done when ARP resolution is failing. One option is to configure static ARP resolution at the [edit interfaces <...> family inet address <...> arp] hierarchy, but let's use a different approach in which we also need to know the MAC address of the local interface at VRF1:

```
user@PE1> show interfaces ge-1/0/1 | match hardware
Current address: 5c:5e:ab:0a:c3:61, Hardware address: 5c:5e:ab:0a:c3:61
```

Now, the following ping command allows you to specify the destination MAC address while bypassing the forwarding table lookup:

```
user@PE1> ping 10.1.1.1 bypass-routing interface ge-1/0/0.101 mac-address 5c:5e:ab:0a:c3:61 count 1
```



```
PING 10.1.1.1 (10.1.1.1): 56 data bytes
```

```
--- 10.1.1.1 ping statistics ---
```

```
1 packets transmitted, 0 packets received, 100% packet loss
```

NOTE The mac-address ping option has been supported since Junos OS 11.1. It's mutually exclusive with the routing-instance option.

Although ping still fails, the echo request is sent this time. You can see it in Capture 1.1, that was taken at H:

```
[timestamp] IP (tos 0x0, ttl 64, id 60312, offset 0, flags [none], proto ICMP (1), length 84)
  10.1.1.2 > 10.1.1.1: ICMP echo request, id 3526, seq 0, length 64
```

TRY THIS Increase the packet count value. Unlike the IPv4 ID, the ICMPv4 ID value remains constant in all the same-ping packets and identifies the *ping flow*. The sequence number increases one by one.

The ping command originates IP packets with IP protocol #1, corresponding to ICMPv4. It's also interesting to see the packet at the VRF1 end, by running tcpdump in a parallel session at PE1, while you launch ping again:

```
user@PE1> monitor traffic interface ge-1/0/1.101 no-resolve size 2000
verbose output suppressed, use <detail> or <extensive> for full protocol decode
Address resolution is OFF.
Listening on ge-1/0/1.101, capture size 2000 bytes
```

```
<timestamp> In IP 10.1.1.2 > 10.1.1.1: ICMP echo request, id 3530, seq 0, length 64
<timestamp> Out arp who-has 10.1.1.2 tell 10.1.1.1
<timestamp> Out arp who-has 10.1.1.2 tell 10.1.1.1
<timestamp> Out arp who-has 10.1.1.2 tell 10.1.1.1
<timestamp> Out arp who-has 10.1.1.2 tell 10.1.1.1
<timestamp> Out arp who-has 10.1.1.2 tell 10.1.1.1
```

VRF1 at PE1 cannot resolve the MAC address associated to 10.1.1.2, hence ping fails. There is an interesting difference in behavior between CE1 and VRF1. While CE1 sends ARP requests periodically (every ~15 seconds), VRF1 only does it if triggered by a ping command. Can you spot the reason? Actually, CE1 has a static route pointing to 10.1.1.1 as next hop: as long as it is in hold state, CE1 will keep trying to resolve it over and over again.

Now it's time to remove the ARP filter from the switch. While keeping the packet capture active at H, execute:

```
user@X> configure
user@X# delete interfaces ge-0/0/11 unit 0 family ethernet-switching filter
user@X# delete interfaces ge-0/0/14 unit 0 family ethernet-switching filter
user@X# commit and-quit
```

The Junos OS kernel keeps trying to complete the pending ARP resolutions for the next hops in its forwarding table. At some point you should see in Capture 1.2 the ARP reply, as shown in Figure 1.3:

```
<timestamp> ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.1.1.1 tell 10.1.1.2, length 42
-----
<timestamp> ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.1.1.1 is-at 5c:5e:ab:0a:c3:61, length 42
```

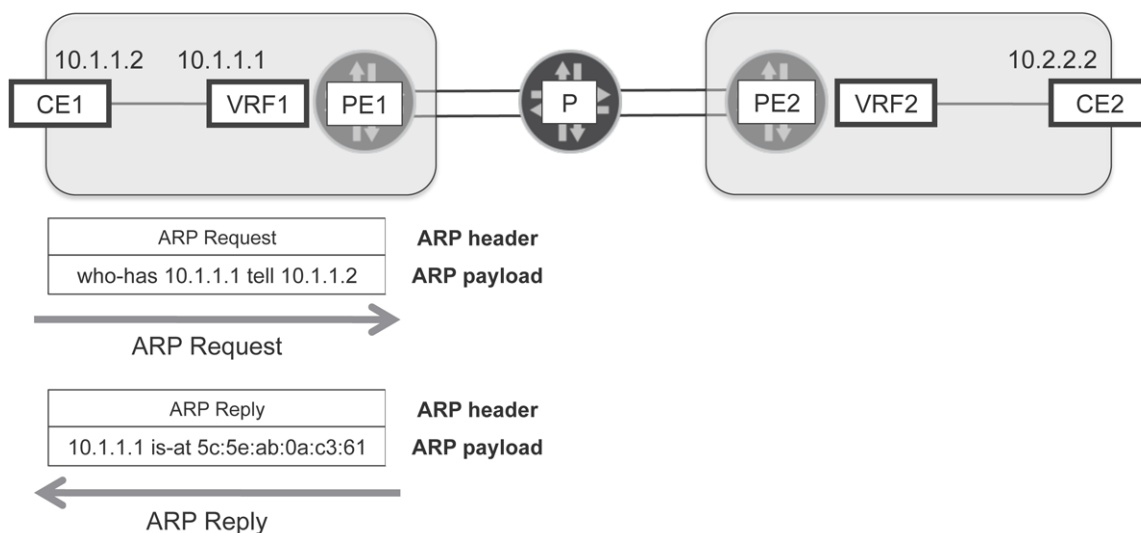


Figure 1.3 ARP Resolution at the First Hop

MORE? Turn on the `-e` and `-x` options of `tcpdump`, and check that the ARP packets have a specific Ethertype (0x0806), which is different from IPv4 (0x0800) and IPv6 (0x86dd) packets. Also, look carefully at the source and destination MAC addresses: the request is sent to the broadcast link address `ff:ff:ff:ff:ff:ff`, whereas the reply is sent unicast to the requester.

Try It Yourself: Managing ARP Cache Entries

With the capture at H active, execute at PE1:

```
clear arp vpn CE1 hostname 10.1.1.1
clear arp vpn VRF1 hostname 10.1.1.2
```

Can you explain the different behavior? You can also try this with and without ARP filtering at X.

Everything falls into place regarding the one-hop ping, as shown here:

```
user@PE1> show arp vpn CE1
MAC Address      Address      Name          Interface      Flags
5c:5e:ab:0a:c3:61 10.1.1.1    10.1.1.1      ge-1/0/0.101   none

user@PE1> show route forwarding-table destination 10.1.1.1 table CE1
Routing table: CE1.inet
Internet:
Destination      Type RtRef Next hop          Type Index NhRef Netif
10.1.1.1/32      dest  0 5c:5e:ab:a:c3:61  ucst  532   3 ge-1/0/0.101

user@PE1> ping 10.1.1.1 routing-instance CE1 count 1
PING 10.1.1.1 (10.1.1.1): 56 data bytes
64 bytes from 10.1.1.1: icmp_seq=0 ttl=64 time=0.533 ms

--- 10.1.1.1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.533/0.533/0.533/0.000 ms
```

And capture 1.3 at H shows both the one-hop ICMPv4 request and the reply:

```
<timestamp> IP (tos 0x0, ttl 64, id 7523, offset 0, flags [none], proto ICMP (1), length 84)
  10.1.1.2 > 10.1.1.1: ICMP echo request, id 3546, seq 0, length 64
-----
<timestamp> IP (tos 0x0, ttl 64, id 7526, offset 0, flags [none], proto ICMP (1), length 84)
  10.1.1.1 > 10.1.1.2: ICMP echo reply, id 3546, seq 0, length 64
```

Taking the ICMPv4 Echo Request to its Destination

Now that CE1 can finally send the packet to the next hop, check out what comes next:

```
user@PE1> show route forwarding-table destination 10.2.2.2 table CE1
Routing table: CE1.inet
Internet:
Destination      Type RtRef Next hop          Type Index NhRef Netif
10.0.0.0/8       user   0 5c:5e:ab:a:c3:61  ucst  532   3 ge-1/0/0.101

user@PE1> ping 10.2.2.2 routing-instance CE1 count 1
PING 10.2.2.2 (10.2.2.2): 56 data bytes
36 bytes from 10.1.1.1: Destination Net Unreachable
Vr HL TOS Len  ID Flg  off TTL Pro  cks      Src      Dst
 4  5  00 0054 231a  0 0000  40  01 4089 10.1.1.2 10.2.2.2

--- 10.2.2.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

TIP If you configured a name-server, PE1 tries to resolve 10.1.1.1, 10.1.1.2, and 10.2.2.2 into hostnames. If the server is not responsive, you can avoid significant delay with the no-resolve ping option.

So ping still fails because PE1 does not have a route to the destination yet. Let's look at the ICMPv4 packets in Capture 1.4, taken at H:

```
[timestamp] IP (tos 0x0, ttl 64, id 13549, offset 0, flags [none], proto ICMP (1), length 84)
  10.1.1.2 > 10.2.2.2: ICMP echo request, id 3559, seq 0, length 64
-----
[timestamp] IP (tos 0x0, ttl 255, id 0, offset 0, flags [none], proto ICMP (1), length 56)
  10.1.1.1 > 10.1.1.2: ICMP net 10.2.2.2 unreachable, length 36
      IP (tos 0x0, ttl 64, id 13549, offset 0, flags [none], proto ICMP (1), length 84)
  10.1.1.2 > 10.2.2.2: ICMP echo request, id 3559, seq 0, length 64
```

As you can see, the ping command originates IP packets with IP protocol #1, corresponding to ICMPv4. The ICMP protocol is also used to report errors, such as the one indicating net unreachable. Figure 1.4 shows the complete message exchange.

Look closely at the ICMPv4 header with a graphical analyzer like Wireshark, then match the echo request and net unreachable messages to Table 1.1 near the end of this chapter.

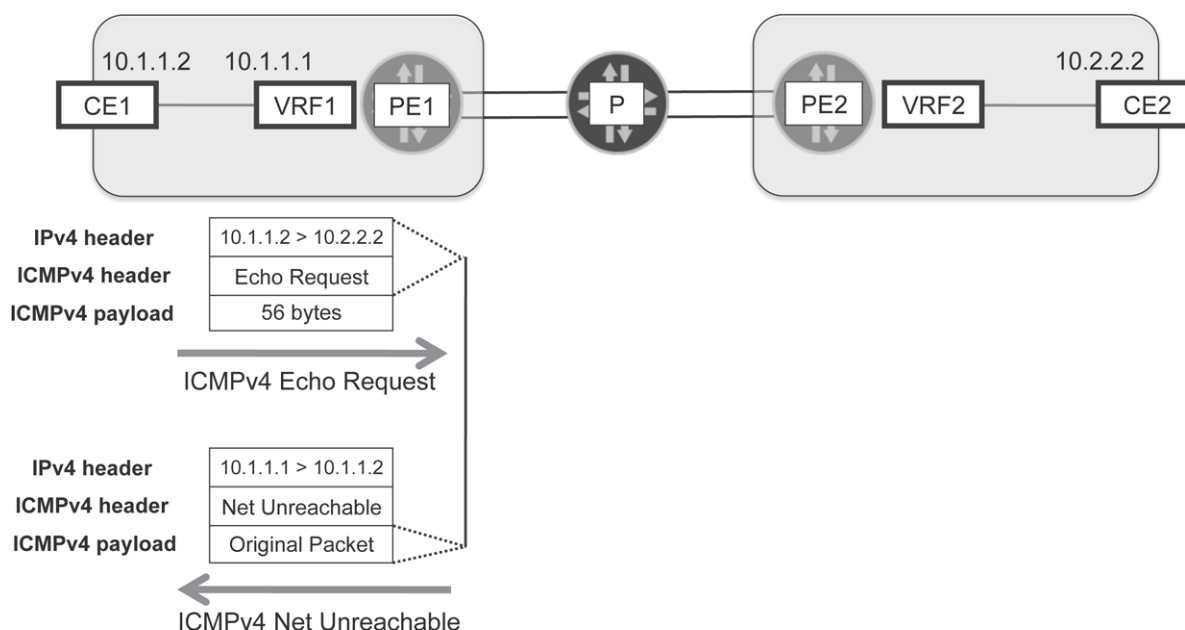


Figure 1.4 Generation of an ICMPv4 Net Unreachable Message

Why is PE1 generating net unreachable messages at VRF1? Execute at PE1:

```
user@PE1> show route forwarding-table destination 10.2.2.2 table VRF1
Routing table: VRF1.inet
Internet:
Destination      Type RtRef Next hop      Type Index NhRef Netif
default          perm  0          rjct      554    1
```

TRY THIS

It is considered a best practice to configure an explicit 0/0 route (typically with next hop discard), in order to avoid the flooding of net unreachable packets. Try to configure it at VRF1 and observe the difference.

It's time to configure a BGP session between PE1 and PE2 in order to have connectivity through the core. Configure at PE1:

```
user@PE1> configure
user@PE1# set protocols bgp group IBGP neighbor 10.111.2.2
user@PE1# commit and-quit
```

Now, configure at PE2:

```
user@PE2> configure
user@PE2# set protocols bgp group IBGP neighbor 10.111.1.1
user@PE2# commit and-quit
```

Verify at PE1 and PE2 that the BGP session is established:

```
user@PE1> show bgp summary
Groups: 1 Peers: 1 Down peers: 0
Table Tot Paths Act Paths Suppressed History Damp State Pending
inet.0 0 0 0 0 0 0 0
bgp.l3vpn.0 1 1 0 0 0 0 0
Peer AS InPkt OutPkt OutQ Flaps Last Up/Dwn State|#Active/Received/
Accepted/Damped...
10.111.2.2 65000 41 33 0 2 6:41 Establ
  bgp.l3vpn.0: 1/1/1/0
  VRF1.inet.0: 1/1/1/0
```

Send one more single-packet ping:

```
user@PE1> ping 10.2.2.2 routing-instance CE1 count 1
PING 10.2.2.2 (10.2.2.2): 56 data bytes

--- 10.2.2.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

Now the capture 1.5 at H shows the same ICMPv4 echo request packet four times, one at each segment of its trip from CE1 to CE2. What follows is just one single packet at different stages of its life through the network:

```
[timestamp] IP (tos 0x0, ttl 64, id 40308, offset 0, flags [none], proto ICMP (1), length 84)
  10.1.1.2 > 10.2.2.2: ICMP echo request, id 3649, seq 0, length 64
-----
[timestamp] MPLS (label 299840, exp 0, ttl 63)
  (label 16, exp 0, [S], ttl 63)
  IP (tos 0x0, ttl 63, id 40308, offset 0, flags [none], proto ICMP (1), length 84)
  10.1.1.2 > 10.2.2.2: ICMP echo request, id 3649, seq 0, length 64
-----
[timestamp] MPLS (label 16, exp 0, [S], ttl 62)
  IP (tos 0x0, ttl 63, id 40308, offset 0, flags [none], proto ICMP (1), length 84)
  10.1.1.2 > 10.2.2.2: ICMP echo request, id 3649, seq 0, length 64
-----
[timestamp] IP (tos 0x0, ttl 61, id 40308, offset 0, flags [none], proto ICMP (1), length 84)
  10.1.1.2 > 10.2.2.2: ICMP echo request, id 3649, seq 0, length 64
```

ALERT! This is just one packet! The proof is in the IPv4 header ID, which remains the same as it travels along the router hops. In addition, as you will see in Chapter 4, when an IPv4 packet is fragmented somewhere in the path, all the resulting fragments also keep the original IPv4 ID.

Figure 1.5 provides detail of the different encapsulations at each stage of the packet. MPLS labels may have different values in your scenario.

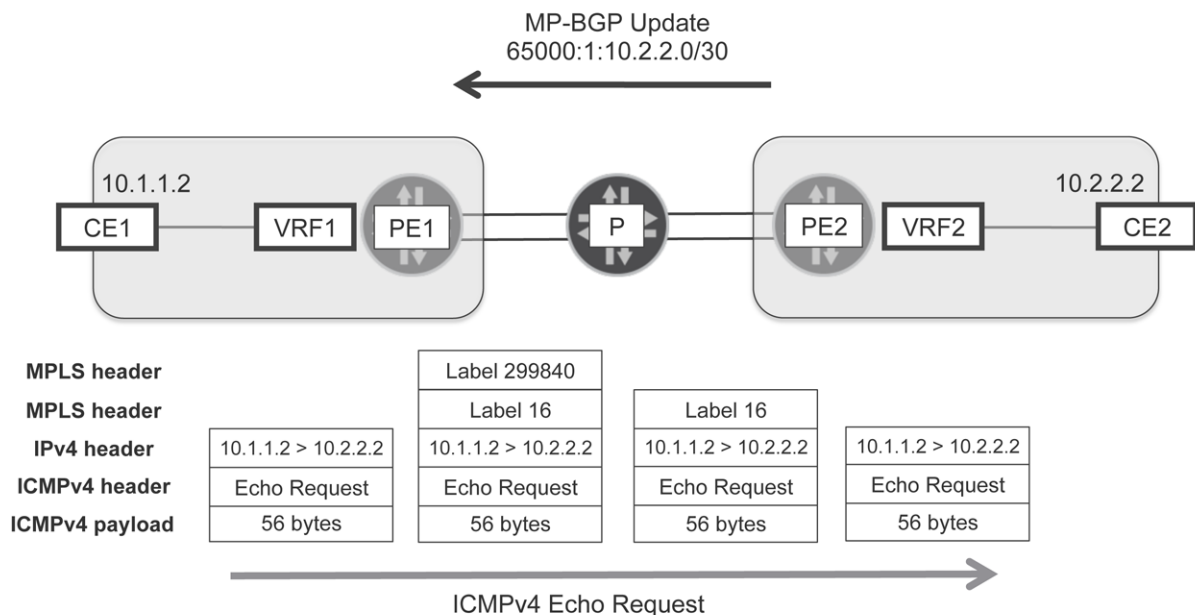


Figure 1.5 Life of an ICMPv4 Echo Request from CE1 to CE2

Try It Yourself: ping is a Return Trip

You expect the ICMP echo reply later in the capture, but it doesn't show up. Can you figure out why ping is failing at this point? Try to fix the issue! The answer is at the end of this chapter.

Bringing the ICMPv4 Echo Reply Back to the Source

Even though you've fixed the end-to-end connectivity issue, ping is still failing. This is because PE2 has a firewall filter called BLOCK-ICMP applied to lo0.0. This filter has a reject action that generates an ICMPv4 message reporting communication administratively filtered.

TRY THIS

If you change the discard action for reject, the echo request is silently dropped. Although this makes troubleshooting more difficult, from a security perspective discard is a much better design option than reject.

Let's remove the filter from PE2:

```
user@PE2> configure
user@PE2# delete interfaces lo0.0 family inet filter input BLOCK-ICMP
user@PE2# commit and-quit
```

The end-to-end ping should succeed now:

```
user@PE1> ping 10.2.2.2 routing-instance CE1 count 1
PING 10.2.2.2 (10.2.2.2): 56 data bytes
64 bytes from 10.2.2.2: icmp_seq=0 ttl=61 time=0.751 ms

--- 10.2.2.2 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.751/0.751/0.751/0.000 ms
```

If you are puzzled at this point, that's okay! The filter was applied at 100.0, but 100.0 is in the master instance inet.0, so why did it affect traffic addressed to CE2 at all? Well, in Junos, if a routing instance (in this case, CE2) has no local 100 unit, all its input control traffic is evaluated by the master instance's 100 filter.

TRY THIS Put the 100.0 filter back in place, create 100.1, and associate it to routing-instance CE2. The end-to-end ping should still succeed!

The ICMPv4 echo reply now shows up once at each of the four network segments in Capture 1.6 at H, as illustrated in Figure 1.6:

```
[timestamp] IP (tos 0x0, ttl 64, id 40242, offset 0, flags [none], proto ICMP (1), length 84)
10.2.2.2 > 10.1.1.2: ICMP echo reply, id 3691, seq 0, length 64
-----
[timestamp] MPLS (label 299872, exp 0, ttl 63)
(label 16, exp 0, [S], ttl 63)
IP (tos 0x0, ttl 63, id 40242, offset 0, flags [none], proto ICMP (1), length 84)
10.2.2.2 > 10.1.1.2: ICMP echo reply, id 3691, seq 0, length 64
-----
[timestamp] MPLS (label 16, exp 0, [S], ttl 62)
IP (tos 0x0, ttl 63, id 40242, offset 0, flags [none], proto ICMP (1), length 84)
10.2.2.2 > 10.1.1.2: ICMP echo reply, id 3691, seq 0, length 64
-----
[timestamp] IP (tos 0x0, ttl 61, id 40242, offset 0, flags [none], proto ICMP (1), length 84)
10.2.2.2 > 10.1.1.2: ICMP echo reply, id 3691, seq 0, length 64
```

Look closely at Capture 1.6 with a graphical analyzer like Wireshark, and start by matching the echo reply message to Table 1.1.

The sequence number is 0 for the first packet of a ping flow, and it increments sequentially for the following packets (if the count is greater than 1 or not specified). In this way, you can match an echo reply to the request that triggered it. What if you simultaneously execute the same ping command from different terminals? The ID inside the ICMPv4 header (not to be mistaken for the IPv4 ID) is copied from the echo request into the echo reply: in this way, the sender can differentiate several ping streams.

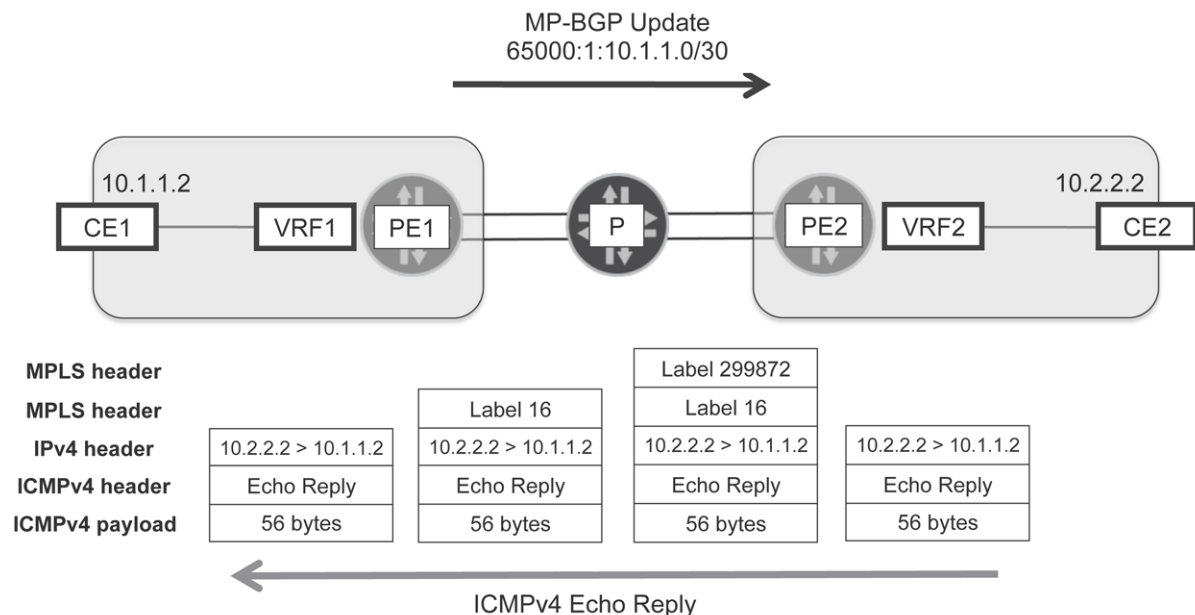


Figure 1.6 Life of an ICMPv4 Echo Reply from CE2 to CE1

This *video camera* assists you with analyzing any (ICMP is just one example) protocol conversation in the complete forwarding path, including the endpoints as well as the transit segments.

Life of an IPv6 Unicast Ping

There are many points in common between ICMPv6 and ICMPv4, but there are differences, too. Let's start with the IPv6-to-MAC address resolution.

Troubleshooting One-Hop IPv6 Connectivity

With the capture started at H, execute:

```
user@PE1> clear ipv6 neighbors
fc00::1:1          5c:5e:ab:0a:c3:61  deleted
[...]
```

And capture 1.7 at H shows (you can also see in Figure 1.7):

```
<timestamp> IP6 (hlim 255, next-header ICMPv6 (58) payload length: 32) fe80::5e5e:ab00:650a:c360 >
ff02::1:ff01:1: [icmp6 sum ok] ICMP6, neighbor solicitation, length 32, who has fc00::1:1
source link-address option (1), length 8 (1): 5c:5e:ab:0a:c3:60
-----
<timestamp> IP6 (hlim 255, next-header ICMPv6 (58) payload length: 32) fe80::5e5e:ab00:650a:c361 >
fe80::5e5e:ab00:650a:c360: [icmp6 sum ok] ICMP6, neighbor advertisement, length 32, tgt is
fc00::1:1, Flags [router, solicited, override]
destination link-address option (2), length 8 (1): 5c:5e:ab:0a:c3:61
```

In the IPv6 world ICMPv6 is also used to achieve the same functionality as ARP in IPv4, and this process is part of Neighbor Discovery (ND).

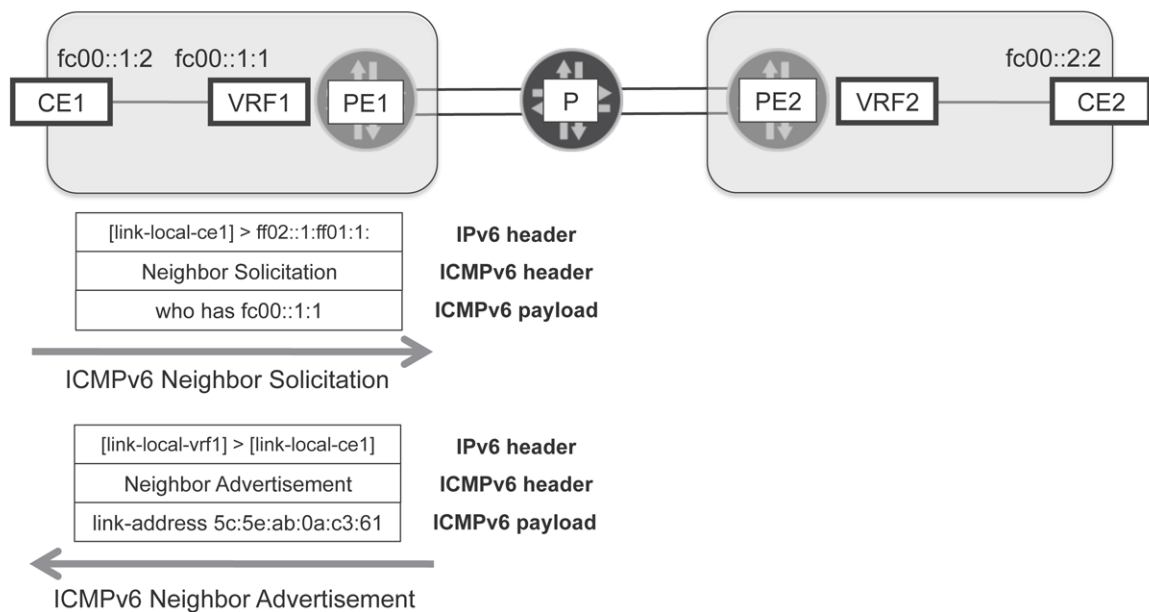


Figure 1.7 IPv6 Neighbor Discovery at the First Hop

REVIEW Can you spot why the ND process to learn about fc00::1:1 MAC from CE1 restarts immediately without any ping to trigger it? You already saw the answer in the equivalent IPv4 scenario.

Look closely at the ICMP payload with a graphical analyzer like Wireshark, then match the neighbor solicitations and advertisements to Table 1.1.

NOTE The multicast destination IPv6 address ff02::1:ff01:1: is the Solicited-Node address associated to fc00::1:1, according to RFC 429: *IP Version 6 Addressing Architecture*. The associated MAC address is further constructed according to RFC 6085: *Address Mapping of IPv6 Multicast Packets on Ethernet*. Finally, you can have a look at RFC 4861: *Neighbor Discovery for IP Version 6 (IPv6)*.

As an exercise to complete the one-hop connectivity check, verify that ping fc00::1:1 routing-instance CE1 count 1 succeeds at PE1, and interpret the capture.

ICMPv6 Echo Request and Reply Between CE1 and CE2

Try to ping from CE1 to CE2:

```
user@PE1> ping fc00::2:2 routing-instance CE1 count 1
PING6(56=40+8+8 bytes) fc00::1:2 --> fc00::2:2
64 bytes from fc00::1:1: No Route to Destination
Vr TC Flow Plen Nxt Hlim
 6 00 00000 0010 3a 40
fc00::1:2->fc00::2:2
ICMP6: type = 128, code = 0

--- fc00::2:2 ping6 statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

Check Capture 1.8 at H, illustrated in Figure 1.8:

```
<timestamp> IP6 (hlim 64, next-header ICMPv6 (58) payload length: 16) fc00::1:2 > fc00::2:2: [icmp6
sum ok] ICMP6, echo request, length 16, seq 0
-----
<timestamp> IP6 (hlim 64, next-header ICMPv6 (58) payload length: 64) fc00::1:1 > fc00::1:2: [icmp6
sum ok] ICMP6, destination unreachable, length 64, unreachable route fc00::2:2
```

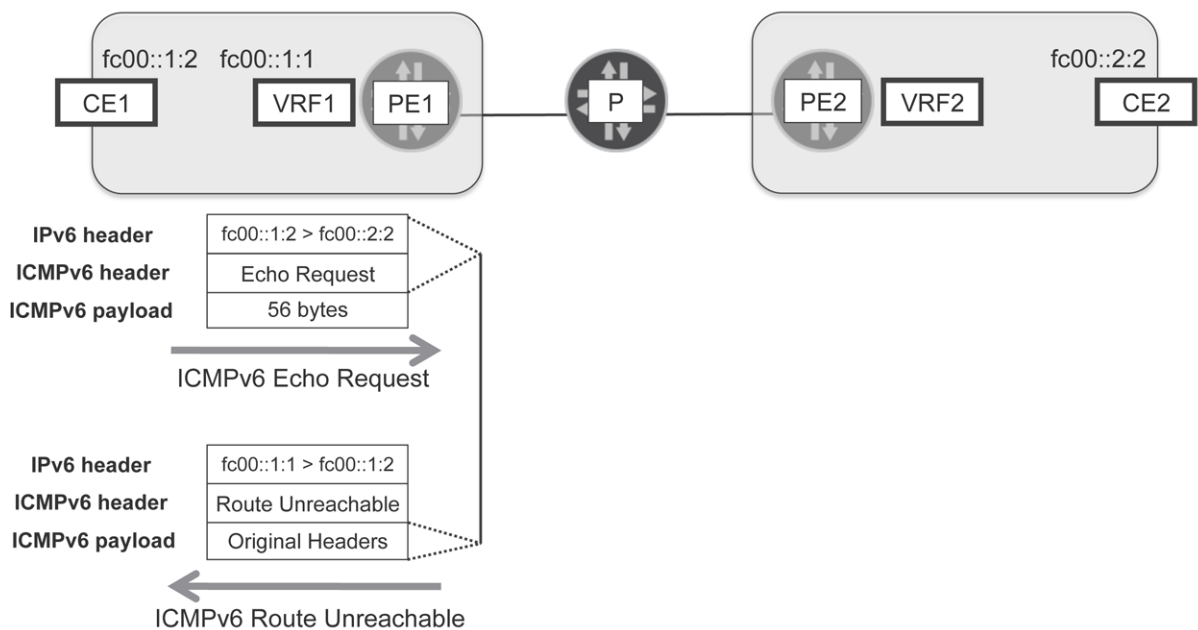


Figure 1.8 Generation of an ICMPv6 Route Unreachable Message

Look closely at the ICMP payload with a graphical analyzer like Wireshark, then match the echo request and route unreachable messages to Table 1.1.

It's time to exchange IPv6 VPN routes between PE1 and PE2. This model, often referred to as 6VPE, relies on a MPLS backbone with IPv4 Control Plane for data transport.

MORE? Read RFC 4659: *BGP-MPLS IP Virtual Private Network (VPN) Extension for IPv6 VPN*.

Configure at PE1:

```
user@PE1> configure
user@PE1# set protocols bgp group IBGP family inet6-vpn unicast
user@PE1# commit and-quit
```

Now, configure at PE2:

```
user@PE2> configure
user@PE2# set protocols bgp group IBGP family inet6-vpn unicast
user@PE2# commit and-quit
```

Check that inet6-vpn routes are being exchanged between VRF1 and VRF2:

```
user@PE1> show bgp summary
Groups: 1 Peers: 1 Down peers: 0
Table Tot Paths Act Paths Suppressed History Damp State Pending
bgp.13vpn.0 1 1 0 0 0 0 0
Peer AS InPkt OutPkt OutQ Flaps Last Up/Dwn State|#Active/Received/
Accepted/Damped...
10.111.2.2 65000 6 6 0 0 35 Estab1
bgp.13vpn.0: 1/1/1/0
VRF1.inet.0: 1/1/1/0
```

```
bgp.l3vpn-inet6.0: 1/1/1/0
VRF1.inet6.0: 1/1/1/0
```

Once end to end connectivity is ensured, ping should work fine between CE1 and CE2:

```
user@PE1> ping fc00::2:2 routing-instance CE1 count 1
PING6(56=40+8+8 bytes) fc00::1:2 --> fc00::2:2
16 bytes from fc00::2:2, icmp_seq=0 hlim=61 time=0.859 ms

--- fc00::2:2 ping6 statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/std-dev = 0.859/0.859/0.859/0.000 ms
```

Take the time to read Capture 1.9 and map it to Figures 1.9 and 1.10. Also try to map echo reply messages to Table 1.1. Figures 1.9 and 1.10 illustrate echo request and echo reply from CE1 & CE2.

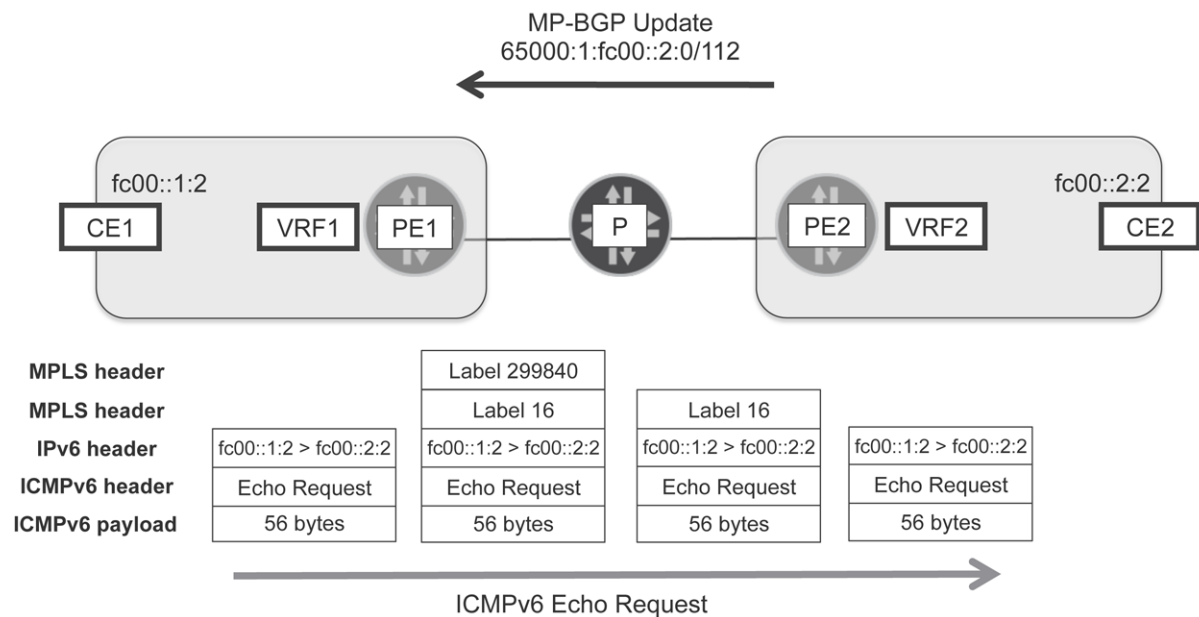


Figure 1.9 Life of an ICMPv6 Echo Request from CE1 to CE2

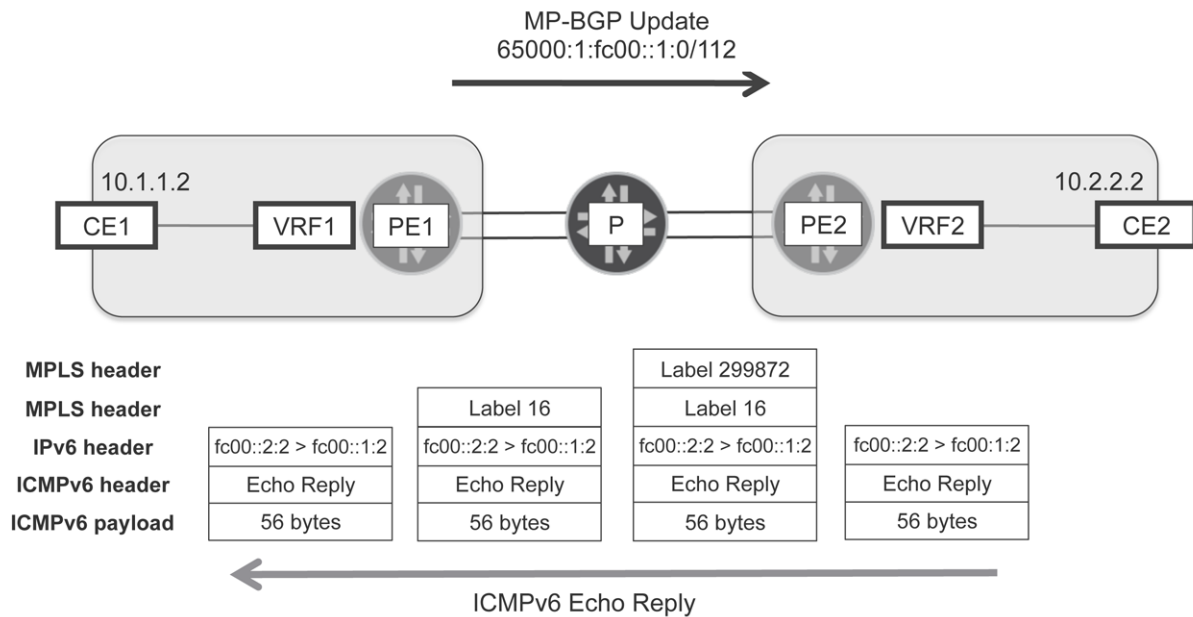


Figure 1.10 Life of an ICMPv6 Echo Reply from CE2 to CE1

Life of an IPv4 Multicast Ping

One useful application of multicast ping is checking multicast MAC and IP bindings of a neighbor. For example, a neighboring router typically replies to the all-routers link-local IPv4 multicast address 224.0.0.2:

```
user@PE1> ping 224.0.0.2 interface ge-1/0/0.101 count 1
PING 224.0.0.2 (224.0.0.2): 56 data bytes
64 bytes from 10.1.1.1: icmp_seq=0 ttl=64 time=1.052 ms

--- 224.0.0.2 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 1.052/1.052/1.052/0.000 ms
```

TIP If a link selectively drops certain multicast packets like OSPF or LDP hellos, and is perfectly fine with unicast traffic, then ping to the relevant 224.0.0.x address.

How about routable IPv4 multicast addresses? Let's configure a complete scenario by first building a Multicast VPN infrastructure. Configure at PE1:

```
user@PE1> configure
user@PE1# set protocols bgp group IBGP family inet-mvpn signaling
user@PE1# set routing-instances VRF1 protocols mvpn
user@PE1# set routing-instances VRF1 protocols pim interface all mode sparse
user@PE1# set routing-instances VRF1 provider-tunnel rsdp-te label-switched-path-template default-
template
user@PE1# set routing-instances VRF1 routing-options multicast ssm-groups 239/8
user@PE1# commit and-quit
```

Now, configure at PE2:

```

user@PE2> configure
user@PE2# set protocols bgp group IBGP family inet-mvpn signaling
user@PE2# set routing-instances VRF2 protocols mvpn
user@PE2# set routing-instances VRF2 protocols pim interface all mode sparse
user@PE2# set routing-instances VRF2 routing-options multicast ssm-groups 239/8
user@PE2# set protocols igmp interface ge-1/0/1.102 version 3 static group 239.1.1.1 source
10.1.1.2
user@PE2# commit and-quit

```

Check that the PIM join state is magically propagated up to VRF1:

```

user@PE1> show pim join instance VRF1
Instance: PIM.VRF1 Family: INET
R = Rendezvous Point Tree, S = Sparse, W = Wildcard

Group: 239.1.1.1
  Source: 10.1.1.2
  Flags: sparse,spt
  Upstream interface: ge-1/0/1.101

Instance: PIM.VRF1 Family: INET6
R = Rendezvous Point Tree, S = Sparse, W = Wildcard

```

MORE? Puzzled? If you want a better understanding of the configuration above, have a look at the *This Week: Deploying BGP Multicast VPN, 2nd Edition* book, available at <http://www.juniper.net/dayone>.

Launch a multicast ping from CE1:

```

user@PE1> ping 239.1.1.1 routing-instance CE1 count 1
PING 239.1.1.1 (239.1.1.1): 56 data bytes

--- 239.1.1.1 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss

```

Nothing in the packet capture, right? At this point it's very common to start configuring IPv4 multicast protocols (PIM, IGMP) at CE1, but that is useless if you are trying to simulate a multicast source, which is stateless. You just need a few more ping options:

```

user@PE1> ping 239.1.1.1 interface ge-1/0/0.101 count 1
PING 239.1.1.1 (239.1.1.1): 56 data bytes

--- 239.1.1.1 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss

```

Capture 1.10 shows an ICMPv4 packet that only survives one hop:

```

<timestamp> IP (tos 0x0, ttl 1, id 9133, offset 0, flags [none], proto ICMP (1), length 84)
172.30.77.181 > 239.1.1.1: ICMP echo request, id 8941, seq 0, length 64

```

You can see that the destination MAC address is 01:00:5e:01:01:01, mapped from 239.1.1.1 in accordance with RFC 1112. In other words, no ARP is required in IPv4 multicast, as the mapping is deterministic.

Try It Yourself: Taking the Multicast ping Further

Can you spot why the packet is stopping at VRF1 and does not move beyond it? Don't look for a complex reason, it's all about ping!

Once you manage to get the ICMPv4 echo request all the way up to CE1, have a look at Capture 1.11:

```

10.1.1.2 > 239.1.1.1: ICMP echo request, id 3818, seq 0, length 64
-----
<timestamp> MPLS (label 299888, exp 0, [S], ttl 9)
  IP (tos 0x0, ttl 9, id 42264, offset 0, flags [none], proto ICMP (1), length 84)
  10.1.1.2 > 239.1.1.1: ICMP echo request, id 3818, seq 0, length 64
-----
<timestamp> MPLS (label 16, exp 0, [S], ttl 8)
  IP (tos 0x0, ttl 9, id 42264, offset 0, flags [none], proto ICMP (1), length 84)
  10.1.1.2 > 239.1.1.1: ICMP echo request, id 3818, seq 0, length 64
-----
<timestamp> IP (tos 0x0, ttl 7, id 42264, offset 0, flags [none], proto ICMP (1), length 84)
  10.1.1.2 > 239.1.1.1: ICMP echo request, id 3818, seq 0, length 64

```

It's normal *not* to see any echo reply, as there is no real receiver at CE2. In production networks, you can use ping to generate multicast traffic, and use input firewall filters to count the traffic in a per-flow basis.

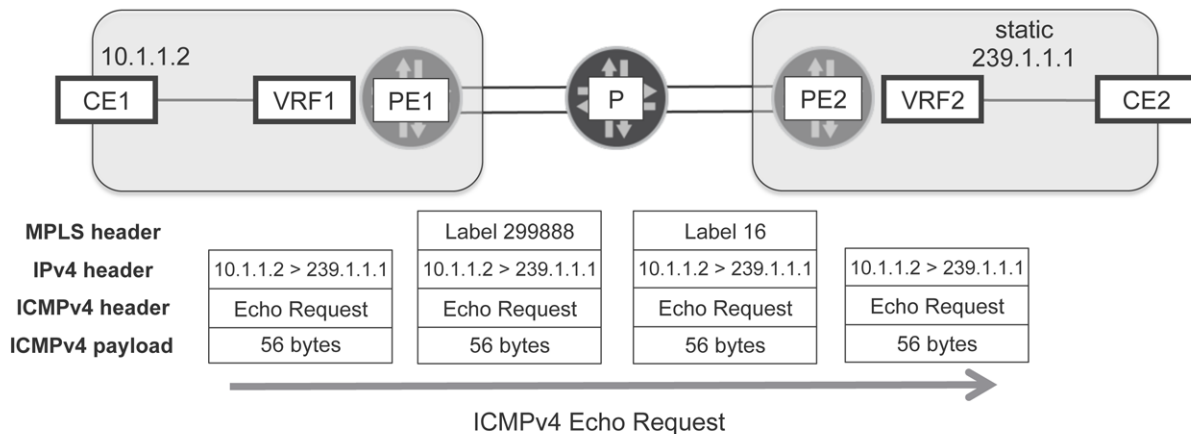


Figure 1.11 Life of an ICMPv4 Multicast Echo Request from CE1 to CE2

Life of an IPv4 Traceroute

With the capture started at H, launch a traceroute from CE1 to CE2 (when the no-resolve option is used, no attempts are made to resolve IPs into hostnames):

```

user@PE1> traceroute 10.2.2.2 routing-instance CE1 no-resolve
traceroute to 10.2.2.2 (10.2.2.2), 30 hops max, 40 byte packets
 1  10.1.1.1  0.510 ms  0.303 ms  0.299 ms
 2  * * *
 3  10.2.2.1  0.524 ms  0.382 ms  0.359 ms
 4  10.2.2.2  0.542 ms  0.471 ms  0.481 ms

```

What just happened behind the scenes? Have a look at Capture 1.12. CE1 first sent three UDP packets (IP protocol #17) with different UDP destination port numbers [33434, 33435, 33436], and TTL=1 (Time To Live in the IP header). Then three more packets with UDP destination ports [33437, 33438, 33439], and TTL=2. This process is iterated, incrementing the TTL value by 1 at each line, as well as the UDP destination port of each packet.

Let's look at the life of the first UDP probe sent for each TTL value. The following movie has several parts: action #1 for TTL=1, action #2 for TTL=2, etc. Each chapter or *action* shows the life of an UDP probe, as well as the reply (if any) sent by one of the routers in the path.

NOTE The source and destination UDP ports you get in your lab may be different from the ones shown in this book.

Action #1

CE1 sends an UDP probe with TTL=1, and PE1 replies with an ICMP time exceeded packet as shown in Figure 1.12:

```
[timestamp] IP (tos 0x0, ttl 1, id 36596, offset 0, flags [none], proto UDP (17), length 40)
10.1.1.2.36595 > 10.2.2.2.33434: UDP, length 12
-----
[timestamp] IP (tos 0x0, ttl 255, id 0, offset 0, flags [none], proto ICMP (1), length 56)
10.1.1.1 > 10.1.1.2: ICMP time exceeded in-transit, length 36
    IP (tos 0x0, ttl 1, id 36596, offset 0, flags [none], proto UDP (17), length 40)
    10.1.1.2.36595 > 10.2.2.2.33434: UDP, length 12
```

The original UDP probe is included in the ICMPv4 time exceeded packet, as shown in *italics* in the packet captures.

Look closely at the ICMPv4 header with a graphical analyzer like Wireshark, then match the time exceeded messages to Table 1.1.

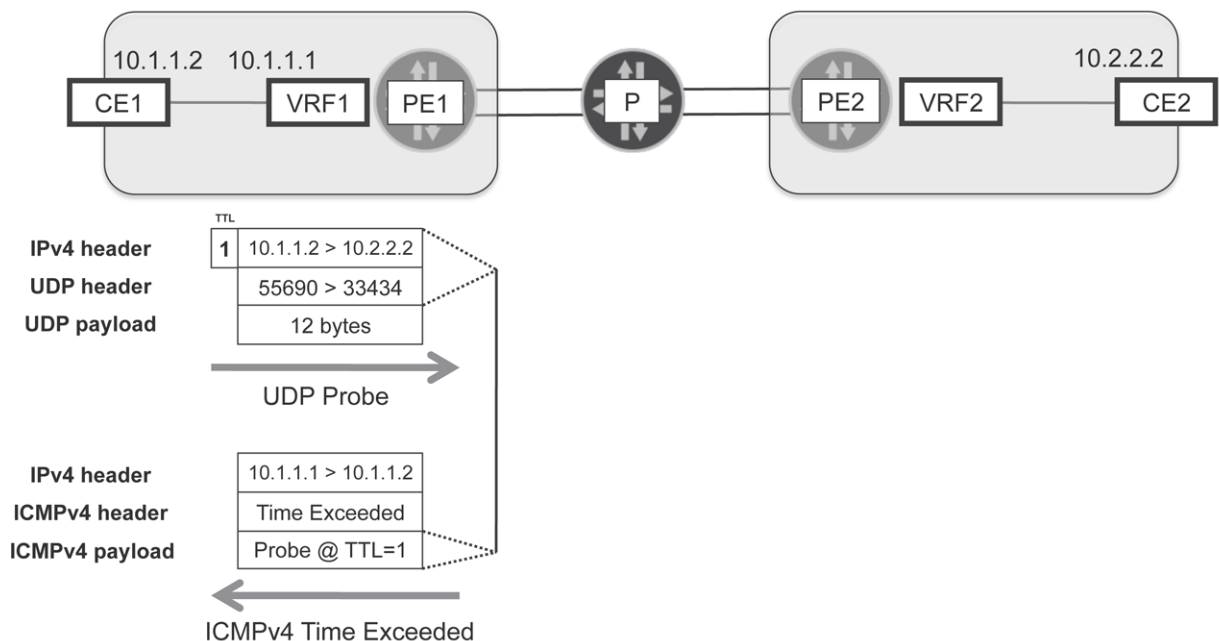


Figure 1.12 Life of a TTL=1 UDP Packet Probe

Action #2

CE1 sends an UDP probe with TTL=2, which gets silently discarded by P as shown in Figure 1.13:

```
[timestamp] IP (tos 0x0, ttl 2, id 36599, offset 0, flags [none], proto UDP (17), length 40)
10.1.1.2.36595 > 10.2.2.2.33437: UDP, length 12
```

```
-----
[timestamp] MPLS (label 299840, exp 0, ttl 1)
(label 16, exp 0, [S], ttl 1)
IP (tos 0x0, ttl 1, id 36599, offset 0, flags [none], proto UDP (17), length 40)
10.1.1.2.36595 > 10.2.2.2.33437: UDP, length 12
```

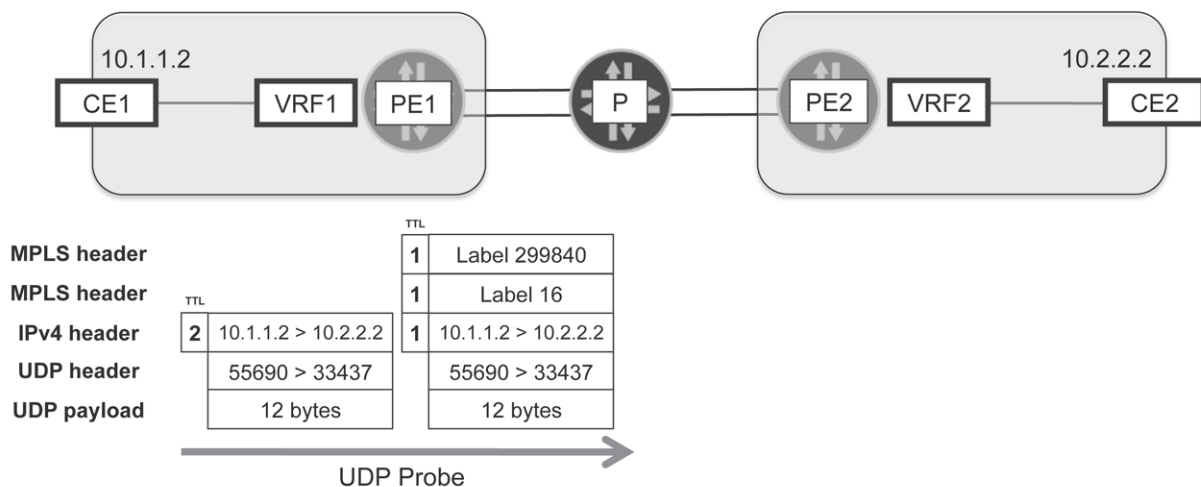


Figure 1.13 Life of a TTL=2 UDP Packet Probe

Action #3

CE1 sends an UDP probe with TTL=3, and PE2 replies with an ICMPv4 time exceeded packet as shown in Figure 1.14:

```
[timestamp] IP (tos 0x0, ttl 3, id 36602, offset 0, flags [none], proto UDP (17), length 40)
10.1.1.2.36595 > 10.2.2.2.33440: UDP, length 12
```

```
-----
[timestamp] MPLS (label 299840, exp 0, ttl 2)
(label 16, exp 0, [S], ttl 2)
IP (tos 0x0, ttl 2, id 36602, offset 0, flags [none], proto UDP (17), length 40)
10.1.1.2.36595 > 10.2.2.2.33440: UDP, length 12
```

```
-----
[timestamp] MPLS (label 16, exp 0, [S], ttl 1)
IP (tos 0x0, ttl 2, id 36602, offset 0, flags [none], proto UDP (17), length 40)
10.1.1.2.36595 > 10.2.2.2.33440: UDP, length 12
```

```
=====
[timestamp] MPLS (label 299872, exp 0, ttl 255)
(label 16, exp 0, [S], ttl 255)
IP (tos 0x0, ttl 255, id 0, offset 0, flags [none], proto ICMP (1), length 56)
10.2.2.1 > 10.1.1.2: ICMP time exceeded in-transit, length 36
IP (tos 0x0, ttl 1, id 36602, offset 0, flags [none], proto UDP (17), length 40)
10.1.1.2.36595 > 10.2.2.2.33440: UDP, length 12
```

```
-----
[timestamp] MPLS (label 16, exp 0, [S], ttl 254)
IP (tos 0x0, ttl 255, id 0, offset 0, flags [none], proto ICMP (1), length 56)
10.2.2.1 > 10.1.1.2: ICMP time exceeded in-transit, length 36
IP (tos 0x0, ttl 1, id 36602, offset 0, flags [none], proto UDP (17), length 40)
10.1.1.2.36595 > 10.2.2.2.33440: UDP, length 12
```



```
[timestamp] IP (tos 0x0, ttl 253, id 0, offset 0, flags [none], proto ICMP (1), length 56)
10.2.2.1 > 10.1.1.2: ICMP time exceeded in-transit, length 36
    IP (tos 0x0, ttl 1, id 36602, offset 0, flags [none], proto UDP (17), length 40)
10.1.1.2.36595 > 10.2.2.2.33440: UDP, length 12
```

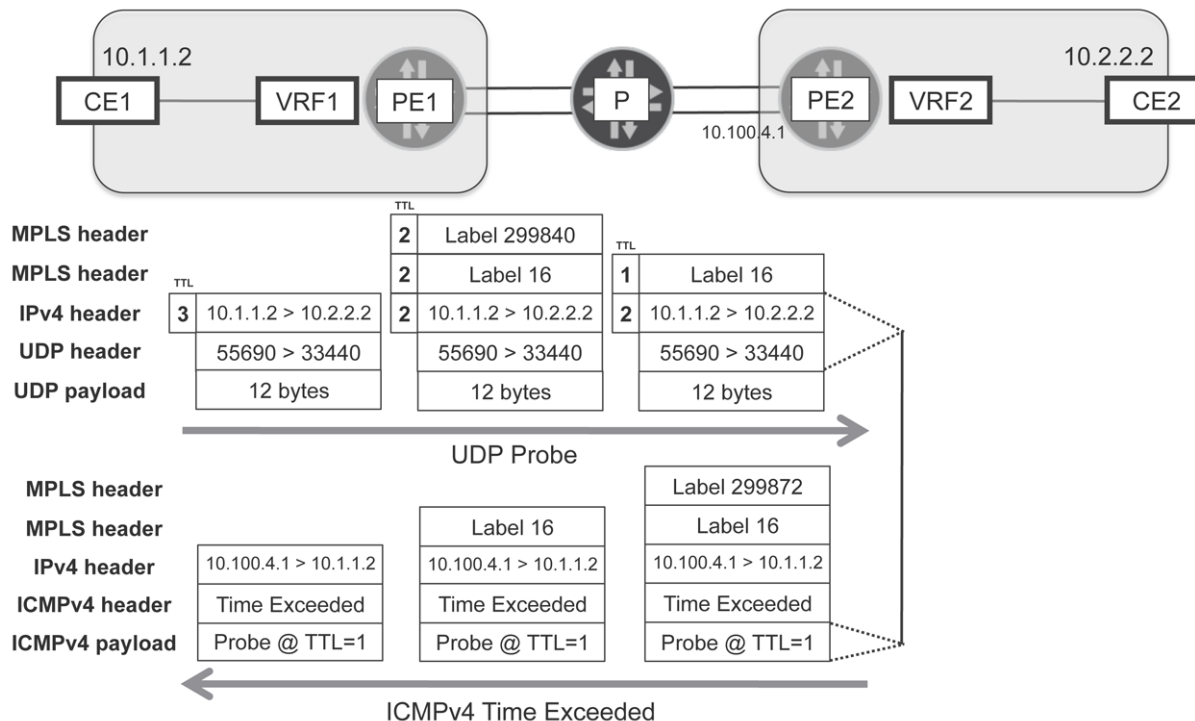


Figure 1.14 Life of a TTL=3 UDP Packet Probe

Action #4

CE1 sends an UDP probe with TTL=4, and CE2 replies with an ICMPv4 port unreachable packet, as shown in Figure 1.15:

```
[timestamp] IP (tos 0x0, ttl 4, id 36605, offset 0, flags [none], proto UDP (17), length 40)
10.1.1.2.36595 > 10.2.2.2.33443: UDP, length 12
-----
[timestamp] MPLS (label 299840, exp 0, ttl 3)
(label 16, exp 0, [S], ttl 3)
IP (tos 0x0, ttl 3, id 36605, offset 0, flags [none], proto UDP (17), length 40)
10.1.1.2.36595 > 10.2.2.2.33443: UDP, length 12
-----
[timestamp] MPLS (label 16, exp 0, [S], ttl 2)
IP (tos 0x0, ttl 3, id 36605, offset 0, flags [none], proto UDP (17), length 40)
10.1.1.2.36595 > 10.2.2.2.33443: UDP, length 12
-----
[timestamp] IP (tos 0x0, ttl 1, id 36605, offset 0, flags [none], proto UDP (17), length 40)
10.1.1.2.36595 > 10.2.2.2.33443: UDP, length 12
=====
[timestamp] IP (tos 0x0, ttl 255, id 27376, offset 0, flags [DF], proto ICMP (1), length 56)
10.2.2.2 > 10.1.1.2: ICMP 10.2.2.2 udp port 33443 unreachable, length 36
    IP (tos 0x0, ttl 1, id 36605, offset 0, flags [none], proto UDP (17), length 40)
10.1.1.2.36595 > 10.2.2.2.33443: UDP, length 12
-----
```

```

[timestamp] MPLS (label 299872, exp 0, ttl 254)
      (label 16, exp 0, [S], ttl 254)
      IP (tos 0x0, ttl 254, id 27376, offset 0, flags [DF], proto ICMP (1), length 56)
10.2.2.2 > 10.1.1.2: ICMP 10.2.2.2 udp port 33443 unreachable, length 36
      IP (tos 0x0, ttl 1, id 36605, offset 0, flags [none], proto UDP (17), length 40)
10.1.1.2.36595 > 10.2.2.2.33443: UDP, length 12
-----
[timestamp] MPLS (label 16, exp 0, [S], ttl 253)
      IP (tos 0x0, ttl 254, id 27376, offset 0, flags [DF], proto ICMP (1), length 56)
10.2.2.2 > 10.1.1.2: ICMP 10.2.2.2 udp port 33443 unreachable, length 36
      IP (tos 0x0, ttl 1, id 36605, offset 0, flags [none], proto UDP (17), length 40)
10.1.1.2.36595 > 10.2.2.2.33443: UDP, length 12
-----
[timestamp] IP (tos 0x0, ttl 252, id 27376, offset 0, flags [DF], proto ICMP (1), length 56)
10.2.2.2 > 10.1.1.2: ICMP 10.2.2.2 udp port 33443 unreachable, length 36
      IP (tos 0x0, ttl 1, id 36605, offset 0, flags [none], proto UDP (17), length 40)
10.1.1.2.36595 > 10.2.2.2.33443: UDP, length 12

```

ALERT! These are just two packets!

Look closely at the ICMPv4 payload with a graphical analyzer like Wireshark, then match the port unreachable messages to Table 1.1.

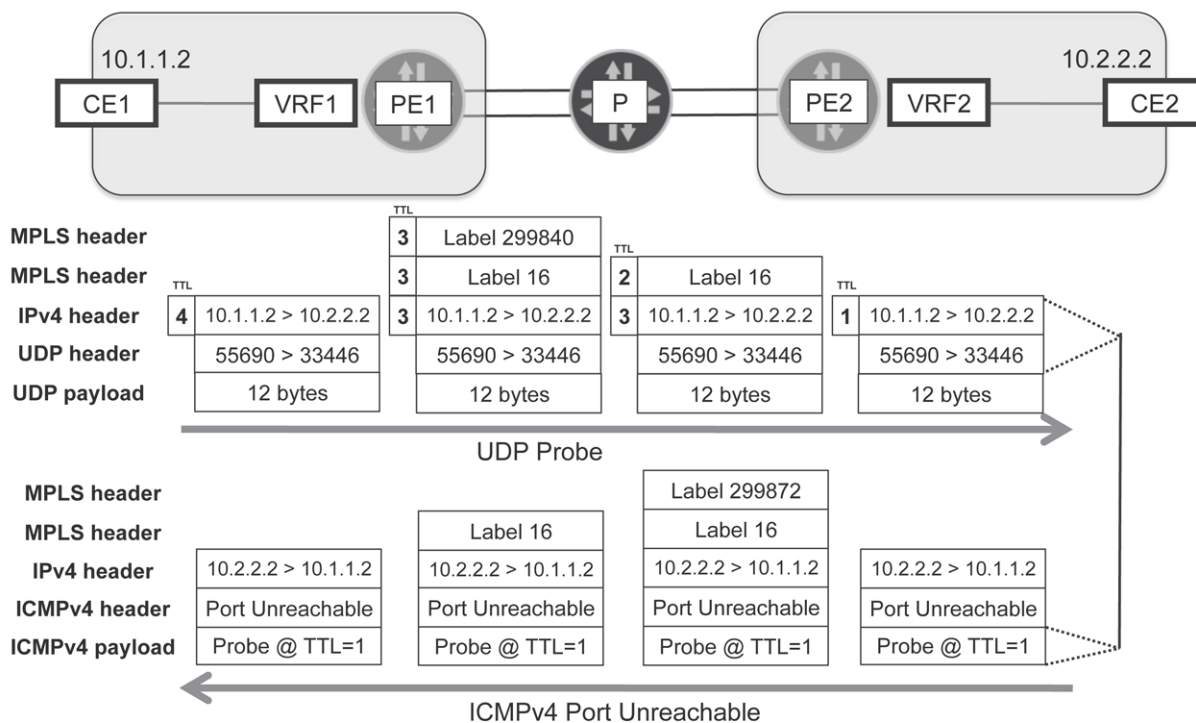


Figure 1.15 Life of a TTL=4 UDP Packet Probe

Try It Yourself: traceroute in MPLS Networks

There are several ways to avoid this silent discard and hence get rid of the 2 * * * line in the traceroute output. Turn your *video camera* on, execute the following procedure, and interpret the different movies.

1. Execute at PE1:

```

user@PE1> configure
user@PE1# set protocols mpls no-propagate-ttl
user@PE1# commit and-quit
user@PE1> traceroute 10.2.2.2 routing-instance CE1 no-resolve
user@PE1# delete protocols mpls no-propagate-ttl
user@PE1# commit and-quit
user@PE1> traceroute 10.2.2.2 routing-instance CE1 no-resolve

```

2. Carefully execute the following procedure, paying attention to which router you execute each command at:

```

user@P> configure
user@P# set protocols mpls icmp-tunneling
user@P# commit and-quit

```

```

user@PE1> traceroute 10.2.2.2 routing-instance CE1 no-resolve

```

3. If PE2 supports tunnel services, load the following configuration change at PE2 (adapting the FPC/PIC slots to the actual hardware). Then traceroute from PE1 again:

```

user@PE2> configure
user@PE2# load patch terminal
[Type ^D at a new line to end input]
[edit chassis]
+ fpc 0 {
+   pic 0 {
+     tunnel-services;
+   }
+ }
[edit interfaces]
+ vt-0/0/0 {
+   unit 0 {
+     family inet;
+     family inet6;
+   }
+ }
[edit routing-instances VRF2]
+ interface vt-0/0/0.0;
[edit routing-instances VRF2]
- vrf-table-label;
^D
user@PE2# commit and-quit

```

```

user@PE1> traceroute 10.2.2.2 routing-instance CE1 no-resolve

```

```

user@PE2> configure
user@PE2# delete interfaces vt-0/0/0
user@PE2# delete routing-instances VRF2 interface vt-0/0/0.0
user@PE2# set routing-instances VRF2 vrf-table-label
user@PE2# commit and-quit

```

Try It Yourself: IPv6 traceroute

Execute from PE1:

```

user@PE1> traceroute fc00::2:2 routing-instance CE1 no-resolve

```

Map the time exceeded and port unreachable messages to Table 1.1. If you want to have visibility of the MPLS core, you need to do something at P. This is a tricky question. Can you guess what?

Table 1.1 ICMP Types and Codes (seen so far)

ICMPv4 (IP Protocol 1)			ICMPv6 (IPv6 Next Header 58)		
Type	Code	Description	Type	Code	Description
0	0	Echo Reply	129	0	Echo Reply
3	0	Destination / Net Unreachable	1	0	Route Unreachable
	3	Destination / Port Unreachable		4	Port Unreachable
	13	Destination Unreachable, Adm. Filtered		1	Unreachable, Adm. Prohibited
8	0	Echo (Request)	128	0	Echo Request
11	0	Time Exceeded	3	0	Time Exceeded
Not an ICMPv4 function (ARP)			135	0	Neighbor Solicitation
			136	0	Neighbor Discovery

MORE? Have a look at RFC 792 and 4443, the specifications for ICMPv4 and ICMPv6, respectively. They are all at <https://datatracker.ietf.org/>

Life of a MPLS Ping

The name *ping* is used for a wide range of tools that have little or nothing to do with ICMP, or even with IP. Just type `ping ?` at the command line and you will see some exotic options like `atm`, `clns`, `ethernet`, `mpls`, `vpls`. The Junos OS ping command is actually launching different FreeBSD binaries depending on the options and address specified: `/sbin/ping` (IPv4), `/sbin/ping6` (IPv6), `/usr/sbin/atmping`, `/usr/sbin/clnsping` (CLNS), `/sbin/ethping` (ethernet), `/usr/sbin/lsping` (MPLS & VPLS). There is also a translation between Junos OS ping options and FreeBSD command options. The only thing these different binaries have in common is the strategy of sending probes and expecting replies.

CAUTION Ethernet ping option requires Ethernet OAM configuration to work.

Let's have a look at the `mpls` option. It supports several protocol or service specific options, and a generic one: `lsp-end-point`. Its internal implementation is similar to the protocol-specific options.

MPLS ping is a nice tool that performs an in-band health check of a Label Switched Path. The logic behind this action is: "let's put a packet in this LSP and check whether it reaches its destination."

With the capture active at H, execute at PE1:

```
user@PE1> ping mpls lsp-end-point 10.111.2.2 count 1
!
--- lsping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
```

Since the capture is extensive, let's show each of the packets once: the request on its way from PE1 to P, and the reply on its way from PE2 to P. Please refer to Figure 1.16 and to Capture 1.13 for the whole movie.

```

<timestamp> MPLS (label 299824, exp 0, [S], ttl 255)
  IP (tos 0x0, ttl 1, id 54181, offset 0, flags [none], proto UDP (17), length 80, options
(RA))
  10.111.1.1.52885 > 127.0.0.1.3503:
    LSP-PINGv1, msg-type: MPLS Echo Request (1), length: 48
      reply-mode: Reply via an IPv4/IPv6 UDP packet (2)
      Return Code: No return code or return code contained in the Error Code TLV (0)
      Return Subcode: (0)
      Sender Handle: 0x00000000, Sequence: 1
      Sender Timestamp: -15:-6:-36.6569 Receiver Timestamp: no timestamp
      Target FEC Stack TLV (1), length: 12
        LDP IPv4 prefix subTLV (1), length: 5
          10.111.2.2/32
=====
<timestamp> IP (tos 0xc0, ttl 64, id 35947, offset 0, flags [none], proto UDP (17), length 60)
  10.111.2.2.3503 > 10.111.1.1.52885:
    LSP-PINGv1, msg-type: MPLS Echo Reply (2), length: 32
      reply-mode: Reply via an IPv4/IPv6 UDP packet (2)
      Return Code: Replying router is an egress for the FEC at stack depth 1 (3)
      Return Subcode: (1)
      Sender Handle: 0x00000000, Sequence: 1
      Sender Timestamp: -15:-6:-36.6569 Receiver Timestamp: no timestamp

```

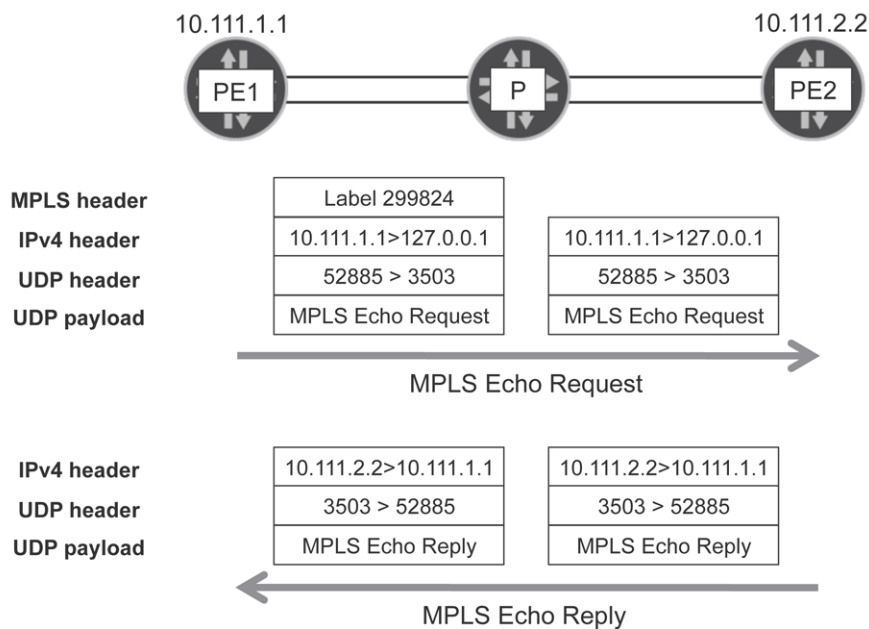


Figure 1.16 Life of a MPLS ping

The key aspects of MPLS ping are:

- MPLS echo requests are encapsulated in MPLS, however, MPLS echo replies are plain IP/UDP packets. This is consistent with the unidirectional nature of LSPs.
- MPLS echo requests are sent to 127.0.0.1, which in most Junos OS releases does not need to be explicitly configured. These packets carry RA (Route Alert) option in the IPv4 header. When PE2 receives this packet, it sees the RA

option so it decides to process it at the Control Plane level, regardless of the destination IPv4 address.

- MPLS echo requests are UDP packets with a well-known destination port 3503, identical to the source port of the MPLS echo replies.

MORE? Have a look at RFC 4379 to better understand the motivation and specifics of MPLS ping.

Answers to Try It Yourself Sections of Chapter 1

Try It Yourself: Managing ARP Cache Entries

The key is that CE1 has a static route pointing to 10.1.1.1, while VRF1 does not. The first command triggers a new ARP resolution: if it succeeds, nothing is done; if it fails, the entry is silently deleted, with the resulting impact in the forwarding table. The second command just deletes the unreferenced entry, requiring an IP packet at VRF1 towards 10.1.1.2 in order to trigger ARP again.

Try It Yourself: ping is a Return Trip

Even though the ICMP echo request reached its destination, CE2 does not have a route to send the ICMP echo replies back to CE1. You need to add this route in order to make ping work. Golden Rule #1: the return path is at least as important as the forward path!

```
user@PE2> configure
user@PE2# set routing-instances CE2 routing-options static route 0/0 next-hop 10.2.2.1
user@PE2# commit and-quit
```

Try to ping CE2 from CE1 again:

```
user@PE1> ping 10.2.2.2 routing-instance CE1 count 1 no-resolve
PING 10.2.2.2 (10.2.2.2): 56 data bytes
36 bytes from 10.2.2.2: Communication prohibited by filter
Vr HL TOS Len ID Flg off TTL Pro cks Src Dst
4 5 00 0054 9b9c 0 0000 3e 01 ca06 10.1.1.2 10.2.2.2
```

```
--- 10.2.2.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

Ping still fails, but at least it's getting an answer! Capture 1.14 contains the details. Review this chapter to find the way to correct this issue.

Try It Yourself: Taking the Multicast ping Further

When the destination address is multicast, ping uses Time To Live (TTL)=1 by default. You need to explicitly set a different value in order to take the ICMPv4 echo request all the way up to CE2:

```
user@PE1> ping 239.1.1.1 bypass-routing interface ge-1/0/0.101 ttl 10 count 1
PING 239.1.1.1 (239.1.1.1): 56 data bytes
```

```
--- 239.1.1.1 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

Try It Yourself: traceroute in MPLS Networks

1. Configuring no-propagate-ttl at PE1:

```
user@PE1> traceroute 10.2.2.2 routing-instance CE1 no-resolve
traceroute to 10.2.2.2 (10.2.2.2), 30 hops max, 40 byte packets
 1 10.1.1.1 0.493 ms 0.299 ms 0.297 ms
 2 10.2.2.1 0.392 ms 0.363 ms 0.368 ms
 3 10.2.2.2 0.534 ms 0.478 ms 0.473 ms
```

The end customer CE1 does not have any visibility of the MPLS backbone, which is seen as one hop. The movie in Capture 1.15 now has only three parts, as compared to the original movie which had four parts. Does action #2 (TTL=2) of your movie look like Figure 1.17? Well done!

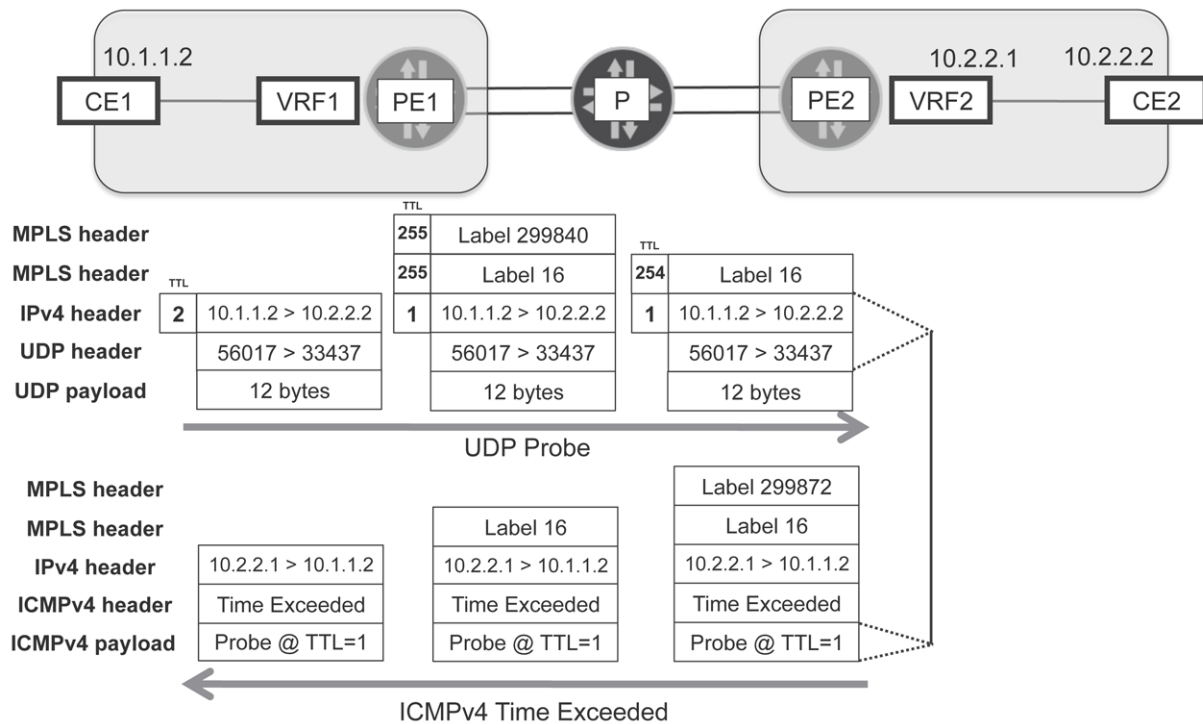


Figure 1.17 Effect of no-propagate-ttl at the Ingress PE

2. Configuring icmp-tunneling at P:

```
user@PE1> traceroute 10.2.2.2 routing-instance CE1 no-resolve
traceroute to 10.2.2.2 (10.2.2.2), 30 hops max, 40 byte packets
 1 10.1.1.1 0.460 ms 0.299 ms 0.296 ms
 2 10.100.1.2 0.647 ms 10.100.2.2 0.548 ms 10.100.1.2 0.561 ms
    MPLS Label=299840 CoS=0 TTL=1 S=0
    MPLS Label=16 CoS=0 TTL=1 S=1
 3 10.2.2.1 0.395 ms 0.370 ms 0.368 ms
 4 10.2.2.2 0.537 ms 0.482 ms 0.741 ms
```

NOTE PE1 should no longer have no-propagate-ttl configured!

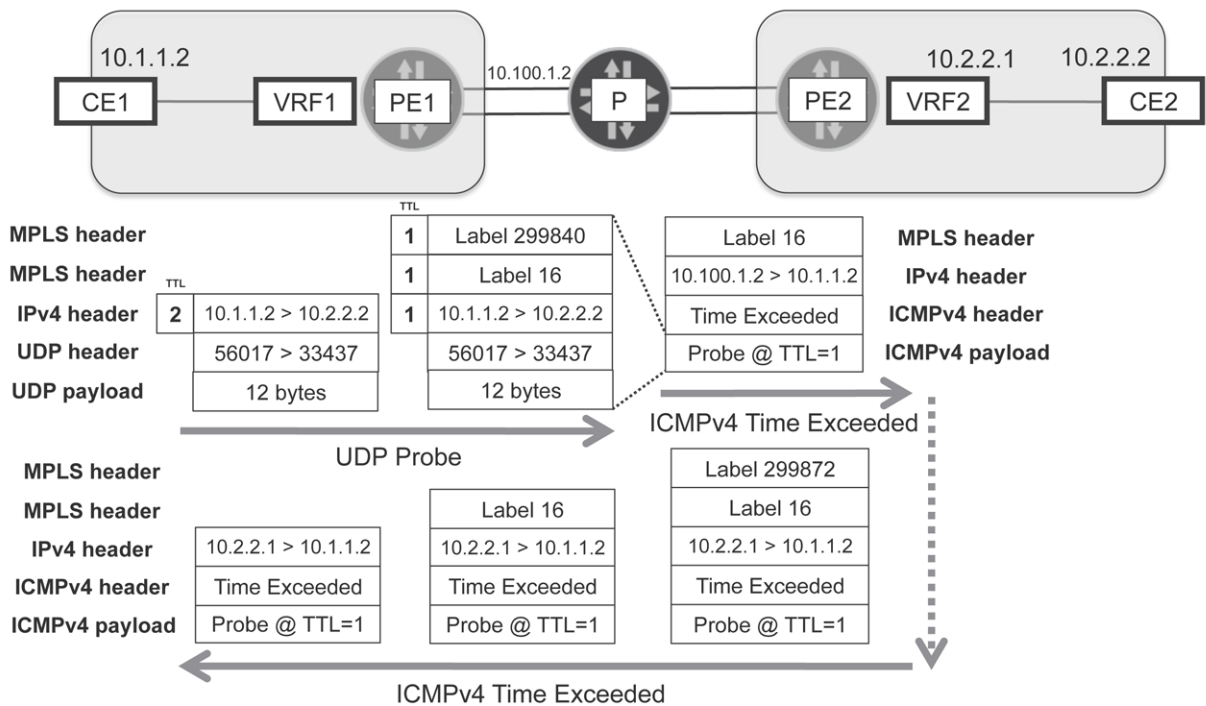


Figure 1.18 Effect of icmp-tunneling at the Transit P

With icmp-tunneling, the Round Trip Time (RTT) of all the packets whose TTL expires in the MPLS core is similar. Chapter 4 covers the ping latency topic in more detail.

MORE? Original MPLS headers are encoded as MPLS extensions in the tunneled TTL Exceeded packet. These extensions are defined in RFC 4950. Check this with a protocol analyzer on Capture 1.16.

3. With icmp-tunneling still configured at P, then moving VRF2 at PE2 to tunnel mode:

```
user@PE1> traceroute 10.2.2.2 no-resolve routing-instance CE1
traceroute to 10.2.2.2 (10.2.2.2), 30 hops max, 40 byte packets
 1 10.1.1.1 0.477 ms 0.302 ms 0.299 ms
 2 10.100.1.2 0.638 ms 0.546 ms 0.543 ms
    MPLS Label=299840 CoS=0 TTL=1 S=0
    MPLS Label=303088 CoS=0 TTL=1 S=1
 3 10.100.4.1 0.536 ms 0.456 ms 10.100.3.1 0.444 ms
    MPLS Label=303088 CoS=0 TTL=1 S=1
 4 10.2.2.1 0.379 ms 0.368 ms 0.374 ms
 5 10.2.2.2 0.503 ms 0.474 ms 0.470 ms
```

The TTL=4/5 packets need to loop in the vt- interface, hence the extra IP hop displayed. Check the details at Capture 1.17.

TRY THIS Execute ping 10.2.2.2 routing-instance CE1 count 1 ttl <value>, with an increasing value starting with TTL=1.

Try It Yourself: IPv6 traceroute

You need to enable IPv6 at the P core links:

```
user@P> configure
user@P# set interfaces ge-2/3/0.111 family inet6 address fc00::100:1:2/112
user@P# set interfaces ge-2/3/0.113 family inet6 address fc00::100:3:2/112
user@P# set interfaces xe-2/0/0.112 family inet6 address fc00::100:2:2/112
user@P# set interfaces xe-2/0/0.114 family inet6 address fc00::100:4:2/112
user@P# commit and-quit
```

This effectively enables the icmp-tunneling mechanism for IPv6. Check the details at Capture 1.18, taken with VRF2 back to vrf-table-label mode:

```
user@PE1> traceroute fc00::2:2 routing-instance CE1 no-resolve
traceroute6 to fc00::2:2 (fc00::2:2) from fc00::1:2, 64 hops max, 12 byte packets
 1 fc00::1:1 7.978 ms 0.385 ms 0.332 ms
 2 fc00::100:1:2 0.639 ms 0.562 ms 0.554 ms
   MPLS Label=299824 CoS=0 TTL=1 S=0
   MPLS Label=16 CoS=0 TTL=1 S=1
 3 fc00::2:1 0.431 ms 0.519 ms 0.385 ms
 4 fc00::2:2 0.613 ms 0.629 ms 0.505 ms
```

Don't worry if you see different MPLS labels, they can change upon routing events in a network.

Chapter 2

Tracking and Influencing the Path of a Packet

<i>Ping to a Local Address</i>	<i>38</i>
<i>Ping to a Neighbor (One Hop IPv4)</i>	<i>39</i>
<i>Ping from PE to PE (Multihop IPv4)</i>	<i>43</i>
<i>Ping from CE to CE (Multihop IPv4/MPLS)</i>	<i>49</i>
<i>Traceroute from CE to CE (Multihop IPv4/MPLS)</i>	<i>52</i>
<i>Steering Traffic on a Specific Multihop Path.</i>	<i>54</i>
<i>Conclusions</i>	<i>57</i>
<i>Answers to Try It Yourself Sections of Chapter 2.</i>	<i>58</i>

There are a lot of misconceptions about ping and traceroute with respect to their most frequent applications: network operation and troubleshooting. Here is a brief preview of some of the myths that this chapter will address:

- Myth #1: *Traceroute tells you hop-by-hop the path that ping would use to reach its destination.* But, why would ping and traceroute follow the same path? Many times they don't!
- Myth #2: *Tcpdump on a Junos OS router captures all the control traffic sent and received on an interface.*

Get ready for a quick tour of classic, but hopefully surprising, ping scenarios!

The *video* captures in Chapter 1 show a packet at different points in the network. However, the ethernet headers were stripped from the view and there is no information on the actual path being followed. When going from PE1 to P, did the ICMP echo request go via VLAN 111 or 112? You only know how it looked at layer 3 and above, but not what logical link it was transiting.

By using tcpdump option -e, or decoding the capture with Wireshark, you can see the actual ethernet headers and precisely describe the path taken by the packet. This kind of analysis can become a bit laborious when more packets or flows are involved, so let's take a different approach.

NOTE Although traceroute may look like a good fit here, let's stick to ping for the moment.

Ping to a Local Address

With the capture started at H (terminal #0), open two additional terminals, each with a different session at PE1. At terminal #1, leave running:

```
user@PE1> monitor traffic interface ge-1/0/0.111 size 2000 no-resolve matching icmp
```

As you know, this launches tcpdump in the background. At terminal #2, execute:

```
user@PE1> ping 10.100.1.1
PING 10.100.1.1 (10.100.1.1): 56 data bytes
64 bytes from 10.100.1.1: icmp_seq=0 ttl=64 time=0.138 ms
64 bytes from 10.100.1.1: icmp_seq=1 ttl=64 time=0.087 ms
64 bytes from 10.100.1.1: icmp_seq=2 ttl=64 time=0.087 ms
^C
--- 10.100.1.1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.087/0.104/0.138/0.024 ms
```

No ICMPv4 packets are captured at H, which is expected. But do you see any such packets at terminal #1? You shouldn't see any here, either, since as shown in Figure 2.1, the packets never cross the line that separates the Routing Engine from the other Control Plane components of the router. And it's precisely at this border where tcpdump at terminal #1 is capturing.

TIP From a troubleshooting perspective, it's important to note that if you have bad hardware connections between the Routing Engine and the Line Cards, or bad hardware connections within the Line Cards themselves, or if you have a physical port in a bad state, you will never catch these failures by just pinging the local address of an interface at the Line Card (unless the interface is down).

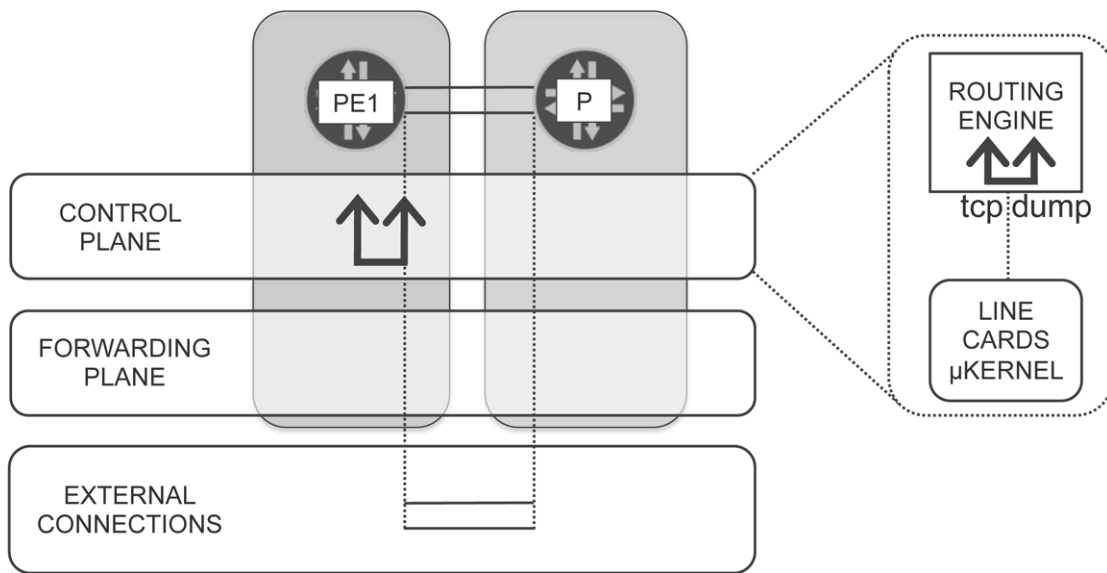


Figure 2.1 Forwarding Path of a Local ping

Ping to a Neighbor (One Hop IPv4)

This setup has `load-balance per-packet` configured in all routers, as is the case in the vast majority of production networks. Actually, that configuration knob translates to `per-flow` for transit traffic; you will see it in the two-hop ping scenario, covered later in this chapter.

Load Balancing at the Control Plane

Close terminals #1 and #2, and open two new terminals, each with a different CLI session at P.

At terminal #1, leave running:

```
user@P> monitor traffic interface ge-2/3/0.111 size 2000 no-resolve matching icmp
```

At terminal #2, leave running:

```
user@P> monitor traffic interface xe-2/0/0.112 size 2000 no-resolve matching icmp
```

Open a new terminal #3 in PE1 and ping P's 100.0 address:

```
user@PE1> ping 10.111.11.11 rapid count 4
```

Both PE1 and P are treating these packets at the Control Plane level, as shown in Figure 2.2. When the `load-balance per-packet` knob is configured, the Control Plane uses a round robin algorithm to load balance the outgoing packets: two packets out to link #1, then two packets out to link #2, etc.

CAUTION

If the latency of both paths is different, then the packet order of the flow is not guaranteed at the destination. To avoid this, you can define static routes towards management systems or collectors using the `qualified-next-hop` knob.

This decision by the Control Plane is respected by the Forwarding Plane components, which just use the next hop dictated by the Control Plane. Note that both routers are also originating other control traffic (linked to routing protocols), so it could well be that the ICMPv4 echo requests and replies in your test are not evenly load balanced, but since you are using the rapid option with little time between packets, you should likely see an even distribution like this one:

```
=====TERMINAL #1=====
<timestamp> In IP 10.111.1.1 > 10.111.11.11: ICMP echo request, id 13147, seq 1
<timestamp> In IP 10.111.1.1 > 10.111.11.11: ICMP echo request, id 13147, seq 2
<timestamp> Out IP 10.111.11.11 > 10.111.1.1: ICMP echo reply, id 13147, seq 2
<timestamp> Out IP 10.111.11.11 > 10.111.1.1: ICMP echo reply, id 13147, seq 3

=====TERMINAL #2=====
<timestamp> In IP 10.111.1.1 > 10.111.11.11: ICMP echo request, id 13147, seq 0
<timestamp> Out IP 10.111.11.11 > 10.111.1.1: ICMP echo reply, id 13147, seq 0
<timestamp> Out IP 10.111.11.11 > 10.111.1.1: ICMP echo reply, id 13147, seq 1
<timestamp> In IP 10.111.1.1 > 10.111.11.11: ICMP echo request, id 13147, seq 3
```

NOTE The *id* displayed in these captures is in the ICMP payload and identifies a ping flow, whose packets have an incremental *seq* number (reply echoes request's *seq* value).

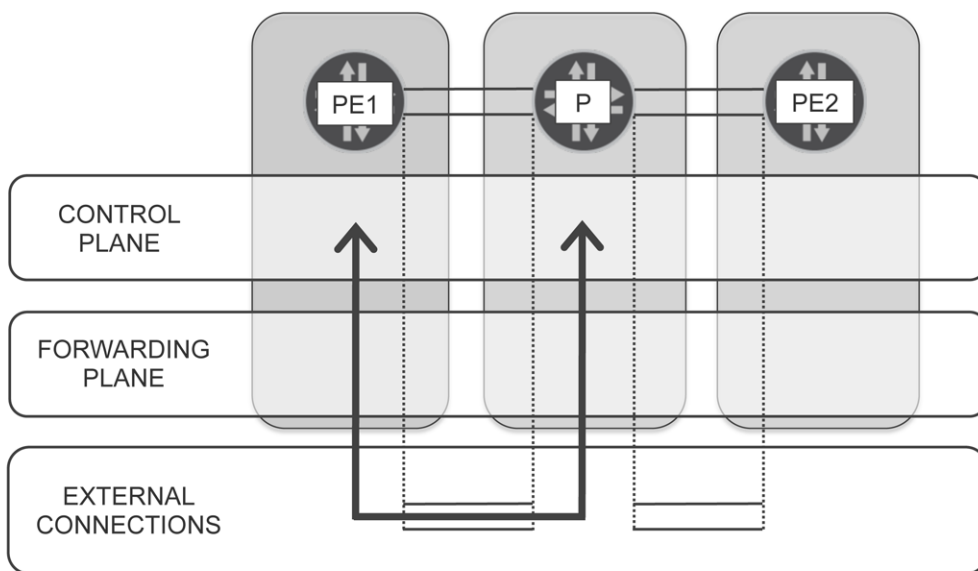


Figure 2.2 Forwarding Path of a One-hop Ping

Since you are not specifying any routing-instance option, Junos OS is picking the master instance, where the loopback interface `lo0.0` lies. As you can see, echo requests are sourced precisely from `lo0.0`.

TRY THIS What if `lo0.0` has more than one IPv4 address configured? Junos would pick the primary address, but if no primary is specified, what is the tie breaker?

Influencing the Path Followed by One-Hop Ping

Both the requests and the replies are load balanced across the equal cost links. Let's see how to influence that decision. Execute at terminal #3:

```
user@PE1> ping 10.111.11.11 interface ge-1/0/0.111 rapid count 4
```

This time, the echo requests are still load balanced, while the echo replies are all received at interface ge-1/0/0.111, can you spot why?

```
=====TERMINAL #1=====
<timestamp> In IP 10.100.1.1 > 10.111.11.11: ICMP echo request, id 13436, seq 0
<timestamp> Out IP 10.111.11.11 > 10.100.1.1: ICMP echo reply, id 13436, seq 0
<timestamp> Out IP 10.111.11.11 > 10.100.1.1: ICMP echo reply, id 13436, seq 1
<timestamp> Out IP 10.111.11.11 > 10.100.1.1: ICMP echo reply, id 13436, seq 2
<timestamp> In IP 10.100.1.1 > 10.111.11.11: ICMP echo request, id 13436, seq 3
<timestamp> Out IP 10.111.11.11 > 10.100.1.1: ICMP echo reply, id 13436, seq 3
```

```
=====TERMINAL #2=====
<timestamp> In IP 10.100.1.1 > 10.111.11.11: ICMP echo request, id 13436, seq 1
<timestamp> In IP 10.100.1.1 > 10.111.11.11: ICMP echo request, id 13436, seq 2
```

When used alone, the interface option simply changes the source IP address of the echo requests, very much like the source option (try it!). So you are actually attracting the echo replies on the specified interface, but the echo requests are still sent in a round robin fashion according to the forwarding table.

You need something else to send all the packets out of the same interface. So at terminal #3, execute:

```
user@PE1> show arp hostname 10.100.1.2
MAC Address      Address      Name      Interface      Flags
00:23:9c:9a:d3:8a 10.100.1.2    10.100.1.2 ge-1/0/0.111    none

user@PE1> ping 10.111.11.11 interface ge-1/0/0.111 bypass-routing mac-address 00:23:9c:9a:d3:8a
rapid count 4
PING 10.111.11.11 (10.111.11.11): 56 data bytes
!!!!
--- 10.111.11.11 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.483/0.560/0.778/0.126 ms
```

Now you are driving both the requests and the replies at the specified interface:

```
=====TERMINAL #1=====
<timestamp> In IP 10.100.1.1 > 10.111.11.11: ICMP echo request, id 13452, seq 0
<timestamp> Out IP 10.111.11.11 > 10.100.1.1: ICMP echo reply, id 13452, seq 0
<timestamp> In IP 10.100.1.1 > 10.111.11.11: ICMP echo request, id 13452, seq 1
<timestamp> Out IP 10.111.11.11 > 10.100.1.1: ICMP echo reply, id 13452, seq 1
<timestamp> In IP 10.100.1.1 > 10.111.11.11: ICMP echo request, id 13452, seq 2
<timestamp> Out IP 10.111.11.11 > 10.100.1.1: ICMP echo reply, id 13452, seq 2
<timestamp> In IP 10.100.1.1 > 10.111.11.11: ICMP echo request, id 13452, seq 3
<timestamp> Out IP 10.111.11.11 > 10.100.1.1: ICMP echo reply, id 13452, seq 3
```

```
=====TERMINAL #2=====
```

NOTE SONET interfaces also support the bypass-routing option, but they obviously do not require the mac-address one.

Default Source Address Assignment

Okay, previously you saw that PE1 sourced echo requests from its local 10.0.0 when pinging P's loopback. What if you ping a directly connected address?

```
user@PE1> ping 10.100.1.2 count 1
PING 10.100.1.2 (10.100.1.2): 56 data bytes
64 bytes from 10.100.1.2: icmp_seq=0 ttl=64 time=0.673 ms

--- 10.100.1.2 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.673/0.673/0.673/0.000 ms
```

```
=====TERMINAL #1=====
<timestamp> In IP 10.100.1.1 > 10.100.1.2: ICMP echo request, id 13484, seq 0
<timestamp> Out IP 10.100.1.2 > 10.100.1.1: ICMP echo reply, id 13484, seq 0

=====TERMINAL #2=====
```

NOTE In this case, Figure 2.2 is valid as well. When you ping a remote non-loopback interface, the echo requests also go up to the remote Routing Engine.

This time, echo requests are sourced from the local IPv4 address at ge-1/0/0.111. This is not strictly related to the fact that the destination address is directly connected. The real reason behind this behavior is that the route to 10.100.1.2 only has one next hop, as compared to the route to 10.111.11.11, which has two possible next hops:

```
user@PE1> show route 10.100.1.2 table inet.0

inet.0: 10 destinations, 10 routes (10 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.100.1.0/30      *[Direct/0] 22:24:46
                  > via ge-1/0/0.111

user@PE1> show route 10.111.11.11 table inet.0

inet.0: 10 destinations, 10 routes (10 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.111.11.11/32   *[IS-IS/18] 00:00:05, metric 10
                  to 10.100.1.2 via ge-1/0/0.111
                  > to 10.100.2.2 via ge-1/0/1.112
```

TRY THIS At PE1, temporarily make the route 10.111.11.11 point only to 10.100.1.2. You can achieve this by disabling ge-1/0/1.112 or by changing its IS-IS metric. Check the source address of the echo requests sent with ping 10.111.11.11, and it should now be 10.100.1.1!

You can change this default behavior with a frequently used configuration knob called `default-address-selection`, which, ironically, *is not on* by default!

```
user@PE1> configure
user@PE1# set system default-address-selection
user@PE1# commit and-quit
commit complete
Exiting configuration mode
```



```

user@PE1> ping 10.100.1.2 count 1
PING 10.100.1.2 (10.100.1.2): 56 data bytes
64 bytes from 10.100.1.2: icmp_seq=0 ttl=64 time=0.673 ms

--- 10.100.1.2 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.673/0.673/0.673/0.000 ms

=====TERMINAL #1=====
22:30:41.630231 In IP 10.111.1.1 > 10.100.1.2: ICMP echo request, id 15010, seq 0

=====TERMINAL #1 OR #2=====
22:30:41.630269 Out IP 10.100.1.2 > 10.111.1.1: ICMP echo reply, id 15010, seq 0

```

ALERT! It is easy to believe that a ping to a directly connected IPv4 address does not require any routing protocol. However, if the `default-address-selection` knob is configured and the destination does not have a route towards the local `100.0`, the ping will fail. Admittedly, this is odd behavior.

Ping from PE to PE (Multihop IPv4)

At terminal #3, connected to PE1, run:

```
user@PE1> ping 10.111.2.2 rapid count 4
```

You don't see anything in terminal #1 and #2 (at P), right? The reason for this is illustrated in Figure 2.3. Junos OS `tcpdump` can only capture control traffic that is sent or received by the Routing Engine. This is a limit for funny lab tests, but that's why you bought a *video camera*! On the other hand, it is a safety protection measure: it would be too easy to collapse the router Control Plane otherwise.

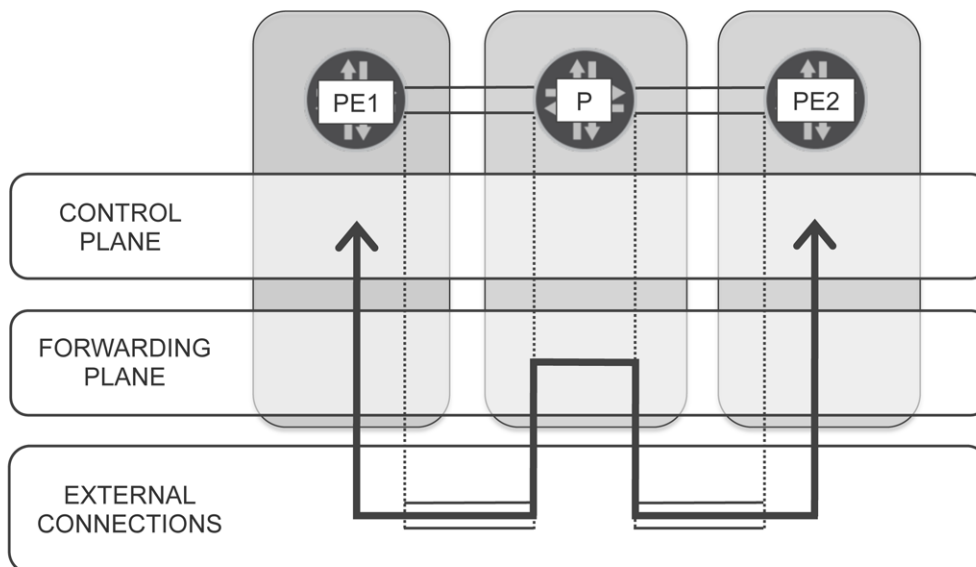


Figure 2.3 Forwarding Path of a Two-hop PE-to-PE Ping

Tracking and Counting Transit Packets

You need to use a different method to track the path followed by ping, and it's not traceroute! Open a new terminal #4 and connect it to P, then execute:

```
user@P> configure
user@P# edit firewall family inet filter ICMP
user@P# set interface-specific
user@P# set term ECHO-REQUEST from protocol icmp
user@P# set term ECHO-REQUEST from icmp-type echo-request
user@P# set term ECHO-REQUEST then count icmp-echo-request
user@P# set term ECHO-REPLY from protocol icmp
user@P# set term ECHO-REPLY from icmp-type echo-reply
user@P# set term ECHO-REPLY then count icmp-echo-reply
user@P# set term TIME-EXCEEDED from protocol icmp
user@P# set term TIME-EXCEEDED from icmp-type time-exceeded
user@P# set term TIME-EXCEEDED then count icmp-time-exceeded
user@P# set term UNREACHABLE from protocol icmp
user@P# set term UNREACHABLE from icmp-type unreachable
user@P# set term UNREACHABLE then count icmp-unreachable
user@P# set term DEFAULT then accept
user@P# top
user@P# set interfaces ge-2/3/0 unit 111 family inet filter input ICMP
user@P# set interfaces ge-2/3/0 unit 111 family inet filter output ICMP
user@P# set interfaces ge-2/3/0 unit 113 family inet filter input ICMP
user@P# set interfaces ge-2/3/0 unit 113 family inet filter output ICMP
user@P# set interfaces xe-2/0/0 unit 112 family inet filter output ICMP
user@P# set interfaces xe-2/0/0 unit 112 family inet filter input ICMP
user@P# set interfaces xe-2/0/0 unit 114 family inet filter output ICMP
user@P# set interfaces xe-2/0/0 unit 114 family inet filter input ICMP
user@P# commit and-quit
```

NOTE Other networking OSes (by default), and Junos OS (with certain non-default configurations) can encapsulate control IP traffic in MPLS. Default Junos OS behavior is shown here: control IP traffic is sent unlabeled.

Try It Yourself: Matching ICMP Traffic with a Firewall Filter

Temporarily modify the firewall filter definition as follows:

```
user@P> configure
user@P# deactivate firewall family inet filter ICMP term ECHO-REPLY from protocol
user@P# commit and-quit
```

Now, just wait a couple of minutes without running any ping command, and then check the counters:

```
user@P> show firewall | match ^icmp.*i | except " 0"
icmp-echo-reply-ge-2/3/0.113-i          317          5
icmp-echo-reply-xe-2/0/0.114-i          298          5
```

The counters are increasing! Why is that? Answers are at the end of the chapter.

Load Balancing at the Forwarding Plane

Clean the counters at P (terminal #4):

```
user@P> clear firewall all
```

Launch a 100-packet ping from PE1 to PE2 (terminal #3), specifying ge-1/0/0.111 as the next hop interface:

```
user@PE1> ping 10.111.2.2 interface ge-1/0/0.111 bypass-routing mac-address 00:23:9c:9a:d3:8a
rapid count 100
PING 10.111.2.2 (10.111.2.2): 56 data bytes
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!
--- 10.111.2.2 ping statistics ---
100 packets transmitted, 100 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.419/0.545/5.167/0.670 ms
```

Now, check the input (-i) and output (-o) firewall filter counters at P (terminal #4), looking for echo requests:

```
user@P> show firewall | match request | except " 0"
icmp-echo-request-ge-2/3/0.111-i          8400          100
icmp-echo-request-ge-2/3/0.113-o          8400          100
```

TRY THIS The first numeric column in the output represents bytes and the second represents packets. This means P receives and sends 84-byte packets. Can you explain why? Use a packet capture and write down the size of all headers.

P only uses one of its links to PE2 to forward the echo requests of your ping. This proves that the Forwarding Plane uses a different load-balancing algorithm from the one used by the Control Plane. In this case, the `load-balance per-packet` knob is translated as `per-flow`. If there are several candidate next-hops with equal cost to the destination, the ingress Packet Forwarding Engine (PFE) of P picks one of them by calculating a hash value using certain – some fixed, some configurable – packet properties. From P's perspective, all packets are received from the same interface (ge-2/3/0.111) and share common properties: same IPv4 protocol, same source and destination address, and no port information. As a result, the hash value is the same for all the packets of the stream.

NOTE The hash value depends on multiple factors including the type of Packet Forwarding Engine, so it is possible that you can get xe-2/0/0.114 as the next hop. However, one thing is certain, you should see one and only one output interface.

Now, let's clear counters at P again and launch a 100-packet ping from PE1 to PE2, but this time don't specify the next hop interface:

```
user@P> clear firewall all

user@PE1> ping 10.111.2.2 rapid count 100
PING 10.111.2.2 (10.111.2.2): 56 data bytes
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!
--- 10.111.2.2 ping statistics ---
100 packets transmitted, 100 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.404/0.428/0.698/0.038 ms
```

Now, check the firewall filter counters at P, looking for echo requests:

```
user@P> show firewall | match request | except " 0"
icmp-echo-request-ge-2/3/0.111-i          4200          50
icmp-echo-request-xe-2/0/0.112-i          4200          50
icmp-echo-request-xe-2/0/0.114-o          8400         100
```

PE1 is generating the echo requests from its Control Plane, so it uses a round robin algorithm to spread them across the two links to P. On the other hand, in this example, P only picks one next hop, so it must be calculating the same hash value for all the packets in the echo request stream. There are two possible reasons for this: either the ingress PFE is ignoring the input interface for hash computation (the default for Trio), or it is taking it into account but the mathematical result turns out to be the same. It's also fairly possible that you get different hash results anyway, and hence a different output interface. If you see P effectively load balancing in your setup across two interfaces, then your PFEs are definitely taking the input interface into account for the hash.

Now, check the firewall filter counters at P, looking for echo replies:

```
user@P> show firewall | match request | except " 0"
icmp-echo-reply-ge-2/3/0.113-i          4200          50
icmp-echo-reply-xe-2/0/0.112-o          8400         100
icmp-echo-reply-xe-2/0/0.114-i          4200          50
```

The logic here is the same as for the echo requests.

TRY THIS

If you are using Trio PFEs in your setup, configure `set forwarding-options enhanced-hash-key family inet incoming-interface-index` and watch the outcome. If you are lucky enough, P will balance the echo requests, or the echo replies, or both. If you are unlucky, then the hash results are not changing because two flows are not statistically significant to achieve effective load balancing.

The Record-Route Option

Give the *record-route* option a try (isn't its name appealing?):

```
user@PE1> ping 10.111.2.2 record-route count 1 no-resolve
PING 10.111.2.2 (10.111.2.2): 56 data bytes
64 bytes from 10.111.2.2: icmp_seq=0 ttl=63 time=27.855 ms
RR:      10.111.11.11
         10.111.2.2
         10.111.11.11
         10.111.1.1

--- 10.111.2.2 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 27.855/27.855/27.855/0.000 ms
```

With this option, each router in the path of the ICMP echo request, as well as the ICMP echo reply, stamps its own IP address *somewhere* in the packet. As a result, the output shows the complete two-way trip.

Let's see how these packets are forwarded. Clean firewall filter counters at P:

```
user@P> clear firewall all
```

Launch a 100-packet ping from PE1 to PE2, with the *record-route* option:

```
user@PE1> ping 10.111.2.2 rapid count 100 record-route no-resolve
PING 10.2.2.1 (10.2.2.1): 56 data bytes
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
--- 10.2.2.1 ping statistics ---
100 packets transmitted, 100 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.409/0.514/8.827/0.836 ms
```

Now, check the firewall filter counters at P:

```
user@P> show firewall | match request | except " 0"
icmp-echo-request-ge-2/3/0.111-i          6200          50
icmp-echo-request-ge-2/3/0.113-o          6200          50
icmp-echo-request-xe-2/0/0.112-i          6200          50
icmp-echo-request-xe-2/0/0.114-o          6200          50

user@P> show firewall | match reply | except " 0"
icmp-echo-reply-ge-2/3/0.111-o            6200          50
icmp-echo-reply-ge-2/3/0.113-i            6200          50
icmp-echo-reply-xe-2/0/0.112-o            6200          50
icmp-echo-reply-xe-2/0/0.114-i            6200          50
```

The traffic is perfectly balanced at each hop. And, if you keep your terminals #1 and #2 with tcpdump at P, you can see the ICMP packets in the captures!

The ICMP echo request and reply packets need special treatment in order to record the transit IP at each step, so the Control Plane is involved at each hop (as shown in Figure 2.4), which explains the round robin traffic distribution that you can see here. Also, the tcpdump captures at P prove that the packets go up and down the Routing Engine of the transit routers.

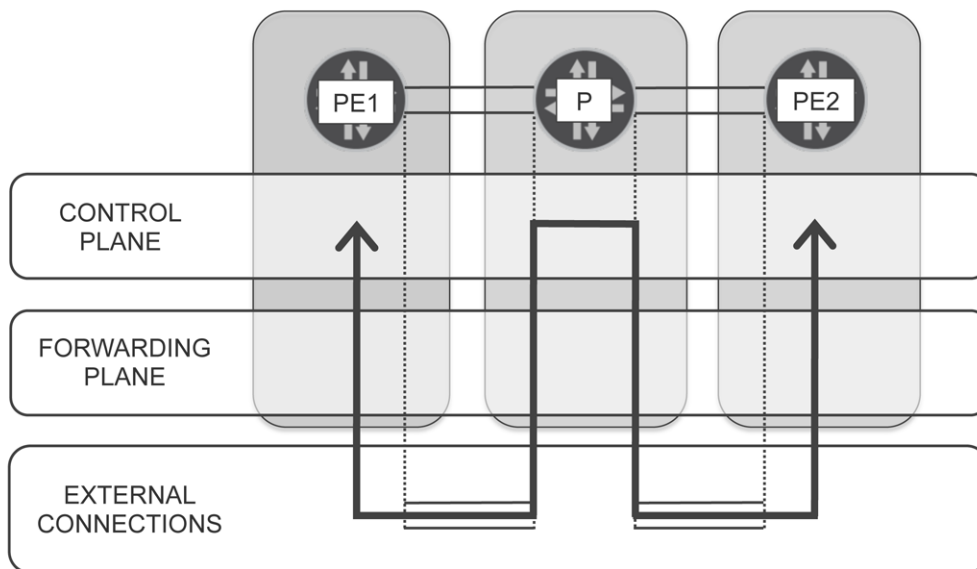


Figure 2.4 Effect of the record-route ping option

One interesting application of the *record-route* option is that, unlike traceroute, it shows both the forward and the return path, so it can be useful in spotting asymmetrical routing scenarios. Not only can you detect plain IGP metric asymmetries, you can also identify more complex interdomain routing patterns. For example, if the IP list shows that the Autonomous System (AS) path transited by the echo requests is different from the one followed by the echo replies (conveniently reversed), you can definitely assume routing is asymmetrical. Depending on the agreed EBGp policies, this may be completely expected, or alternatively, it may point to incorrect configuration or unexpected backdoors.

Try It Yourself: The Magic Behind record-route Option

How is it possible that a single packet gets processed by the Control Plane of all the transit routers? Turn on your *video camera* at H and see!

Traffic Polarization

You saw in the preceding example that in a loopback-to-loopback ping test from PE1 to PE2, echo requests are first load balanced between both of the PE1-P links, but then they all go to the same P-PE2 link because the hash result is the same. This behavior is called *traffic polarization*.

Suppose you have a topology similar to Figure 2.5. There are $16 = 2^4$ possibilities for a packet to go from PE1 to PE2.



Figure 2.5 Dual-link Core Topology

If you ping from PE1 to PE2 loopback-to-loopback:

- By default, echo requests explore only two of the 16 paths (~50% of the packets take path #1, ~50% take path #2). In the previous example, echo replies explored two paths, but these are not necessarily the reverse paths taken by the echo requests.
- If you use the *record-route* option, over time, all the 16 forward and the 16 reverse paths are explored. However, the forwarding path inside the transit routers is not the same: this is an important point, as you will see in the upcoming *Failure Scenarios* section.

For now, let's go to a different scenario. Let's have a look at the topology in Figure 2.6.

PE1 receives traffic from the CEs connected to it, and forwards half of the flows to each of the directly connected routers (P1 and P5). Let's focus on P1. All the packets that PE1 decides to forward to P1 result in the same hash calculation at P1. What if PE1 and P1 have the same hashing algorithm? P1 also calculates the same hash value to all of them, so it only uses one next-hop of the two it has available. As a result, the links with dotted lines in Figure 2.6 are not utilized for PE1-to-PE2 traffic.

The strategies necessary to address this design challenge are beyond the scope of this book, which focuses on ping, but it's important to mention this concept not only because end user traffic is typically polarized to some extent, but because ping is often *more* polarized than end user traffic since it has less randomization in its headers. This is just something to keep it in mind when interpreting ping test results.

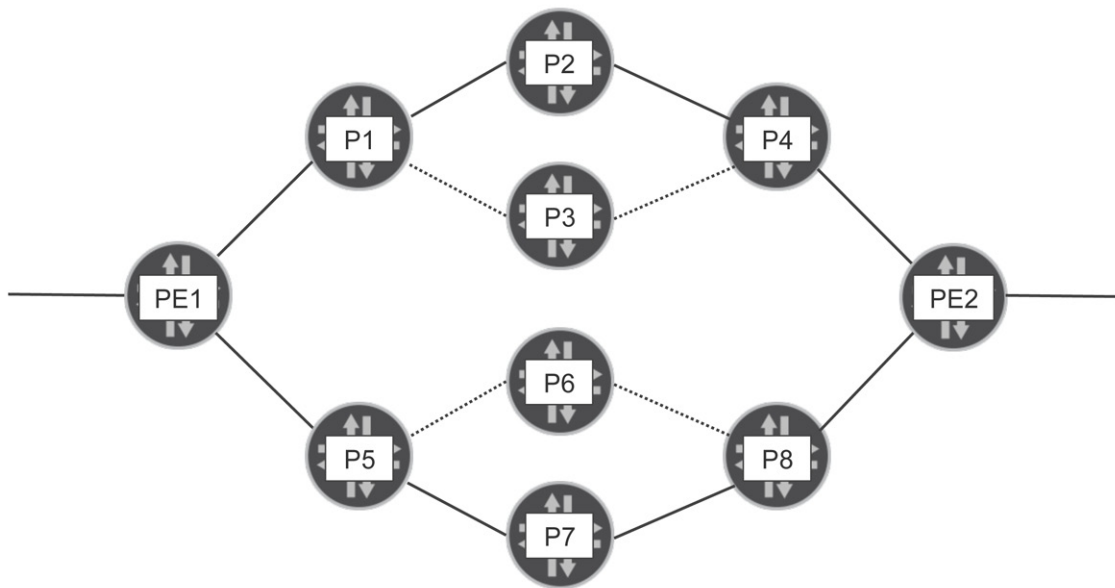


Figure 2.6 Traffic Polarization

Ping from CE to CE (Multihop IPv4/MPLS)

In this scenario, PE1 performs an IPv4->MPLS encapsulation of the ICMPv4 echo request, whereas PE2 does its MPLS->IPv4 decapsulation. The hash is calculated according to the properties of the packet as it enters the router. From PE1's perspective, the ICMPv4 echo request is an IPv4 packet, whereas it is a MPLS packet for PE2. The hash calculated for MPLS packets also takes into account some fields of the inner headers, in this case IPv4.

MORE? The fields taken into account for hash calculation by Trio PFEs are configurable at `[edit forwarding-options enhanced-hash-key family inet|inet6|mpls]` for IPv4, IPv6 and MPLS packets, respectively. In older PFEs, the hierarchy is `[edit forwarding-options hash-key family inet|inet6|mpls]`.

TIP Some of the pre-Trio platforms take the MPLS EXP bits into account by default to perform load balancing. If the configuration knob `set forwarding-options hash-key family mpls no-label-1-exp` is available, and not hidden, your router is one of them!

Clean the firewall filter counters at P:

```
user@P> clear firewall all
```

Launch a 100-packet ping from CE1 to CE2:

```
user@PE1> ping 10.2.2.2 routing-instance CE1 rapid count 100
PING 10.2.2.2 (10.2.2.2): 56 data bytes
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!
--- 10.2.2.2 ping statistics ---
100 packets transmitted, 100 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.463/0.809/21.974/2.408 ms
```

Now, check the input firewall filter counters at P:

```
user@P> show firewall | match request | except " 0"

user@P> show firewall | match reply | except " 0"

user@P>
```

Try It Yourself: CE-to-CE ping Tracking at P-router

Why aren't the packets being matched by the filters? Try to fix it (answers at the end of the chapter)!

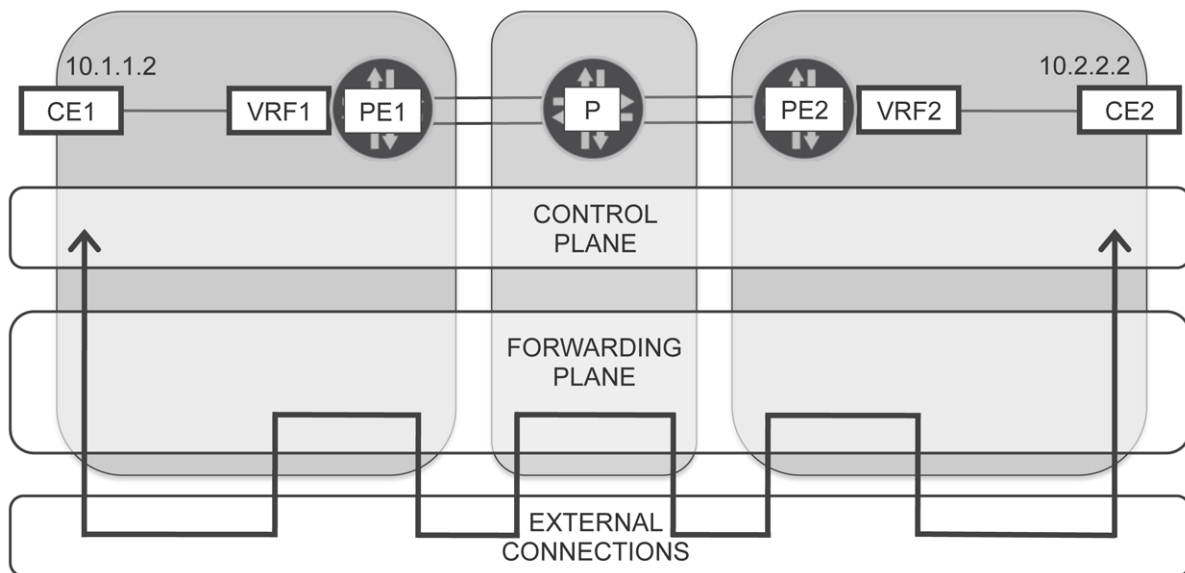


Figure 2.7 Forwarding Path of a Multihop CE-to-CE Ping

Once you get the right filters in place, launch a 100-packet ping from CE1 to CE2:

```
user@PE1> ping 10.2.2.2 routing-instance CE1 rapid count 100
PING 10.2.2.2 (10.2.2.2): 56 data bytes
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!
--- 10.2.2.2 ping statistics ---
100 packets transmitted, 100 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.460/0.680/5.655/0.940 ms
```

In order to see how PE1 and PE2 are balancing the ICMPv4 traffic out of their core uplinks, check the input firewall filter counters at P:

```
user@P> show firewall | match mpls-packets.*i
mpls-packets-ge-2/3/0.111-i          0          0
mpls-packets-ge-2/3/0.113-i          0          0
mpls-packets-xe-2/0/0.112-i        9200        100
mpls-packets-xe-2/0/0.114-i        9200        100
```


Finally, see how P load balances the MPLS-encapsulated ICMPv4 traffic by checking the output firewall filter counters at P:

```
user@P> show firewall | match mpls-packets.*o
mpls-packets-ge-2/3/0.111-o      8800      100
mpls-packets-ge-2/3/0.113-o        0         0
mpls-packets-xe-2/0/0.112-o        0         0
mpls-packets-xe-2/0/0.114-o      8800      100
```

Due to the forwarding path outlined in Figure 2.7, it's all about hashing now! You may see other interfaces chosen by the hashing algorithm, but there should definitely be the same degree of polarization as shown here.

TRY THIS

Can you explain the 92-byte and 88-byte packet size? The captures taken in Chapter 1 can help you.

If you use the `record-route` option, the forwarding path changes as shown in Figure 2.8. The traffic is evaluated at the Control Plane by PE1 and PE2, but not by P. The reason is that IPv4 options are not exposed in MPLS packets. As a result, PEs use round robin algorithm, while P uses a hash:

```
user@P> clear firewall all
```

```
user@PE1> ping 10.2.2.2 routing-instance CE1 rapid count 100 record-route no-resolve
PING 10.2.2.2 (10.2.2.2): 56 data bytes
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!
--- 10.2.2.2 ping statistics ---
100 packets transmitted, 100 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.460/0.680/5.655/0.940 ms
```

```
user@P> show firewall | match mpls-packets.*i
mpls-packets-ge-2/3/0.111-i      6600      50
mpls-packets-ge-2/3/0.113-i      6600      50
mpls-packets-xe-2/0/0.112-i      6600      50
mpls-packets-xe-2/0/0.114-i      6600      50
```

```
user@P> show firewall | match mpls-packets.*o
mpls-packets-ge-2/3/0.111-o      12800     100
mpls-packets-ge-2/3/0.113-o      12800     100
mpls-packets-xe-2/0/0.112-o        0         0
mpls-packets-xe-2/0/0.114-o        0         0
```

NOTE If you see that P is load balancing, then the input interface is likely being taken into account for the hash calculations.

Try It Yourself: Capturing MPLS-tagged Traffic with tcpdump

Ping packets with `record-route` option are processed by the Control Plane at the PEs. However, the command `monitor traffic interface <core-interface> matching icmp` wouldn't catch them. One alternative is to capture at the access interface PE-CE instead, and you also have the *video camera* at H. But what if you want to see the packets at the core interface, and at the router? Which `monitor traffic` options should you use?

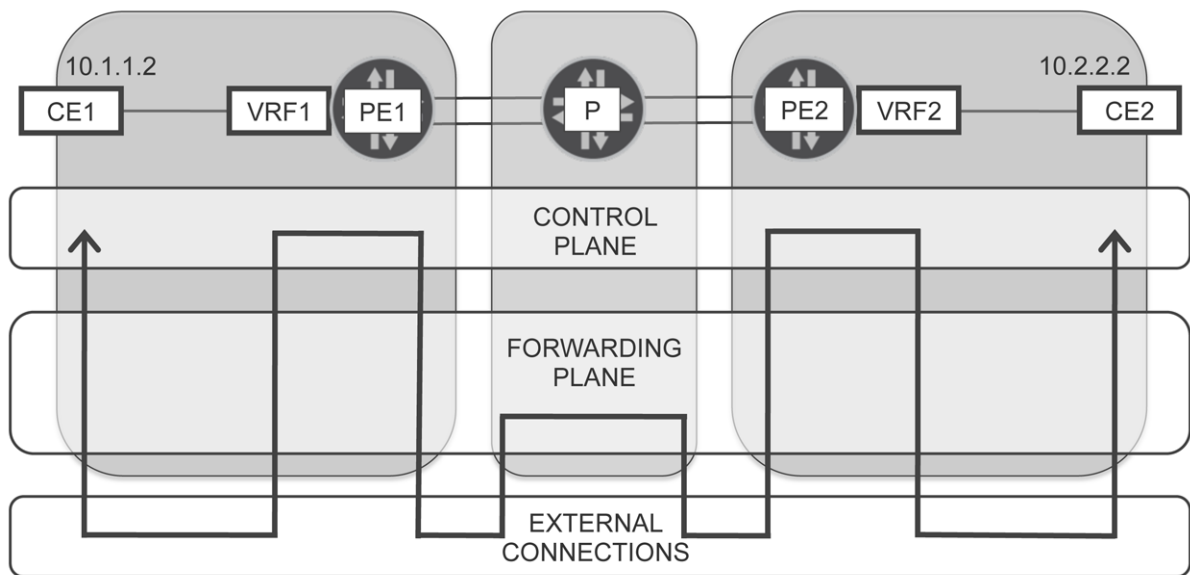


Figure 2.8 Forwarding Path of a Multihop CE-to-CE Ping with Record-route Option

Traceroute from CE to CE (Multihop IPv4/MPLS)

With [protocols mpls icmp-tunneling] configured at P, execute the following command at PE1 several times:

```
user@PE1> traceroute 10.2.2.2 routing-instance CE1 no-resolve
traceroute to 10.2.2.2 (10.2.2.2), 30 hops max, 40 byte packets
 1 10.1.1.1 0.520 ms 0.303 ms 0.295 ms
 2 10.100.1.2 5.624 ms 0.552 ms 0.537 ms
   MPLS Label=299904 CoS=0 TTL=1 S=0
   MPLS Label=16 CoS=0 TTL=1 S=1
 3 10.2.2.1 0.450 ms 0.357 ms 0.352 ms
 4 10.2.2.2 0.524 ms 0.455 ms 0.454 ms

user@PE1> traceroute 10.2.2.2 routing-instance CE1 no-resolve
traceroute to 10.2.2.2 (10.2.2.2), 30 hops max, 40 byte packets
 1 10.1.1.1 0.455 ms 0.306 ms 0.291 ms
 2 10.100.2.2 0.633 ms 4.121 ms 0.536 ms
   MPLS Label=299904 CoS=0 TTL=1 S=0
   MPLS Label=16 CoS=0 TTL=1 S=1
 3 10.2.2.1 0.406 ms 0.366 ms 0.361 ms
 4 10.2.2.2 0.529 ms 0.454 ms 0.475 ms
```

You can see that the good thing about traceroute is that it tells you the path upfront, but the logic behind it is a bit more complex than it might seem at first.

For example, you can see that the hop displayed at the first line is always the same, which is natural since there is only one connection CE1-PE1. On the other hand, the second line is changing from one packet to another. Let's see why.

As you know, each line of the traceroute output corresponds to an initial TTL value that starts from 1 and grows by 1 at each line. Figure 2.9 shows the forwarding path of the various packets sent from CE1, depending on the initial TTL value. You can see that the second and third hops are actually determined by the Forwarding Plane of PE1, and P, respectively.

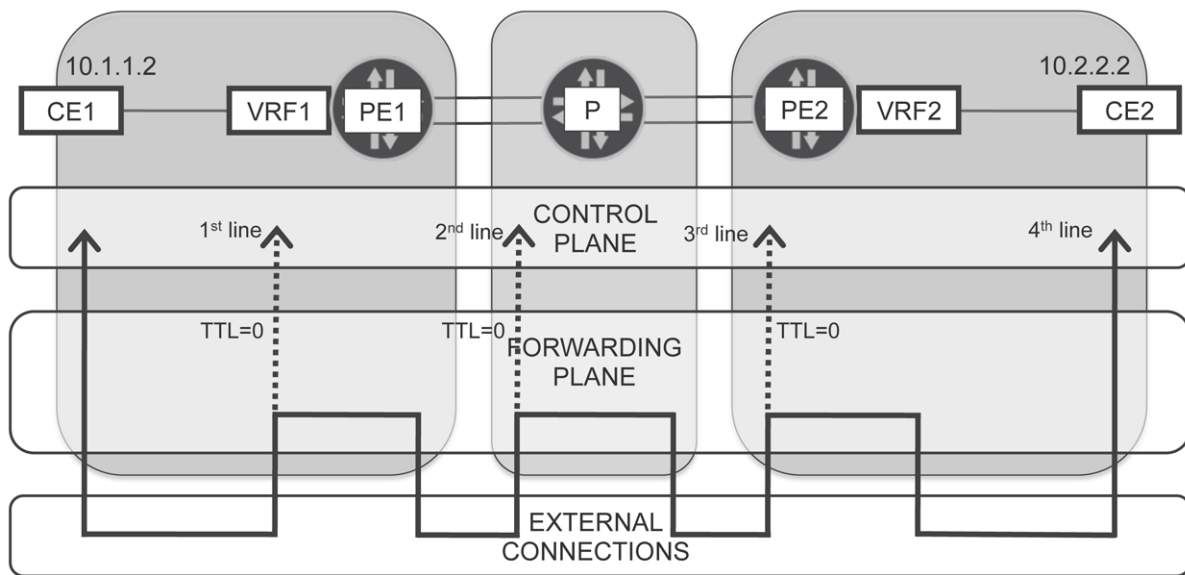


Figure 2.9 Forwarding Path of the Various Packets Involved in Traceroute from CE1 to CE2

The UDP packets with expired TTL value and the resulting ICMP time exceeded messages, are both processed locally by the microkernel of the Line Cards. This is also a component of the Control Plane, as you can see in Figure 2.1. You can verify that during traceroute executions, the following Routing Engine counter does not increase at transit routers:

```
user@P> show system statistics icmp | match exceeded
<x> time exceeded
```

While the following Line Card counter does increase:

```
user@P> show pfe statistics ip icmp | match expired
<x> ttl expired
```

CAUTION Since the packets do not go up to the Routing Engine, they are not visible with local tcpdump at PE1 and PE2.

TIP Execute the `traceroute monitor 10.111.2.2 no-resolve` command at PE1, which is the Junos OS CLI equivalent of a popular tool with different names (*MTR*, *Matt's TraceRoute*, or *my traceroute*):

```
user@PE1> traceroute monitor 10.111.2.2 no-resolve
```

Try It Yourself: Unveiling the traceroute Load Balancing Logic

The Forwarding Plane applies a hash to the packets. How is it possible that the hash results vary from one traceroute execution to another? Use the *video camera* to get more data.

Steering Traffic on a Specific Multihop Path

It's been a long time since IPv4 source routing was deprecated. So, if you miss an alternative way to make ping follow the exact path that you decide beforehand, or a way to guarantee symmetry in the path followed by echo requests and replies, you can define static paths in several ways. These techniques are not always scalable as design options, but they can be used ad-hoc during troubleshooting or network testing. Let's start by exploring Table 2.1.

Table 2.1 Evaluation of Traffic Steering Strategies

Advantages	IP Static Routes	MPLS Static LSPs	RSVP Strict LSPs	Firewall Filter Actions (*)
Is it transparent to PE-routers configuration?	No	No	No	No
Is it transparent to P-routers configuration?	No	No	Yes	No
Is it compatible with all the protocol choices in the core?	Yes	Yes (**)	No	Yes
Can it be used to transport edge VPN services?	No	Yes	Yes	No
Does it preserve the encapsulation of PE-to-PE ping?	Yes	No	No	Yes
Is it supported by all Junos platforms?	Yes	Yes	Yes	No

(*) Actions `next-interface`, `next-ip`, `next-ip6` are supported since Junos 12.2 in Trio.

(**) Junos OS supports static MPLS label range: 1000000 - 1048575

All the options in Table 2.1 show a relative tie, so whether to use one option or another depends on your context. For big networks with RSVP enabled, RSVP strict LSPs are the way to go as you don't want to configure all the P-routers in the path. And that's what we'll use for the following test. (There is a scenario discussed with MPLS Static LSPs later on in Chapter 5, and we'll leave the remaining options with you as an optional exercise to explore.)

A key rule of troubleshooting is to avoid breaking existing services, so you need to find a way to create a static path without changing the current forwarding logic between PE1 and PE2 100.0 addresses. An efficient approach to accomplish this is to define a secondary 100.0 address in both routers:

```
user@PE1> configure
user@PE1# edit interfaces lo0 unit 0 family inet
user@PE1# set address 10.111.1.1/32 primary
user@PE1# set address 10.222.1.1/32
user@PE1# commit and-quit
```

```
user@PE2> configure
user@PE2# edit interfaces lo0 unit 0 family inet
user@PE2# set address 10.111.2.2/32 primary
user@PE2# set address 10.222.2.2/32
user@PE2# commit and-quit
```

CAUTION

Make sure the new address doesn't overlap any of the already allocated addresses in the network in production environments. Also, consider blocking it at the IGP export policy. Finally, it's critical to flag the original 100.0 address as `primary`, otherwise a router ID change may cause serious impact.

Traffic Steering with RSVP Static Paths

The RSVP LSPs need to be targeted to the remote PE router ID, but since you don't want to affect existing services, use the `no-install-to-address` option. On the other hand, the active knob allows you to install the routes towards the remote secondary 100.0 addresses in the global routing table `inet.0`:

```
user@PE1> configure
user@PE1# edit protocols mpls
user@PE1# set path to-PE2 10.100.1.2
user@PE1# set path to-PE2 10.100.4.1
user@PE1# edit label-switched-path PE1-to-PE2
user@PE1# set to 10.111.2.2 no-install-to-address
user@PE1# set install 10.222.2.2/32 active
user@PE1# set primary to-PE2
user@PE1# commit and-quit
user@PE1> show rsvp session name PE1-to-PE2 ingress
Ingress RSVP: 2 sessions
To          From          State   Rt Style Labelin Labelout LSPname
10.111.2.2   10.111.1.1   Up      0  1 FF      -   300176 PE1-to-PE2
Total 1 displayed, Up 1, Down 0
```

```
user@PE2> configure
user@PE2# edit protocols mpls
user@PE2# set path to-PE1 10.100.4.2
user@PE2# set path to-PE1 10.100.1.1
user@PE2# edit label-switched-path PE2-to-PE1
user@PE2# set to 10.111.1.1 no-install-to-address
user@PE2# set install 10.222.1.1/32 active
user@PE2# set primary to-PE1
user@PE2# commit and-quit
user@PE2> show rsvp session name PE2-to-PE1 ingress
Ingress RSVP: 1 sessions
To          From          State   Rt Style Labelin Labelout LSPname
10.111.1.1   10.111.2.2   Up      0  1 FF      -   300160 PE2-to-PE1
Total 1 displayed, Up 1, Down 0
```

MORE? Check the effect of the `no-install-to-address` and `install active` knobs by executing at PE1: `show route 10.111.2.2/32` and `show route 10.222.2.2/32`.

Time to make ping go through the newly engineered path. Execute the following procedure, paying attention to which router gets which command:

```
user@P> clear firewall all

user@PE1> ping 10.222.2.2 source 10.222.1.1 count 100 rapid
PING 10.222.2.2 (10.222.2.2): 56 data bytes
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
--- 10.222.2.2 ping statistics ---
100 packets transmitted, 100 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.429/0.596/15.963/1.544 ms

user@P> show firewall | match request | except " 0"
icmp-echo-request-xe-2/0/0.114-o          8400          100

user@P> show firewall | match reply | except " 0"
icmp-echo-reply-ge-2/3/0.111-o          8400          100

user@P> show firewall | match ^mpls | except " 0"
mpls-packets-ge-2/3/0.111-i             8800          100
mpls-packets-xe-2/0/0.114-i             8800          100
```

```

user@PE1> traceroute 10.222.2.2 no-resolve
traceroute to 10.222.2.2 (10.222.2.2), 30 hops max, 40 byte packets
 1 10.100.1.2 1.794 ms 0.502 ms 0.382 ms
    MPLS Label=300144 CoS=0 TTL=1 S=1
 2 10.222.2.2 0.553 ms 0.446 ms 0.433 ms

user@PE2> traceroute 10.222.1.1 no-resolve
traceroute to 10.222.1.1 (10.222.1.1), 30 hops max, 40 byte packets
 1 10.100.4.2 0.827 ms 0.527 ms 0.491 ms
    MPLS Label=300160 CoS=0 TTL=1 S=1
 2 10.222.1.1 39.423 ms 1.575 ms 0.428 ms

```

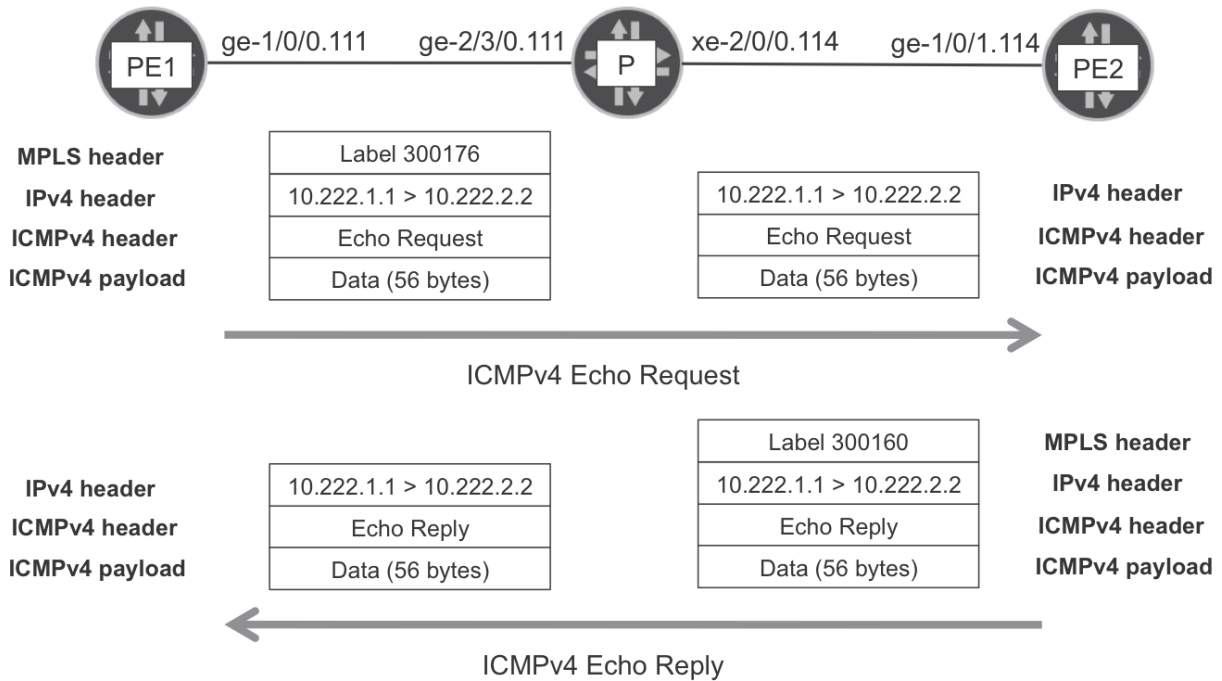


Figure 2.10 Forwarding Path of an ICMPv4 Echo Request and Reply Steered with RSVP LSPs

TRY THIS Apply a `vrf-export` policy at PE1 and PE2. This policy should have an action `then next-hop <local-secondary-lo0.0-address>`. In this way, you can also steer the CE-to-CE and VRF-to-VRF traffic through the static path.

MORE? The Junos technical documentation contains an excellent example describing the `next-interface`, `next-ip`, `next-ip6` options, give it a try in your lab! http://www.juniper.net/techpubs/en_US/junos12.2/topics/topic-map/filter-based-forwarding-policy-based-routing.html

Conclusions

Although ping and traceroute are key tools, it's important to note that due to the load balancing mechanisms discussed:

- Regular end-to-end ping does not necessarily explore all the possible equal cost paths between a source and a destination. This is especially true in large networks.
- Traceroute typically explores more paths, but these do not necessarily include the set of paths explored by ping. Also, a packet loss in traceroute is often difficult to interpret: this is where `traceroute monitor` can be helpful.
- The ping with `record-route` option is nice because it tracks both the forward and return paths. These can definitely be different from the ones followed by traceroute, or even regular ping, including the internal forwarding path inside the routers. On the other hand, when using this option transit routers often stamp packets with their own loopback address (especially if default-address-selection is configured), which is not precise enough to determine the exact physical path.

So, the best approach is: combine the tools wisely, keep in mind what each tool does, and know exactly what you are testing with every command.

Answers to Try It Yourself Sections of Chapter 2

Try It Yourself: Matching ICMP Traffic with a Firewall Filter

The firewall filter defined above is matching on the ICMP type, which is the first octet in the IP payload. Temporarily configure at P:

```
user@P> configure
user@P# edit firewall family inet filter ICMP
user@P# set term ECHO-REPLY then log
user@P# commit and-quit
```

Check the packets matching the ECHO-REPLY term and making it to the firewall log:

```
user@P> show firewall log detail
```

TIP This is a circular log, with newest packets displayed first. Starting with Junos OS 12.1, the firewall log can be cleared with command `clear firewall log`. If you want more persistent logging, use the then `syslog firewall filter` action instead.

Once you spot the packets, capture them with the *video camera* (terminal #0) and analyze them carefully. Can you see why they are matching the ECHO-REPLY term? The `icmp-type` keyword looks at a first byte of the IP payload, since that is precisely the position of the ICMP type field. For ICMP echo replies: `icmp-type = 0`.

On the other hand, the firewall filter term ECHO-REPLY is matching on certain TCP packets, precisely those with source port 179 (BGP). The TCP source port is encoded at the first two bytes of the IP payload. In hexadecimal: 179 = 00:b3. What a coincidence, the most significant byte of the TCP source port is the same as the ICMP type of echo replies!

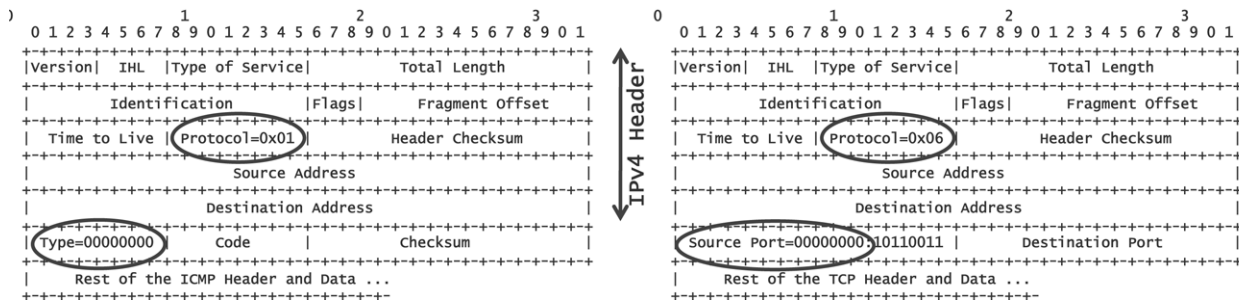


Figure 2.11 Header Comparison Between ICMPv4 Echo Request (left) and BGP Packet (right)

Apply the correct configuration again:

```
user@P> configure
user@P# edit firewall family inet filter ICMP
user@P# delete term ECHO-REPLY then log
user@P# activate term ECHO-REPLY from protocol
user@P# commit and-quit
```


Try It Yourself: The Magic Behind record-route Option

With the *record-route* option, the ICMP packets go up the Control Plane at each hop. But why does this happen for packets with a remote destination IP? As you can check in the capture, the packets have the Record Route Option (#7) in the IP header, which means precisely that: Go up the Control Plane. This option has an IP address list field that gets filled in by transit routers, as you can see in Capture 2.1:

```
<timestamp> IP (tos 0x0, ttl 64, id 8878, offset 0, flags [none], proto ICMP (1), length 124,
options (RR 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0,EOL))
  10.111.1.1 > 10.111.2.2: ICMP echo request, id 4019, seq 0, length 64
<timestamp> IP (tos 0x0, ttl 63, id 8878, offset 0, flags [none], proto ICMP (1), length 124,
options (RR 10.111.11.11, 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0,EOL))
  10.111.1.1 > 10.111.2.2: ICMP echo request, id 4019, seq 0, length 64
<timestamp> IP (tos 0x0, ttl 64, id 56377, offset 0, flags [none], proto ICMP (1), length 124,
options (RR 10.111.11.11, 10.111.2.2, 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0,EOL))
  10.111.2.2 > 10.111.1.1: ICMP echo reply, id 4019, seq 0, length 64
<timestamp> IP (tos 0x0, ttl 63, id 56377, offset 0, flags [none], proto ICMP (1), length 124,
options (RR 10.111.11.11, 10.111.2.2, 10.111.11.11, 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0,EOL))
  10.111.2.2 > 10.111.1.1: ICMP echo reply, id 4019, seq 0, length 64
```

NOTE The RR option has a limited size, or number of hops that it can record.

Try It Yourself: CE-to-CE Ping Tracking at P-router

The answer is very simple, which doesn't mean it's easy! As you saw in Chapter 1, the ICMPv4 packets are encapsulated in MPLS when they reach P, so you need a MPLS filter:

```
user@P> configure
user@P# edit firewall family mpls filter COUNT-MPLS
user@P# set interface-specific
user@P# set term DEFAULT then count mpls-packets
user@P# top
user@P# set interfaces ge-2/3/0 unit 111 family mpls filter input COUNT-MPLS
user@P# set interfaces ge-2/3/0 unit 111 family mpls filter output COUNT-MPLS
user@P# set interfaces ge-2/3/0 unit 113 family mpls filter input COUNT-MPLS
user@P# set interfaces ge-2/3/0 unit 113 family mpls filter output COUNT-MPLS
user@P# set interfaces xe-2/0/0 unit 112 family mpls filter input COUNT-MPLS
user@P# set interfaces xe-2/0/0 unit 112 family mpls filter output COUNT-MPLS
user@P# set interfaces xe-2/0/0 unit 114 family mpls filter input COUNT-MPLS
user@P# set interfaces xe-2/0/0 unit 114 family mpls filter output COUNT-MPLS
user@P# commit and-quit
```

CAUTION This method is not quite as useful in production routers with end-user MPLS transit traffic, as that traffic would be counted, too. So one other option would be to use a different EXP value for ping and match it with the filter, but even that gets complex. Be cautious here.

Try It Yourself: Capturing MPLS-tagged Traffic with tcpdump

In order to see MPLS control traffic sent from PE2 to the core:

```
user@PE2> monitor traffic interface ge-1/0/0.113 no-resolve size 2000 matching mpls
user@PE2> monitor traffic interface ge-1/0/1.114 no-resolve size 2000 matching mpls
```

In order to see MPLS control traffic received by PE2 from the core, there are two options. If you use a *vt-* interface at the VRF, then you can capture directly there. But if you use *vrf-table-label* instead, you need to find the Label Switched Interface (lsi) unit associated to the VRF first:

```
user@PE2> show route instance VRF2 detail | match lsi  
lsi.0
```

```
user@PE2> monitor traffic interface lsi.0 no-resolve size 2000 matching icmp
```

With `vrf-table-label`, a PE allocates a MPLS label for the VRF. All the incoming MPLS packets arriving with that MPLS label at the top of the stack are processed by a receive-only interface coupled to the VRF: in this case, `lsi.0`.

Try It Yourself: Unveiling the traceroute Load Balancing Logic

The UDP ports are taken by default into account to calculate the hash in Trio PFEs. Since both the source and destination UDP ports change at each execution of traceroute, so does the hash calculation.

Chapter 3

Spying on the Private Life of a Packet

<i>The Control Plane in a Nutshell.....</i>	<i>62</i>
<i>The Forwarding Plane in a Nutshell.....</i>	<i>67</i>
<i>The Hidden Hops of One-Hop Ping.....</i>	<i>73</i>
<i>The Hidden Hops of Intra-PFE Transit Ping</i>	<i>74</i>
<i>The Hidden Hops of Inter-PFE Transit Ping.....</i>	<i>77</i>
<i>Failure Scenario</i>	<i>79</i>
<i>Answers to Try It Yourself Sections of Chapter 3.....</i>	<i>81</i>



Let's put ping aside for a moment and go a bit deeper into the architecture of Junos devices. You can see the shaping code used in this book in Figure 3.1. There are basically two functional systems, the Control Plane and the Forwarding Plane, made up of a *network* of functional components. Each physical card can hold one or several functional components, and in some cases (for example, Line Cards) these components belong to different functional systems. In the particular case of the Routing Engine, this book does not represent the functional components (processors and buffers) for the sake of simplicity and brevity.



Figure 3.1 Functional and Physical Shapes

The key word in the previous paragraph is *network*. That's because you can view the router as a set of functional components that are internally connected, and troubleshoot it very much like a real network, made up of different devices linked with external connections. Let's zoom in on the different functional systems.

The Control Plane in a Nutshell

Figure 3.2 shows the internal details of the Control Plane in a M320, a single-chassis T-Series and a high-end MX-Series. These platforms support Routing Engine (RE) redundancy, and have master and backup Control Planes interconnected. You can easily infer the non-redundant scenario from the picture.

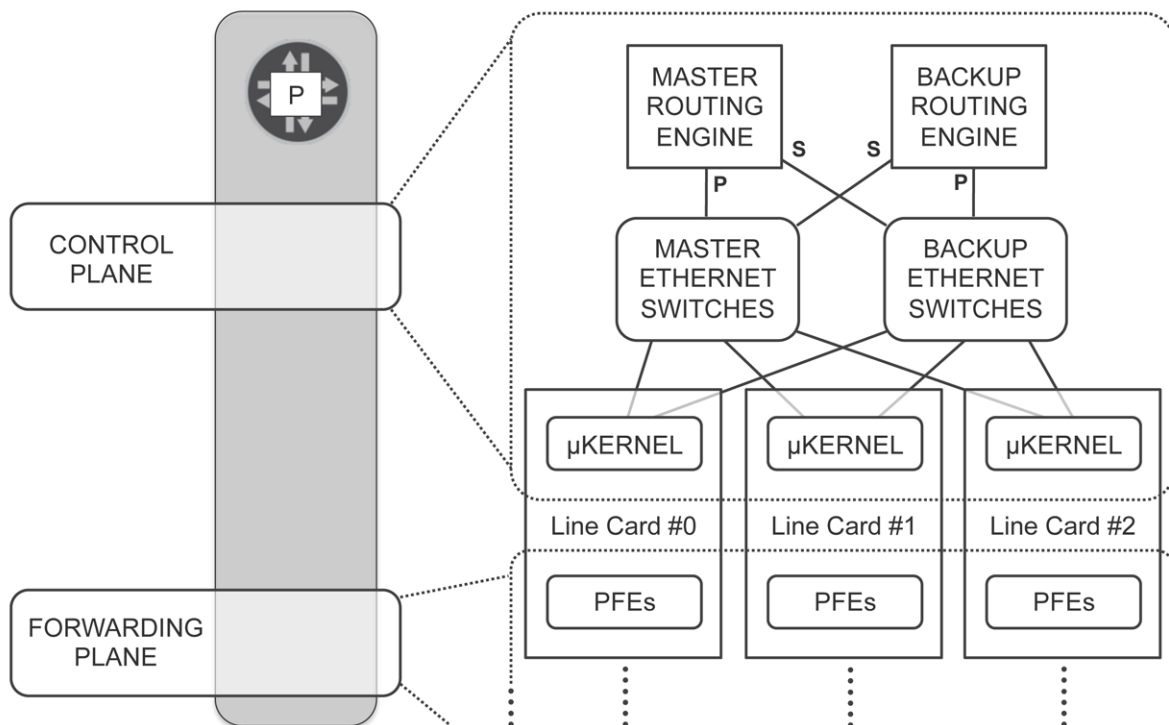


Figure 3.2 Implementation Details of the Control Plane in Redundant Platforms

If P is one of these platforms and it has two REs, you can check the current redundancy state with the following command:

```
user@P-re0> show chassis routing-engine | match "slot|state"
Slot 0:
  Current state           Master
Slot 1:
  Current state           Backup
```

Each *Ethernet Switches* box consists of one or more FE/GE/10GE switches that lie inside a physical card, as detailed in Table 3.1. Since the internal ethernet switching complex uses the midplane and other internal connections, you won't see the cables as in a regular switch! The only exception is the inter-chassis control connections in multi-chassis environments.

Table 3.1 Architecture Details of the Internal Ethernet Switch

Series	Model	Switch Location	Details
MX-Series	MX-5/10/40/80	No switch	No RE redundancy. Direct 1GE connection RE-TFEB.
	MX-240/480/960	Switch Control Board (SCB)	1 switch per plane with all ports 1GE.
M-Series	M7i	No switch	No RE redundancy. Direct FE connection RE-CFEB.
	M10i	HA Chassis Manager (HCM)	1 switch per plane with all ports FE. Master and Backup HCM interconnected. Each HCM only connected to its RE.
	M120	Control Board (CB)	2 switches per plane. GE switch connects via 1GE to same-plane RE, to FEBs and to FE switch. FE switch connects via 1GE to GE switch, and via FE to FPCs. REs can only see line cards (FEBs/FPCs) via same-plane CB.
	M320		1 switch per plane with FE ports connected to FPCs and other-plane RE. Same-plane RE connected via 1GE.
T-Series	T640/T1600/T4000		1 switch per plane with FE ports connected to same-plane LCC-CBs via external FE cables. 1GE ports to same-plane and other-plane SCC-RE via the midplane.
	TX SCC	Control Board (CB)	2 switches per plane. XE switch connects via 10GE to GE switch, and via 1GE to same-plane and other-plane SFC-RE. GE switch connects via 10GE to XE switch, and via 1GE external connections to same-plane LCC-CBs°.
	TXP SFC		

CAUTION The internal ethernet network used for inter-component communication is loop-free by design. There is no Spanning Tree Protocol running on it. That's why the external cabling of multi-chassis Control Plane is a very sensitive task!

You can check the ports and statistics of the internal ethernet switches very easily, provided that there are any switches! The following output is from a MX480:

```
user@P-re0> show chassis ethernet-switch | match "switch|port|speed"
Displaying summary for switch 0
Link is good on GE port 0 connected to device: FPC0
  Speed is 1000Mb
Link is good on GE port 1 connected to device: FPC1
  Speed is 1000Mb
Link is good on GE port 2 connected to device: FPC2
```

```

Speed is 1000Mb
Link is good on GE port 12 connected to device: Other RE
Speed is 1000Mb
Link is good on GE port 13 connected to device: RE-GigE
Speed is 1000Mb

```

```

user@P-re0> show chassis ethernet-switch statistics ?
Possible completions:
<[Enter]>      Execute this command
<port>        Port number (0..27)
|             Pipe through a command

```

In this example, the internal switch sees the REs simply as hosts connected to its ports 12 and 13. Let's run a quick connectivity test between two Control Plane functional components. How about a simple ping from the master RE to the microkernel of a Line Card? This is possible starting with Junos OS 10.4. You first need to know the target IP address:

```

user@P-re0> show arp vpn __juniper_private1__ no-resolve
MAC Address      Address      Interface    Flags
02:00:00:00:00:04 10.0.0.4     em1.0        none
02:00:00:00:00:04 128.0.0.4    em1.0        none
02:00:01:00:00:04 128.0.0.6    em0.0        none
02:00:00:00:00:04 128.0.0.6    em1.0        none
02:00:00:00:00:10 128.0.0.16   em0.0        none
02:00:00:00:00:11 128.0.0.17   em0.0        none
02:00:00:00:00:12 128.0.0.18   em0.0        none
Total entries: 7

```

The `__juniper_private1__` is a private routing instance that the device uses only for internal communication between the different boards. Since it's not meant to be configured, it's often hidden and it only shows up when it's explicitly invoked.

TRY THIS Execute: `show route forwarding-table table __juniper_private1__ all` and `show route table __juniper_private1__`. Try to remove some of the options and you'll likely stop seeing the 128/2 routes, which are not displayed by default.

Out of all the addresses above, which one corresponds to the Line Card in slot #2? You'll need to use one more command, like this:

```

user@P-re0> file show /etc/hosts.junos | match " fpc2$"
128.0.0.18      fpc2

```

So `fpc2` has MAC address `02:00:00:00:00:12`, and that corresponds to IPv4 address `128.0.0.18`, according to the ARP table. You are now ready to go:

```

user@P-re0> ping 128.0.0.18 routing-instance __juniper_private1__ count 100 rapid
PING 128.0.0.18 (128.0.0.18): 56 data bytes
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!
--- 128.0.0.18 ping statistics ---
100 packets transmitted, 100 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.136/0.151/0.281/0.025 ms

```

MORE? Raise the count and spot with `show chassis ethernet-switch statistics` the ports of the internal ethernet switch that are used by the flow. Don't forget that these ports are also used for all the internal control communication and keep increasing in steady mode.

TIP In order to thoroughly test any internal or external forwarding path, you can choose the same bit patterns that are generally used by manufacturing teams while stressing components. See the last Try It Yourself exercise in Chapter 4 for details.

You can try the same ping from the backup RE and it should work from there, too. If RE0 is master (check it with `show chassis routing-engine`), go to RE1:

```
user@P-re0> request routing-engine login other-routing-engine
```

```
user@P-re1> ping 128.0.0.18 routing-instance __juniper_private1__ count 100 rapid
PING 128.0.0.18 (128.0.0.18): 56 data bytes
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!
--- 128.0.0.18 ping statistics ---
100 packets transmitted, 100 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.119/0.142/0.270/0.026 ms
```

```
user@P-re1> exit
```

```
user@P-re0>
```

What do `em0` and `em1` stand for? Both are network interfaces at the Routing Engine. They connect to the same-plane and to the other-plane internal ethernet switches, respectively. In some sense, you can call them primary and secondary interfaces as in Table 3.2. They are flagged as P and S in Figure 3.2.

Table 3.2 Names of Ethernet Interfaces at the Routing Engine

Series	Model	Primary Interface @ RE	Secondary Interface @ RE	Backup RE sends Packets to Line Cards via ... Interface
MX-Series	MX-5/10/40/80	em0 (1GE)	-	-
	MX-240/480/960	em0 (1GE)	em1 (1GE)	Secondary (em1)
M-Series	M7i	fxp1 (FE)	-	-
	M10i	fxp1 (FE)	-	-
	M120	em0 (1GE)	em1 (*) (1GE)	Primary (em0)
	M320			
T-Series	T640/T1600/T4000	bcm0 (1GE)	em0 / em1 (**) (FE)	Secondary (em0/em1)
	TX SCC			
	TXP SFC	ixgbe0 (1GE)	ixgbe1 (1GE)	Secondary (ixgbe1)

(*) Line Cards are not accessible through that interface, only other RE

(**) Depending on the RE model

At terminal #1, connected to P, start a packet capture at the RE primary interface:

```
user@P-re0> monitor traffic interface em0 no-resolve size 2000 matching icmp
verbose output suppressed, use <detail> or <extensive> for full protocol decode
Address resolution is OFF.
Listening on em0, capture size 2000 bytes
```

At terminal #2, also connected to P, run an internal ping to `fpc2`:

```
user@P-re0> ping 128.0.0.18 routing-instance __juniper_private1__
```

The capture should show the ICMPv4 exchange:

```
<timestamp> Out IP 128.0.0.1 > 128.0.0.18: ICMP echo request, id 23088, seq 0, length 64
<timestamp> In IP 128.0.0.18 > 128.0.0.1: ICMP echo reply, id 23088, seq 0, length 64
[...]
```

TIP The `monitor traffic interface` command has a hidden option `write-file <file-name>`. It's unsupported, so if you use it, do it in the lab at your own risk.

Check Capture 3.1 for details of the previous packet exchange. This capture was taken with the `write-file` option at the router itself.

Try It Yourself: Internal vs External Control Traffic

Would the capture show the external ICMPv4 traffic sent and received by the Routing Engine? Execute at terminal #2: `user@P-re0> ping 10.100.1.1`, and see if you catch it at terminal #1. No luck? Then remove matching `icmp` and use the extensive or `write-file` options. If you look *very* carefully at the capture file, you can see the P-PE1 ping traffic and even the ISIS and LDP packets exchanged by P with its neighbors!

As you might have noticed in the last *Try It Yourself* section, a mix of traffic flows in and out the RE ethernet interfaces:

- Internal traffic is not tunneled. The most used transport protocol is TCP. The internal TCP sessions are used for varied functions like transferring the forwarding table from the RE to the Line Cards, maintaining consistency between REs when graceful-switchover is enabled, retrieving statistics and state from Line Cards, (re)booting the Line Cards, and more.
- External traffic is tunneled in TTP (TNP Tunneling Protocol). This is the traffic sent or received by the router from the outside world, namely routing, management, and exception packets exchanged with the neighboring (physical or logical) devices.

Try It Yourself: Testing Control Plane Redundancy Without a RE Switchover

If your test P-router is one whose *Backup RE connects to Line Cards using Secondary Interface* according to Table 3.2, and if it has two REs and (S)CBs, then it has full Control Plane redundancy. If there is a RE switchover, then the new master RE would activate its primary interface. But, would it have connectivity to all the Line Cards? You can know the answer before the RE switchover. How?

MORE? TNP stands for *Trivial Network Protocol*, a legacy Layer-3 protocol traditionally used for internal communication between Control Plane components in Junos OS. In newer versions, the tendency is to migrate towards TCP/IP stack. Here's a sample:

```
user@P-re0> show tnp addresses
  Name      TNPaddr      MAC address  IF  MTU  E  H  R
master      0x1 02:01:00:00:00:05 em0  1500 0 0 3
master      0x1 02:01:01:00:00:05 em1  1500 0 1 3
re0         0x4 02:00:00:00:00:04 em1  1500 3 0 3
re1         0x5 02:01:00:00:00:05 em0  1500 0 0 3
re1         0x5 02:01:01:00:00:05 em1  1500 0 1 3
backup      0x6 02:00:00:00:00:04 em1  1500 3 0 3
fpc0        0x10 02:00:00:00:00:10 em0  1500 5 0 3
fpc1        0x11 02:00:00:00:00:11 em0  1500 4 0 3
fpc2        0x12 02:00:00:00:00:12 em0  1500 5 0 3
bcast       0xffffffff ff:ff:ff:ff:ff:ff em0  1500 0 0 3
bcast       0xffffffff ff:ff:ff:ff:ff:ff em1  1500 0 1 3
```


The Forwarding Plane in a Nutshell

Figure 3.3 illustrates the internal details of the Forwarding Plane in Junos platforms with more than one Packet Forwarding Engine (PFE). When a packet enters the router on one PFE, the result of the route lookup may be a next hop located in a different PFE. In order to move traffic from one PFE to another, the router needs fabric planes interconnecting PFEs, as you can see in the following Figure 3.4. The source PFEs *spray* the traffic across all the available planes, in a round-robin fashion, towards the destination PFEs.

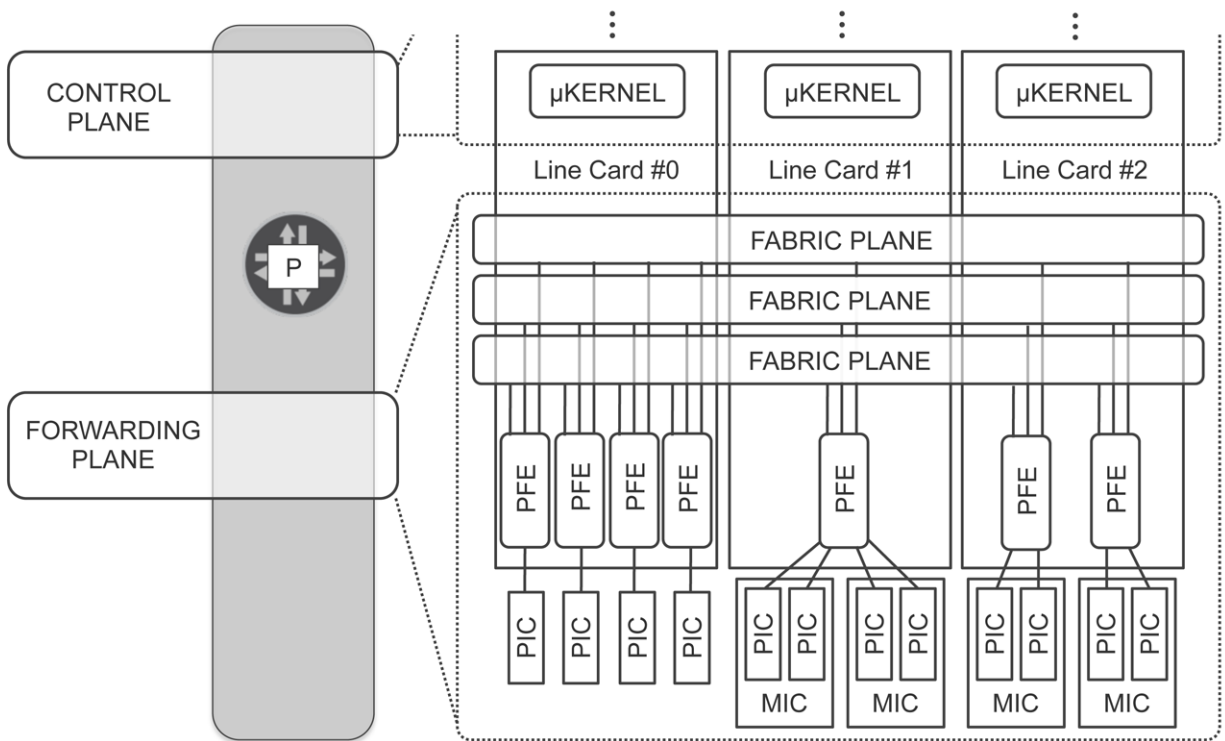


Figure 3.3 Architecture Details of the Forwarding Plane in Multi-PFE Platforms

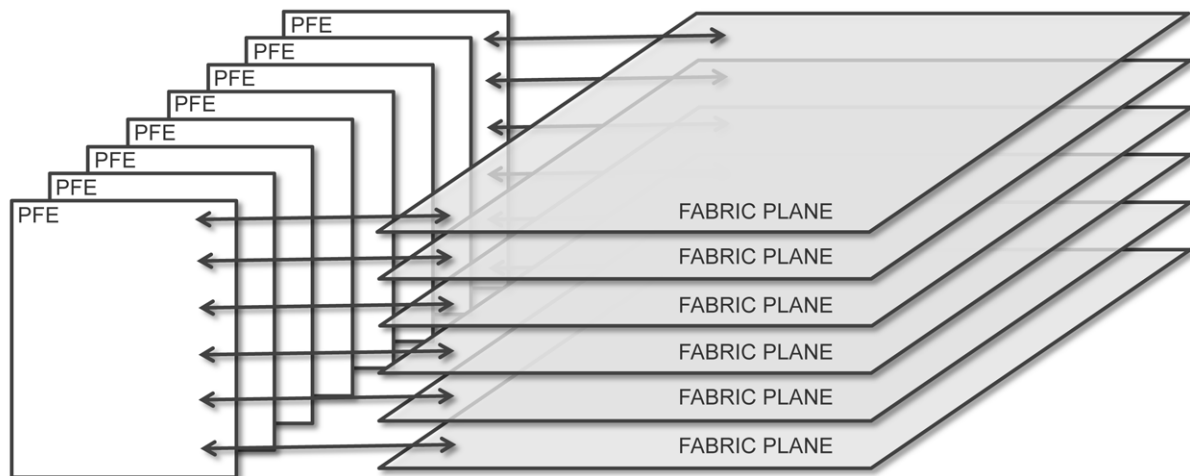


Figure 3.4 Model of PFE Interconnection with Fabric Planes

During troubleshooting, it is essential to know where PFEs are located, so check out Table 3.3 for some details in a per-platform basis.

Table 3.3 Packet Forwarding Engine (PFE) Distribution per Platform

Series	Model	PFE Location	Number of PFEs
MX-Series	MX-5/10/40/80	Built-In Trio Forwarding Engine Board (TFEB)	1 global (built-in)
	MX-240/480/960	Modular PIC Concentrator (MPC) (*) - or - Dense PIC Concentrator (DPC) - or - Flexible PIC Concentrator (FPC)	1, 2 or 4 per MPC/DPC/FPC, depending on the model
M-Series	M7i	Compact Forwarding Engine Board (CFEB)	1 global
	M10i		1 global (redundant)
	M120	Forwarding Engine Board (FEB) (**)	1 per FEB
	M320	Flexible PIC Concentrator (FPC)	1 or 2 per FPC, depending on the model
T-Series	T640/T1600/T4000		
	TX SCC / TXP SFC		

(*) MPC is based on Trio. DPC and FPC are pre-Trio. Junos OS CLI operational command syntax refer to all of them as fpc.

(**) M120 also has FPCs, but they don't contain any PFEs. Their function is to interconnect PICs with a FEB

REMEMBER The pre-Trio term is used for the PFEs that were developed before the Trio chipset.

Execute the following command at P, provided that you are using a multi-PFE device:

```
user@P-re0> show chassis fabric fpcs | match "fpc|pfe"
Fabric management FPC state:
FPC 0
  PFE #0
  PFE #1
  PFE #2
  PFE #3
FPC 1
  PFE #0
  PFE #1
  PFE #2
  PFE #3
FPC 2
  PFE #0
  PFE #1
```

In this particular example, the FPCs in slots (0, 1, 2) each have (4, 4, 2) PFEs, respectively. This is useful to know because it can help you determine the internal forwarding path of a transit packet based on the input and output ports.

TIP In a M120, each FEB has a PFE and you can find the mapping with the command `show chassis fpc-feb-connectivity`.

Table 3.4 provides details about where the fabric planes are physically located in each platform.

Table 3.4 Fabric Plane Distribution per Platform

Series	Model	Fabric Location	Number of Fabric Planes
MX-Series	MX-5/10/40/80	-	-
	MX-240	Switch Control Board (SCB)	4 per SCB (up to 2 active SCBs = 8 active planes)
	MX-480		4 per SCB (up to 2 active SCBs = 8 active planes)
	MX-960		2 per SCB (up to 3 active SCBs = 6 active planes)
M-Series	M7i	-	-
	M10i		
	M120	Control Board (CB)	2 per CB (up to 2 active CBs = 4 active planes)
	M320	Switch Interface Board (SIB)	1 per SIB (up to 4 active SIBs = 4 active planes)
T-Series	T640/T1600/T4000		
	TX SCC / TXP SFC	Switch Interface Board (SIB)	Three-stage CLOS fabric with 4 active planes

And let's look at the layout of a MX480 with redundant Switch Control Boards:

```
user@P-re0> show chassis fabric summary
Plane  State    Uptime
0      Online    1 day, 19 hours, 50 minutes, 3 seconds
1      Online    1 day, 19 hours, 50 minutes, 3 seconds
2      Online    1 day, 19 hours, 50 minutes, 3 seconds
3      Online    1 day, 19 hours, 50 minutes, 2 seconds
4      Online    1 day, 19 hours, 50 minutes, 2 seconds
5      Online    1 day, 19 hours, 50 minutes, 2 seconds
6      Online    1 day, 19 hours, 50 minutes, 2 seconds
7      Online    1 day, 19 hours, 50 minutes, 2 seconds
```

```
user@P-re0> show chassis fabric plane-location
-----Fabric Plane Locations-----
Plane 0      Control Board 0
Plane 1      Control Board 0
Plane 2      Control Board 0
Plane 3      Control Board 0
Plane 4      Control Board 1
Plane 5      Control Board 1
Plane 6      Control Board 1
Plane 7      Control Board 1
```

Before moving deeper into the Forwarding Plane, let's talk briefly about PICs (Physical Interface Cards) and MICs (Modular Interface Cards). From an architectural point of view, a PIC is a physical card containing a set of functional components called ports or interfaces. Stated simply, PICs provide connectivity between PFEs and the outside world. Depending on the Line Card model, PICs can be either independent physical cards, or physically bound to Line Cards, or bundled in a MIC. The show commands typically take pic as an argument, but the commonly used request chassis pic [offline|online] operation only works if the PIC is an independent physical card. Otherwise you need to act on the card it is bound to (fpc or mic).

Execute at P:

```
user@P-re0> show chassis fpc pic-status
Slot 0  Online    DPCE 40x 1GE R
  PIC 0  Online    10x 1GE(LAN)
  PIC 1  Online    10x 1GE(LAN)
```

```

PIC 2 Online      10x 1GE(LAN)
PIC 3 Online      10x 1GE(LAN)
Slot 1 Online     MPC 3D 16x 10GE
PIC 0 Online      4x 10GE(LAN) SFP+
PIC 1 Online      4x 10GE(LAN) SFP+
PIC 2 Online      4x 10GE(LAN) SFP+
PIC 3 Online      4x 10GE(LAN) SFP+
Slot 2 Online     MPC Type 2 3D Q
PIC 0 Online      2x 10GE XFP
PIC 1 Online      2x 10GE XFP
PIC 2 Online      10x 1GE(LAN) SFP
PIC 3 Online      10x 1GE(LAN) SFP

```

Combining the output of the previous commands, you can conclude that the interfaces xe-2/0/0 and ge-2/3/0 are handled by PFEs #2/0 and #2/1, respectively. Note that interface naming follows the logic <line_card#>/<pic#>/<port#>, while <mic#> and <pfe#> and are irrelevant as far as port numbering is concerned.

CAUTION

In this example, the Line Cards had all the PIC/MIC slots full, but that will not always be the case. Consider the possibility of empty PIC slots when doing the PFE-to-PIC mapping.

Inside the Forwarding Plane of a router, the packets are processed in a way that optimizes features and performance. Some extra overhead is added to the original packets to signal internal information like the Forwarding Class after COS classification. This extra overhead is removed before sending the packet out of the router. On the other hand, because it is not optimal to transport large data units across the internal fabric, big packets are fragmented per design. So Ingress PFE typically cuts them into pieces, which are reassembled later in the forwarding path. Figures 3.5 and 3.6 show this activity in detail. Note that the grey lines represent the points of segmentation and reassembly.

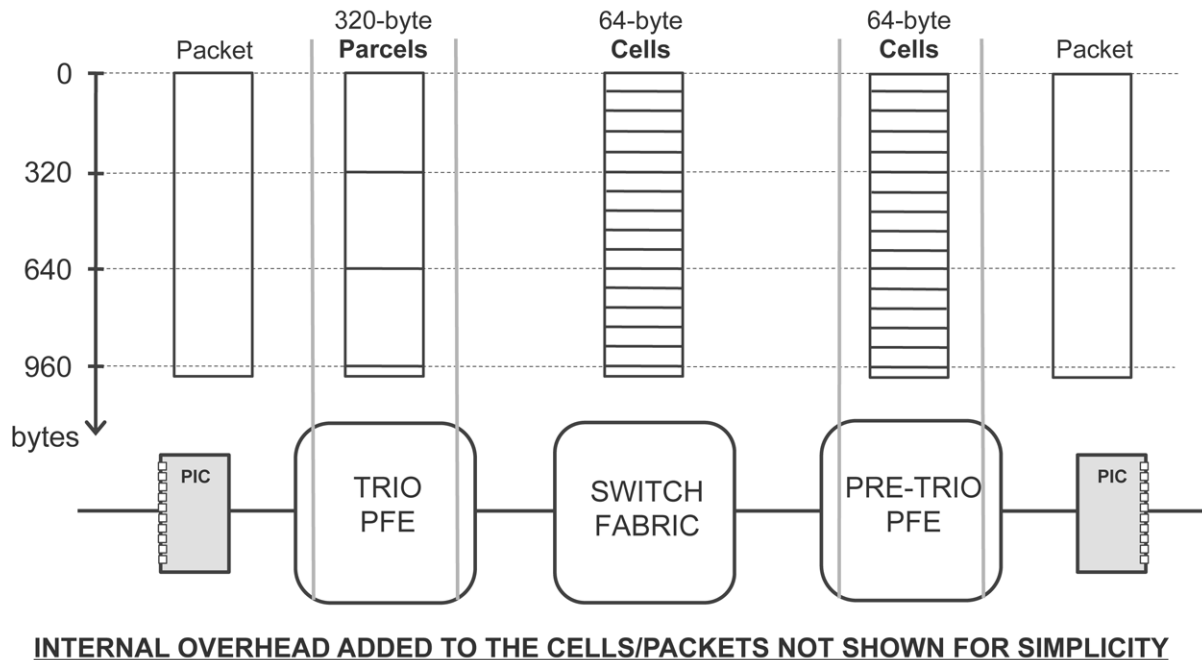


Figure 3.5 Internal Processing of Transit Packets

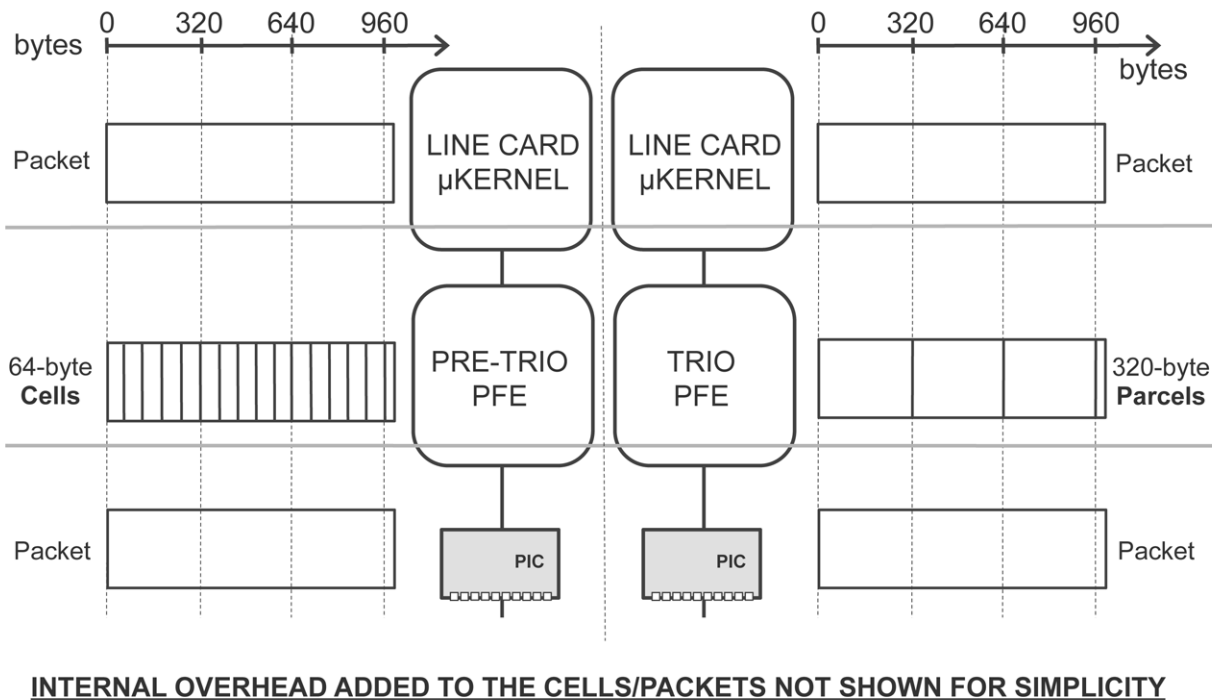


Figure 3.6 Internal Processing of Control and Exception Packets

NOTE For intra-PFE transit traffic, Trio and pre-Trio PFEs also use parcels and cells, respectively.

With the network in a steady state and no ping running, let's see what's actually going on at the PFEs and the switch fabric:

```
user@P-re0> clear pfe statistics traffic
Traffic statistics cleared.
```

Wait for one or two minutes, then execute:

```
user@P-re0> show pfe statistics traffic fpc 2 | except "^ .* 0"
Packet Forwarding Engine traffic statistics:
  Input packets:          775          4 pps
  Output packets:         504          2 pps
Packet Forwarding Engine local traffic statistics:
  Local packets input      :          511
  Local packets output     :          463
/* Output Suppressed */
Packet Forwarding Engine hardware discard statistics:
  Normal discard           :          224
```

How might you interpret this data? The Input packets and Output packets account for all the traffic that enters/exits the PFE from/to the PICs. This includes all the transit, control, and exception traffic, but that's not all. The Input packets also account for Normal discard, in this specific setup untagged traffic received from the switch. In a router with no transit traffic at all, these formulas should apply:

Input packets = Local packets input + Normal discard + <Other Drops>

Output packets = Local packets output

However, in this example, the numbers on the left of the equal sign are slightly larger (by around 40 packets) than the numbers on the right: 775 > 735, and 504 > 463.

Try It Yourself: Interpreting PFE Statistics

Why aren't the formulas exact? The answer is very simple, but not that easy to figure out! Think about it, figure it out on your own, or cheat and go to the end of the chapter to find out.

In the current situation, it's natural to expect there to be no traffic flowing through the fabric at all. However, in this case there is some traffic. Let's look:

```
user@P-re0> show class-of-service fabric statistics source 2 destination 2 | match "stat|pps"
Total statistics:  High priority      Low priority
Pps      :      39                  0
Tx statistics:   High priority      Low priority
Pps      :      39                  0
Drop statistics: High priority      Low priority
Pps      :      0                   0
```

The reason there is some traffic is because the software release used during this writing – Junos OS 11.4R4.4 – supports a nice feature on MX-Series called *fabric self-ping*. Here, each PFE periodically pings itself using its fabric loopback stream: the path that takes cells from the PFE to the switch fabric, and back to the same PFE. And it does so for every fabric plane. The goal is to detect PFE-to-fabric connectivity issues in an independent way. Before the new feature was implemented, end-user transit traffic was necessary to start detecting fabric failures.

Figure 3.7 illustrates the implementation details of this feature. In pre-Trio PFEs, the self-ping packets are generated by the microkernel and sent at a low rate using the PFE-to-fabric queue #7, which has low priority by default but can and should be changed (with `set class-of-service forwarding-classes <name> queue-num 7 priority high`, for example). On the other hand, Trio PFEs generate the packets themselves at a high rate, using a high priority PFE-to-fabric queue. In other words, self-ping is software-based for pre-Trio, and hardware-based for Trio.

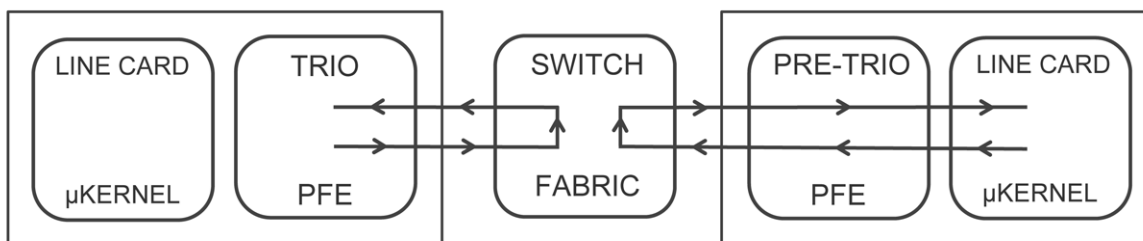


Figure 3.7 PFE Self-ping Through Fabric Forwarding Path

NOTE At the time of this writing T-Series also supports fabric self-ping.

MORE? *PFE liveness* is yet another related feature that periodically sends traffic to *other* PFEs through the fabric. This allows for fast internal forwarding path restoration when a Line Card fails or is abruptly pulled from the chassis. This feature can be enabled in Junos OS 12.1 or higher in all-Trio MX Series with the `[set chassis network-services enhanced-ip]` knob.

The Hidden Hops of One-Hop Ping

Now that the internal secrets of the Control and Forwarding Planes are unveiled, traffic between PE1 and P is no longer seen as one hop. Figure 3.8 shows that there are indeed quite a few internal hops inside the router. Since the TTL of the packets doesn't change, these extra hops often go unnoticed, but for purposes of good network design, operation, and troubleshooting, you should definitely take them into account.

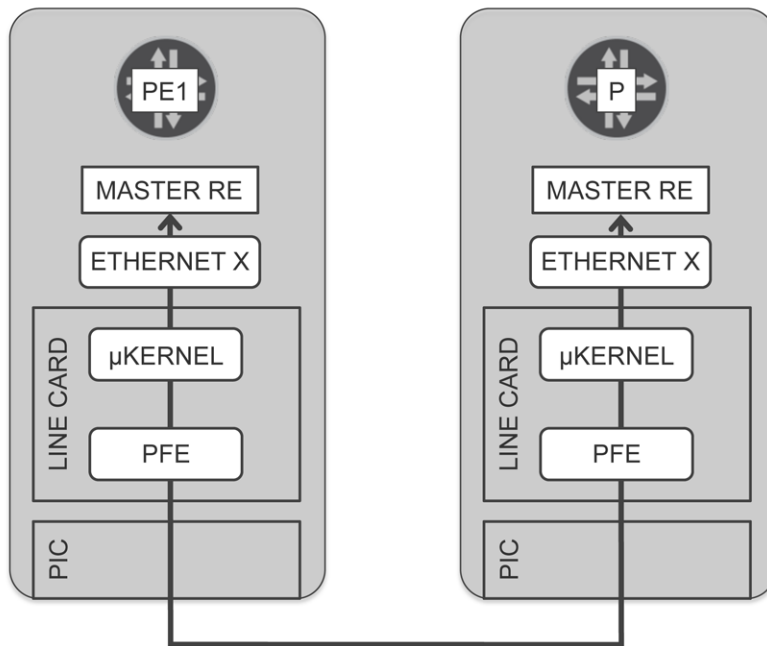


Figure 3.8 Forwarding Path of One-hop Ping from PE1 to P

Open terminal #1 connected to PE1 and terminal #2 connected to P. You are about to execute PE1-to-P ping repeatedly at terminal #1, and each time you will look at a different section of the forwarding path inside P at terminal #2.

Here is the ping operation to be repeated at terminal #1:

```
user@PE1> ping 10.100.1.2 source 10.100.1.1 rapid count 10000
--- 10.100.1.2 ping statistics ---
10000 packets transmitted, 10000 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.395/0.742/92.370/1.870 ms
```

IMPORTANT If at some point in this chapter you see single-packet periodic drops such as
 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!, it is to be expected. The reason why lies in Chapter 4.

You should first notice that interface packet statistics increment by 10,000, plus some routing protocol packets exchanged during the ping exercise:

```
user@P-re0> clear interfaces statistics all

<--- ping PE1-to-P starts and completes here

user@P-re0> show interfaces statistics ge-2/3/0.111 | match packets
Input packets : 10013
Output packets: 10011
```

The PFE input and output stats also increment by 10,000, as expected:

```
user@P-re0> clear pfe statistics traffic

<--- ping PE1-to-P starts here

user@P-re0> show pfe statistics traffic fpc 2 | match pps
Packet Forwarding Engine traffic statistics:
  Input  packets:      8500      1266 pps
  Output packets:      7135      1224 pps

<--- ping PE1-to-P completes here

user@P-re0> show pfe statistics traffic fpc 2 | match packets
Packet Forwarding Engine traffic statistics:
  Input  packets:      10047      3 pps
  Output packets:      10030      1 pps
Packet Forwarding Engine local traffic statistics:
  Local packets input      :      10034
  Local packets output     :      10027
```

This is local control traffic, so it should by no means go through the switch fabric. Check to see that during the PE1-to-P ping, only fabric self-ping crosses the fabric, as usual:

```
user@P-re0> show class-of-service fabric statistics source 2 destination 2 | match "stat|pps"
Total statistics:  High priority      Low priority
Pps      :          40                0
Tx statistics:   High priority      Low priority
Pps      :          40                0
Drop statistics: High priority      Low priority
Pps      :          0                0
```

As seen previously, external control and exception traffic travels disguised as TTP on its way between the RE and the Line Card microkernel. The TTP stats are also increased by 10,000, accordingly:

```
user@P-re0> show system statistics ttp | match "sent$| ipv4 "
258530 Packets sent
196626 IPv4 L3 packets received

<--- ping PE1-to-P starts and completes here

user@P-re0> show system statistics ttp | match "sent$| ipv4 "
268556 Packets sent
206642 IPv4 L3 packets received
```

The Hidden Hops of Intra-PFE Transit Ping

Let's bring down some links to ensure that ping between PE1 and PE2 enters and exits P at the same PFE. Execute at P:

```
user@P-re0> configure
user@P-re0# set interfaces xe-2/0/0.112 disable
user@P-re0# set interfaces xe-2/0/0.114 disable
user@P-re0# commit and-quit
user@P-re0> show isis adjacency
Interface      System      L State      Hold (secs) SNPA
ge-2/3/0.111   PE1         2 Up          24
ge-2/3/0.113   PE2         2 Up          26
```


This is the ping operation at terminal #1, to be repeated:

```
user@PE1> ping 10.111.2.2 rapid count 10000
--- 10.111.2.2 ping statistics ---
/* Output suppressed */
10000 packets transmitted, 10000 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.422/0.535/44.918/1.109 ms
```

The forwarding path is shown in Figure 3.9. As expected, interface packet statistics increment by 10,000, plus some routing protocol packets exchanged during the ping exercise:

```
user@P-re0> clear interfaces statistics all

<--- ping PE1-to-P starts and completes here

user@P-re0> show interfaces statistics ge-2/3/0.111 | match packets
Input packets : 10014
Output packets: 10014

user@P-re0> show interfaces statistics ge-2/3/0.113 | match packets
Input packets : 10017
Output packets: 10015
```

The PFE input and output stats, shown next, increment by 20,000, because this time both ICMPv4 echo requests and replies enter *and* exit the router:

```
user@P-re0> clear pfe statistics traffic

<--- ping PE1-to-P starts here

user@P-re0> show pfe statistics traffic fpc 2 | match pps
Packet Forwarding Engine traffic statistics:
Input packets:      12310      3155 pps
Output packets:     12312      3155 pps

<--- ping PE1-to-P completes here

user@P-re0> show pfe statistics traffic fpc 2 | match packets
Packet Forwarding Engine traffic statistics:
Input packets:      20036      3 pps
Output packets:     20021      1 pps
Packet Forwarding Engine local traffic statistics:
Local packets input :          24
Local packets output:         21
```

Since the traffic enters and exits the router at the same PFE #2/0, it doesn't need to cross the switch fabric. Let's check to ensure that during the PE1-to-PE2 ping, only fabric self-ping crosses the fabric at P:

```
user@P-re0> show class-of-service fabric statistics source 2 destination 2 | match "stat|pps"
Total statistics:  High priority  Low priority
Pps      :          38            0
Tx statistics:   High priority  Low priority
Pps      :          38            0
Drop statistics: High priority  Low priority
Pps      :           0            0
```

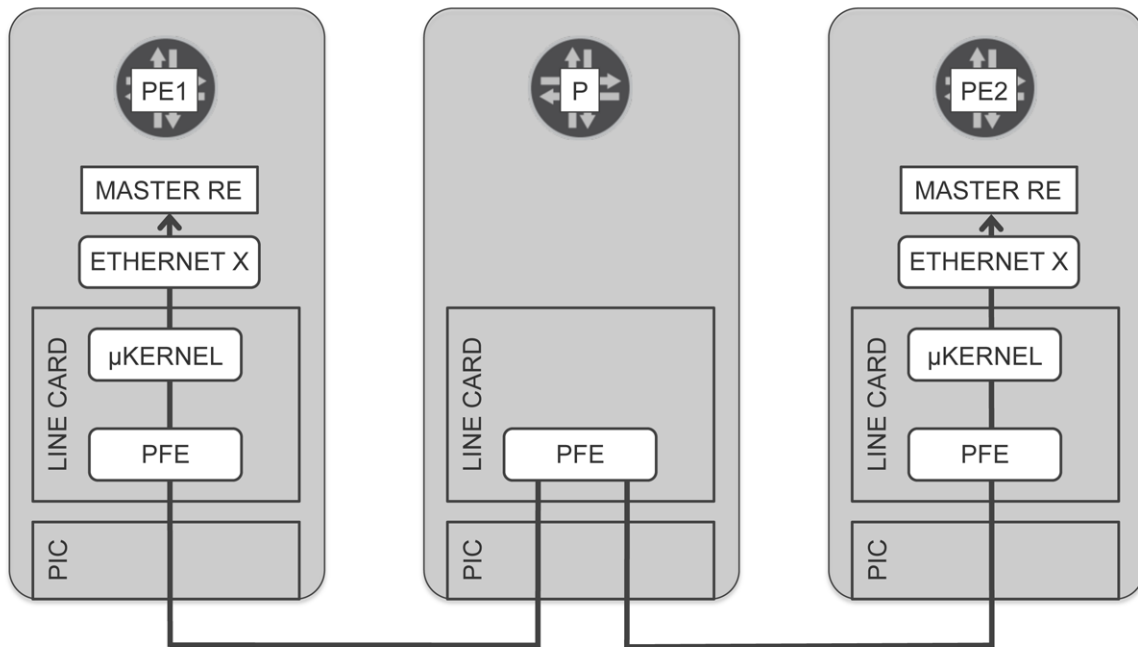


Figure 3.9 Forwarding Path of Intra-PFE Transit Ping from PE1 to PE2

The sentence: *Since the traffic enters and exits the router at the same PFE, it doesn't need to cross the switch fabric*, is actually misleading because it is not always true. Table 3.5 shows a subset of the actions that do require the packet to be looped once in the PFE-fabric-PFE path, even for intra-PFE traffic with the same source and destination PFE.

Table 3.5 Intra-PFE Transit Traffic Handling in Different Line Cards Types

Action	Requires additional loop through the fabric?	
	TRIO	Pre-TRIO
Input Firewall Filter	No(*)	No
Output Firewall Filter	No	Yes
MPLS push/swap/pop	No	Yes
Input Sampling-like Actions	Yes	No

(*) The only exception is Sampling-like Actions.

In general, pre-Trio PFEs can only perform a limited number/type of actions in one single routing chip lookup. Yet in some situations, further lookups are required: for example, one route/label lookup followed by a MPLS label operation. In these cases, the PFE needs to receive the packet several times and it uses the fabric loop in order to do so, as shown in Figures 3.10 and 3.11.

CONCLUSION

A healthy switch fabric is essential for any transit MPLS packet to *egress* a pre-Trio PFE.

For Trio PFEs, the fabric loop is only used in one corner case: input sampling-like firewall filter actions. These actions are: `then sample`, `log`, `syslog`, `port-mirror`. When the actions are invoked from an input filter, the current implementation

requires using the PFE-fabric-PFE loopback path. In fabric-less routers like MX-5/10/40/80, there is a dedicated loopback PFE-PFE path for that purpose.

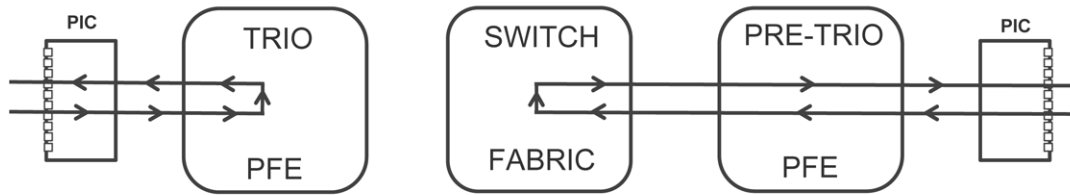


Figure 3.10 Intra-PFE MPLS Transit Traffic Internal Path in Different Line Card Types

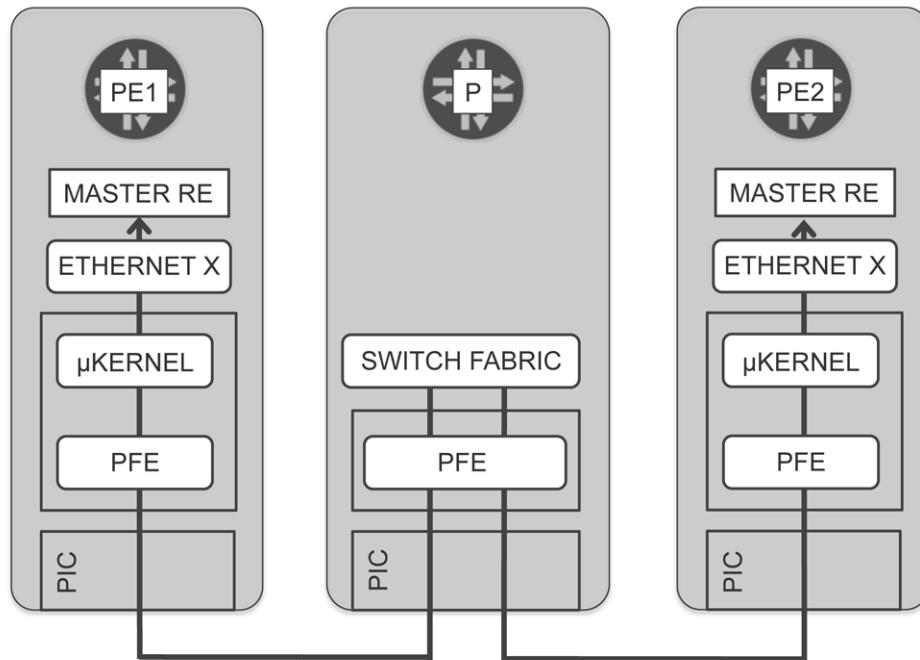


Figure 3.11 Forwarding Path of Intra-PFE Transit Ping When the PFE-fabric Loop is Used

TRY THIS Check to ensure that TTP statistics do not increase at P during regular PE1-PE2 ping.

The Hidden Hops of Inter-PFE Transit Ping

Let's change the active links to ensure that ping between PE1 and PE2 enters and exits P at different PFEs. Execute at P:

```
user@P-re0> configure
user@P-re0# delete interfaces xe-2/0/0.114 disable
user@P-re0# set interfaces ge-2/3/0.113 disable
user@P-re0# commit and-quit
user@P-re0> show isis adjacency
Interface      System      L State      Hold (secs) SNPA
ge-2/3/0.111   PE1         2 Up         25
xe-2/0/0.114   PE2         2 Up         24
```

This is the ping operation at terminal #1, to be repeated elsewhere:

```
user@PE1> ping 10.111.2.2 rapid count 10000
--- 10.111.2.2 ping statistics ---
```

```
/* Output suppressed */
10000 packets transmitted, 10000 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.422/0.535/44.918/1.109 ms
```

The PFE input and output stats increment by 20,000. As was the case for intra-PFE ping, both ICMPv4 echo requests and replies enter *and* exit the router:

```
user@P-re0> clear pfe statistics traffic

<--- ping PE1-to-P starts here

user@P-re0> show pfe statistics traffic fpc 2 | match pps
Input packets:      10389      3185 pps
Output packets:     11562      3188 pps

<--- ping PE1-to-P completes here

user@P-re0> show pfe statistics traffic fpc 2 | match packets
Packet Forwarding Engine traffic statistics:
Input packets:      20057      2 pps
Output packets:     20036      2 pps
Packet Forwarding Engine local traffic statistics:
Local packets input :          40
Local packets output:         33
```

Since the traffic enters and exits the router at different PFEs, #2/0 and #2/1, it needs to cross the switch fabric. Execute during PE1-to-PE2 ping, at P:

```
user@P-re0> show class-of-service fabric statistics source 2 destination 2 | match "stat|pps"
Total statistics:  High priority  Low priority
Pps :              40             3139
Tx statistics:    High priority  Low priority
Pps :              40             3139
Drop statistics:  High priority  Low priority
Pps :              0             0
```

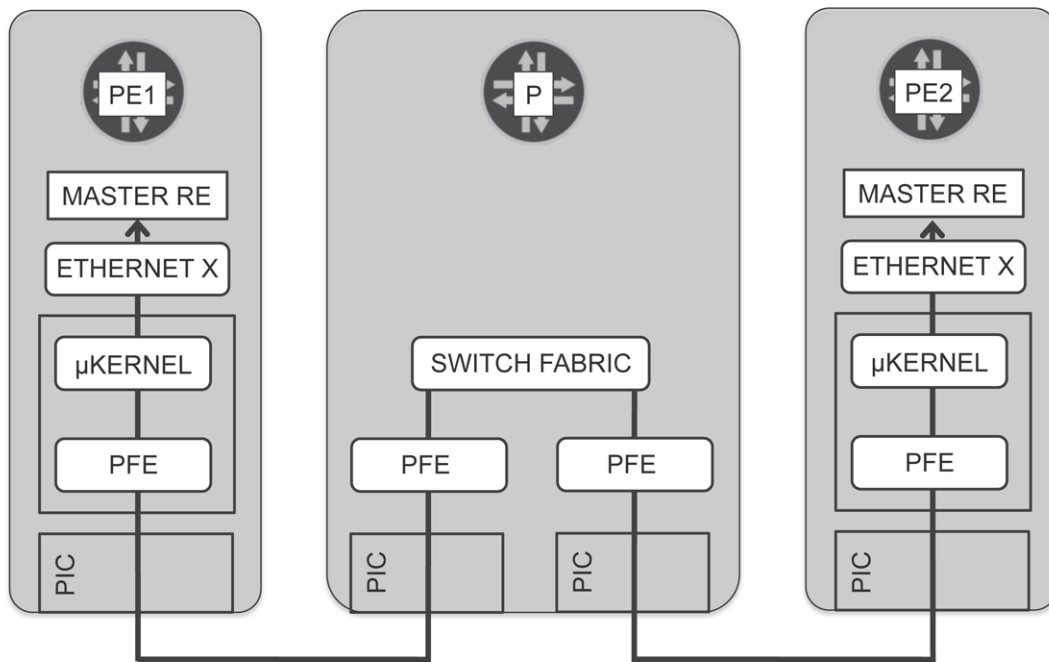


Figure 3.12 Forwarding Path of Inter-PFE Transit Ping from PE1 to PE2

Try It Yourself: The Private Life of record-route ping

This time, try it with the record-route option, and compare with the previous test. Can you identify the three differences? Two of the differences are relatively straightforward, but how about the third one? As always, the answers are at the end of each chapter.

Failure Scenario

One of the inspirations for this book is the following real-life scenario; hope you like it! Let's keep the topology with single PE-P links in different PFEs:

```
user@P-re0> show isis adjacency
```

Interface	System	L State	Hold (secs)	SNPA
ge-2/3/0.111	PE1	2 Up	23	
xe-2/0/0.114	PE2	2 Up	26	

Figure 3.13 shows the forwarding path taken by echo requests and replies. The dotted line shows the case where the record-route option is used.

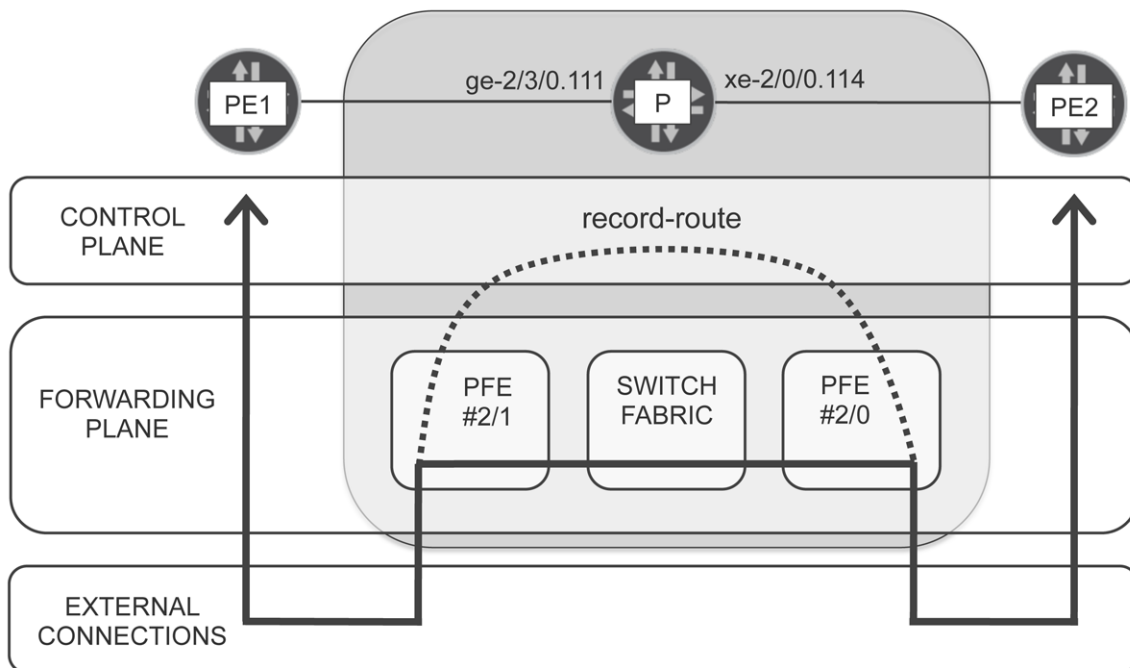


Figure 3.13 Forwarding Path of Inter-PFE Transit Ping With and Without Record-route Option

Open three terminals, terminal #1 and #2 connected to PE1, and terminal #3 connected to P.

At terminal #1, leave running:

```
user@PE1> ping 10.111.2.2 interval 0.1
```

At terminal #2, leave running:

```
user@PE1> ping 10.111.2.2 interval 0.1 record-route no-resolve
```

At terminal #3, execute the necessary procedure to bring down all the fabric planes at P. The command `show chassis fabric summary` provides you with the fabric

plane numbers, and you can then offline them by executing `request chassis fabric plane offline <x>`, where <x> is a number going from 0 to the highest numbered plane.

ALERT! Don't execute these commands in a live system, unless you are very sure of what the impact of bringing the fabric planes down will be on the network and you are certain that result is what you want.

You will definitely see some impact on terminal #1, depending on the release and platform you are running. During the writing of this book, Junos OS version 11.4R4.4 was used. This version of Junos has fabric robustness enhancements for MX-series that automatically try to bring up the fabric planes if all of them are detected as down. So the measured impact is several seconds instead of *forever*. If you see no impact at all, probably both uplinks are in the same PFE.

TIP You can measure the impact by typing `ctrl-C` at terminals #1 and #2 once both pings are up and running again. If you multiply the number of lost packets by 10, then you have the number of seconds of impact.

You should see no impact on terminal #2 at all, since the forwarding path there doesn't use the fabric. This is also the case for one-hop control traffic, like ISIS, LDP, and RSVP. During the time all the fabric planes are down, P is attracting traffic that it's unable to forward, so blackholing occurs. If the fabric down condition lasts long enough, though, the BGP session between PE1 and PE2 eventually goes down. But in more complex topologies, the BGP sessions between the PEs and the Route Reflectors may not go through P. In this case, the sessions may stay up indefinitely while the forwarding path between PE1 and PE2 is broken, leading to a long traffic blackholing. That's why the fabric robustness enhancements are so important.

CAUTION If you offline all planes several times, the fabric robustness logic will take more aggressive actions, like reloading the Line Cards, and ultimately offlining them. The idea behind it all: better to be down than blackholing traffic.

MORE? Search for *Traffic Black Hole Caused by Fabric Degradation* in the Junos technical guides within your version of Junos OS.

Try It Yourself: Selectively Breaking the Control Plane

The previous test showed a failure in ping at terminal #1, while ping at terminal #2 was fine. Try to figure out a way to trigger the reverse failure: one that impacts terminal #2, but leaves terminal #1 up and running.

Answers to Try It Yourself Sections of Chapter 3

Try It Yourself: Internal vs External Control Traffic

Get a packet capture directly on the router internal interface:

```
user@P-re0> monitor traffic interface em0 no-resolve size 2000 write-file /var/tmp/em0.pcap
```

The external ICMPv4, LDP, ISIS traffic is in fact tunneled inside a TTP header. TTP stands for TNP Tunneling Protocol and was originally designed to work over TNP. With the migration to TCP/IP stack, TTP has been redefined as IPv4 protocol 84 and registered at IANA. Figures 3.14 and 3.15 show the *secret life* of an ICMPv4 echo request going from P to PE1.

Capture 4.2 contains the packets exchanged with: user@P> ping 10.100.1.1 count 1

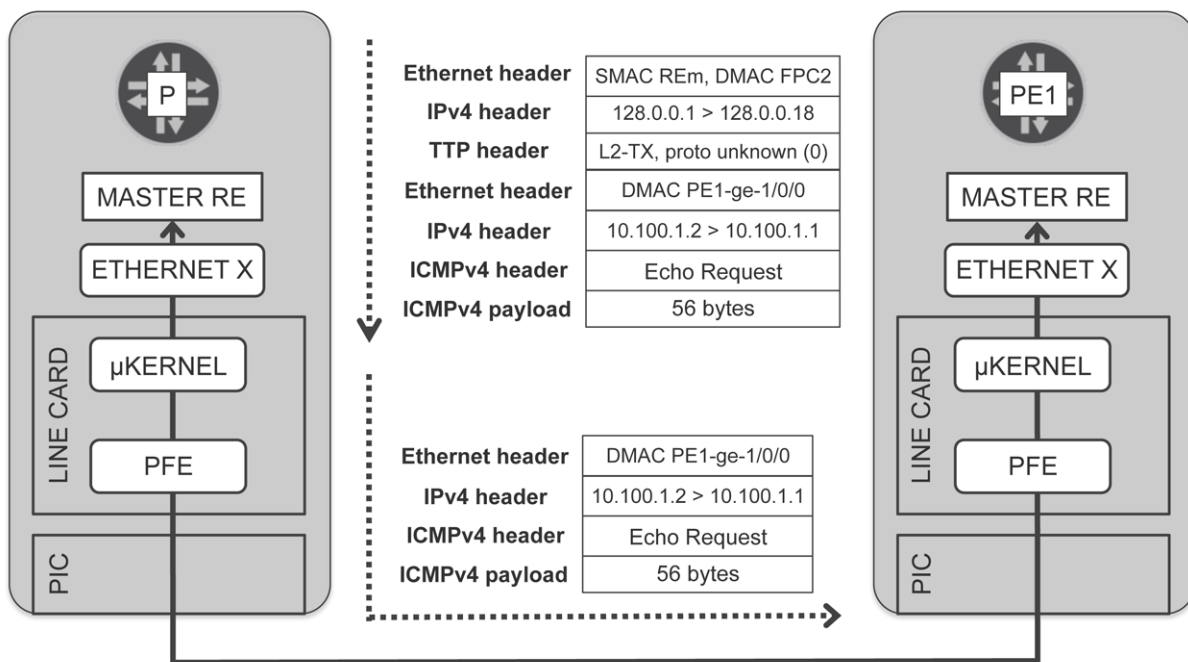


Figure 3.14 Secret Life of an ICMPv4 Echo Request from P Master RE to the Wire

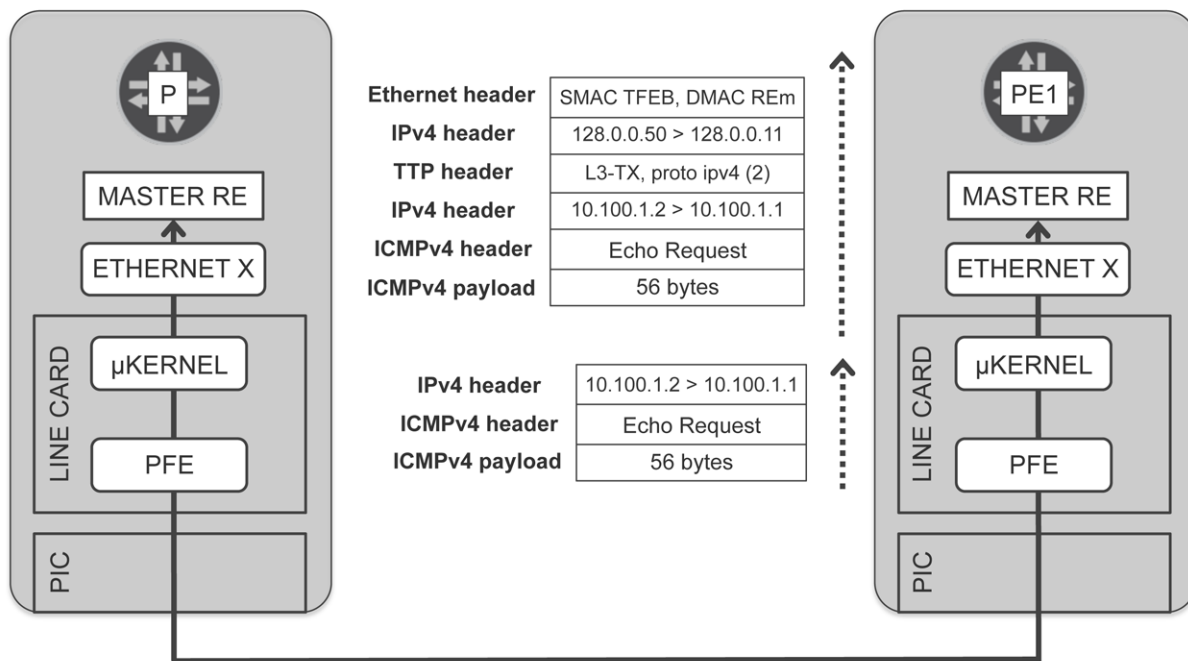


Figure 3.15 Secret Life of an ICMPv4 Echo Request Received at PE1

Try It Yourself: Testing Control Plane Redundancy without a RE Switchover

Open three terminals connected to P backup RE. If RE0 is master, then RE1 is backup:

```
user@PE-re0> request routing-engine login other-routing-engine
```

At terminals #1 and #2, leave running, respectively:

```
user@P-re1> monitor traffic interface em0 no-resolve size 2000 matching icmp
```

```
user@P-re1> monitor traffic interface em1 no-resolve size 2000 matching icmp
```

At terminal #3, execute:

```
user@P-re1> ping 128.0.0.18 routing-instance __juniper_private1__
```

The ICMPv4 echo requests should go out of em1 while the ICMPv4 echo replies come back at em0. But when RE1 becomes master, it will use em0 to send the packets. How can you check to ensure that em0 is bidirectionally healthy before that happens? Going back to terminal #3:

```
user@P-re1> ping 128.0.0.18 routing-instance __juniper_private1__ interface em0 bypass-routing mac-address 02:00:00:00:00:12
```

Try It Yourself: Interpreting PFE Statistics

“In a router *with no transit traffic at all*, these formulas should apply.” You are not running ping, but there is transit traffic going through P – the BGP keepalives in the session between PE1 and PE2!

Try It Yourself: The Private Life of record-route ping

With the record-route option, the TTP statistics increase but the fabric statistics don't. The reason for this is the different forwarding path shown in Figure 3.13.

What's the third difference? PE1-to-PE2 ping takes a longer time to complete, and as a result the pps numbers in show pfe statistics are sensibly lower. With the rapid option, ping waits for the echo reply of the current packet to move the next one. It doesn't wait forever: if the reply doesn't arrive within the specified interval (1 second by default), ping moves to the next packet. But as long as the echo reply arrives within the interval, the average combined latency of the echo request and its reply determines the time it takes for ping to complete, and the resulting pps. With the record-route option, the packets need to transit the control plane of P, and that does add up for the latency due to the non-real-time nature of the Junos OS kernel.

Try It Yourself: Selectively Breaking the Control Plane

Just configure a firewall filter that blocks ICMPv4 and apply it at P's lo0.0 in the input direction. At terminal #3, execute:

```
user@P-re0> configure
user@P-re0# set firewall family inet filter BLOCK-ICMP term ICMP from protocol icmp
user@P-re0# set firewall family inet filter BLOCK-ICMP term ICMP then discard
user@P-re0# set firewall family inet filter BLOCK-ICMP term REST then accept
user@P-re0# set interfaces lo0.0 family inet filter input BLOCK-ICMP
user@P-re0# commit and-quit
```

CAUTION Don't forget to remove the filter attachment from lo0.0 when done.

Chapter 4

Classical Network Applications of Ping

<i>Measuring Link Latency.....</i>	<i>86</i>
<i>Measuring Link Reliability.....</i>	<i>87</i>
<i>Packet Size, Fragmentation, and MTU.....</i>	<i>90</i>
<i>Answers to Try It Yourself Sections of Chapter 4</i>	<i>103</i>



You've already read about *what* ping is and *where* it goes; this chapter deals more with *how* it goes: reliably or intermittently? Fast or slow? Big or small? As a whole or in pieces? The ping options covered in this chapter are widely known and used, but frequent assumptions are made about them that are not always true. For example:

- Myth #1: *Intermittent one-hop ping packet loss is an indication of either hardware issues at the endpoints or physical problems in the link.*
- Myth #2: *If ping shows high latency and jitter, then we have an explanation to end-user slow network complaints.*

Let's confront these and other myths in this chapter!

Measuring Link Latency

Many network operators use ping to measure the latency of a WAN link. Let's see how reliable this is. Execute at PE1:

```
user@PE1> ping 10.100.1.2 source 10.100.1.1 interval 0.1 count 100
PING 10.100.1.2 (10.100.1.2): 56 data bytes
64 bytes from 10.100.1.2: icmp_seq=0 ttl=64 time=0.704 ms
64 bytes from 10.100.1.2: icmp_seq=1 ttl=64 time=0.553 ms
64 bytes from 10.100.1.2: icmp_seq=2 ttl=64 time=31.200 ms
[...]

--- 10.100.1.2 ping statistics ---
100 packets transmitted, 100 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.513/3.150/54.792/7.904 ms
```

TIP For latency tests, always make sure you use the `source` or `interface` options, just in case you have `default-address-selection` configured.

The `time` values displayed are for the Round Trip Time (RTT) between the time the echo request is sent, and the echo reply is received. So, assuming symmetrical routing, which isn't always the case, the unidirectional latency is typically half of the `time`.

As you can see, there is a high level of jitter. These latency variations are actually introduced by the Control Plane components of the sender and receiver, especially by the Routing Engines, which run non-real-time FreeBSD kernels. Due to the forwarding architecture in Junos OS devices, this extra delay introduced by the Control Plane does not affect end-user transit traffic at all, so you don't need to worry that much about ping jitter.

The average latency can even go higher up if there is a lot of control or exception traffic queued in the PFE-to-host path. Typical scenarios where this can happen are aggressive RE-based traffic sampling, or firewall filter `log/syslog/reject` actions. The ICMP echo requests/reply compete with this other exception traffic and may be impacted with higher latency or even discards.

TIP Look for input/output drops in `show pfe statistics traffic`, which indicate the health of the communication channel between Control and Forwarding planes.

So is ping useless for link latency measurements? Not at all. In big international networks with high latency links, it is a great tool to use. Just take the minimum value rather than the average/maximum and you'll have a rough idea of the real link latency.

CAUTION If you use ping to measure latency in multi-hop paths, load balancing comes into play and you may be looking at different paths at the same time. Also, beware of asymmetrical routing.

MORE? M/MX/T-Series routers with Multi-Services PIC/DPC, in addition to the EX-Series, support hardware timestamping of ping flows, in the context of a functionality called Real-Time Performance Monitoring (RPM). This allows for accurate latency measurements.

Measuring Link Reliability

Scenario: You are told that the remaining link between PE1 and P is having some issues. End users report traffic loss and slow Internet connectivity, and their traffic is transiting that link. So you decide to run a stressing ping test to check whether the link drops packets or not:

```
user@P> clear firewall all
```

```
user@PE1> ping 10.100.1.2 source 10.100.1.1 rapid count 1000
/* output suppressed */
--- 10.100.1.2 ping statistics ---
1000 packets transmitted, 1000 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.398/0.787/55.515/2.575 ms
```

So far so good, but keep raising the count until you see drops. The test should last at least five seconds in order to start seeing drops. During this writing, the authors started to see drops at around count 30000:

```
user@P> clear firewall all
```

```
user@PE1> ping 10.100.1.2 source 10.100.1.1 rapid count 30000
/* output suppressed */
--- 10.100.1.2 ping statistics ---
30000 packets transmitted, 29988 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.369/0.696/89.295/1.877 ms
```

NOTE The value 30000 is specific to the setup and really depends on the RTT. In this case, all devices are in the same room so the RTT is very low. With high latency links, you may never see drops unless you increase the packet size.

The pattern of the drops is periodic. At some point in time, there is one packet lost every N packets, with a fixed N value. Does this necessarily mean there is an issue with the link? Absolutely not. Execute this at PE1:

```
user@P> show firewall | match request | except " 0"
icmp-echo-request-ge-2/3/0.111-i          2520000          30000
```

```
user@P> show firewall | match reply | except " 0"
icmp-echo-reply-ge-2/3/0.111-o           2518992          29988
```

Here the packets are discarded inside P! Let's look for the cause of the drops. If your P router has Trio Line Cards and is running Junos OS 11.2 or higher, it has a feature enabled by default called *Distributed Denial of Service* (DDoS) Protection. It implements control and exception traffic policers at different levels of the host-bound path, both at the Line Cards and at the Routing Engine.

Let's check to see how the ICMPv4 policers are programmed and what their statistics are during a stressing ping. Execute at P and PE1:

```
user@P> show ddos-protection protocols icmp
Protocol Group: ICMP

Packet type: aggregate (Aggregate for all ICMP traffic)
Aggregate policer configuration:
  Bandwidth:      20000 pps
  Burst:          20000 packets
  Recover time:   300 seconds
  Enabled:        Yes
System-wide information:
  Aggregate bandwidth is never violated
  Received: 36036          Arrival rate: 1099 pps
  Dropped: 0              Max arrival rate: 1178 pps
Routing Engine information:
  Bandwidth: 20000 pps, Burst: 20000 packets, enabled
  Aggregate policer is never violated
  Received: 493942         Arrival rate: 1187 pps
  Dropped: 0              Max arrival rate: 1362 pps
  Dropped by individual policers: 0
FPC slot 1 information:
  Bandwidth: 100% (20000 pps), Burst: 100% (20000 packets), enabled
  Aggregate policer is never violated
  Received: 0              Arrival rate: 0 pps
  Dropped: 0              Max arrival rate: 0 pps
  Dropped by individual policers: 0
FPC slot 2 information:
  Bandwidth: 100% (20000 pps), Burst: 100% (20000 packets), enabled
  Aggregate policer is never violated
  Received: 36036          Arrival rate: 1099 pps
  Dropped: 0              Max arrival rate: 1178 pps
  Dropped by individual policers: 0
```

NOTE Here icmp stands for ICMPv4. There is a separate protocol group called icmpv6.

As you can see, the Max arrival rate reported is below the policer Bandwidth, hence the ping drops are not caused by the DDoS Protection feature. There must be something else.

Try It Yourself: DDoS Protection Protocols

DDoS Protection classifies control and exception traffic into protocols and protocol groups. Launch ping from PE1 to PE2, and see how the ICMPv4 echo requests are classified at P in the following cases: *normal* transit; with record-route; TTL expiring at P; and Destination Unreachable generated at P.

In this case the drops are caused by an additional ICMPv4-specific policer implemented at the Routing Engine kernel. This policer, more aggressive by default, was implemented in Junos OS earlier than the DDoS Protection feature. Execute at PE1 and at P:

```
user@PE1> show system statistics icmp | match limit
0 drops due to rate limit

user@P> show system statistics icmp | match limit
12 drops due to rate limit

user@P> start shell

% sysctl -a | grep icmp.bucket
```

```
net.inet.icmp.bucketsize: 5
% sysctl -a | grep icmp.token
net.inet.icmp.tokenrate: 1000
% exit
```

CAUTION Using the freeBSD shell is unsupported. The idea is to show the default policer values only: bucketsize in seconds and tokenrate in packets per second.

You can see above that P dropped a few incoming echo requests.

TIP Also look for drops in `show interfaces queue`, `show pfe statistics traffic`, `show pfe statistics ip icmp`, `show system statistics ttp`, `show system queues`, and full buffers in `show system buffers`. They can point to a bottleneck in the internal forwarding path within the routers, and may definitely account for ping packet loss.

For the sake of testing only, let's raise the policing rate at P:

```
user@P> configure
user@P# set system internet-options icmpv4-rate-limit packet-rate 10000
user@P# commit and-quit

user@PE1> ping 10.100.1.2 source 10.100.1.1 rapid count 100000
/* output suppressed */
--- 10.100.1.2 ping statistics ---
100000 packets transmitted, 100000 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.363/0.697/92.908/1.875 ms
```

These rate limiters are a protection measure and you configured a different rate just to see the feature in action. But in production routers, it's better to leave the default (or lower) values in place unless careful fine tuning is required and tested.

Try It Yourself: Rate Limiting ICMPv4 Packets

Find a way to rate limit the incoming ICMPv4 packets addressed to P to 100kbps. You should apply a global policer to the aggregate ICMPv4 packets, regardless of the input interface. Don't use DDoS Protection or the global kernel ICMPv4 policer.

MORE? If you want to learn about protecting the Routing Engine in a Junos OS device, have a look at the *Day One: Securing the Routing Engine on M, MX, and T Series*, available at <http://www.juniper.net/dayone>.

The bottom line is: keep using ping to measure link reliability but take your time to interpret the drop statistics. Not every packet discarded in a one-hop ping is the cause of poor link quality.

ALERT! Don't use ping to measure a production link bandwidth. Some people like to launch several aggressive pings in parallel, with a big packet size. This is simply a bad idea: it stresses the endpoints more than the link itself!

Try It Yourself: Data Patterns

Certain link issues show up only with certain bit patterns. This is especially true in synchronous transport technologies like SONET/SDH. Send ICMPv4 echo requests with the following bit patterns: all 0's, all 1's, alternating 0 and 1. Also try the following patterns, typically used by manufacturing teams: 0xd6bd6b (MOD3), 0x294294 (inverse MOD3), 0x8080 (lonely_1 8), 0x8000 (lonely_1 16), 0x7F7F (lonely_0 8), 0x7FFF (lonely_0 16). Are ICMPv4 echo replies reflecting the patterns? Think of ping as an *echo* request.

Packet Size, Fragmentation, and MTU

When end users report *slow* connectivity to network applications, the reasons behind it can be varied: packet drops, packet reordering, latency, jitter, and more. One of the most typical causes occurs when a host sends a packet with size *S* to a destination, and at some point on the path there is a link whose Maximum Transmission Unit (MTU) is smaller than *S*.

NOTE Due to the possible encapsulations or decapsulations along the path, the value of *S* may vary on a hop-by-hop basis.

At this point, the behavior is completely different in IPv4 and IPv6. Before looking at the implementation details, let's configure different MTUs in the single-uplink topology depicted in Figure 4.1. Execute at PE1:

```
user@PE1> show interfaces ge-1/0/0 | match link.*mtu
Link-level type: Ethernet, MTU: 1518, Speed: 1000mbps, BPDU Error: None, MAC-REWRITE Error: None,

user@PE1> show interfaces ge-1/0/0.111 | match mtu
Protocol inet, MTU: 1500
Protocol iso, MTU: 1497
Protocol inet6, MTU: 1500
Protocol mpls, MTU: 1488, Maximum labels: 3
Protocol multiservice, MTU: Unlimited

user@PE1> configure
user@PE1# set interfaces ge-1/0/0 mtu 1528
user@PE1# commit and-quit

user@PE1> show interfaces ge-1/0/0 | match link.*mtu
Link-level type: Ethernet, MTU: 1528, Speed: 1000mbps, BPDU Error: None, MAC-REWRITE Error: None,

user@PE1> show interfaces ge-1/0/0.111 | match mtu
Protocol inet, MTU: 1510
Protocol iso, MTU: 1507
Protocol inet6, MTU: 1510
Protocol mpls, MTU: 1498, Maximum labels: 3
Protocol multiservice, MTU: Unlimited
```

In order to make MTUs symmetric, execute at P:

```
user@P> configure
user@P# set interfaces ge-2/3/0 mtu 1528
user@P# commit and-quit
```

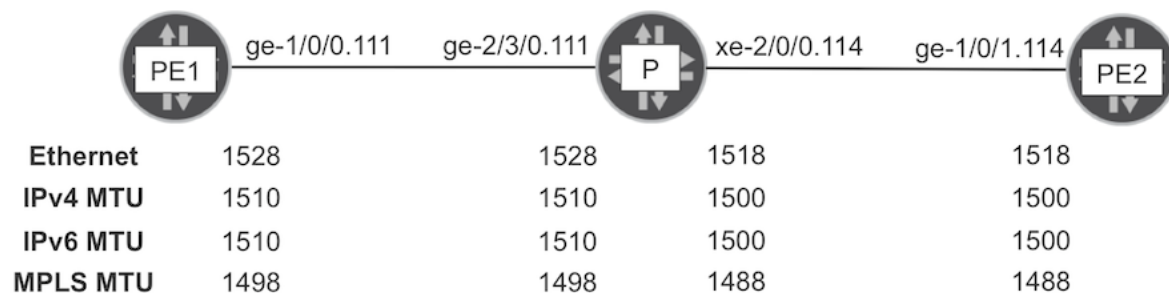


Figure 4.1 Scenario With Mixed MTU Values

The MTU is an unidirectional concept: it can have different values at each end of a link. However, the effective MRU (Maximum Receive Unit) is typically derived from the MTU, so it is much better to set it symmetrically.

A very important concept is the Path MTU (PMTU); it's the minimum MTU of the links making up an end-to-end path. In this case, the IPv4/v6 MTU of PE1-to-P link is 1510 bytes, but the PE1-to-PE2 Path MTU (PMTU) is 1500 bytes.

One of the most frequent applications of ping is manually determining the Path Maximum Transmission Unit (PMTU) between two endpoints.

IPv4 Packet Size Tests

Unlike IPv6, the IPv4 specification (RFC 791) allows for packet fragmentation by a transit router, and not only by the originating host. On the other hand, packet reassembly is always performed by the receiving host.

In general, one important aspect of fragmentation is that load balancing in the network can cause multiple fragments of the same packet to follow different paths. This can result in fragments arriving at the receiving host out of order, which means a buffer is required to hold the out-of-order fragments before reassembling them.

As a first task, adapt the ICMP filter at P to deal with IPv4 fragments:

```
user@P> configure
user@P# edit firewall family inet filter ICMP
user@P# set term FRAGMENTS from is-fragment
user@P# set term FRAGMENTS from protocol icmp
user@P# set term FRAGMENTS then count icmp-fragments
user@P# insert term FRAGMENTS before term ECHO-REQUEST
user@P# commit and-quit
```

NOTE The `is-fragment` knob matches IPv4 packets with a non-zero fragment offset. So it doesn't match the first fragment of a fragmented packet: you can use `first-fragment` in that case.

Apply this filter at PE2 as well:

```
user@PE2> configure
user@PE2# /* ADD YOUR ICMP FILTER HERE */
user@PE2# set interfaces ge-1/0/1 unit 114 family inet filter input ICMP
user@PE2# set interfaces ge-1/0/1 unit 114 family inet filter output ICMP
user@PE2# commit and-quit
```

The ping `size` option specifies the number of bytes in the ICMP payload. In the case of IPv4, you need to add 8 and 20 bytes for the ICMPv4 and IPv4 headers, respectively.

NOTE If the IPv4 header has options, then it's longer than 20 bytes. This is the case if you use the `record-route` option.

Since the PE1-to-PE2 Path MTU is 1500 bytes, the maximum value of the `size` option without fragmentation is $1500 - 20 - 8 = 1472$. Let's go 1 byte above:

```
user@P> clear firewall all

user@PE1> ping 10.111.2.2 count 1 size 1473
PING 10.111.2.2 (10.111.2.2): 1473 data bytes
1481 bytes from 10.111.2.2: icmp_seq=0 ttl=63 time=1.114 ms
```

```

--- 10.111.2.2 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 1.114/1.114/1.114/0.000 ms

```

Note that 1481 bytes is including the 8-byte ICMPv4 header. Check the firewall counters at P:

```

user@P> show firewall | match request | except " 0"
icmp-echo-request-ge-2/3/0.111-i          1501          1
icmp-echo-request-xe-2/0/0.114-o          1501          1

user@P> show firewall | match reply | except " 0"
icmp-echo-reply-ge-2/3/0.111-o           1500          1
icmp-echo-reply-xe-2/0/0.114-i           1500          1

user@P> show firewall | match fragment | except " 0"
icmp-fragments-ge-2/3/0.111-o             21            1
icmp-fragments-xe-2/0/0.114-i             21            1

```

Check the firewall counters at PE2:

```

user@PE2> show firewall | match request | except " 0"
icmp-echo-request-ge-1/0/1.114-i          1500          1

user@PE2> show firewall | match reply | except " 0"
icmp-echo-reply-ge-1/0/1.114-o           1501          1

user@PE2> show firewall | match fragment | except " 0"
icmp-fragments-ge-1/0/1.114-i             21            1

```

In Trio, and in virtually all the M/MX/T-series PFE chipsets, the packets are fragmented after the output firewall filter is evaluated. So, even though P's counter `icmp-echo-request-xe-2/0/0.114-o` and PE2's counter `icmp-echo-reply-ge-1/0/1.114-o` say they are sending a 1501-byte IPv4 packet, that value exceeds the MTU and the packet will be fragmented later before leaving the router. The input filter at the other end is more trustworthy.

Figures 4.2 and 4.3 show the complete flow that you can reconstruct with your *video camera* at H. You can check Capture 4.1 for details. Remember that `default-address-selection` is configured.

TIP By default, Wireshark reassembles the IPv4 fragments in the view. Don't forget to disable it by unselecting: Edit -> Preferences -> Protocols -> IPv4 -> Reassemble fragmented IPv4 datagrams.

TRY THIS Look at the IPv4 ID of the echo request fragments. Is it the same in both? Also have a look at the IPv4 flags and at the fragment offset field.

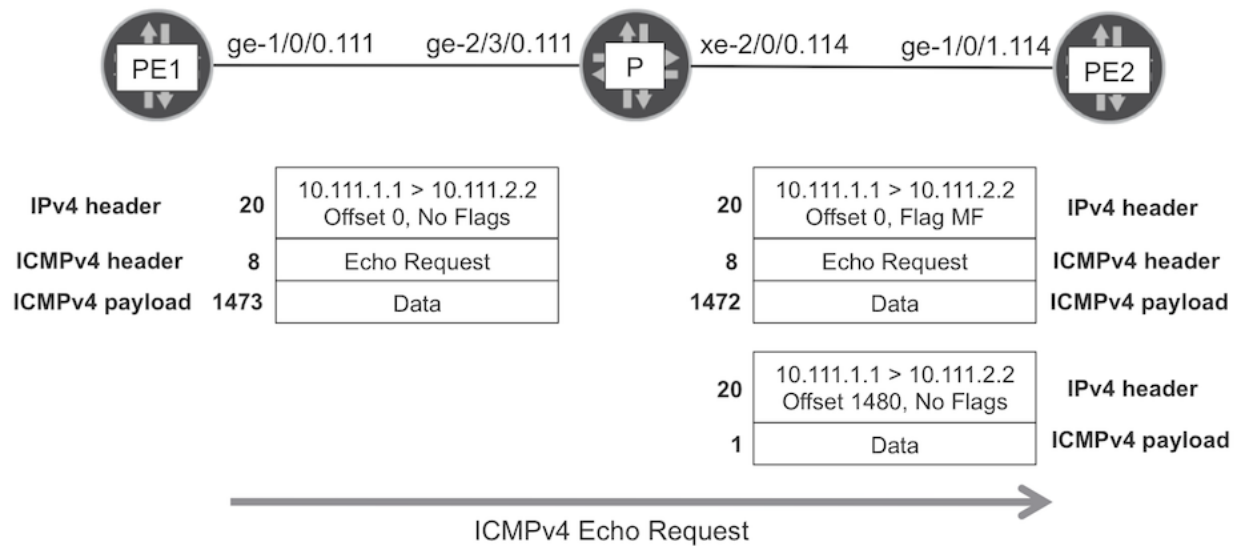


Figure 4.2 Fragmentation of ICMPv4 Echo Request in Transit

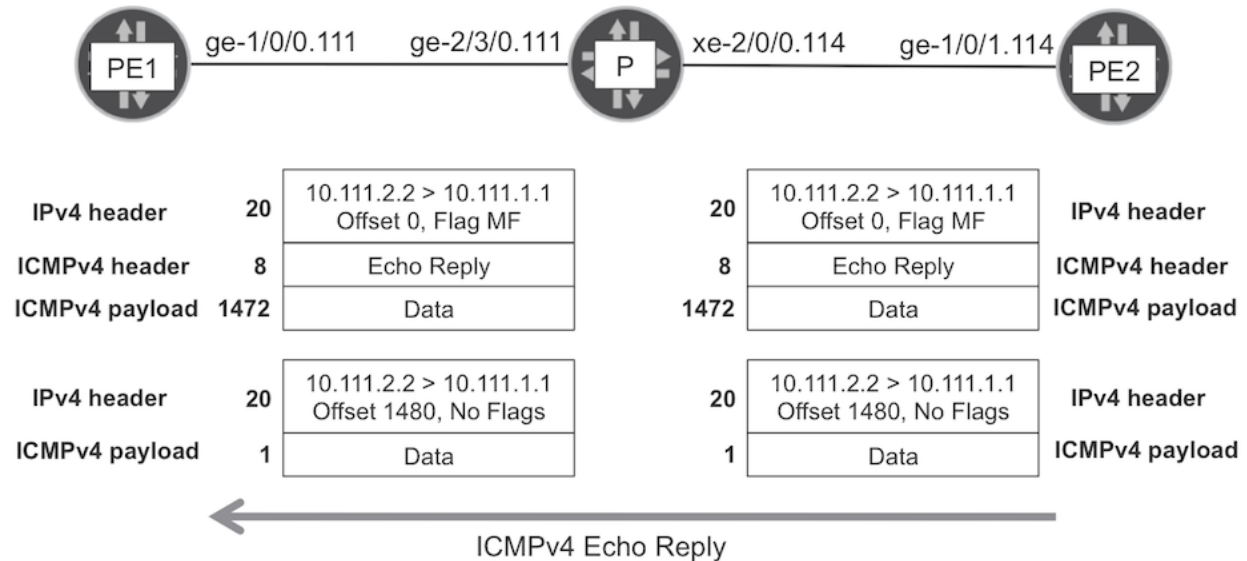


Figure 4.3 Fragmentation of ICMPv4 Echo Reply at the Source

Your previous ping worked because P and PE2 were able to fragment the ICMPv4 echo request and reply, respectively. But some end-user applications set the DF (Don't Fragment) flag in the IPv4 header. You can simulate that with the do-not-fragment option:

```
user@P> clear firewall all
```

```
user@PE2> clear firewall all
```

```

user@PE1> ping 10.111.2.2 count 1 size 1473 do-not-fragment no-resolve
PING 10.111.2.2 (10.111.2.2): 1473 data bytes
36 bytes from 10.100.1.2: frag needed and DF set (MTU 1500)
Vr HL TOS Len ID Flg off TTL Pro cks Src Dst
4 5 00 05dd 8c6c 2 0000 40 01 90d3 10.111.1.1 10.111.2.2

```

```

--- 10.111.2.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss

```

```

user@P> show firewall | match ^icmp- | except " 0"
icmp-echo-request-ge-2/3/0.111-i          1501          1
icmp-unreachable-ge-2/3/0.111-o           56            1
icmp-echo-request-xe-2/0/0.114-o         1501          1

```

```

user@PE2> show firewall | match ^icmp- | except " 0"

```

```

user@PE2>

```

As you know, the `icmp-echo-request-xe-2/0/0.114-o` counter is updated before fragmentation is attempted. However, the ICMPv4 echo request is never forwarded to PE2, as shown in Figure 4.4. Check Capture 4.2 for details.

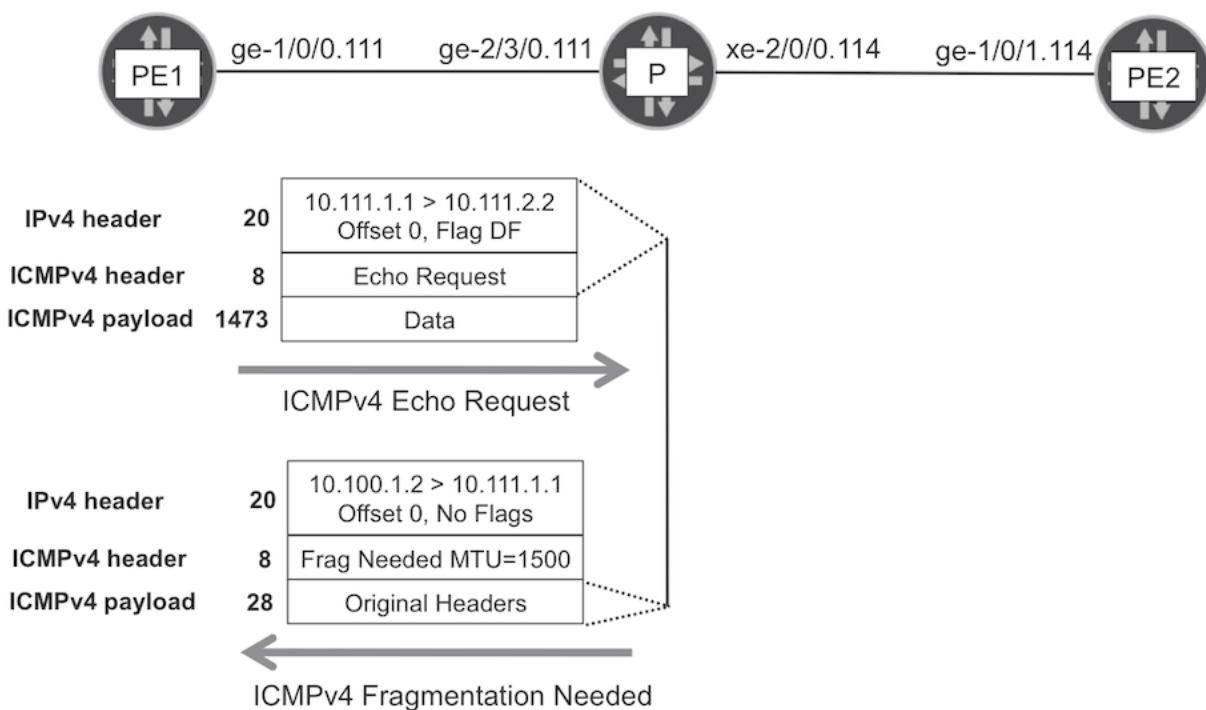


Figure 4.4 Too Big ICMPv4 Echo Request with DF Flag Discarded by Transit IPv4 Router

Let's remove the DF flag and increase the IPv4 packet size to over 1510 bytes, the PE1-P link MTU. In order to send a 1511-byte IPv4 packet, the ping size value is 1483:

```
user@P> clear firewall all
```

```
user@PE2> clear firewall all
```

```
user@PE1> ping 10.111.2.2 size 1483 count 1
PING 10.111.2.2 (10.111.2.2): 1483 data bytes
1491 bytes from 10.111.2.2: icmp_seq=0 ttl=63 time=1.144 ms
```

```
--- 10.111.2.2 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 1.144/1.144/1.144/0.000 ms
```

Check the firewall counters at P:

```
user@P> show firewall | match request | except " 0"
icmp-echo-request-ge-2/3/0.111-i          1508          1
icmp-echo-request-xe-2/0/0.114-o          1508          1

user@P> show firewall | match reply | except " 0"
icmp-echo-reply-ge-2/3/0.111-o            1500          1
icmp-echo-reply-xe-2/0/0.114-i            1500          1

user@P> show firewall | match fragment | except " 0"
icmp-fragments-ge-2/3/0.111-i             23           1
icmp-fragments-ge-2/3/0.111-o             31           1
icmp-fragments-xe-2/0/0.114-i             31           1
icmp-fragments-xe-2/0/0.114-o             23           1
```

Check the firewall counters at PE2:

```
user@PE2> show firewall | match request | except " 0"
icmp-echo-request-ge-1/0/1.114-i          1500          1

user@PE2> show firewall | match reply | except " 0"
icmp-echo-reply-ge-1/0/1.114-o            1511          1

user@PE2> show firewall | match fragment | except " 0"
icmp-fragments-ge-1/0/1.114-i             51           2
```

As you can see in Capture 4.3 and Figure 4.5, this time the ICMPv4 echo request is fragmented twice: once at the source (PE1), and one more time at a transit router (P). The ICMPv4 echo reply is only fragmented by the source (PE2) this time: similar to Figure 4.3 but with 11 bytes in the last fragment's ICMPv4 payload.

NOTE The first IPv4 fragment in the PE1-P link has 1508 bytes instead of 1510. The reason is that the Fragment Offset in the IPv4 header is measured in units of 8 bytes, so the first fragment size must be an exact multiple of that quantity.

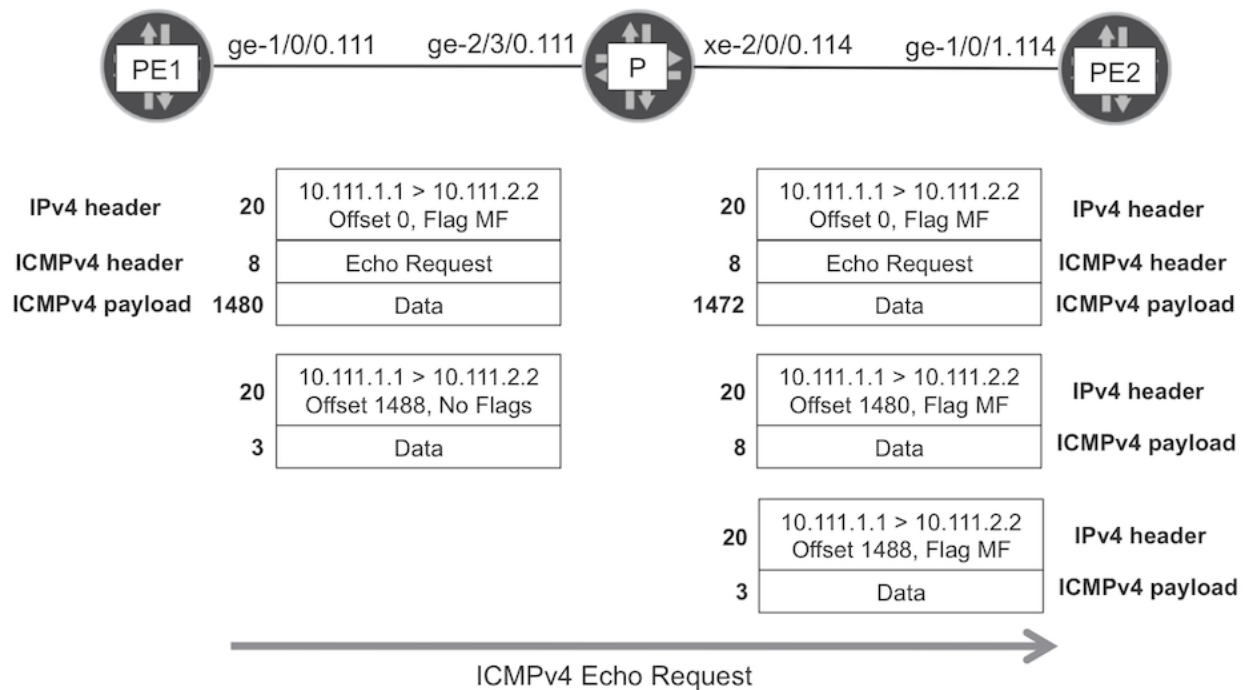


Figure 4.5 Fragmentation of ICMPv4 Echo Request Both at the Source and in Transit

CAUTION Try to avoid using unrealistically high size values. One lost fragment is enough to keep all the other fragments waiting in the reassembly buffer of the destination. This is a classical DoS attack! Of course, you can guard against it with several techniques: DDoS Protection, IP fragment policers in 100.0 filters, etc.

Finally, try to send from PE1 an IPv4 packet bigger than PE1-P link MTU, but with the DF flag:

```
user@PE1> ping 10.111.2.2 size 1483 count 1 do-not-fragment
PING 10.111.2.2 (10.111.2.2): 1483 data bytes
ping: sendto: Message too long

--- 10.111.2.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

Naturally, no packet is sent out of PE1 this time.

IPv6 Packet Size Tests

Unlike IPv4, the IPv6 specification (RFC 2460) does not allow for packet fragmentation by transit routers. Only the originating host is allowed to fragment packets. But, how does it know the maximum fragment size? By using a Path MTU Discovery mechanism, described in RFC 1981. Let's see how it works.

First, configure IPv6 addresses in the links. Don't worry about IPv6 routing: IS-IS is automatically advertising the prefixes.

```

user@PE1> configure
user@PE1# set interfaces ge-1/0/0 unit 111 family inet6 address fc00::100:1:1/112
user@PE1# commit and-quit

```

```

user@P> configure
user@P# set interfaces ge-2/3/0 unit 111 family inet6 address fc00::100:1:2/112
user@P# set interfaces xe-2/0/0 unit 114 family inet6 address fc00::100:4:2/112
user@P# commit and-quit

```

```

user@PE2> configure
user@PE2# set interfaces ge-1/0/1 unit 114 family inet6 address fc00::100:4:1/112
user@PE2# commit and-quit

```

The sizes of the IPv6 packet and ICMPv6 headers are 40 and 8 bytes, respectively. In order to send an IPv6 with exactly the inet6 MTU of the PE1-P link in bytes, the size value should be: $1510 - 40 - 8 = 1462$ bytes.

With the *video camera* on, execute at PE1:

```

user@PE1> ping fc00::100:4:1 size 1462 count 1
PING6(1510=40+8+1462 bytes) fc00::100:1:1 --> fc00::100:4:1
1240 bytes from fc00::100:1:2: Packet too big mtu = 1500
Vr TC Flow Plen Nxt Hlim
 6 00 00000 05be 3a 40
fc00::100:1:1->fc00::100:4:1
ICMP6: type = 128, code = 0

```

```

--- fc00::100:4:1 ping6 statistics ---
1 packets transmitted, 0 packets received, 100% packet loss

```

This first IPv6 ping just triggered PMTU Discovery, as you can see in Capture 4.4 and in Figure 4.6. PE1 just learned about the existence of a remote link in the path with a lower inet6 MTU = 1500 than the local link. The next packet to fc00::100:4:1 will be sent taking this information into account.

Let's execute again at PE1:

```

user@PE1> ping fc00::100:4:1 size 1462 count 1
PING6(1510=40+8+1462 bytes) fc00::100:1:1 --> fc00::100:4:1
1470 bytes from fc00::100:4:1, icmp_seq=0 hlim=63 time=1.352 ms

--- fc00::100:4:1 ping6 statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/std-dev = 1.352/1.352/1.352/0.000 ms

```

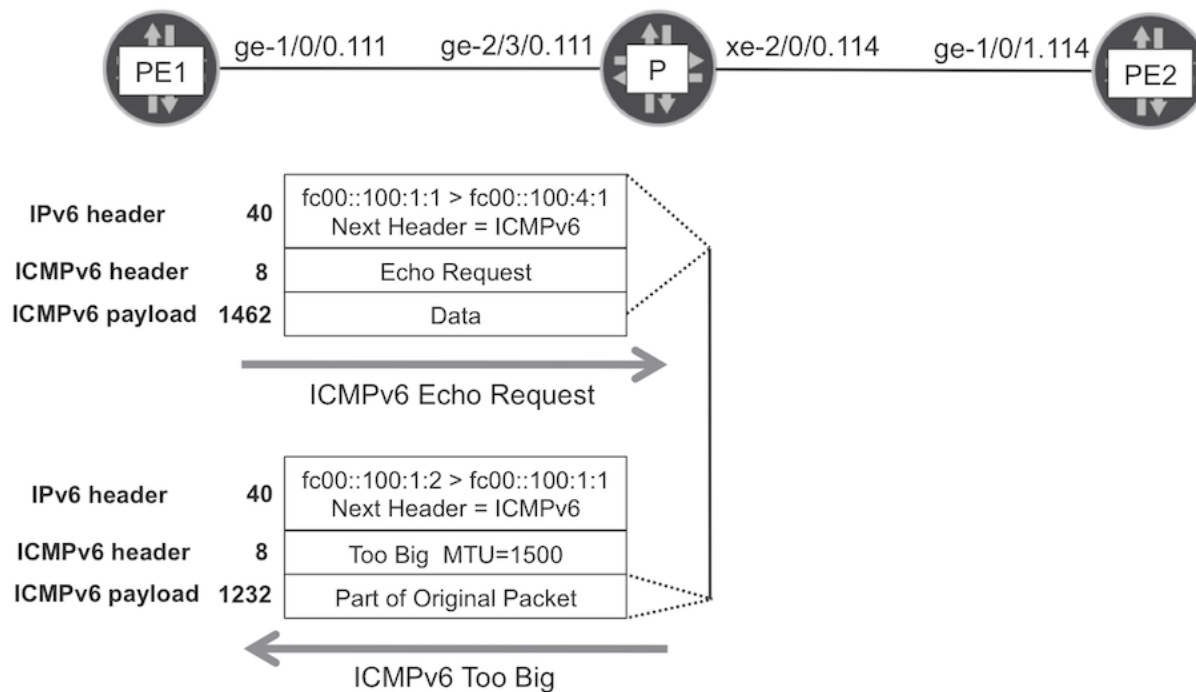


Figure 4.6 Too Big ICMPv6 Echo Request Discarded by Transit IPv6 Router

This time PE1 has enough information to make the ping succeed. Figure 4.7 shows the ICMPv6 echo request. The ICMPv6 echo reply flow is very similar. You can see all packets at each stage in Capture 4.5.

TIP By default, Wireshark reassembles the IPv6 fragments in the view. Don't forget to disable it by unselecting: Edit -> Preferences -> Protocols -> IPv6 -> Reassemble fragmented IPv6 datagrams.

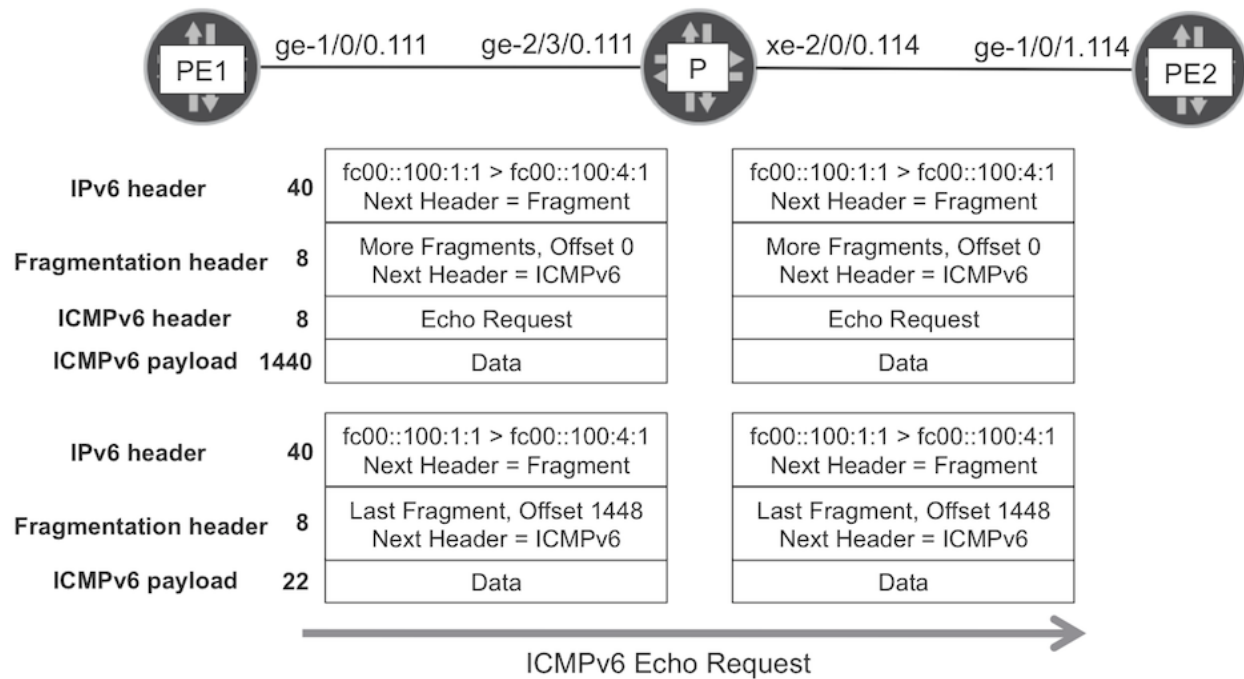


Figure 4.7 Fragmentation of ICMPv6 Echo Request at the Source

If, in another terminal connected to PE1, you launch tcpdump to capture the ping flow, you should see something like this:

```
user@PE2> monitor traffic interface ge-1/0/1.114 size 2000 no-resolve matching inet6
[...]
<timestamp> In IP6 fc00::100:1:1 > fc00::100:4:1: frag (0|1448) ICMP6, echo request, seq 0, length 1448
<timestamp> In IP6 fc00::100:1:1 > fc00::100:4:1: frag (1448|22)
<timestamp> Out IP6 fc00::100:4:1 > fc00::100:1:1: frag (0|1448) ICMP6, echo reply, seq 0, length 1448
<timestamp> Out IP6 fc00::100:4:1 > fc00::100:1:1: frag (1448|22)
```

NOTE If you use the matching `icmp6` keyword instead, only the first fragments (0|1448) show up.

Note that there is no DF flag in IPv6. Even if the do-not-fragment option exists, it makes no change in the IPv6 packet so there is no point in using it.

Table 4.1 New ICMP Types and Codes (seen in this chapter)

ICMPv4 (IP Protocol 1)			ICMPv6 (IPv6 Next Header 58)		
Type	Code	Description	Type	Code	Description
3	4	Destination Unreachable, Fragmentation Needed	2	0	Too Big

MPLS Packet Size Tests

The MPLS protocol does not provide any fragmentation support by itself. Network designers should make sure that whenever a MPLS packet enters the backbone, it already has the right size to reach the egress PE.

The way MPLS MTU is implemented in Junos OS is a bit complex, as the headers depend on a number of factors, including the label operation (push/swap/pop) being performed, in addition to whether the family `mpls mtu` is explicitly configured (Flags: User-MTU) or derived from physical MTU. In practice, make sure the value of the header is big enough to fit both the largest possible IPv4/IPv6 packet and the maximum number of MPLS labels expected to be in the stack.

Let's start with a fragmentation example at the Ingress PE (PE1). Turn on the *video camera* at H and execute at PE1:

```
user@PE1> ping 10.2.2.2 routing-instance CE1 size 1472 count 1 do-not-fragment no-resolve
PING 10.2.2.2 (10.2.2.2): 1472 data bytes
36 bytes from 10.1.1.1: frag needed and DF set (MTU 1498)
Vr HL TOS Len ID Flg off TTL Pro cks Src Dst
4 5 00 05dc 45b8 2 0000 40 01 d862 10.1.1.2 10.2.2.2
```

```
--- 10.2.2.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

The behavior is shown in Figure 4.8. Doesn't it look familiar? Check Capture 4.6 for details.

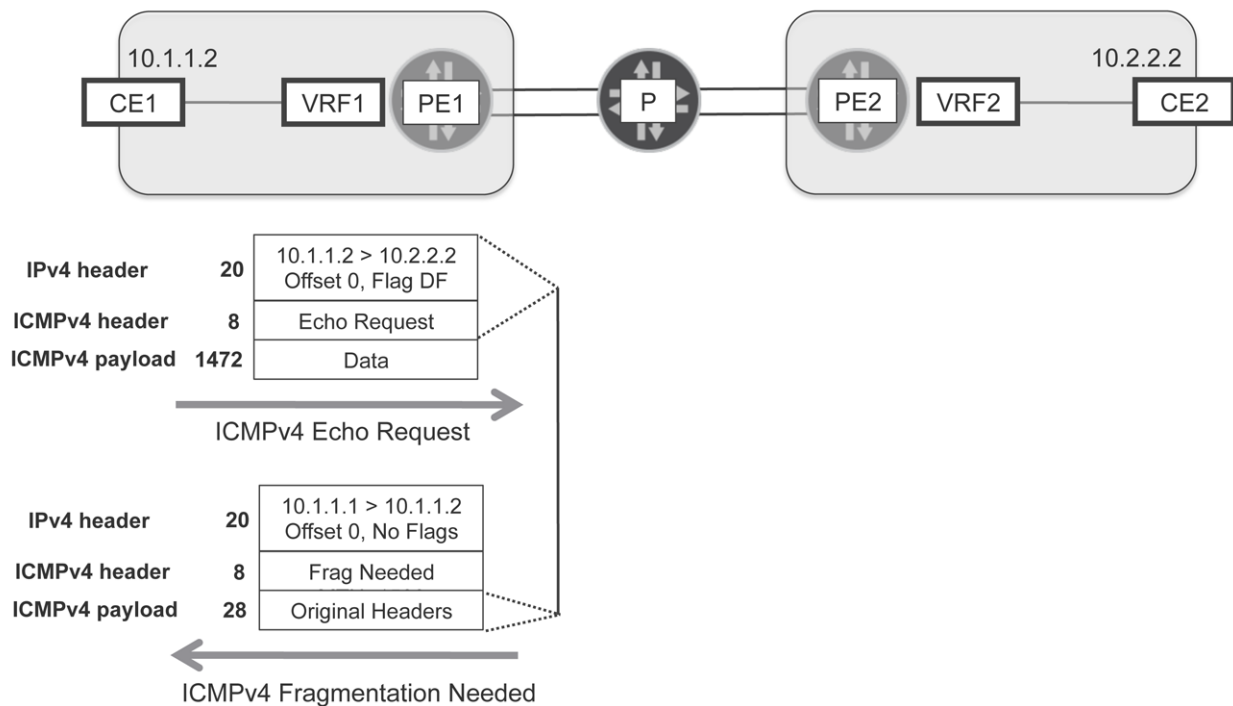


Figure 4.8 Too Big ICMPv4 Echo Request with DF Flag Discarded by Ingress MPLS PE

Now remove the DF flag and try again:

```
user@PE1> ping 10.2.2.2 routing-instance CE1 size 1472 count 1
PING 10.2.2.2 (10.2.2.2): 1472 data bytes
1480 bytes from 10.2.2.2: icmp_seq=0 ttl=61 time=1.286 ms

--- 10.2.2.2 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 1.286/1.286/1.286/0.000 ms
```

In this case, PE1 is fragmenting the ICMPv4 echo request at the IPv4 layer. The two fragments are then encapsulated into MPLS packets, as shown in Figure 4.9. In Capture 4.7 you can see both the echo request and the reply. Try to draw a figure like Figure 4.9 with the latter!

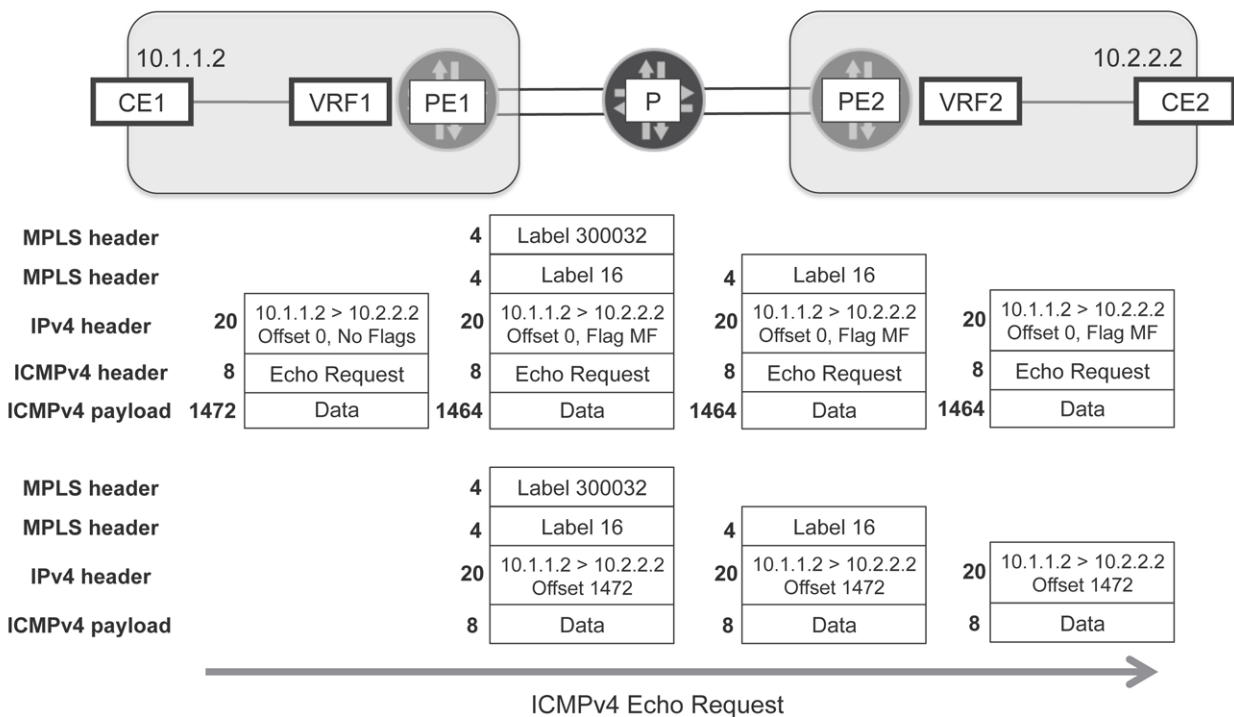


Figure 4.9 Fragmentation of ICMPv4 Echo Request at the Ingress MPLS PE

There is (at least) one interesting scenario still left to consider. Imagine an ICMPv4 echo request that perfectly fits on its way to the destination, but whose echo reply requires fragmentation on its way back. If the DF flag is set in the echo request, it should also be set (echoed) in the echo reply. You can guess the outcome.

Execute at PE1:

```
user@PE1> ping 10.2.2.2 routing-instance CE1 size 1462 count 1 do-not-fragment no-resolve
PING 10.2.2.2 (10.2.2.2): 1462 data bytes

--- 10.2.2.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

The flow is shown in Figures 4.10 and 4.11. Capture 4.8 has the whole *movie*.

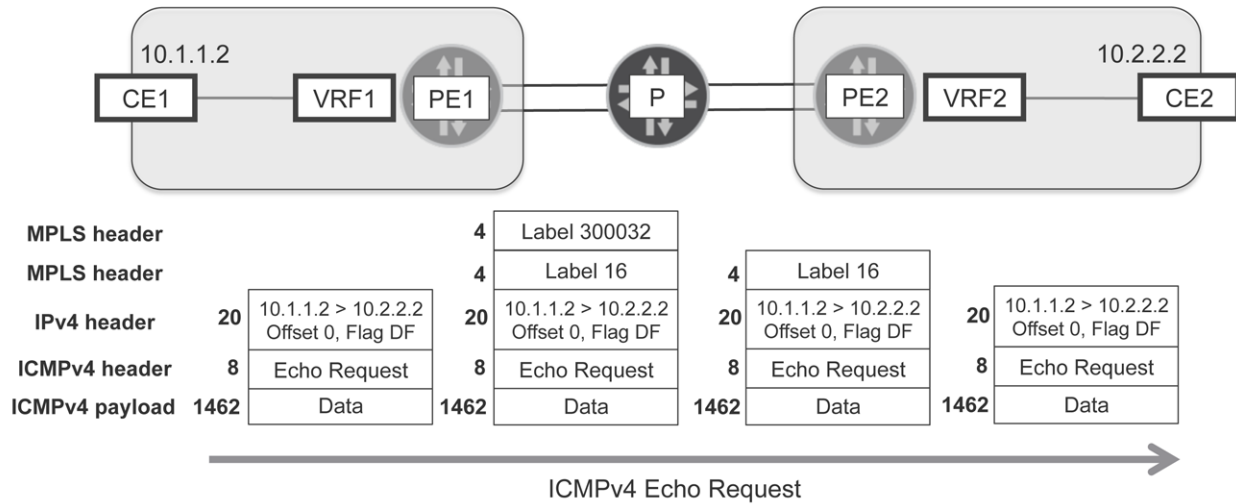


Figure 4.10 ICMPv4 Echo Request with DF Flag Reaching the Destination

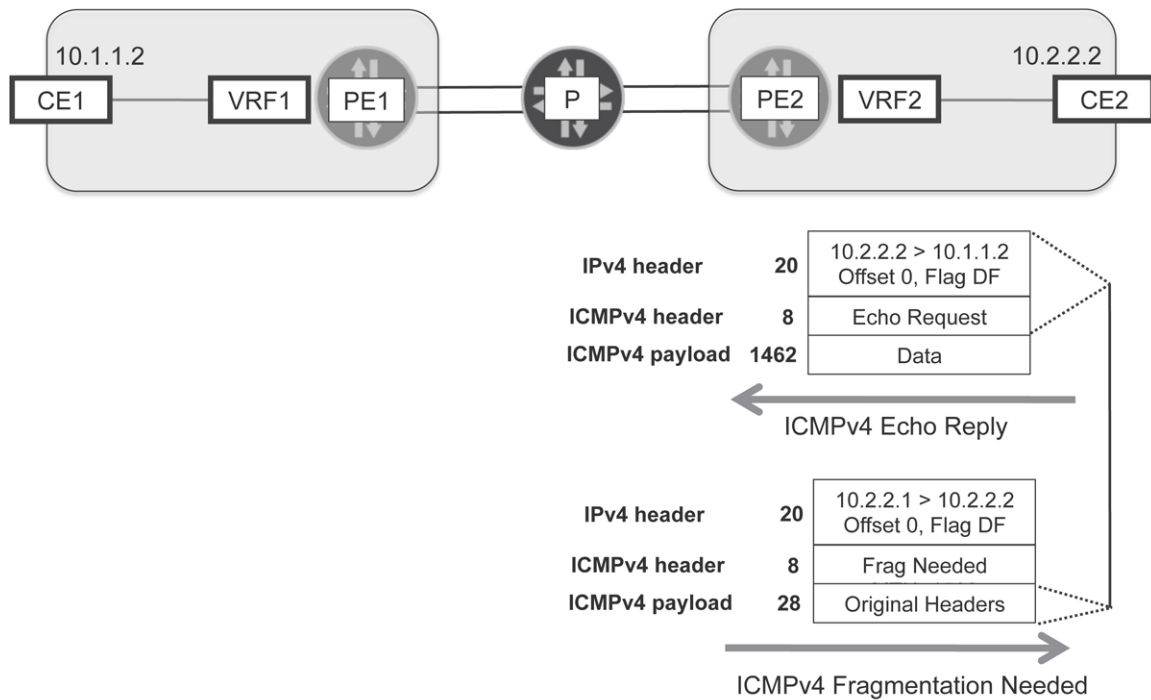


Figure 4.11 ICMPv4 Echo Reply with DF Flag Dropped by Ingress MPLS PE

MORE? Check RFC 3209, section 2.6, for an explanation on how RSVP implements Path MTU Discovery. All the RFCs are viewable at <https://datatracker.ietf.org/>

Answers to Try It Yourself Sections of Chapter 4

Try It Yourself: Rate Limiting ICMPv4 Packets

Find a way to rate limit the incoming ICMPv4 packets without using DDoS Protection or the global kernel ICMPv4 policer.

```

user@P> configure
user@P# edit firewall
user@P# set policer 100K if-exceeding bandwidth-limit 100k burst-size-limit 10k
user@P# set policer 100K then discard
user@P# set family inet filter PROTECT-RE term ICMP from protocol icmp
user@P# set family inet filter PROTECT-RE term ICMP then policer 100K
user@P# set family inet filter PROTECT-RE term ICMP then count lo0-ICMP
user@P# set family inet filter PROTECT-RE term ICMP then accept
user@P# set family inet filter PROTECT-RE term REST then count lo0-REST
user@P# set family inet filter PROTECT-RE term REST then accept
user@P# top
user@P# set interfaces lo0.0 family inet filter input PROTECT-RE
user@P# commit and-quit

user@PE1> ping 10.100.1.2 source 10.100.1.1 rapid count 1000
PING 10.100.1.2 (10.100.1.2): 56 data bytes
/* output suppressed */
--- 10.100.1.2 ping statistics ---
1000 packets transmitted, 989 packets received, 1% packet loss
round-trip min/avg/max/stddev = 0.407/0.772/58.697/2.830 ms

```

CAUTION This is not a good filter from a security perspective, since it accepts virtually any packets. But it's fine for the purpose of the test. Note that if the packets are fragmented, the non-first fragments wouldn't match from protocol icmp and wouldn't be evaluated by the policer.

Try It Yourself: DDoS Protection Protocols

With the stressing ICMPv4 ping launched from PE1 to PE2, execute at P the command:

```
user@P> show ddos-protection protocols statistics | match "protocol| arrival" | except " 0 pps"
```

Here are the results:

Normal transit echo requests are not classified in any protocol group, since they are not control or exception traffic from the perspective of P, so they bypass the Control Plane. Command:

```
user@PE1> ping 10.111.2.2 rapid count 30000
```

Transit echo requests with record-route option are classified in protocol group ICMP. Command:

```
user@PE1> ping 10.111.2.2 rapid count 30000 record-route
```

Transit echo requests with TTL expiring at P are classified in protocol group TTL. Command:

```
user@PE1> ping 10.111.2.2 ttl 1 interval 0.1 no-resolve
```

Transit echo requests with Destination Unreachable generated at P are classified in protocol group Reject. This scenario requires extra configuration. Out of all the various options available, a simple one is:

```
user@P> configure
user@P# set routing-options static route 10.111.2.2 reject
user@P# commit and-quit
```

```
user@PE1> ping 10.111.2.2 interval 0.1 no-resolve
```

```
user@P> configure
user@P# delete routing-options static route 10.111.2.2
user@P# commit and-quit
```

Try It Yourself: Data Patterns

ICMPv4 echo replies are echoing the patterns, as expected. Here are the ping commands:

All 0's:

```
user@PE1> ping 10.100.1.2 source 10.100.1.1 pattern 00 count 1 size 1400
```

All 1's:

```
user@PE1> ping 10.100.1.2 source 10.100.1.1 pattern FF count 1 size 1400
```

Alternating 0 and 1:

```
user@PE1> ping 10.100.1.2 source 10.100.1.1 pattern 55 count 1 size 1400
```

```
user@PE1> ping 10.100.1.2 source 10.100.1.1 pattern AA count 1 size 1400
```

The remaining patterns are: d6bd6b, 294294, 8080, 8000, 7F7F, 7FFF.

All the flows are in Capture 4.9.

Chapter 5

One Loop to Rule Them All

<i>Theory of Loops</i>	106
<i>Single-Link Loops</i>	110
<i>Snake Tests at Work</i>	115
<i>Particle Accelerators</i>	123
<i>Class of Service in a Loop</i>	132
<i>Answers to Try It Yourself Sections of Chapter 5</i>	152



In this chapter you are going to dramatically reduce the size and complexity of the lab: you will just keep P.

Imagine your network is experiencing traffic loss and a thorough troubleshooting activity based on ping causes you to conclude that, for whatever reason, a particular router is blackholing traffic. At this point, and if there is redundancy, you should deviate the traffic so that it doesn't transit the suspect router, but do so before restarting any router component, in order not to lose valuable info. A typical first step in a Service Provider network is to set IGP overload bit or high metrics, but this only repels transit core traffic at P-routers – in PE-routers you also need to deactivate BGP sessions and make sure the local router keeps no egress or transit RSVP LSPs active. Once the router is isolated from end-user traffic, you can conduct further troubleshooting for a limited time... but what can you do?

Another frequent scenario occurs when you connect to a remote router to perform some proof-of-concept tests, but you don't know if the remote router is connected to other devices. It could be a lab router, or even a pre-production router that you want to health-check. You are unaware of any back-to-back connections yet the hardware doesn't support tunnel services. Let's also assume there is nobody locally next to the devices to perform the cabling.

If you think there are very few things you can do with the router, continue reading!

Virtualization is a great strategy and the support of Logical Systems, Virtual Routers, and Logical Tunnels with Junos OS makes it very easy to simulate a whole network with many routers and switches interconnected. However, the approach in this book is radically different (and complementary): you can test most Forwarding Plane features and components in a protocol-less scenario. No need to think about addressing schemes, routing protocols or MPLS signaling. Just set one or more ports in loopback mode and you will be able to steer and manipulate traffic at will.

But what generates the traffic? Ping. Elementary!

Here are just a couple of the common misconceptions that are addressed in this chapter:

- Myth #1: *If a router is dropping traffic and you suspect a hardware issue, you are most likely to find the root cause only if you keep the end-user traffic flowing through the router.*
- Myth #2: *You need several (at least logical) routers and a traffic generator to run Class of Service tests.*

Prepare to have these and other myths dispelled in this chapter!

Theory of Loops

This book differentiates between two types of loops:

- In *half-duplex* loops the packets sent out of a port are received on the same port. In other words, transmission (TX) and reception (RX) signals are looped in an individual port.
- On the other hand, *full-duplex* loops involve two ports connected to each other back-to-back: naturally, port A (TX, RX) is connected to port B (RX, TX).

Let's explore each loop in a little more detail.

Half-Duplex Loops

Looping interfaces is a traditional troubleshooting method to verify the integrity of a link. Figure 5.1 shows three half-duplex loopback flavors, from top to bottom:

- **Local software loopback:** the physical interface is configured as `loopback [local]` at R1. The outgoing packets reach the PIC and are returned back to the PFE, where they are looked up and forwarded again. This chapter focuses on this mode.
- **Physical half-duplex loopback:** With the help of optical equipment, the two strands of a fiber are connected to each other. No configuration is required on the routers. The loop can be completed next to R1, next to R2, or anywhere else along the transmission path.
- **Remote software loopback:** physical interface in remote loopback mode at R2 (or at a transmission device in the path). The incoming signal at R2 reaches the PIC and is returned back to the fiber/cable. One copy of the packets is sent to the PFE at R2, so packets can be counted there as well (interface statistics and firewall filter counters).

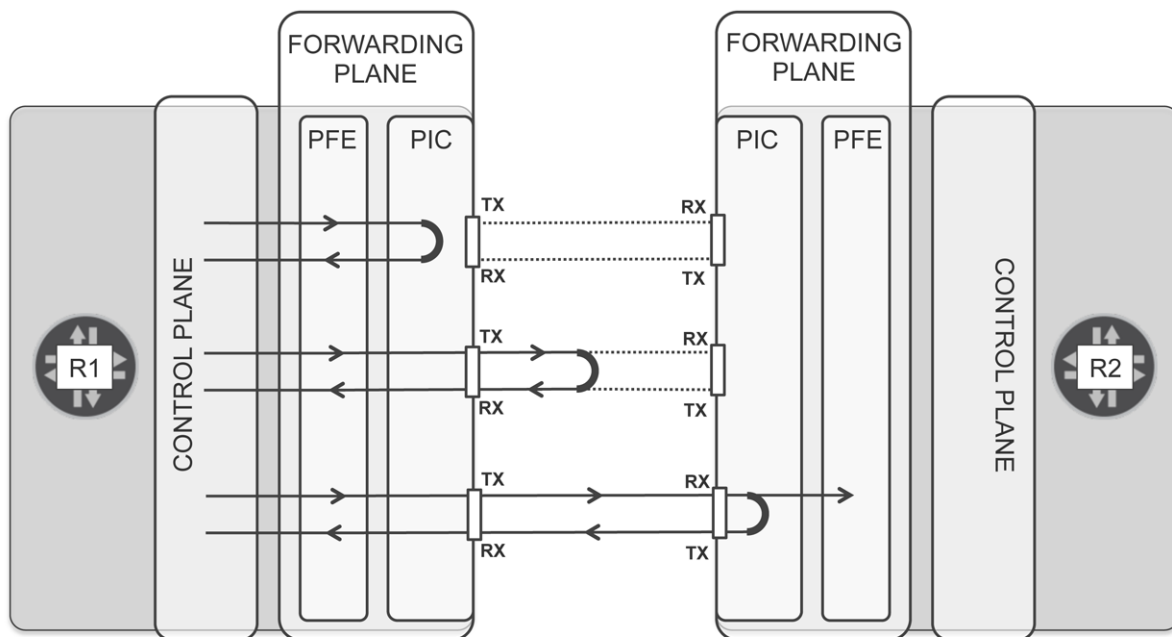


Figure 5.1 Half-duplex Port Loopback Techniques

Due to their great flexibility and low requirements, this book uses local software half-duplex loops as the preferred option for general applications beyond link troubleshooting. It's true, they are not suitable for Control Plane tests involving different Logical Systems or Virtual Routers running routing protocols with each other, but in the context of pure Forwarding Plane tests, they simply rock!

In order to generate traffic, there are two classic networking options. If you ping an IPv4/IPv6 address locally configured at the router's interface, make sure you use the interface and bypass-routing options, otherwise the packet won't leave the Control Plane. If you ping the remote address of the link, the packet will loop until TTL expires. (Both options are covered in detail later in this chapter.)

TIP Some interfaces have keepalives that will detect the loop and bring the interface down. Turn them off to bring the port up. For example, in SONET/SDH interfaces, you can use `encapsulation cisco-hdlc` and `no-keepalive` options.

MORE? You will see an example of ethernet remote loopback later. In SONET/SDH interfaces, you need to configure the interface at router B with `sonet-options loopback remote`.

Full-Duplex Loops

Connecting two ports of the same router back to back is a traditional testing approach, especially in environments supporting virtualization like Junos OS. Figure 5.2 shows three full-duplex loopback flavors, from left to right:

- **Physical back-to-back connection:** This is a regular external connection whose endpoints are in the same physical device, but typically in different Logical Systems (LS) or Virtual Routers (VR).
- **Offline Logical Tunnel:** in pre-Trio PFEs with PICs that support `lt-` interfaces (Tunnel PICs or MX DPCs), you can connect two different `lt-` units and put them in different LS's or VR's.
- **Inline Logical Tunnel:** Trio PFEs implement the `lt-` functionality directly on the PFE. You can offline the MICs, if there is a `lt-` configured it will stay up!

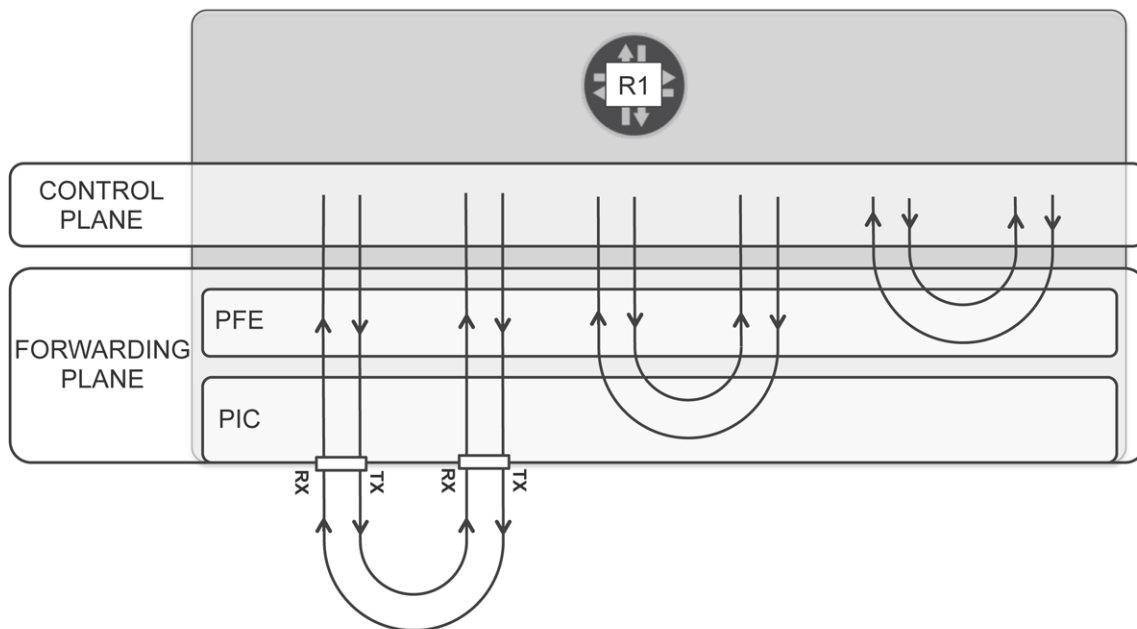


Figure 5.2 Full-duplex Looping Techniques

NOTE Logical Systems have a separate routing process (up to 16) and completely support PE and P router functionality. Virtual Routers, on the other hand, are very much like CE equipment with IPv4/IPv6 features, but no MPLS support.

Full-duplex loops have the flexibility to enable different logical entities to peer with each other. For example, one back-to-back connection with VLANs is enough to create a complex topology with CEs, VRFs, PEs, Ps, etc. Full-duplex loops are a great asset for testing, especially when the focus is on Control Plane.

But do you really need to go full-duplex every time? If you just need to test a Forwarding Plane feature, most often the answer is *no*. And in troubleshooting scenarios, where you are left with a production router for a limited amount of time, half-duplex loops are the way to go because they allow you to test the internal forwarding paths in an unmatched simple way.

Snake Tests

Snakes are test setups where each packet has a long life inside the router, transiting several interfaces and functional components. Traffic is steered in a deterministic way with the help of techniques you are about to peruse.

The classical use case of *snakes* is illustrated in Figure 5.3:

- In the upper and lower part of Figure 5.3, you can see half-duplex and full-duplex tests, respectively. In this chapter, the half-duplex scenarios are built with local software loopback, but physical loops could be an option, too.
- On the left hand side, the snake scenarios are simple in that each packet only transits each link once. On the right hand side, packets can be looped several times at the same interface. Why? Well imagine you have a 1Gbps stream and you want to fill a 10GE or a 100GE interface!

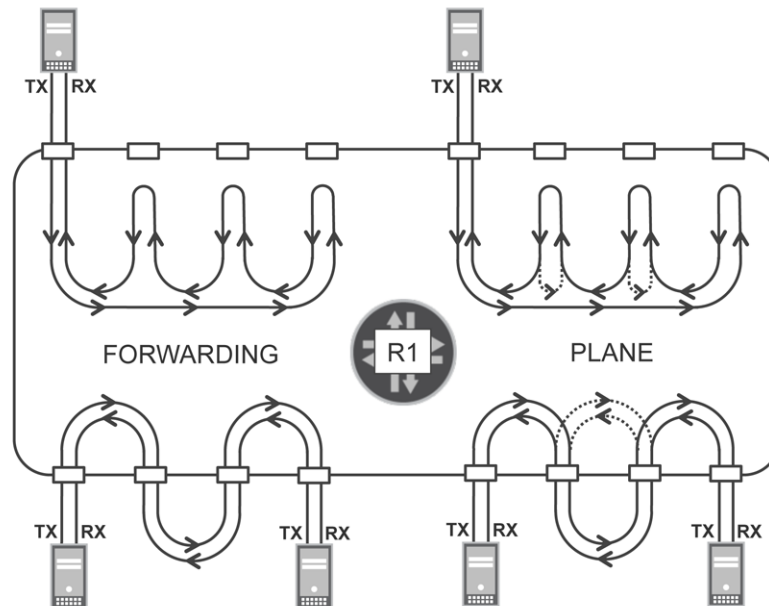


Figure 5.3 Snake Tests With an External Traffic Generator

The traffic steering techniques are varied. With full-duplex scenarios you can group interfaces together in a logical way. If, in Figure 5.3, you number the ports (0,1,2,3) from left to right, port groups (0,1) and (2,3) would belong to different LS's or VR's. At this point, you have a fully featured logical network that you can configure with dynamic or static routing to steer traffic between the endpoints.

This chapter focuses on other strategies that are applicable to both half-duplex and full-duplex loop scenarios. These models are less intuitive at the beginning, but more powerful and flexible when the focus is on the Forwarding Plane.

Actually, there is no external traffic generator in any of the following examples – one single ping executed locally is more than enough. Only when several different streams are needed (for example, different Type of Service values) will you be executing more than one simultaneous instance of ping.

It is relatively common to use full duplex snake setups to verify the capacity of a Forwarding Plane component. If you get 100% throughput, it's easy to interpret. But what if you get 90% of the original packets at the receiver end? Does it mean the component has only 90% of the tested forwarding capacity? Not at all! The packets that jumped some of the coils of the snake and did not reach the end are also consuming resources, and they need to be taken into account, too.

Try It Yourself: Measuring Throughput with Snake Tests

Imagine you are performing a bidirectional snake test like the one on the lower left corner of Figure 5.3. This test involves a 1Gbps flow across twenty different 1GE physical interfaces in the same PFE. You measure a 90% throughput, in other words, a total 10% loss. What is the real percentage of PFE throughput?

TIP Snake tests may involve many links. Verify their integrity individually, watching for framing or link-level errors that could alter the forwarding capacity measurements.

CAUTION Don't underestimate the impact that snakes can have on Forwarding Plane resources, and even on the Control Plane, especially if a high rate of exceptions are generated in the test. In a production router, that will compete with your production traffic!

Single-Link Loops

Let's configure two independent half-duplex loopback scenarios. First, pick two unused ports, if possible each in a different Packet Forwarding Engine (PFE), and configure them in loopback mode. Finally, associate them to different Virtual Routers in order to avoid a potential address overlap with the global routing table. Initially, the two ports are down:

```
user@P> show interfaces ge-2/3/1 | match physical
Physical interface: ge-2/3/1, Enabled, Physical link is Down
```

```
user@P> show interfaces xe-2/0/1 | match physical
Physical interface: xe-2/0/1, Enabled, Physical link is Down
```

Configure them in loopback mode:

```
user@P> configure
user@P# set interfaces ge-2/3/1 vlan-tagging giether-options loopback
user@P# set interfaces ge-2/3/1 unit 201 vlan-id 201 family inet address 10.201.1.1/30
user@P# set interfaces xe-2/0/1 vlan-tagging giether-options loopback
user@P# set interfaces xe-2/0/1 unit 201 vlan-id 201 family inet address 10.201.2.1/30
user@P# set routing-instances VR1 instance-type virtual-router interface ge-2/3/1.201
user@P# set routing-instances VR2 instance-type virtual-router interface xe-2/0/1.201
user@P# commit and-quit
```

```
user@P> show interfaces ge-2/3/1 | match physical
Physical interface: ge-2/3/1, Enabled, Physical link is Up
```

```
user@P> show interfaces xe-2/0/1 | match physical
Physical interface: xe-2/0/1, Enabled, Physical link is Up
```

Send a 100-packet ping to the *local* interface IPv4 address:

```
user@P> clear interfaces statistics all

user@P> ping 10.201.1.1 routing-instance VR1 rapid count 100
PING 10.201.1.1 (10.201.1.1): 56 data bytes
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!
--- 10.201.1.1 ping statistics ---
100 packets transmitted, 100 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.036/0.041/0.294/0.026 ms

user@P> show interfaces ge-2/3/1.201 statistics | match packets
Input packets : 0
Output packets: 0
```

At this point the counters should be at zero. If not, the interface is either sending routing protocol hellos, management traffic or ARP requests to resolve the next hop of a static route.

TIP Each time you do loopback tests, don't forget to temporarily disable the port at the protocol's configuration, and also unneeded address families, or else it's tough to interpret the results, especially if you want to measure packet drop rate.

Why are the counters at zero? As you might remember from Figure 2.1, these packets never make it to the Forwarding Plane. You need to do something else to achieve that:

```
user@P> ping 10.201.1.1 bypass-routing interface ge-2/3/1.201 mac-address 00:23:9c:9a:d3:8b rapid
count 100
PING 10.201.1.1 (10.201.1.1): 56 data bytes
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!
--- 10.201.1.1 ping statistics ---
100 packets transmitted, 100 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.286/0.454/5.710/0.747 ms

user@P> show interfaces ge-2/3/1.201 statistics | match packets
Input packets : 100
Output packets: 100
```

Repeat this procedure for the other interface, in this case xe-2/0/1.201. You can see the forwarding path in Figure 5.4.

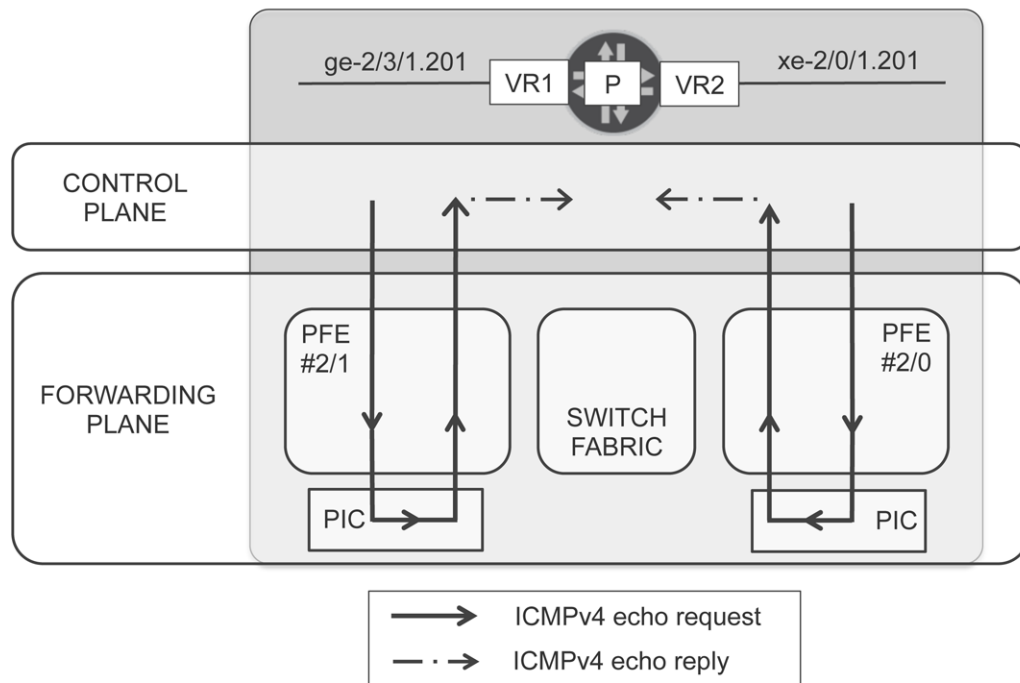


Figure 5.4 Forwarding Path of Ping bypass-routing to the Local IPv4 Address of a Half-duplex Loop

IMPORTANT

This is *not* genuine transit traffic. When the ICMPv4 echo request is received from the interface stream, it goes up to the Control Plane.

What if you ping a remote IPv4 address of the link?

```
user@P> show interfaces ge-2/3/1 | match hardware
Current address: 00:23:9c:9a:d3:8b, Hardware address: 00:23:9c:9a:d3:8b

user@P> show interfaces xe-2/0/1 | match hardware
Current address: 00:23:9c:9a:d2:95, Hardware address: 00:23:9c:9a:d2:95

user@P> clear interfaces statistics all

user@P> ping 10.201.1.2 bypass-routing interface ge-2/3/1.201 mac-address 00:23:9c:9a:d3:8b count 1
PING 10.201.1.2 (10.201.1.2): 56 data bytes

--- 10.201.1.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss

user@P> show interfaces ge-2/3/1.201 statistics | match packets
Input packets : 6
Output packets: 6
```

On another terminal, perform a packet capture during the one-packet ping:

```
user@P> monitor traffic interface ge-2/3/1.201 no-resolve size 2000
verbose output suppressed, use <detail> or <extensive> for full protocol decode
Address resolution is OFF.
Listening on ge-2/3/1.201, capture size 2000 bytes

<timestamp> Out IP 10.201.1.1 > 10.201.1.2: ICMP echo request, id 43079, seq 0, length 64
<timestamp> Out arp who-has 10.201.1.2 tell 10.201.1.1
<timestamp> In arp who-has 10.201.1.2 tell 10.201.1.1
```

```

<timestamp> Out arp who-has 10.201.1.2 tell 10.201.1.1
<timestamp> In arp who-has 10.201.1.2 tell 10.201.1.1
<timestamp> Out arp who-has 10.201.1.2 tell 10.201.1.1
<timestamp> In arp who-has 10.201.1.2 tell 10.201.1.1
<timestamp> Out arp who-has 10.201.1.2 tell 10.201.1.1
<timestamp> In arp who-has 10.201.1.2 tell 10.201.1.1
<timestamp> Out arp who-has 10.201.1.2 tell 10.201.1.1
<timestamp> In arp who-has 10.201.1.2 tell 10.201.1.1

```

Let's reconstruct the movie. The mac-address option allows the Control Plane to send the ICMPv4 echo request, but after the first loop trip, it's the Forwarding Plane's turn. ARP resolution fails, and the packet is finally discarded. Here is the fix:

```

user@P> configure
user@P# set interfaces ge-2/3/1.201 family inet address 10.201.1.1/30 arp 10.201.1.2 mac
00:23:9c:9a:d3:8b
user@P# set interfaces xe-2/0/1.201 family inet address 10.201.2.1/30 arp 10.201.2.2 mac
00:23:9c:9a:d2:95
user@P# commit and-quit

```

NOTE Dynamic ARP resolution works fine in full duplex loops, so you don't need static ARP there. This is a clear advantage for you to consider.

Check the outcome of the fix:

```

user@P> clear interfaces statistics all

user@P> ping 10.201.1.2 no-resolve routing-instance VR1 count 1 ttl 100
PING 10.201.1.2 (10.201.1.2): 56 data bytes
36 bytes from 10.201.1.1: Time to live exceeded
Vr HL TOS Len ID Flg off TTL Pro cks Src Dst
 4 5 00 0054 259f 0 0000 01 01 7c76 10.201.1.1 10.201.1.2

--- 10.201.1.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss

user@P> show interfaces ge-2/3/1.201 statistics | match packets
Input packets : 100
Output packets: 100

user@P> ping 10.201.2.2 no-resolve routing-instance VR2 count 1 ttl 100
PING 10.201.2.2 (10.201.2.2): 56 data bytes
36 bytes from 10.201.2.1: Time to live exceeded
Vr HL TOS Len ID Flg off TTL Pro cks Src Dst
 4 5 00 0054 300b 0 0000 01 01 700a 10.201.2.1 10.201.2.2

--- 10.201.2.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss

user@P> show interfaces xe-2/0/1.201 statistics | match packets
Input packets : 100
Output packets: 100

```

In this test, you get real transit traffic going through the loops, as you can see in Figure 5.5.

The Control Plane sends the ICMPv4 echo request for the first time. At that point, every time the packet is received from the loop (with TTL > 1) the Forwarding Plane decreases its TTL by one just before again sending it to the loop. This is done without involving the Control Plane at all, except for the final time, when TTL expires and the transit packet becomes an exception.

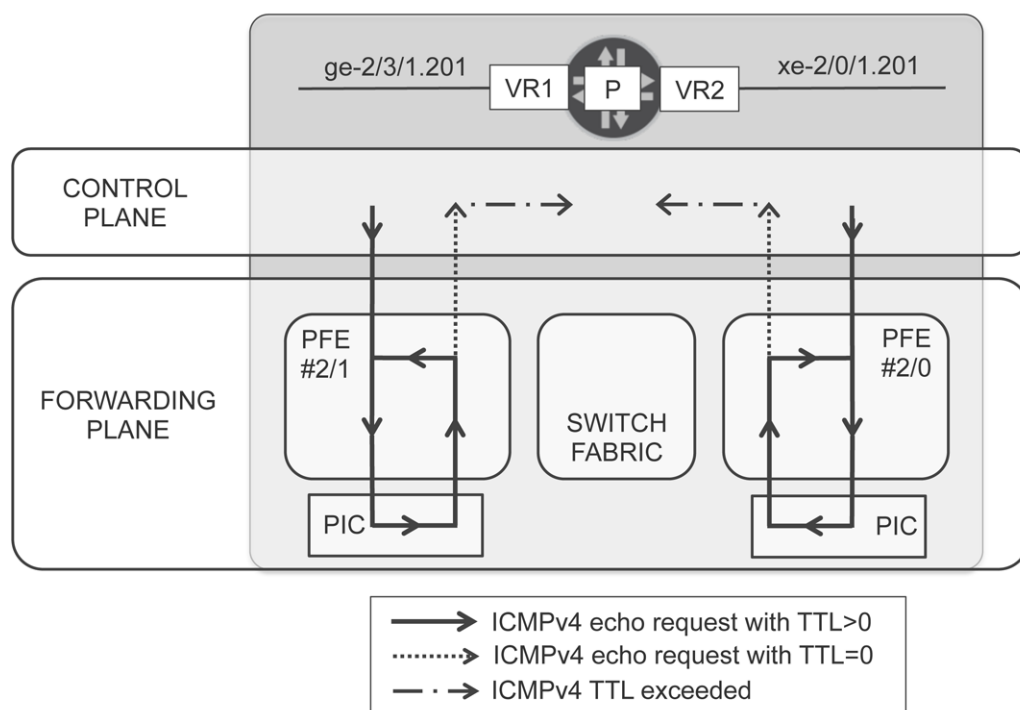


Figure 5.5 Forwarding Path of Ping to the Remote IPv4 Address of a Half-duplex Loop

From the point of view of the ping application, receiving a time exceeded message is considered a failure. However, it's the expected result of this particular test: the packet was looped 100 times! So let's consider it a success. On the other hand, one echo request is typically not enough to check the integrity of the path. Let's send 1000 echo requests, and see if you get 1000 time exceeded messages in exchange:

```
user@P> show system statistics icmp | match "put|exceed"
Input Histogram
104 time exceeded
```

```
user@P> ping 10.201.1.2 no-resolve routing-instance VR1 count 1000 ttl 100 interval 0.1
PING 10.201.1.2 (10.201.1.2): 56 data bytes
/* Output suppressed */
--- 10.201.1.2 ping statistics ---
1000 packets transmitted, 0 packets received, 100% packet loss
```

```
user@P> show system statistics icmp | match "put|exceed"
Input Histogram
1104 time exceeded
```

This time, the time exceeded counter increased by 1000, which means there was zero packet loss and the link passed the integrity check!

CAUTION

The time exceeded counter is global for the system, so it may increase for other reasons not related to your test – it only accounts for received ICMPv4 time exceeded messages at the RE.

TIP Line Cards typically have circular buffers. If you want to spot faulty buffer sectors, you need to leave the ping running a long time with interval 0.1, big size, and no count value. Do it from the console if you want to avoid session disconnections!

TRY THIS In certain conditions (visit Table 3.5), intra-PFE transit traffic is looped through the switch fabric. You can easily adapt the previous test to include the fabric in the transit path.

Try It Yourself: Remote Loopback Mode

Configure PE1 and the X switch so that each packet sent by PE1 at ge-1/0/0 is looped by X ge-0/0/11 back to PE1. Tip: look for *Configuring Ethernet LFM with Loopback Support* in the Junos technical guides: <http://www.juniper.net/techpubs>.

Snake Tests at Work

You are about to create a two-loop snake with two different strategies: IPv4 and MPLS static paths (you can apply the same techniques discussed in this section to create a snake with an arbitrary number of loops). As mentioned previously, it's more interesting to choose two ports in different PFEs, so that's what we'll do. Chapter 3 contains tips on how to map PFEs to ports.

Also revisit Chapter 3 for reminders about checking interface, PFE, and fabric statistics at each step in order to verify the accuracy of the forwarding path figures.

Plain IPv4 Half-Duplex Snake Tests

This test relies on chaining Virtual Routers: for example, a snake of five loops would consist of this chain: VR1 -> VR2 -> VR3 -> VR4 -> VR5 -> VR1. For simplicity, let's just take two Virtual Routers so the chain becomes: VR1 -> VR2 -> VR1. If that is puzzling it should be much clearer soon!

Configure the snake logic at P:

```
user@P> configure
user@P# set firewall family inet filter to-VR1 term ALL then routing-instance VR1
user@P# set firewall family inet filter to-VR2 term ALL then routing-instance VR2
user@P# set interfaces ge-2/3/1 unit 201 family inet filter input to-VR2
user@P# set interfaces xe-2/0/1 unit 201 family inet filter input to-VR1
user@P# set routing-instances VR1 routing-options static route 0/0 next-hop 10.201.1.2
user@P# set routing-instances VR2 routing-options static route 0/0 next-hop 10.201.2.2
user@P# commit and-quit
```

In order to understand the way it works, let's increment TTL gradually, starting with TTL=1, as illustrated in Figure 5.6:

```
user@P> ping 10.201.2.2 source 10.201.1.1 no-resolve routing-instance VR1 count 1 ttl 1
PING 10.201.2.2 (10.201.2.2): 56 data bytes
36 bytes from 10.201.1.1: Time to live exceeded
Vr HL TOS Len ID Flg off TTL Pro cks Src Dst
4 5 00 0054 fd16 0 0000 01 01 a3fe 10.201.1.1 10.201.2.2
```

```
--- 10.201.2.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

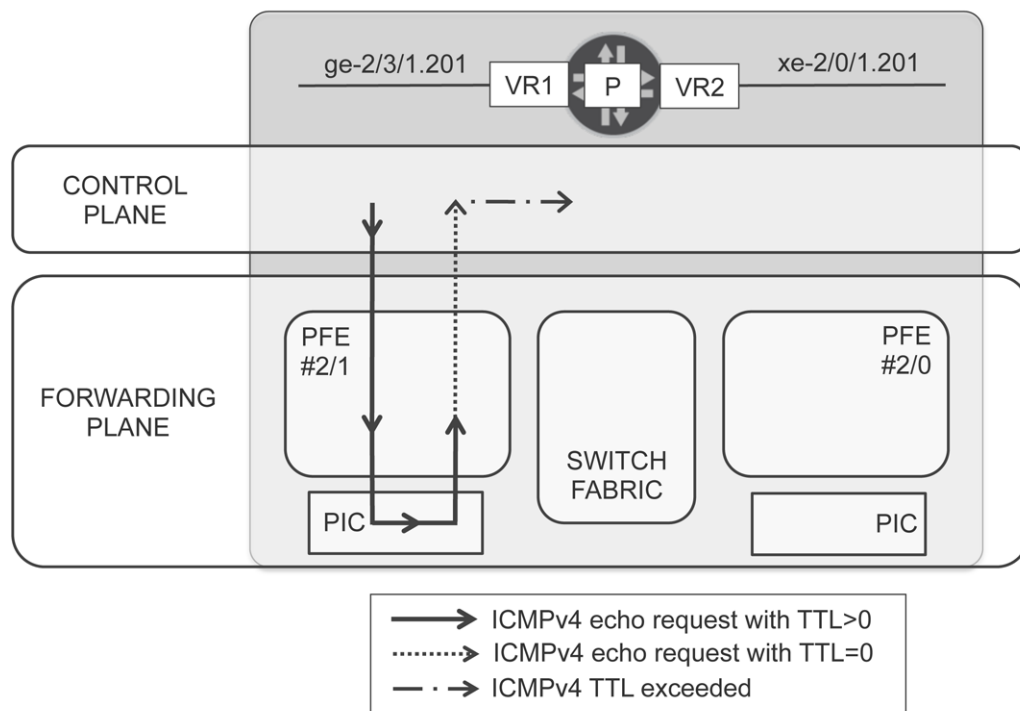


Figure 5.6 Forwarding Path of a Half-duplex IPv4 Snake Ping with TTL=1

Up until now, it wasn't really a snake, but just by increasing TTL to 2, the path becomes more interesting, as shown in Figure 5.7. Let's jump from VR1 to VR2:

```
user@P> ping 10.201.2.2 source 10.201.1.1 no-resolve routing-instance VR1 count 1 tt1 2
PING 10.201.2.2 (10.201.2.2): 56 data bytes
36 bytes from 10.201.2.1: Time to live exceeded
Vr HL TOS Len ID Flg off TTL Pro cks Src Dst
4 5 00 0054 fef1 0 0000 01 01 a223 10.201.1.1 10.201.2.2

--- 10.201.2.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

TIP Always check interface counters to verify the accuracy of the numbers.

As you can see in Figure 5.7, the time exceeded message jumps the other way around, from VR2 to VR1. However, it doesn't transit the fabric. In fact, when PFE #2/0 looks up the packet in the context of VR1, it finds out that the destination address is local and sends the message up to the Control Plane.

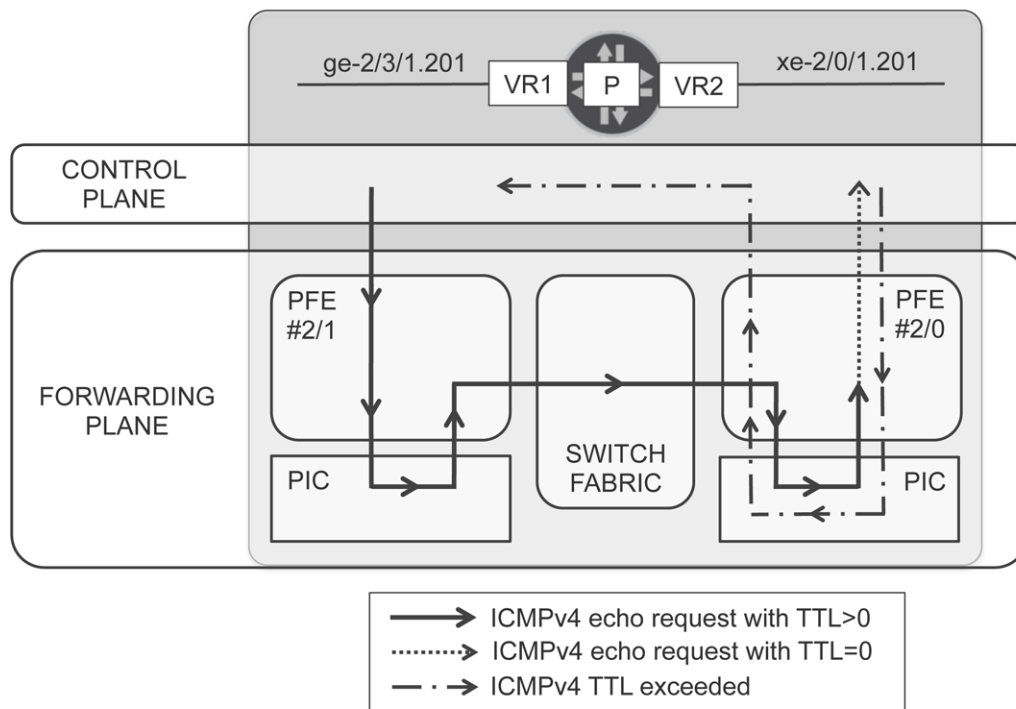


Figure 5.7 Forwarding Path of a Half-duplex IPv4 Snake Ping with TTL=2

Making the ICMPv4 echo request jump additional times is just a matter of increasing the TTL. If you choose an odd TTL value (different from 1), the forwarding path looks like the one in Figure 5.8. For example 99:

```
user@P> ping 10.201.2.2 source 10.201.1.1 no-resolve routing-instance VR1 count 1 ttl 99
PING 10.201.2.2 (10.201.2.2): 56 data bytes
36 bytes from 10.201.1.1: Time to live exceeded
Vr HL TOS Len ID Flg off TTL Pro cks Src Dst
4 5 00 0054 00d2 0 0000 01 01 a043 10.201.1.1 10.201.2.2

--- 10.201.2.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

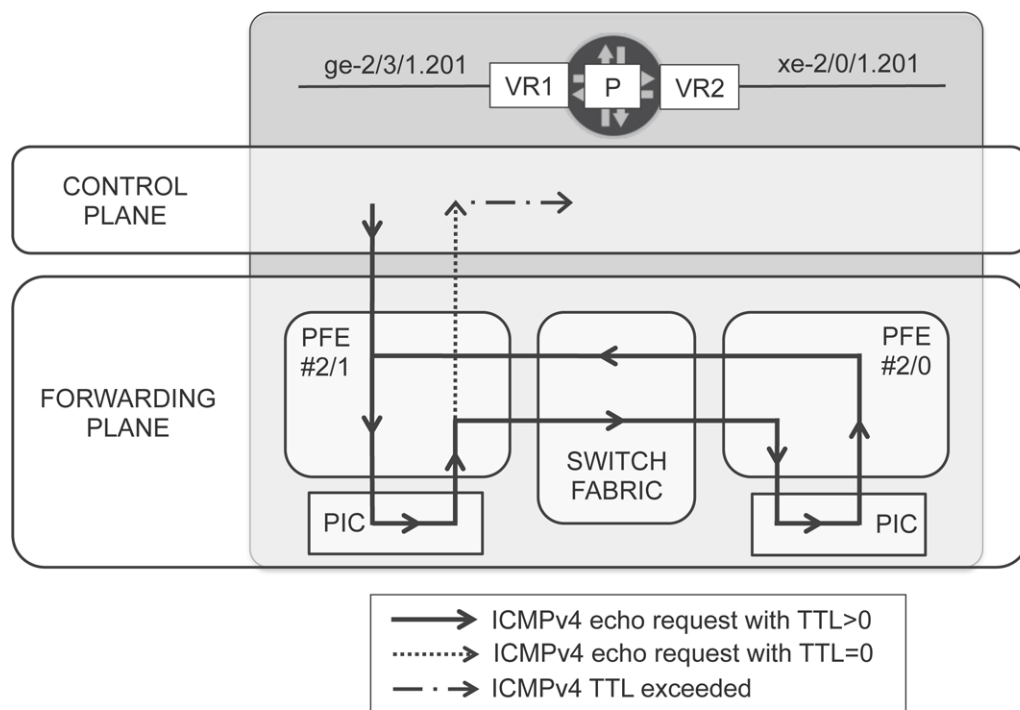


Figure 5.8 Forwarding Path of a Half-duplex IPv4 Snake Ping with an Odd TTL > 2

Or, if you choose an even TTL value different from 2, the outcome is as shown in Figure 5.9. Give it a try:

```
user@P> ping 10.201.2.2 source 10.201.1.1 no-resolve routing-instance VR1 count 1 ttl 100
PING 10.201.2.2 (10.201.2.2): 56 data bytes
36 bytes from 10.201.2.1: Time to live exceeded
Vr HL TOS Len ID Flg off TTL Pro cks Src Dst
4 5 00 0054 02f2 0 0000 01 01 9e23 10.201.1.1 10.201.2.2

--- 10.201.2.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

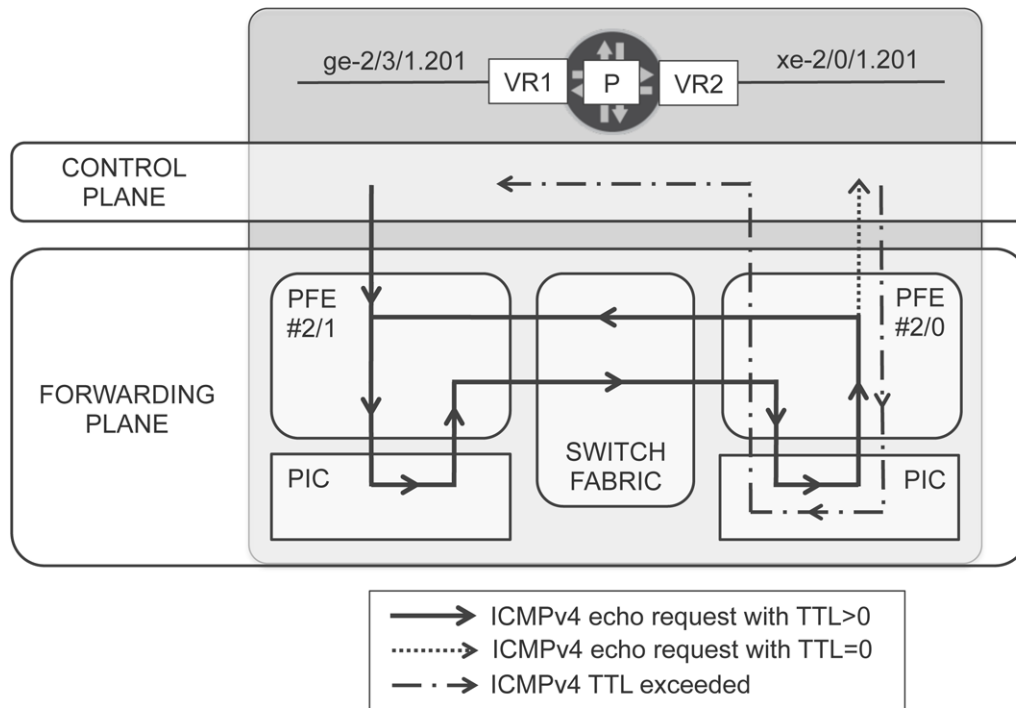


Figure 5.9 Forwarding Path of a Half-duplex IPv4 Snake Ping with an Even TTL > 2

TRY THIS Add another port to the setup and introduce VR3 in the VR chain.

NOTE Starting with Junos OS 12.2, you can use firewall filter `next-interface`, `next-ip` and `next-ip6` options in Trio, as an alternative to using different Virtual Routers.

IPv4 over MPLS Half-Duplex Snake Tests

Steering packets with static MPLS paths is much more intuitive, you just need to make the LSP follow your predetermined path. First, at P, create two new half duplex loops by reusing the existing physical interfaces but this time with a different VLAN ID:

```
user@P> configure
user@P# edit interfaces ge-2/3/1 unit 202
user@P# set vlan-id 202 family mpls
user@P# set family inet address 10.202.1.1/30 arp 10.202.1.2 mac 00:23:9c:9a:d3:8b
user@P# top
user@P# edit interfaces xe-2/0/1 unit 202
user@P# set vlan-id 202 family mpls
user@P# set family inet address 10.202.2.1/30 arp 10.202.2.2 mac 00:23:9c:9a:d2:95
user@P# top
user@P# edit protocols mpls
user@P# set interface ge-2/3/1.202
user@P# set interface xe-2/0/1.202
user@P# commit and-quit
```

Now, configure a 4-hop half duplex snake as follows:

```

user@P> configure
user@P# edit protocols mpls
user@P# set static-label-switched-path SNAKE-A-1 ingress to 10.202.100.1 push 1000001 next-hop
10.202.1.2
user@P# set static-label-switched-path SNAKE-A-2 transit 1000001 swap 1000002 next-hop 10.202.2.2
user@P# set static-label-switched-path SNAKE-A-3 transit 1000002 swap 1000003 next-hop 10.202.1.2
user@P# set static-label-switched-path SNAKE-A-4 transit 1000003 pop next-hop 10.202.2.2
user@P# top
user@P# commit and-quit

```

```

user@P> show route 10.202.100.1 exact

```

inet.3: 5 destinations, 5 routes (5 active, 0 holddown, 0 hidden)

+ = Active Route, - = Last Active, * = Both

```

10.202.100.1/32    *[MPLS/6/1] 00:00:14, metric 0
                  > to 10.202.1.2 via ge-2/3/1.202, Push 1000001

```

You need a route in inet.0 to target your ping. There are several ways to achieve that, let's choose probably the simplest one:

```

user@P> configure
user@P# set routing-options static route 10.202.100.100 next-hop 10.202.100.1 resolve
user@P# commit and-quit

```

```

user@P> show route 10.202.100.100 exact

```

inet.0: 45 destinations, 56 routes (45 active, 0 holddown, 0 hidden)

+ = Active Route, - = Last Active, * = Both

```

10.202.100.100/32 *[Static/5] 00:00:06, metric2 0
                  > to 10.202.1.2 via ge-2/3/1.202, Push 1000001

```

Now you can now follow the snake through all its coils:

```

user@P> show route forwarding-table destination 10.202.100.100 table default

```

Routing table: default.inet

Internet:

Destination	Type	RtRef	Next hop	Type	Index	NhRef	Netif
10.202.100.100/32	user	0		indr	1048582	2	
			10.202.1.2	Push	1000001	810	2 ge-2/3/1.202

```

user@P> show route forwarding-table label 1000001 table default

```

Routing table: default.mpls

MPLS:

Destination	Type	RtRef	Next hop	Type	Index	NhRef	Netif
1000001	user	0	10.202.2.2	Swap	1000002	753	2 xe-2/0/1.202

```

user@P> show route forwarding-table label 1000002 table default

```

Routing table: default.mpls

MPLS:

Destination	Type	RtRef	Next hop	Type	Index	NhRef	Netif
1000002	user	0	10.202.1.2	Swap	1000003	809	2 ge-2/3/1.202

```

user@P> show route forwarding-table label 1000003 table default

```

Routing table: default.mpls

MPLS:

Destination	Type	RtRef	Next hop	Type	Index	NhRef	Netif
1000003	user	0	10.202.2.2	Pop	812	2	xe-2/0/1.202
1000003(S=0)	user	0	10.202.2.2	Pop	813	2	xe-2/0/1.202

Time to send a first packet and see the outcome:

```
user@P> clear interfaces statistics all

user@P> ping 10.202.100.100 no-resolve ttl 1 count 1
PING 10.202.100.100 (10.202.100.100): 56 data bytes
148 bytes from 10.202.1.1: Time to live exceeded
Vr HL TOS Len ID Flg off TTL Pro cks Src Dst
 4 5 00 0054 6ac9 0 0000 01 01 d3e7 10.202.1.1 10.202.100.100

--- 10.202.100.100 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss

user@P> show interfaces ge-2/3/1.202 statistics | match packets
Input packets : 2
Output packets: 2

user@P> show interfaces xe-2/0/1.202 statistics | match packets
Input packets : 2
Output packets: 2
```

Try It Yourself: Counting Packets at MPLS Snakes

A plain IPv4 snake ping with TTL=1 is illustrated in Figure 5.6. However, the interface statistics for the MPLS scenario don't match that picture at all. What is the actual forwarding path in this case?

The results with TTL=4 are easier to interpret, according to Figure 5.10:

```
user@P> ping 10.202.100.100 no-resolve ttl 4 count 1
PING 10.202.100.100 (10.202.100.100): 56 data bytes
36 bytes from 10.202.2.1: Time to live exceeded
Vr HL TOS Len ID Flg off TTL Pro cks Src Dst
 4 5 00 0054 ab13 0 0000 01 01 939d 10.202.1.1 10.202.100.100

user@P> show interfaces xe-2/0/1.202 statistics | match packets
Input packets : 2
Output packets: 2
user@P> show interfaces ge-2/3/1.202 statistics | match packets
Input packets : 2
Output packets: 2
```

If you want to generate more traffic, apart from increasing the initial TTL value, you can also try to maximize the potential of MPLS loops by using the `no-propagate-ttl` option (which is fully explained in Chapter 1):

```
user@P# set protocols mpls no-propagate-ttl
user@P# commit and-quit
user@P> clear interfaces statistics all
```

In those Junos OS versions affected by PR/683616, when you configure the `no-propagate-ttl` knob, it takes immediate effect for dynamic LSPs, but not for the static ones. So you'll need to recreate the static LSP to make it work as expected:

```
user@P> configure
user@P# deactivate protocols mpls static-label-switched-path SNAKE-A-1
user@P# commit
user@P# activate protocols mpls static-label-switched-path SNAKE-A-1
user@P# commit and-quit

user@P> clear interfaces statistics all
```

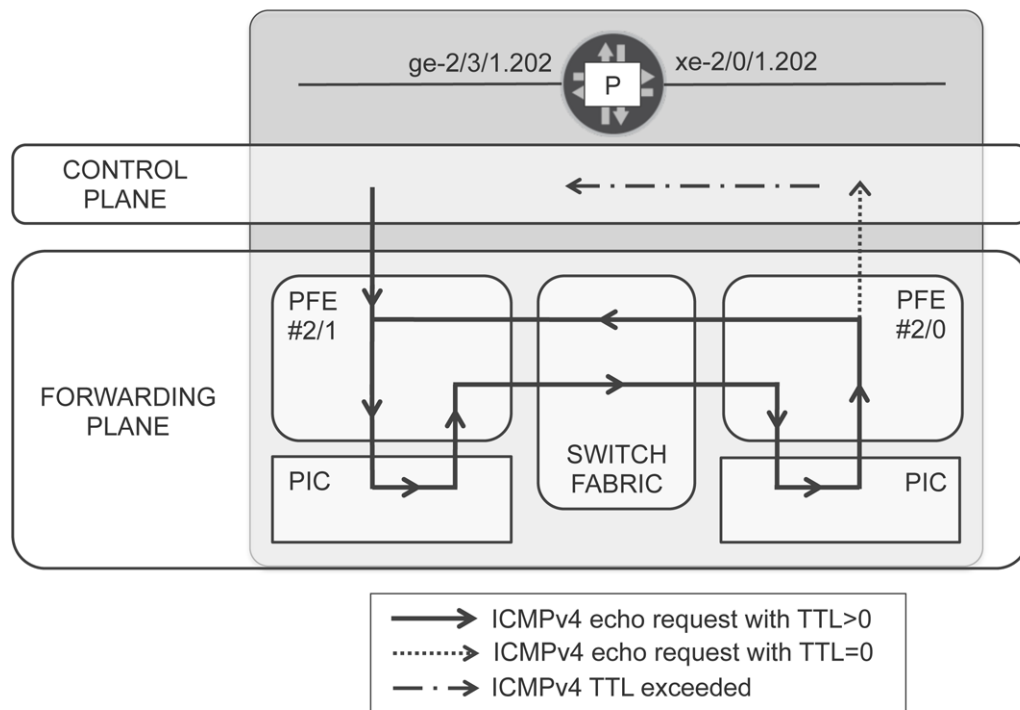


Figure 5.10 Forwarding Path of a Half-duplex IPv4 Over MPLS Snake Ping with TTL=4

```
user@P> ping 10.202.100.100 no-resolve tt1 4 count 1
PING 10.202.100.100 (10.202.100.100): 56 data bytes
36 bytes from 10.202.2.1: Time to live exceeded
Vr HL TOS Len ID Flg off TTL Pro cks Src Dst
4 5 00 0054 c130 0 0000 01 01 7d80 10.202.1.1 10.202.100.100
```

```
--- 10.202.100.100 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

```
user@P> show interfaces xe-2/0/1.202 statistics | match packets
Input packets : 8
Output packets: 8
```

```
user@P> show interfaces ge-2/3/1.202 statistics | match packets
Input packets : 8
Output packets: 8
```

Figure 5.10 still applies in these instances, but this time the TTL value represents the number of times that the packet goes through the *whole* snake. At each iteration, the packet remains in the Forwarding Plane. Only when the IPv4 TTL finally expires (after loop #16 or full trip #4), will the Control Plane come into play again.

REMEMBER It's the Line Card microkernel, and not the Routing Engine, that's generating the time exceeded message: but that's part of the Control Plane, too!

Particle Accelerators

You are now ready to learn several techniques that can allow you to simulate a very powerful traffic generator based on ping, without bothering the Control Plane! While each technique strategy has its own pros and cons, become familiar with them and you'll have a useful toolset that can create a traffic burst out of virtually anywhere.

Of course, these *particle accelerators* are not equivalent to commercial feature-rich traffic generators, but they can be useful in many situations, in both production and lab networks.

ALERT! Please use the following techniques in a responsible manner. Never utilize these methods in a production environment, or without the express permission of the respective owners of the target routers and networks.

NOTE Some techniques are intentionally not included in this book as they are confidential and only available to Juniper Networks' staff.

Loop 'Till it Burns

Depending on the angle from which you look in a mirror, you may see yourself or something else. The classical use of port mirroring in Junos OS involves copying the traffic from one port to another, in other words, set the mirror at an angle that allows you to see something else. We'll do something slightly different here.

This first particle accelerator technique is very simple, but can take a long time to figure out. Start by configuring a half-duplex 10GE loop, and set it as the system-wide port-mirror destination:

```
user@P> show interfaces xe-2/1/0 | match address
  Current address: 00:23:9c:9a:d2:e6, Hardware address: 00:23:9c:9a:d2:e6

user@P> configure
user@P# set system no-redirects
user@P# edit interfaces xe-2/1/0
user@P# set gigether-options loopback
user@P# set unit 0 family inet address 10.200.1.1/30 arp 10.200.1.2 mac 00:23:9c:9a:d2:e6
user@P# top
user@P# edit forwarding-options port-mirroring
user@P# set input rate 1
user@P# set family inet output no-filter-check
user@P# set family inet output interface xe-2/1/0.0 next-hop 10.200.1.2
user@P# commit and-quit

user@P> clear interfaces statistics all

user@P> ping 10.200.1.2 count 1 no-resolve
PING 10.200.1.2 (10.200.1.2): 56 data bytes
36 bytes from 10.200.1.1: Time to live exceeded
Vr HL TOS Len  ID Flg off TTL Pro cks      Src      Dst
 4  5  00 0054 27f2  0 0000  01  01 7a1f 10.200.1.1 10.200.1.2

--- 10.200.1.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss

user@P> show interfaces xe-2/1/0 | match packets
  Input packets : 64
  Output packets: 64
```

The number of packets matches the default `ttl` value (64). So far, it's business as usual.

Actually, no traffic is being mirrored yet. You could copy traffic from any other source interface to the mirror destination `xe-2/1/0.0`, but this just duplicates the packets. If you want to fill a 10GE interface with just one ping, you need to do something *crazier*. How about making `xe-2/1/0.0` not only the mirror destination, but also the mirror *source*?

In that case, if a packet is sent out of `xe-2/1/0.0`, when it's received back from the loop, the Trio PFE performs two actions:

- It processes the packet normally. If `TTL > 1`, then the PFE decreases TTL and sets the packet next hop to `xe-2/1/0.0 mac 00:23:9c:9a:d2:e6`. If `TTL = 1`, then the PFE flags it as an exception packet and sets the host (Line Card microkernel) as next hop.
- It sends the packet to the fabric loopback stream (see the explanation after Table 3.5). Once the packet is back from the fabric, the PFE sends a copy to the mirror destination, which is precisely `xe-2/1/0.0` itself.

NOTE This technique also works in pre-Trio platforms, but the internal implementation is different.

The outcome is one packet whose copies fill up the port:

```
user@P> configure
user@P# set firewall family inet filter MIRROR-INET term MIRROR then port-mirror
user@P# set interfaces xe-2/1/0 unit 0 family inet filter input MIRROR-INET
user@P# commit and-quit

user@P> ping 10.200.1.2 count 1 no-resolve wait 1 ttl 10
PING 10.200.1.2 (10.200.1.2): 56 data bytes
/* Lots of Time to live exceeded messages */

--- 10.200.1.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss

user@P> show interfaces xe-2/1/0 | match pps
Input rate      : 6885362872 bps (10246075 pps)
Output rate     : 6885361520 bps (10246073 pps)

user@P> show interfaces queue xe-2/1/0 egress | except " 0 "
Physical interface: xe-2/1/0, Enabled, Physical link is Up
Interface index: 216, SNMP ifIndex: 607
Forwarding classes: 16 supported, 4 in use
Egress queues: 8 supported, 4 in use
Queue: 0, Forwarding classes: best-effort
Queued:
  Packets      :          278889305          10245653 pps
  Bytes       :          34024495210        9999757968 bps
Transmitted:
  Packets      :          227255947          10245653 pps
  Bytes       :          27725225534        9999757968 bps
[...]
```

NOTE The `show interfaces queue` command is taking into account Layer 2 headers.

Why are there precisely 10Gbps and no drops? How can it be that precise? And if you sent one more `TTL=10` packet, there would probably still be no drops. It's because

the packets are buffered, which brings extra latency so they basically turn fewer times per second in the loop.

TRY THIS If you send a packet with TTL=255, then you'll likely see some tail drops.

One of the drawbacks of this looped technique is that the Control Plane is bothered:

```
user@P> show system statistics icmp | match "put|exceed"
Output Histogram
Input Histogram
694189 time exceeded /* This value increases all the time */

user@P> show ddos-protection protocols violations
Number of packet types that are being violated: 1
Protocol   Packet   Bandwidth Arrival Peak   Policer bandwidth
group      type     (pps)      rate(pps) rate(pps) violation detected at
ttl        aggregate 2000       797196    841040    2012-10-28 12:10:08 CET
Detected on: FPC-2
```

REMEMBER DDoS protection is not currently supported in pre-Trio platforms, however there are also other mechanisms protecting the routers from an avalanche of TTL exceeded messages. These mechanisms take the form of internal policers applied at different stages of the control path.

Try It Yourself: Getting Rid of TTL Exceptions

How can you make the previous *particle accelerator* work without sending TTL expired packets to the Control Plane?

Another drawback of this looped technique is that the traffic somehow gets trapped in the half-duplex loop, and it's not a trivial task to make it go somewhere else, such as, for example, to a multiple-hop snake, in a routable (TTL>1) fashion. You can apply this technique to Class of Service tests related to congestion management at the port, or use it in combination with filter actions like `next-hop-group`, `next-interface`, `next-ip` or `next-ip6`, and try to take the traffic somewhere else. The author has not explored these paths in full, so go for it!

Let's visit a more exotic, but at the same time more flexible, technique.

Decoupling MPLS and IPv4 TTL

Forget about the previous mirror-to-self technique and, if at all possible, try to imagine that you never saw it in action.

You are now about to generate more than 1Gbps of 1500-byte ICMPv4 echo requests. You'll be able to steer that traffic through different interfaces — even send it to another router — with just a single ping, that's all. Isn't it great? Of course the configuration is not trivial, but the effort is worth it.

Let's do some arithmetic first. The minimum interval value is 0.1, which means 10 packets per second. Translated to bits per second, it's $10 \times 1500 \times 8 = 1.2 \times 10^5$ bps. The target is 1Gbps = 10^9 bps. So you need to make around $10^4 = 10,000$ copies of each ICMP echo request to reach the goal. If you make an IPv4 packet turn in a loop, you are limited by the maximum number of hops it can make before the TTL expires. The maximum TTL value is 255, quite far from the 10,000 target.

There are basically two main strategies. The first is to make a packet turn 10,000 times in a loop, while copying the packet (to a different interface) at each turn. The second is to make 10,000 copies of the packet in just one turn. Or you can also combine both models. Let's start with the first strategy, pure and raw!

Let's use MPLS. By default, the IPv4 TTL is copied to the MPLS TTL at the head of a LSP, and then copied back from MPLS to IPv4 at the tail end. As discussed in Chapter 1, you can change this default behavior with the `no-propagate-ttl` knob, which basically sets MPLS TTL=255 at the head of a LSP, regardless of the IPv4 TTL value. So you can set the initial IPv4 TTL = 100, make the packet traverse 100 times a 100-hop static LSP, and you will finally get one packet looping $100 \times 100 = 10,000$ times. Make sense?

There is still one more hurdle to clear. As seen earlier in this chapter, it's better not to cause congestion in your looped ports, or else the impact of the packet drops is amplified. Instead, making a copy of the packet at each trip, and then sending the copy out of another non-looped port is the way to go. This is simply port mirroring in Junos OS. So far so good? But port mirroring is only supported for family `inet` and family `ccc` in the Junos release under test, so MPLS packets cannot be mirrored – unless you use yet another trick! Take a deep breath and check Figure 5.11.

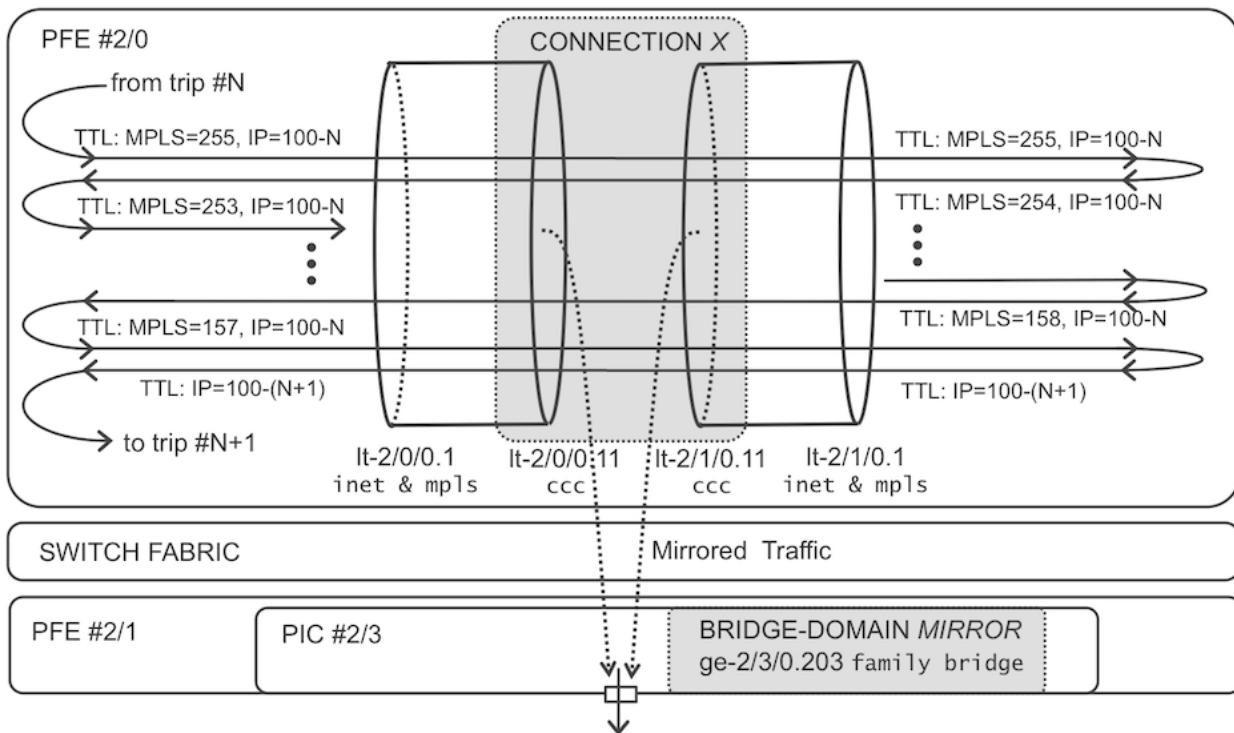


Figure 5.11 Port Mirroring of MPLS Packets Using CCC (circuit cross-connects)

TIP Trio implements `lt-` inline at the PFE. As for pre-Trio PFEs with tunnel services implemented at the PIC, make sure the PICs support at least 2Gbps full duplex per tunnel. Otherwise, it would be better to use full duplex loops with back-to-back connected 1GE ports instead.

NOTE Make sure your Junos OS release is not affected by: PR/685639: TTL check skipped for tunnel packets. Affected maintenance releases are 11.4R1, 11.4R2, 11.4R3, 12.1R1 and 12.1R2.

Let's go step by step. First, configure the Logical Tunnels that will serve as a ring for the snake to roll through many times. Execute at P:

```
user@P> configure
user@P# set chassis fpc 2 pic 0 tunnel-services
user@P# set chassis fpc 2 pic 1 tunnel-services

user@P# set interfaces lt-2/0/0 unit 1 peer-unit 11 encapsulation vlan vlan-id 203
user@P# set interfaces lt-2/0/0 unit 1 family inet address 10.203.1.1/30
user@P# set interfaces lt-2/0/0 unit 1 family mpls
user@P# set protocols mpls interface lt-2/0/0.1

user@P# set interfaces lt-2/0/0 unit 11 peer-unit 1 encapsulation vlan-ccc vlan-id 203 family ccc
user@P# set interfaces lt-2/1/0 unit 11 peer-unit 1 encapsulation vlan-ccc vlan-id 203 family ccc
user@P# set protocols connections interface-switch X interface lt-2/0/0.11
user@P# set protocols connections interface-switch X interface lt-2/1/0.11

user@P# set logical-systems LOOP interfaces lt-2/1/0 unit 1 peer-unit 11 encapsulation vlan vlan-id 203
user@P# set logical-systems LOOP interfaces lt-2/1/0 unit 1 family inet address 10.203.1.2/30
user@P# set logical-systems LOOP interfaces lt-2/1/0 unit 1 family mpls
user@P# set logical-systems LOOP protocols mpls interface lt-2/1/0.1

user@P# commit and-quit
```

NOTE The interconnection between the Logical Tunnels is full duplex.

Now let's build the MPLS snake. You can use the following script to generate the configuration:

```
#!/usr/bin/perl

print "edit protocols mpls static-label-switched-path SNAKE-B-1\n";
print "set ingress to 10.203.100.1 push 1001001 next-hop 10.203.1.2\n";

for (my $i=2; $i<100; $i++) {
    print "top\n";
    if (($i%2) == 1) {
        print "edit protocols mpls static-label-switched-path SNAKE-B-".$i."\n";
        print "set transit 100".($i+999)." swap 100".($i+1000)." next-hop 10.203.1.2\n";
    }
    else {
        print "edit logical-systems LOOP\n";
        print "edit protocols mpls static-label-switched-path SNAKE-B-".$i."\n";
        print "set transit 100".($i+999)." swap 100".($i+1000)." next-hop 10.203.1.1\n";
    }
}

print "top edit logical-systems LOOP\n";
print "edit protocols mpls static-label-switched-path SNAKE-B-100\n";
print "set transit 1001099 pop next-hop 10.203.1.1\n";
```

NOTE The Perl interpreter must be executed in an external server like H, as it's not implemented in Junos OS for security reasons. SLAX is also very good option if you want to do it all locally at the router.

TIP You can load the configuration with `load set terminal` or `load set <filename>`.

Once the configuration is applied, execute at P:

```
user@P> show route 10.203.100.1
```

```
inet.3: 4 destinations, 4 routes (4 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

```
10.203.100.1/32 * [MPLS/6/1] 00:00:02, metric 0
> to 10.203.1.2 via lt-2/0/0.1, Push 1001001
```

You need a route in `inet.0` to use it as the ping target:

```
user@P> configure
```

```
user@P# set routing-options static route 10.203.100.100/32 next-hop 10.203.100.1 resolve
user@P# commit and-quit
```

```
user@P> show route 10.203.100.100
```

```
inet.0: 22 destinations, 23 routes (22 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

```
10.203.100.100/32 * [Static/5] 00:00:27, metric 2 0
> to 10.203.1.2 via lt-2/0/0.1, Push 1001001
```

TRY THIS Follow the snake hop by hop at the forwarding table level.

Everything is ready for the first 10,000-times-replicating packet:

```
user@P> clear interfaces statistics all
```

```
user@P> ping 10.203.100.100 count 1 ttl 100 no-resolve wait 1
```

```
PING 10.203.100.100 (10.203.100.100): 56 data bytes
```

```
36 bytes from 10.203.1.1: Time to live exceeded
```

```
Vr HL TOS Len ID Flg off TTL Pro cks Src Dst
```

```
4 5 00 0054 c0be 0 0000 01 01 7df0 10.203.1.1 10.203.100.100
```

```
--- 10.203.100.100 ping statistics ---
```

```
1 packets transmitted, 0 packets received, 100% packet loss
```

```
user@P> show interfaces lt-2/0/0.1 statistics | match packets
```

```
Input packets : 5000
```

```
Output packets: 5000
```

This is the same ICMPv4 echo request going 5,000 times forth and 5,000 times back.
How about mirroring all the copies out of P?

```
user@P> configure
```

```
user@P# set firewall family ccc filter MIRROR-CCC term MIRROR then port-mirror
```

```
user@P# set interfaces lt-2/0/0 unit 11 family ccc filter input MIRROR-CCC
```

```
user@P# set interfaces lt-2/1/0 unit 11 family ccc filter input MIRROR-CCC
```

```
user@P# set interfaces ge-2/3/0 encapsulation flexible-ethernet-services
```

```
user@P# set interfaces ge-2/3/0 unit 203 vlan-id 203 encapsulation vlan-bridge family bridge
```

```
user@P# set bridge-domains MIRROR interface ge-2/3/0.203
```

```
user@P# set forwarding-options port-mirroring input rate 1
```

```
user@P# set forwarding-options port-mirroring family ccc output interface ge-2/3/0.203
```

```
user@P# commit and-quit
```

```

user@P> ping 10.203.100.100 count 1 ttl 100 no-resolve wait 1
PING 10.203.100.100 (10.203.100.100): 56 data bytes
36 bytes from 10.203.1.1: Time to live exceeded
Vr HL TOS Len ID Flg off TTL Pro cks Src Dst
4 5 00 0054 7108 0 0000 01 01 cda6 10.203.1.1 10.203.100.100

```

```

--- 10.203.100.100 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss

```

```

user@P> show interfaces ge-2/3/0.203 statistics | match packets
Input packets : 0
Output packets: 10000

```

Do you want to see the 10,000-times-multiplying effect in a packet capture? Let's configure the switch X to achieve that!

```

user@X> configure
user@X# set vlans v203 vlan-id 203
user@X# set interfaces ge-0/0/12 unit 0 family ethernet-switching vlan members v203
user@X# set ethernet-switching-options analyzer test-server input ingress vlan v203
user@X# commit and-quit

```

Check out Capture 5.1 to watch the MPLS and IPv4 TTL evolution.

What if you raise the initial TTL value or if you increase the hop number of the snake? Well, up to a value of 127, the number of mirrored packets would increase, but not if you choose higher values. Actually, snakes close to 255 hops are quite useless. This may sound surprising, but a packet capture will tell you why. The answer has to do with Penultimate Hop Popping.

The expected moment has finally arrived! In order to generate more than 1Gbps of traffic with one single ping, leave running:

```

user@P> ping 10.203.100.100 interval 0.1 ttl 100 no-resolve size 1460
/* Output suppressed */

```

And, in another terminal, execute:

```

user@P> show interfaces queue ge-2/3/0 | except " 0"
Physical interface: ge-2/3/0, Enabled, Physical link is Up
Interface index: 287, SNMP ifIndex: 16036
Forwarding classes: 16 supported, 4 in use
Egress queues: 8 supported, 4 in use
Queue: 0, Forwarding classes: best-effort
Queued:
Packets      :          813817          98897 pps
Bytes        :      1248359860      1213651352 bps
Transmitted:
Packets      :          669842          81489 pps
Bytes        :      1027509580      1000017344 bps
Tail-dropped packets :          143975          17408 pps
Queue: 1, Forwarding classes: expedited-forwarding
Queued:
Queue: 2, Forwarding classes: assured-forwarding
Queued:
Queue: 3, Forwarding classes: network-control
Queued:
Packets      :              8              1 pps
Bytes        :          1016          2312 bps
Transmitted:
Packets      :              8              1 pps
Bytes        :          1016          2312 bps

```

The 1GE interface is sending traffic at line rate, and discarding excess traffic. That opens the door to a very juicy set of Class of Service tests that you'll see an example of very soon.

TRY THIS

Check the fabric statistics. If you have a Trio PFE, the stats should be increasing twice as fast as the actual packet rate. The reason for this appears in Table 3.1.

Next Hop Groups

Trio PFEs support 1:N mirroring, in other words, 1 packet mirrored to N destinations. If an IPv4 packet is looped 250 times, you need to make $N=40$ to reach the 10,000 packet target. The idea is illustrated in Figure 5.12.

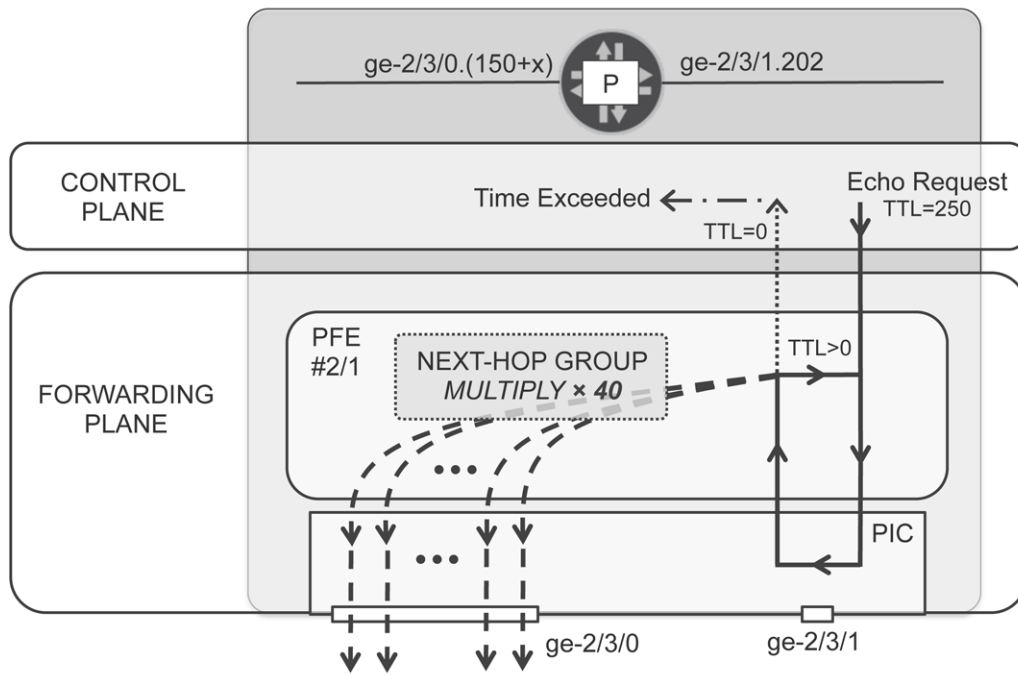


Figure 5.12 Port Mirroring of IPv4 Packets Using Next-hop Groups

Let's use the existing half duplex loop: ge-2/3/1.202, that you configured previously as part of a MPLS snake setup, but this time let's use it for single-port IPv4 loopback tests. Execute at P:

```
user@P> clear interfaces statistics all

user@P> ping 10.202.1.2 count 1 wait 1 no-resolve ttl 250
PING 10.202.1.2 (10.202.1.2): 56 data bytes
36 bytes from 10.202.1.1: Time to live exceeded
Vr HL TOS Len ID Flg off TTL Pro cks Src Dst
4 5 00 0054 d329 0 0000 01 01 cee9 10.202.1.1 10.202.1.2

--- 10.202.1.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss

user@P> show interfaces statistics ge-2/3/1.202 | match packets
Input packets : 250
Output packets: 250
```


You can configure the next hop group using this Perl script:

```
#!/usr/bin/perl

(my $interface, my $mac) = @ARGV;

if (((!$ARGV[0]) || (!$ARGV[1]))) {
    print "Usage: perl next-hop-group.pl <physical-interface> <mac-address>\n";
    exit;
}

for (my $i=1; $i<41; $i++) {

    print "set forwarding-options next-hop-group MULTIPLY interface
".$interface.".".$(i+150)." next-hop 10.202.".$(i+150)." 2\n";
    print "set interfaces ".$interface." unit ".$(i+150)." vlan-id ".$(i+150)." family inet
address 10.202.".$(i+150)." 1/30 arp 10.202.".$(i+150)." 2 mac ".$mac." \n";
}
```

One simple value for the MAC address option is a Juniper generic one:

```
$ perl next-hop-group.pl ge-2/3/0 00:23:9c:00:00:00
/* Output Suppressed */
```

Once the next-hop group is configured, you just need to mirror the input traffic at ge-2/3/1.202:

```
user@P> configure
user@P# delete forwarding-options port-mirroring family inet
user@P# set forwarding-options port-mirroring family inet output next-hop-group MULTIPLY
user@P# set firewall family inet filter MIRROR-INET term MIRROR then port-mirror
user@P# set interfaces ge-2/3/1.202 family inet filter input MIRROR-INET
user@P# commit and-quit
```

Leave running:

```
user@P> ping 10.202.1.2 interval 0.1 ttl 250 no-resolve size 1460
/* Output suppressed */
```

In another terminal, check that the egress port is saturated, and as expected:

```
user@P> show interfaces queue ge-2/3/0 | except " 0"
Physical interface: ge-2/3/0, Enabled, Physical link is Up
  Interface index: 228, SNMP ifIndex: 16036
Forwarding classes: 16 supported, 4 in use
Egress queues: 8 supported, 4 in use
Queue: 0, Forwarding classes: best-effort
  Queued:
    Packets      :          1800006          98550 pps
    Bytes        :       2754000546     1206261240 bps
  Transmitted:
    Packets      :          1477254          80989 pps
    Bytes        :       2260189986     991305560 bps
    Tail-dropped packets :          322752          17561 pps
Queue: 1, Forwarding classes: expedited-forwarding
  Queued:
Queue: 2, Forwarding classes: assured-forwarding
  Queued:
Queue: 3, Forwarding classes: network-control
  Queued:
    Bytes        :          1656          512 bps
    Bytes        :          1656          512 bps
```

NOTE The two techniques described in this section use a two-stage replication process.

How about IP Multicast, the most natural replication technology in the IP world? I thought of that but didn't find a way to do a cascaded or multi-stage replication, only the obvious 1:N approach, where you send 1 packet to N outgoing interfaces. The problem here is that N needs to be very high (~10,000), which makes the configuration heavy and you have to care about scaling limits too. If you find a way to cascade multicast replication, let us know on this book's J-Net landing page: <http://www.juniper.net/dayone!>

Class of Service in a Loop

Most classic Class of Service (COS) scenarios involve a traffic generator, several connected ports, and routing and MPLS signaling protocols: in other words, a full-blown network. If you have that setup ready, great, but what if you are only remotely connected to P via telnet or SSH and want to perform COS tests? Well, most of the forwarding plane features can be tested with half-duplex loops and COS is not an exception.

But this section isn't really about COS. It's about how to use ping and loops to perform virtually any kind of COS tests. It's necessary to keep just a bit of theory in mind to do so, as summarized in Table 5.1. You will see different encodings for the same field in IPv4/IPv6 headers, depending on the context:

- 3-bit bin: used for inet-precedence classifiers, rewrite-rules and code-point-aliases
- 6-bit bin: used for dscp & dscp-ipv6 classifiers, rewrite-rules and code-point-aliases
- 3-bit dec: used for precedence in firewall filter from/term clauses
- 6-bit dec: used for dscp and traffic-class in firewall filter from/term clauses
- 8-bit dec: used in ping tos option, both for IPv4 and for IPv6
- 8-bit hex: present in tcpdump output, both for IPv4 and for IPv6

NOTE MPLS EXP follows the same encoding rules as inet-precedence

Table 5.1 Different Encodings for the Type of Service (ToS) byte

IPv4 Precedence		DSCP				
3 bit		6 bit		8 bit		
bin	dec	bin	dec	Bin	dec	hex
000	0	000000	0	00000000	0	0x00
001	1	001000	8	00100000	32	0x20
010	2	010000	16	01000000	64	0x40
011	3	011000	24	01100000	96	0x60
100	4	100000	32	10000000	128	0x80
101	5	101000	40	10100000	160	0xA0
110	6	110000	48	11000000	192	0xC0
111	7	111000	56	11100000	224	0xE0

IPv4 Classification and Rewrite in a Loop

Firewall filters are your allies. Configure at P a new half-duplex loop able to count packets based on their IPv4 precedence values:

```
user@P> configure
user@P# edit firewall family inet filter PRECEDENCE-CHECK
user@P# set interface-specific
user@P# set term 0 from precedence 0
user@P# set term 0 then count prec-0
user@P# set term 1 from precedence 1
user@P# set term 1 then count prec-1
user@P# set term 2 from precedence 2
user@P# set term 2 then count prec-2
user@P# set term 3 from precedence 3
user@P# set term 3 then count prec-3
user@P# set term 4 from precedence 4
user@P# set term 4 then count prec-4
user@P# set term 5 from precedence 5
user@P# set term 5 then count prec-5
user@P# set term 6 from precedence 6
user@P# set term 6 then count prec-6
user@P# set term 7 from precedence 7
user@P# set term 7 then count prec-7
user@P# top
user@P# edit interfaces ge-2/3/1 unit 204
user@P# set per-unit-scheduler
user@P# set vlan-id 204 family inet address 10.204.1.1/30 arp 10.204.1.2 mac 00:23:9c:9a:d3:8b
user@P# set family inet filter input PRECEDENCE-CHECK
user@P# commit and-quit
```

NOTE The goal of `per-unit-scheduler` in the upcoming one-packet tests is to have separate queue statistics per VLAN.

Send a first IPv4 packet on the loop, with default TTL=64:

```
user@P> clear interfaces statistics all
user@P> clear firewall all
user@P> ping 10.204.1.2 count 1 no-resolve wait 1
PING 10.204.1.2 (10.204.1.2): 56 data bytes
36 bytes from 10.204.1.1: Time to live exceeded
Vr HL TOS Len ID Flg off TTL Pro cks Src Dst
 4  5  00 0054 1813  0 0000  01  01 89fc 10.204.1.1 10.204.1.2

--- 10.204.1.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss

user@P> show interfaces queue ge-2/3/1.204 | except " 0 "
Logical interface ge-2/3/1.204 (Index 425) (SNMP ifIndex 15596)
Forwarding classes: 16 supported, 4 in use
Egress queues: 8 supported, 4 in use
Burst size: 0
Queue: 0, Forwarding classes: best-effort
  Queued:
    Packets      :           64      0 pps
    Bytes        :          8064      0 bps
  Transmitted:
    Packets      :           64      0 pps
    Bytes        :          8064      0 bps
Queue: 1, Forwarding classes: expedited-forwarding
  Queued:
Queue: 2, Forwarding classes: assured-forwarding
```

```

Queued:
Queue: 3, Forwarding classes: network-control
Queued:

```

```

user@P> show firewall | match ge-2/3/1.204 | except " 0" | except filter
prec-0-ge-2/3/1.204-i                    5376                64

```

TIP For interfaces supporting ingress queuing, the output of `show interfaces queue` may be confusing. We are interested in output queuing here, so you can use the `egress` keyword.

As expected, the echo request packet leaves the Control Plane as best-effort and IPv4 precedence 0. This doesn't change during the 63 transit hops it makes in the Forwarding Plane, as you can partially see in Figure 5.13.

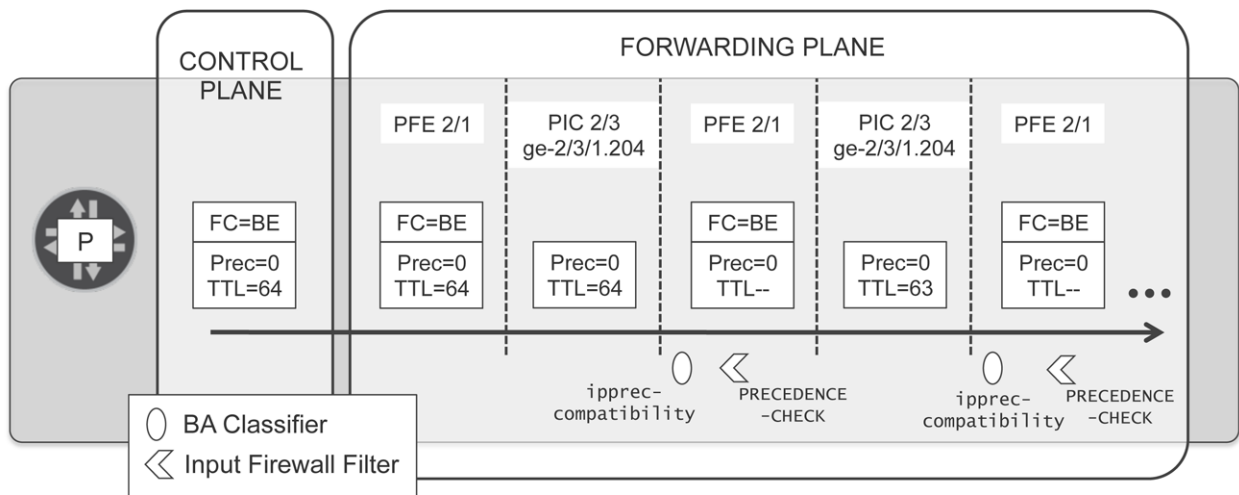


Figure 5.13 Default COS Treatment of a Packet with IPv4 Precedence 0 in a Half-duplex Loop

What if you send the echo request with precedence 5? Let's execute at P, with the help of the information in Table 5.1:

```

user@P> clear interfaces statistics all
user@P> clear firewall all
user@P> ping 10.204.1.2 count 1 no-resolve wait 1 tos 160
/* Output suppressed */

user@P> show interfaces queue ge-2/3/1.204 | except " 0 "
Logical interface ge-2/3/1.204 (Index 425) (SNMP ifIndex 15596)
Forwarding classes: 16 supported, 4 in use
Egress queues: 8 supported, 4 in use
Burst size: 0
Queue: 0, Forwarding classes: best-effort
Queued:
  Packets      :          64          0 pps
  Bytes       :        8064          0 bps
Transmitted:
  Packets      :          64          0 pps
  Bytes       :        8064          0 bps
Queue: 1, Forwarding classes: expedited-forwarding
Queued:
Queue: 2, Forwarding classes: assured-forwarding
Queued:

```

Queue: 3, Forwarding classes: network-control
Queued:

```
user@P> show firewall | match ge-2/3/1.204 | except " 0" | except filter
prec-5-ge-2/3/1.204-i          5376          64
```

There are two conclusions that arise from this test:

- The ping tos option changes the IPv4 precedence, but you do not see it change the Forwarding Class of the echo request when it first goes out of the Control Plane.
- IPv4 packets with precedence 5 seem to be classified by default as best-effort in transit.

```
user@P> show class-of-service interface ge-2/3/1.204
Logical interface: ge-2/3/1.204, Index: 425
Object      Name      Type      Index
Scheduler-map <default> Output      2
Rewrite     exp-default exp (mpls-any) 33
Classifier  exp-default exp      10
Classifier  ipprec-compatibility ip      13
```

As you can see above, there is no default IPv4 rewrite-rule, that's why the echo request remained with IPv4 precedence 5 all the time. The key element here is the `inet-precedence` classifier, which determines the Forwarding Class that packets *received* at ge-2/3/1.204 are mapped to. In other words, it only affects transit traffic in this test. Execute at P:

```
user@P> show class-of-service classifier name ipprec-compatibility
Classifier: ipprec-compatibility, Code point type: inet-precedence, Index: 13
Code point  Forwarding class  Loss priority
000         best-effort    low
001         best-effort    high
010         best-effort    low
011         best-effort    high
100         best-effort    low
101         best-effort    high
110         network-control low
111         network-control high
```

Indeed, precedence 5 is mapped to **best-effort**, which explains the results. Figure 5.13 would apply here, by just replacing Prec=0 with Prec=5 everywhere.

Let's try to change the Forwarding Class of the echo request in its very first hop out of the Control Plane, using an output firewall filter applied to 100.0 interface:

```
user@P> configure
user@P# edit firewall family inet filter 100-COS
user@P# set term ICMP from protocol icmp
user@P# set term ICMP then forwarding-class expedited-forwarding
user@P# set term ICMP then loss-priority low
user@P# set term ICMP then accept
user@P# set term REST then accept
user@P# top
user@P# set interfaces 100.0 family inet filter output 100-COS
user@P# commit and-quit

user@P> clear interfaces statistics all
user@P> clear firewall all
user@P> ping 10.204.1.2 count 1 no-resolve wait 1 tos 160
/* Output suppressed */
```

```

user@P> show interfaces queue ge-2/3/1.204 | except " 0 "
Logical interface ge-2/3/1.204 (Index 425) (SNMP ifIndex 15596)
Forwarding classes: 16 supported, 4 in use
Egress queues: 8 supported, 4 in use
Burst size: 0
Queue: 0, Forwarding classes: best-effort
  Queued:
    Packets      :           63          0 pps
    Bytes        :          7938          0 bps
  Transmitted:
    Packets      :           63          0 pps
    Bytes        :          7938          0 bps
Queue: 1, Forwarding classes: expedited-forwarding
  Queued:
    Packets      :             1          0 pps
    Bytes        :           126          0 bps
  Transmitted:
    Packets      :             1          0 pps
    Bytes        :           126          0 bps
Queue: 2, Forwarding classes: assured-forwarding
  Queued:
Queue: 3, Forwarding classes: network-control
  Queued:

user@P> show firewall | match ge-2/3/1.204 | except " 0" | except filter
prec-5-ge-2/3/1.204-i                    5376          64

```

NOTE This feature - being able to set Forwarding Class in outbound lo0.0 filters - was introduced in Junos OS 11.2 in Trio, and in 10.0 for earlier platforms.

As you can see here, and next in Figure 5.14, the Forwarding Class only changes in the first hop, when the ICMPv4 echo request exits the Control Plane. Subsequent hops are ruled exclusively by the Forwarding Plane, where the lo0.0 filters don't apply, and only the ge-2/3/1.204 classifiers do.

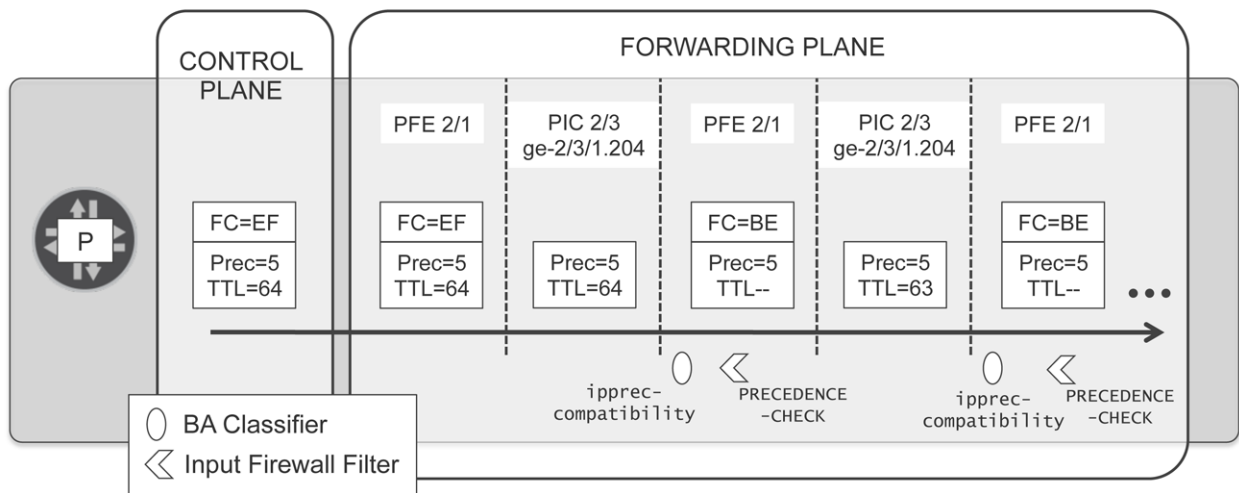


Figure 5.14 Forwarding Class Remap, via an Output lo0.0 Firewall Filter, of a Packet with IPv4 Precedence 5

There is also a possibility to set the DSCP of outgoing control packets using the same lo0.0 output filter. This is an interesting option for all protocols except for ICMP, since it somehow makes the ping tos option useless. Give the feature a try, but don't forget to remove the configuration later:

```

user@P> configure
user@P# set firewall family inet filter lo0-COS term ICMP then dscp 40
user@P> commit and-quit

user@P> clear firewall all
user@P> clear interfaces statistics all
user@P> ping 10.204.1.2 count 1 no-resolve wait 1 tos 0
/* Output suppressed */

user@P> show firewall | match ge-2/3/1.204 | except " 0" | except filter
prec-5-ge-2/3/1.204-i          5376          64

user@P> configure
user@P# delete firewall family inet filter lo0-COS term ICMP then dscp
user@P# commit and-quit

```

NOTE An older knob [edit class-of-service host-outbound-traffic] also allows you to remap outbound Control Traffic to a different Forwarding Class and DSCP than the one initially chosen by the RE. This feature is less flexible since it applies to all the traffic with no granularity.

Going back to Figure 5.14, how can you classify the transit packets as expedited-forwarding? There are several options, of which the most flexible is changing the firewall filter chain currently applied to ge-2/3/1.204. The results are in Figure 5.15. Execute at P:

```

user@P> configure
user@P# edit firewall family inet filter PREC-T0-FC term PREC5
user@P# set from precedence 5
user@P# set then forwarding-class expedited-forwarding
user@P# set then next term

/* since this is the last term, it will go to the next filter in the chain */

user@P# set interfaces ge-2/3/1.204 family inet filter input-list [ PREC-T0-FC PRECEDENCE-CHECK ]
user@P# commit and-quit

user@P> clear interfaces statistics all
user@P> clear firewall all
user@P> ping 10.204.1.2 count 1 no-resolve wait 1 tos 160
/* Output suppressed */
user@P> show interfaces queue ge-2/3/1.204 | except " 0 "
Logical interface ge-2/3/1.204 (Index 425) (SNMP ifIndex 15596)
Forwarding classes: 16 supported, 4 in use
Egress queues: 8 supported, 4 in use
Burst size: 0
Queue: 0, Forwarding classes: best-effort
  Queued:
Queue: 1, Forwarding classes: expedited-forwarding
  Queued:
    Packets      :          64          0 pps
    Bytes        :         8064          0 bps
  Transmitted:
    Packets      :          64          0 pps
    Bytes        :         8064          0 bps
Queue: 2, Forwarding classes: assured-forwarding
  Queued:
Queue: 3, Forwarding classes: network-control
  Queued:

user@P> show firewall | match ge-2/3/1.204 | except " 0" | except filter
prec-5-ge-2/3/1.204-i          5376          64

```

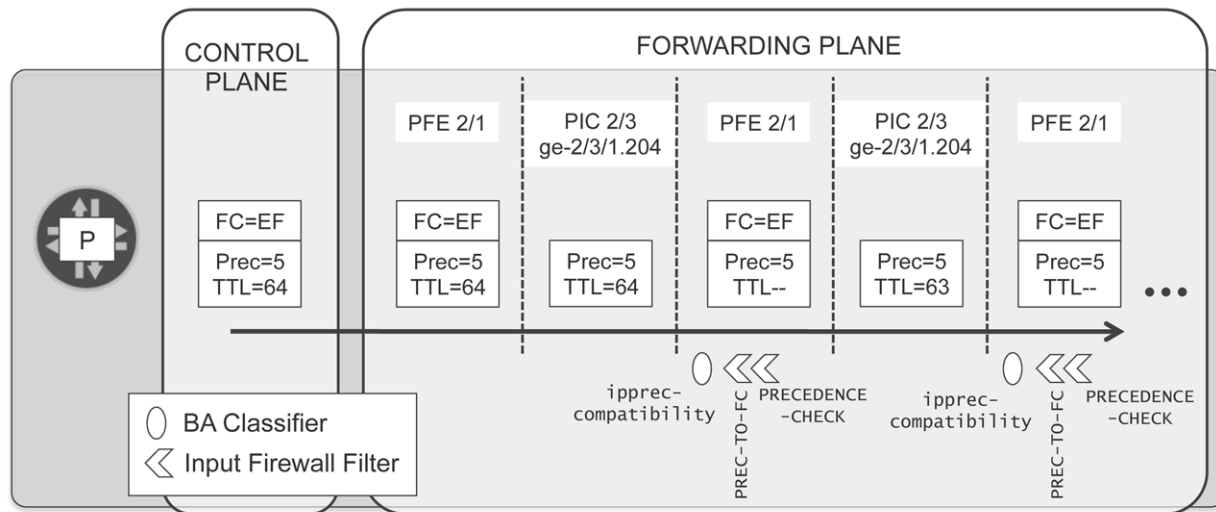


Figure 5.15 Further FC Remap, via an Input ge- filter, of a Packet with IPv4 Precedence 5

MPLS Classification and Rewrite in a Loop

First, let's simplify things a little bit:

```
user@P> configure
user@P# delete protocols mpls no-propagate-ttl
user@P# delete protocols mpls icmp-tunneling
user@P# commit and-quit
```

Now, configure a filter to count MPLS packets based on the experimental bits, and apply it to ge-2/3/1.204:

```
user@P> configure
user@P# edit firewall family mpls filter EXP-CHECK
user@P# set interface-specific
user@P# set term 0 from exp 0
user@P# set term 0 then count exp-0
user@P# set term 1 from exp 1
user@P# set term 1 then count exp-1
user@P# set term 2 from exp 2
user@P# set term 2 then count exp-2
user@P# set term 3 from exp 3
user@P# set term 3 then count exp-3
user@P# set term 4 from exp 4
user@P# set term 4 then count exp-4
user@P# set term 5 from exp 5
user@P# set term 5 then count exp-5
user@P# set term 6 from exp 6
user@P# set term 6 then count exp-6
user@P# set term 7 from exp 7
user@P# set term 7 then count exp-7
user@P# top
user@P# set interfaces ge-2/3/1 unit 204 family mpls filter input EXP-CHECK
user@P# top
user@P# set protocols mpls interface ge-2/3/1.204
user@P# commit and-quit
```


Ready for a new snake test? Configure at P:

```
user@P> configure
user@P# edit protocols mpls
user@P# set static-label-switched-path SNAKE-C-1 ingress to 10.204.100.1 push 1002001 next-hop
10.204.1.2
user@P# set static-label-switched-path SNAKE-C-2 transit 1002001 swap 1002002 next-hop 10.204.1.2
user@P# set static-label-switched-path SNAKE-C-3 transit 1002002 pop next-hop 10.204.1.2
user@P# top
user@P# set routing-options static route 10.204.100.100/32 next-hop 10.204.100.1 resolve
user@P# commit and-quit
```

Let's start first with plain ping with no tos option (by default, it's 0):

```
user@P> clear interfaces statistics all
user@P> clear firewall all
user@P> ping 10.204.100.100 count 1 no-resolve wait 1 ttl 4
PING 10.204.100.100 (10.204.100.100): 56 data bytes
148 bytes from 10.204.1.1: Time to live exceeded
Vr HL TOS Len ID Flg off TTL Pro cks Src Dst
 4  5  00 0054 b6db  0 0000  01  01 87d1 10.204.1.1 10.204.100.100

--- 10.204.100.100 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss

user@P> show interfaces queue ge-2/3/1.204 | except " 0 "
Logical interface ge-2/3/1.204 (Index 425) (SNMP ifIndex 15596)
Forwarding classes: 16 supported, 4 in use
Egress queues: 8 supported, 4 in use
Burst size: 0
Queue: 0, Forwarding classes: best-effort
  Queued:
    Packets      :           3      0 pps
    Bytes        :          386      0 bps
  Transmitted:
    Packets      :           3      0 pps
    Bytes        :          386      0 bps
Queue: 1, Forwarding classes: expedited-forwarding
  Queued:
    Packets      :           1      0 pps
    Bytes        :          130      0 bps
  Transmitted:
    Packets      :           1      0 pps
    Bytes        :          130      0 bps
Queue: 2, Forwarding classes: assured-forwarding
  Queued:
Queue: 3, Forwarding classes: network-control
  Queued:

user@P> show firewall | match ge-2/3/1.204 | except " 0" | except filter
prec-0-ge-2/3/1.204-i      84      1
exp-0-ge-2/3/1.204-i     264      3
```

TIP If you find it difficult to interpret the results, repeat the test with increasing values of TTL, starting from 1 and clearing the counters at each iteration.

Figure 5.16 illustrates the test. When the ICMPv4 echo request is built by the Routing Engine, the MPLS experimental bits are copied from the IPv4 precedence. Once the first hop is over, the classification for transit hops is done upon the code-points – EXP or precedence, depending on the outermost header – which are all 0. So the Forwarding Class becomes best-effort. Have a look at the default MPLS EXP classifier:

```

user@P> show class-of-service classifier name exp-default
Classifier: exp-default, Code point type: exp, Index: 10
Code point      Forwarding class      Loss priority
000             best-effort           low
001             best-effort           high
010             expedited-forwarding  low
011             expedited-forwarding  high
100             assured-forwarding  low
101             assured-forwarding  high
110             network-control    low
111             network-control    high

```

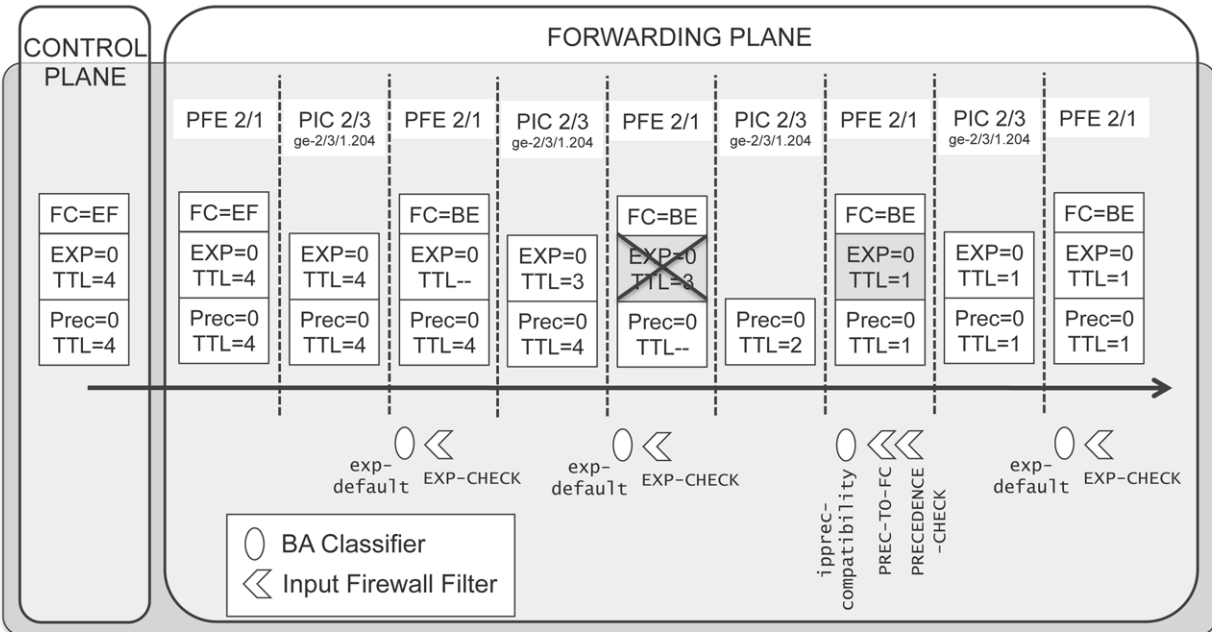


Figure 5.16 Default COS Treatment of a Packet with MPLS EXP=0 in a Half-duplex Loop

Let's try a non-zero tos value now. Execute at P:

```

user@P> clear interfaces statistics all
user@P> clear firewall all
user@P> ping 10.204.100.100 count 1 no-resolve wait 1 ttl 4 tos 160
/* Output suppressed */

```

```

user@P> show interfaces queue ge-2/3/1.204 | except " 0 "
Logical interface ge-2/3/1.204 (Index 425) (SNMP ifIndex 15596)
Forwarding classes: 16 supported, 4 in use
Egress queues: 8 supported, 4 in use
Burst size: 0
Queue: 0, Forwarding classes: best-effort
  Queued:
    Packets      :          2          0 pps
    Bytes        :         260         0 bps
Transmitted:
    Packets      :          2          0 pps
    Bytes        :         260         0 bps
Queue: 2, Forwarding classes: assured-forwarding

```

```

Queued:
Packets      :                2          0 pps
Bytes        :               256          0 bps
Transmitted:
Packets      :                2          0 pps
Bytes        :               256          0 bps
Queue: 3, Forwarding classes: network-control
Queued:

user@P> show firewall | match ge-2/3/1.204 | except " 0" | except filter
exp-5-ge-2/3/1.204-i                264          3
prec-5-ge-2/3/1.204-i              84          1

```

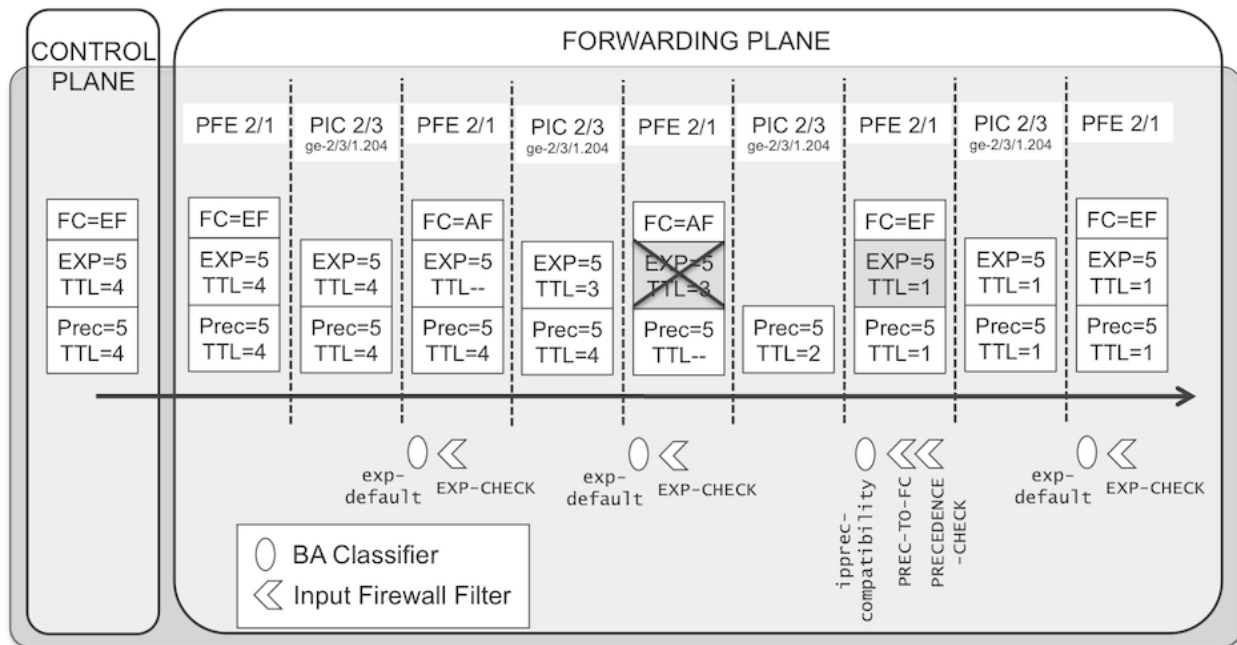


Figure 5.17 Default COS Treatment of a Packet with MPLS EXP=5 in a Half-duplex Loop in Trio

Figure 5.17 illustrates the results. The sequence is: *host outbound* push, *transit* swap, *transit* pop, *transit* push. The *host outbound* operations are performed by the Control Plane, while the *transit* ones rely on the Forwarding Plane. Both cases follow a similar logic: copy the IPv4 precedence into the MPLS EXP bits.

This is a generic fact in all platforms for *host outbound* MPLS push operations. However, *transit* push is a bit different. In pre-Trio platforms, MPLS interfaces always have an EXP `rewrite-rule` applied and the resulting EXP bits depend on the internal COS classification (FC, PLP) of the packet. On the other hand, Trio platforms have no EXP `rewrite-rule` applied by default, which explains the IPv4 precedence to MPLS EXP copy operation you just saw.

NOTE There is a cosmetic bug being tracked under PR/824791. Releases affected show `exp-default rewrite-rule` applied to MPLS interfaces in Trio platforms by default, which is misleading.

Apply an explicit MPLS EXP `rewrite-rule` to the interface. For the moment just copy the values from the `exp-default` ones:

```

user@P> configure
user@P# set class-of-service rewrite-rules exp CUSTOM-EXP-RW import default
user@P# set class-of-service interfaces ge-2/3/1 unit 204 rewrite-rules exp CUSTOM-EXP-RW
user@P# commit and-quit

```

```

user@P> show class-of-service interface ge-2/3/1.204 | match rewrite
Rewrite          CUSTOM-EXP-RW          exp (mpls-any)          21390
user@P> show class-of-service rewrite-rule name CUSTOM-EXP-RW
Rewrite rule: CUSTOM-EXP-RW, Code point type: exp, Index: 21390
Forwarding class      Loss priority      Code point
best-effort           low           000
best-effort           high          001
expedited-forwarding low           010
expedited-forwarding  high          011
assured-forwarding    low           100
assured-forwarding high           101
network-control       low           110
network-control       high          111

```

Check the differences introduced by the EXP rewrite-rule:

```

user@P> clear interfaces statistics all
user@P> clear firewall all
user@P> ping 10.204.100.100 count 1 no-resolve wait 1 ttl 4 tos 160
/* Output suppressed */

```

```

user@P> show interfaces queue ge-2/3/1.204 | except " 0 "
Logical interface ge-2/3/1.204 (Index 425) (SNMP ifIndex 15596)
Forwarding classes: 16 supported, 4 in use
Egress queues: 8 supported, 4 in use
Burst size: 0
Queue: 0, Forwarding classes: best-effort
  Queued:
Queue: 1, Forwarding classes: expedited-forwarding
  Queued:
    Packets      :          2          0 pps
    Bytes        :         260          0 bps
  Transmitted:
    Packets      :          2          0 pps
    Bytes        :         260          0 bps
Queue: 2, Forwarding classes: assured-forwarding
  Queued:
    Packets      :          2          0 pps
    Bytes        :         256          0 bps
  Transmitted:
    Packets      :          2          0 pps
    Bytes        :         256          0 bps
Queue: 3, Forwarding classes: network-control
  Queued:

```

```

user@P> show firewall | match ge-2/3/1.204 | except " 0" | except filter
exp-2-ge-2/3/1.204-i          88          1
exp-5-ge-2/3/1.204-i         176          2
prec-5-ge-2/3/1.204-i        84          1

```

NOTE Although it's not shown in the picture, in this example the Packet Loss Priority (PLP) is low everywhere, except in the hop involving MPLS *swap* operation.

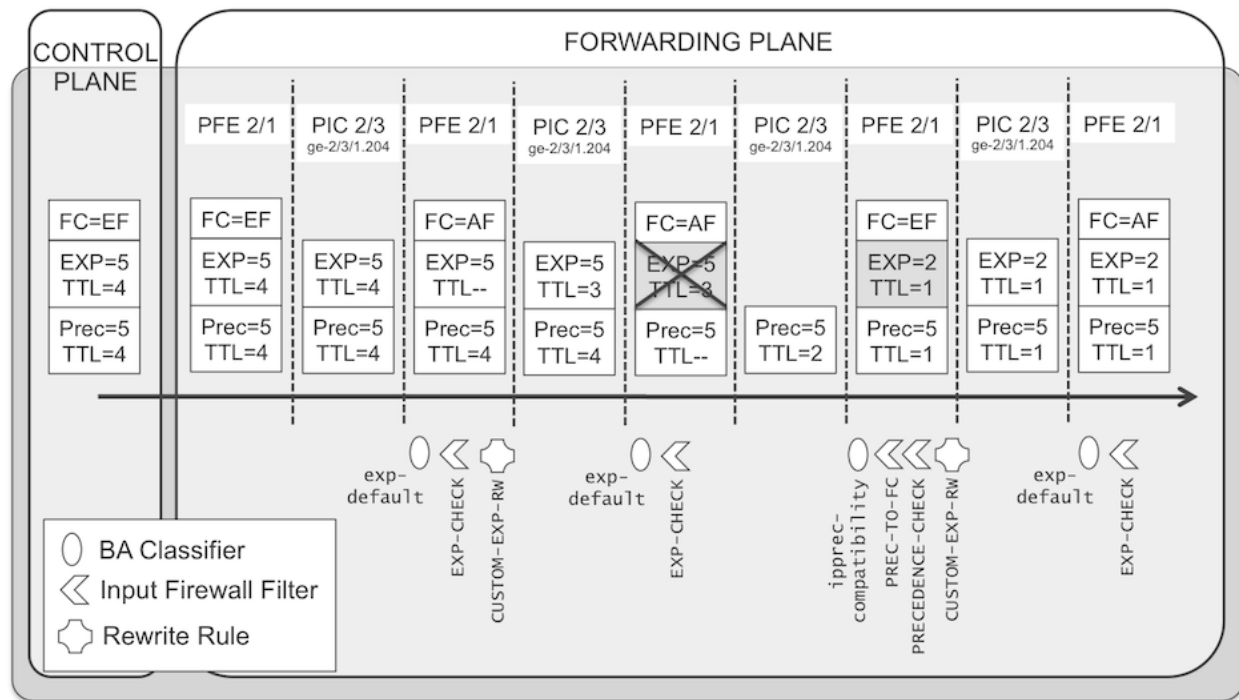


Figure 5.18 COS Treatment of a Packet with MPLS EXP=5 in a Half-duplex Loop with an EXP Rewrite-rule

Try It Yourself: Playing with MPLS Rewrite Rules

Modify CUSTOM-EXP-RW so that the EXP values are 5 all over the forwarding path. One command is enough.

VRFs and Label Switched Interfaces (LSIs) in a Loop

You are about to test COS features of VRFs integrated with a MPLS core. With no BGP session, and no MPLS network with PEs or Ps whatsoever. Just a half-duplex local loopback is enough! First, configure a minimal VRF using a new VLAN on your favorite half-duplex loop:

```
user@P# edit interfaces ge-2/3/1 unit 205
user@P# set vlan-id 205
user@P# set family inet address 10.205.1.1/30 arp 10.205.1.2 mac 00:23:9c:9a:d3:8b
user@P# set family inet filter input PRECEDENCE-CHECK
user@P# top
user@P# edit routing-instances VRF-COS
user@P# set instance-type vrf vrf-table-label
user@P# set route-distinguisher 65000:2
user@P# set vrf-target target:65000:200
user@P# set interface ge-2/3/1.205
user@P# set routing-options static route 10.205.100.100/32 next-hop 10.205.1.2
user@P# commit and-quit
```

The trick here is to send traffic to the VPN label, but you first need to find its numerical value (in this case it's 16):

```
user@P> show route table mpls protocol vpn
```

```
mpls.0: 70 destinations, 70 routes (70 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

```
16          *[VPN/0] 00:00:25
           to table VRF-COS.inet.0, Pop
```

Next, define a static LSP leading to the VRF:

```
user@P> configure
user@P# edit protocols mpls
user@P# set static-label-switched-path SNAKE-D-1 ingress to 10.205.100.1 push 1003001 next-hop
10.204.1.2
user@P# set static-label-switched-path SNAKE-D-2 transit 1003001 swap 16 next-hop 10.204.1.2
user@P# top
user@P# set routing-options static route 10.205.100.100 next-hop 10.205.100.1 resolve
```

Let's follow the snake up to its destination:

```
user@P> show route forwarding-table destination 10.205.100.100 table default
Routing table: default.inet
Internet:
Destination      Type RtRef Next hop          Type Index NhRef Netif
10.205.100.100/32 user    0          10.204.1.2      Push 1003001 1119 2 ge-2/3/1.204
```

```
user@P> show route label 1003001
```

```
mpls.0: 71 destinations, 71 routes (71 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

```
1003001      *[MPLS/6] 00:00:59, metric 1
             > to 10.204.1.2 via ge-2/3/1.204, Swap 16
```

```
user@P> show route label 16
```

```
mpls.0: 71 destinations, 71 routes (71 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

```
16          *[VPN/0] 00:21:42
           to table VRF-COS.inet.0, Pop
```

```
user@P> show route forwarding-table destination 10.205.100.100 table VRF-COS
Routing table: VRF-COS.inet
Internet:
Destination      Type RtRef Next hop          Type Index NhRef Netif
10.205.100.100/32 user    0 10.205.1.2      ucst 1118 3 ge-2/3/1.205
```

Now, make it work with the usual procedure:

```
user@P> clear firewall all
user@P> clear interfaces statistics all
user@P> ping 10.205.100.100 count 1 no-resolve wait 1 ttl 3 tos 160
PING 10.205.100.100 (10.205.100.100): 56 data bytes
```

```
--- 10.205.100.100 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

```
user@P> show interfaces queue ge-2/3/1.205 | except " 0 "
Logical interface ge-2/3/1.205 (Index 378) (SNMP ifIndex 15597)
Forwarding classes: 16 supported, 4 in use
Egress queues: 8 supported, 4 in use
Burst size: 0
```

```

Queue: 0, Forwarding classes: best-effort
  Queued:
Queue: 1, Forwarding classes: expedited-forwarding
  Queued:
Queue: 2, Forwarding classes: assured-forwarding
  Queued:
    Packets      :           1          0 pps
    Bytes        :          126         0 bps
  Transmitted:
    Packets      :           1          0 pps
    Bytes        :          126         0 bps
Queue: 3, Forwarding classes: network-control
  Queued:

user@P> show firewall | match ge-2/3/1.204 | except " 0" | except filter
exp-5-ge-2/3/1.204-i                               176                2

user@P> show firewall | match ge-2/3/1.205 | except " 0" | except filter
prec-5-ge-2/3/1.205-i                               84                1

```

From the Forwarding Plane perspective, the packets are handled in a way very similar to the non-VRF scenario. Figure 5.19 illustrates the whole forwarding path.

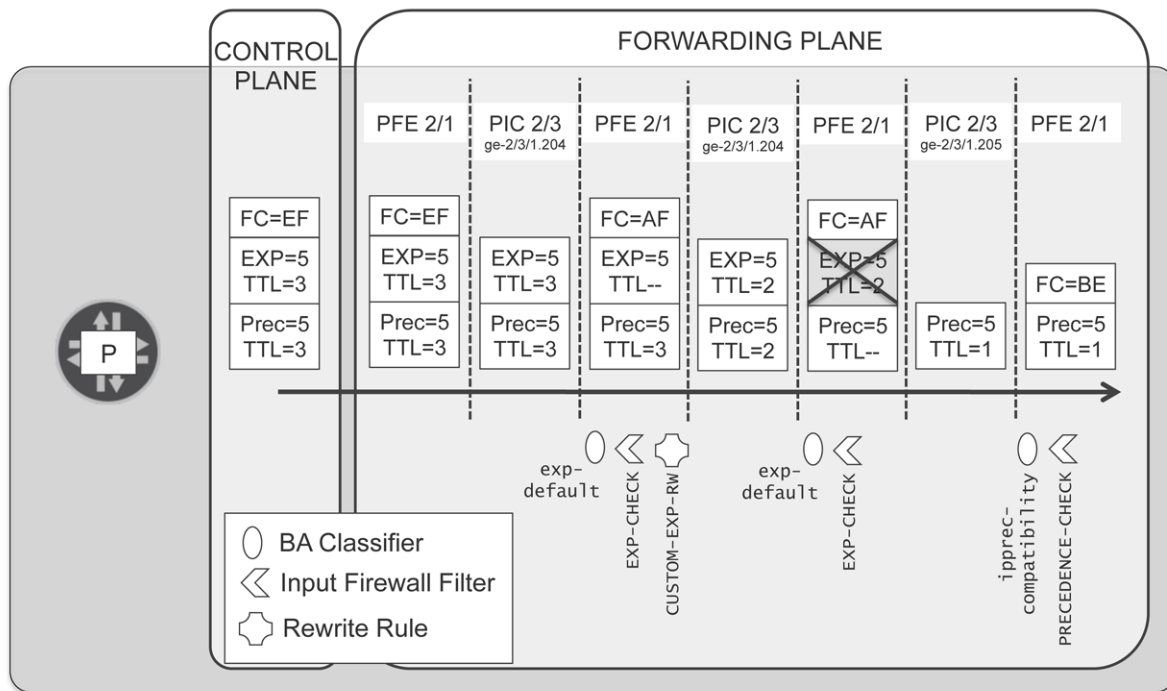


Figure 5.19 COS Treatment of a Packet with MPLS EXP=5 Arriving to a VRF via a Half-duplex Loop

The MPLS packet with EXP=5 is arriving to the VRF from the MPLS core. Suppose that you want to change the default classification, so that the packet is queued as expedited-forwarding before exiting the PE-CE interface ge-2/3/1.205. You can do it with an output firewall filter applied to ge-2/3/1.205, but let's choose a different approach:

```

user@P> configure
user@P# edit class-of-service classifiers exp CUSTOM-EXP-CLAS
user@P# set import default

```

```

user@P# set forwarding-class expedited-forwarding loss-priority low code-points 101
user@P# top
user@P# set class-of-service interfaces ge-2/3/1 unit 204 classifiers exp CUSTOM-EXP-CLAS
user@P# commit and-quit

```

If you send the ICMPv4 echo request again, you should still see it at the assured-forwarding queue. Why? Well, in the special case of VPN traffic coming from the MPLS core, it's the EXP classifier attached to the VRF's Label Switched Interface (LSI) that is the one taking the lead:

```

user@P> show class-of-service routing-instance VRF-COS
Routing instance: VRF-COS

```

```

Logical interface: lsi.0, Index: 354
Object      Name      Type      Index
Classifier  exp-default  exp      10

```

```

user@P> configure
user@P# set class-of-service routing-instances VRF-COS classifiers exp CUSTOM-EXP-CLAS
user@P# commit and-quit

```

```

user@P> show class-of-service routing-instance VRF-COS
Routing instance: VRF-COS

```

```

Logical interface: lsi.0, Index: 354
Object      Name      Type      Index
Classifier  CUSTOM-EXP-CLAS  exp      23794

```

```

user@P> clear interfaces statistics all
user@P> clear firewall all

```

```

user@P> ping 10.205.100.100 count 1 no-resolve wait 1 ttl 3 tos 160
/* Output suppressed */

```

```

user@P> show interfaces queue ge-2/3/1.205 | except " 0 "
Logical interface ge-2/3/1.205 (Index 378) (SNMP ifIndex 15597)
Forwarding classes: 16 supported, 4 in use
Egress queues: 8 supported, 4 in use
Burst size: 0
Queue: 0, Forwarding classes: best-effort
  Queued:
Queue: 1, Forwarding classes: expedited-forwarding
  Queued:
    Packets      :          1      0 pps
    Bytes        :         126      0 bps
  Transmitted:
    Packets      :          1      0 pps
    Bytes        :         126      0 bps
Queue: 2, Forwarding classes: assured-forwarding
  Queued:
Queue: 3, Forwarding classes: network-control
  Queued:

```



```
PING 10.204.101.101 (10.204.101.101): 56 data bytes
36 bytes from 10.204.1.1: Time to live exceeded
Vr HL TOS Len ID Flg off TTL Pro cks Src Dst
4 5 00 0054 4542 0 0000 01 01 f869 10.204.1.1 10.204.101.101
```

```
--- 10.204.101.101 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

```
user@P> show interfaces queue ge-2/3/1.204 | except " 0 "
Logical interface ge-2/3/1.204 (Index 425) (SNMP ifIndex 15596)
Forwarding classes: 16 supported, 4 in use
Egress queues: 8 supported, 4 in use
Burst size: 0
Queue: 0, Forwarding classes: best-effort
  Queued:
    Packets      :           63      0 pps
    Bytes        :          7938      0 bps
  Transmitted:
    Packets      :           63      0 pps
    Bytes        :          7938      0 bps
Queue: 1, Forwarding classes: expedited-forwarding
  Queued:
    Packets      :            1      0 pps
    Bytes        :           126      0 bps
  Transmitted:
    Packets      :            1      0 pps
    Bytes        :           126      0 bps
Queue: 2, Forwarding classes: assured-forwarding
  Queued:
Queue: 3, Forwarding classes: network-control
  Queued:
```

You need to define a mapping between destination-class GOLD to forwarding-class expedited-forwarding, by configuring:

```
user@P> configure
user@P# edit firewall family inet filter COS-CHANGE
user@P# set term SPECIAL-FLOW from destination-class GOLD
user@P# set term SPECIAL-FLOW then forwarding-class expedited-forwarding
user@P# set term DEFAULT then accept
user@P# top
user@P# set forwarding-options family inet filter output COS-CHANGE
user@P# commit and-quit
```

```
user@P> clear interfaces statistics all
user@P> ping 10.204.101.101 count 1 no-resolve wait 1
/* Output suppressed */
```

```
user@P> show interfaces queue ge-2/3/1.204 | except " 0 "
Logical interface ge-2/3/1.204 (Index 425) (SNMP ifIndex 15596)
Forwarding classes: 16 supported, 4 in use
Egress queues: 8 supported, 4 in use
Burst size: 0
Queue: 0, Forwarding classes: best-effort
  Queued:
Queue: 1, Forwarding classes: expedited-forwarding
  Queued:
    Packets      :           64      0 pps
    Bytes        :          8064      0 bps
  Transmitted:
    Packets      :           64      0 pps
    Bytes        :          8064      0 bps
Queue: 2, Forwarding classes: assured-forwarding
```

```

Queued:
Queue: 3, Forwarding classes: network-control
Queued:

```

MORE? Read sections *Configuring SCU or DCU* and *Applying Filters to Forwarding Tables* in the Junos technical guides, <http://www.juniper.net/techpubs>.

This is just an example of a relatively exotic feature that you can easily test with a poor-man scenario. Virtually any feature implemented in the Forwarding Plane can be tested in this manner: unicast RPF checks, complex firewall filtering or policing, and a large list only limited by imagination and... well, also by what's implemented in Junos OS to date.

Queue Scheduling with Just One Ping

Remember the one-ping test filling a 1GE port by decoupling MPLS and IPv4 TTL? Let's start it again in terminal #1:

```

user@P> configure
user@P# edit protocols mpls
user@P# set no-propagate-ttl
user@P# deactivate static-label-switched-path SNAKE-B-1
user@P# commit
user@P# activate static-label-switched-path SNAKE-B-1
user@P# commit and-quit

user@P> ping 10.203.100.100 interval 0.1 ttl 100 no-resolve size 1460
PING 10.203.100.100 (10.203.100.100): 1460 data bytes
36 bytes from 10.203.1.1: Time to live exceeded
Vr HL TOS Len ID Flg off TTL Pro cks Src Dst
 4  5  00 05d0 f768  0 0000  01  01 41ca 10.203.1.1 10.203.100.100

/* Output suppressed */

```

Currently, all the replicated traffic is going out of ge-2/3/0 as best-effort. Looking back to Figure 5.11, there are two interfaces mirroring traffic: lt-2/0/0.11 and lt-2/1/0.11. In terminal #2, make one of the interfaces mirror traffic as best-effort, and the other as expedited-forwarding:

```

user@P> configure
user@P# edit firewall family ccc
user@P# set filter MIRROR-CCC-BE term MIRROR then port-mirror
user@P# set filter MIRROR-CCC-BE term MIRROR then forwarding-class best-effort
user@P# set filter MIRROR-CCC-EF term MIRROR then port-mirror
user@P# set filter MIRROR-CCC-EF term MIRROR then forwarding-class expedited-forwarding
user@P# top
user@P# set interfaces lt-2/0/0 unit 11 family ccc filter input MIRROR-CCC-BE
user@P# set interfaces lt-2/1/0 unit 11 family ccc filter input MIRROR-CCC-EF
user@P# commit and-quit

user@P> clear interfaces statistics all

user@P> show interfaces queue ge-2/3/0 | except " 0 "
Physical interface: ge-2/3/0, Enabled, Physical link is Up
  Interface index: 228, SNMP ifIndex: 16036
Forwarding classes: 16 supported, 4 in use
Egress queues: 8 supported, 4 in use
Queue: 0, Forwarding classes: best-effort

```

```

Queued:
  Packets      :          2770134          49691 pps
  Bytes       :          4249371126       609817336 bps
Transmitted:
  Packets      :          2770134          49691 pps
  Bytes       :          4249371126       609817336 bps
Queue: 1, Forwarding classes: expedited-forwarding
Queued:
  Packets      :          2770123          49689 pps
  Bytes       :          4249147070       609775872 bps
Transmitted:
  Packets      :          1782325          31797 pps
  Bytes       :          2733946054       390200816 bps
Tail-dropped packets :          763229          13649 pps
RED-dropped packets :          224569          4243 pps
  Low         :          224569          4243 pps
RED-dropped bytes   :          344470114       52078592 bps
  Low         :          344470114       52078592 bps
Queue: 2, Forwarding classes: assured-forwarding
Queued:
Queue: 3, Forwarding classes: network-control
Queued:
  Bytes       :          4698          584 bps
  Bytes       :          4698          584 bps

```

NOTE The Bytes row displays total byte count followed by the bps (bits per second).

The drops are taking place in the expedited-forwarding queue, due to the default scheduling:

```

user@P> show class-of-service interface ge-2/3/0 | match scheduler
Scheduler map: <default>, Index: 2

```

```

user@P> show class-of-service scheduler-map | match scheduler | except -ch
Scheduler map: <default>, Index: 2
  Scheduler: <default-be>, Forwarding class: best-effort, Index: 21
    Transmit rate: 95 percent, Rate Limit: none, Buffer size: 95 percent, Buffer Limit: none,
Priority: low
  Scheduler: <default-nc>, Forwarding class: network-control, Index: 23
    Transmit rate: 5 percent, Rate Limit: none, Buffer size: 5 percent, Buffer Limit: none, Priority:
low

```

As a sample of what you can test in this scenario, try to wildly privilege expedited-forwarding in order to shift drops to the best-effort queue:

```

user@P> configure
user@P# set class-of-service schedulers EF priority strict-high
user@P# set class-of-service scheduler-maps EF-BEST forwarding-class expedited-forwarding scheduler
EF
user@P# set class-of-service interfaces ge-2/3/0 scheduler-map EF-BEST
user@P# commit and-quit

```

```

user@P> clear interfaces statistics all
user@P> show interfaces queue ge-2/3/0 | except " 0 "
Physical interface: ge-2/3/0, Enabled, Physical link is Up
  Interface index: 228, SNMP ifIndex: 16036
Forwarding classes: 16 supported, 4 in use
Egress queues: 8 supported, 4 in use
Queue: 0, Forwarding classes: best-effort
Queued:
  Packets      :          316058          49571 pps
  Bytes       :          484830086       608326584 bps
Transmitted:

```

```

Packets      :          203285          31840 pps
Bytes        :          311839190       390740480 bps
Tail-dropped packets :          87654       13764 pps
RED-dropped packets :          25119       3967 pps
  Low        :          25119          3967 pps
RED-dropped bytes :          38532546     48683744 bps
  Low        :          38532546     48683744 bps
Queue: 1, Forwarding classes: expedited-forwarding
Queued:
  Packets      :          316055          49570 pps
  Bytes        :          484803086     608293720 bps
Transmitted:
  Packets      :          316055          49570 pps
  Bytes        :          484803086     608293720 bps
Queue: 2, Forwarding classes: assured-forwarding
Queued:
Queue: 3, Forwarding classes: network-control
Queued:
  Bytes        :          537           520 bps
  Bytes        :          537           520 bps

```

TRY THIS Send traffic in different queues in the *Loop Till it Burns* setup discussed a few pages ago. The packets should be nicely spread according to priorities and bandwidth reservations.

ALERT! This is just a simple example, it by no means represents a design recommendation about how to configure scheduling in production networks. In fact, this is a bad choice for a scheduler-map.

Answers to Try It Yourself Sections of Chapter 5

Try It Yourself: Measuring Throughput with Snake Tests

First, in a full duplex test involving 20 ports, the number of hops in one direction is half of the number of links: 10.

The original question is tough to answer from a mathematical perspective. The reverse is easier, though: if the probability of a packet to be dropped in a hop is $N\%$, what would be the measured loss in the snake test? Assume that the physical links are healthy and all the routes are in place. If a packet doesn't reach the receiver, there are 10 possibilities: it may have been dropped in hop #1, or in hop #2, etc., or finally in hop #10. You already know the probability for the packet to be dropped in hop #1: $N\%$. For the packet to be dropped in hop #2, two things must happen: first, that it wasn't dropped in hop #1, and second, that once it reaches hop #2 it is dropped there. The probabilities of these two events are $(100-N)\%$ and $N\%$, respectively. The probability for the combined event is: $[(100-N)/100]*N\%$. You can iterate this logic with the following perl script:

```
#!/usr/bin/perl

(my $number_of_hops, my $per_hop_drop_percent) = @ARGV;

if (($ARGV[0] < 1) || ($ARGV[1] < 0) || ($ARGV[1] > 100)) {
    print "Usage: perl snake-drop-probability.pl <number-of-hops> <per-hop-drop-percent>\n";
    exit;
}

my $drop_probability = $per_hop_drop_percent / 100;

if ($number_of_hops > 0) {
    for ($i = 1; $i < $number_of_hops; $i++) {
        $drop_probability += ($per_hop_drop_percent / 100) * (1 - $drop_probability);
    }
}

my $drop_percent = 100 * $drop_probability;

print "The total drop probability in the path is ".$drop_percent."%\n";
```

Then do a binary search, trying different possible values of `$per_hop_drop_percent` until you hit a `$drop_percent` of 10%. The `$number_of_hops` is always 10:

```
$ perl snake-drop-probability.pl 10 10
The total drop probability in the path is 65.13215599%

$ perl snake-drop-probability.pl 10 1
The total drop probability in the path is 9.56179249911955%

$ perl snake-drop-probability.pl 10 1.05
The total drop probability in the path is 10.0175144319432%

$ perl snake-drop-probability.pl 10 1.04
The total drop probability in the path is 9.92653573725331%
```

So the real throughput is $100 - 1.05 = 98.95\%$. Much better than 90%, wouldn't you say?

CAUTION This theory assumes that there is a common bottleneck for all hops. If that's not the case, things get more complex. Conclusion: if possible, try not to measure throughput with snake tests!

Try It Yourself: Remote Loopback Mode

First, request remote loopback from PE1:

```
user@PE1> configure
user@PE1# set protocols oam ethernet link-fault-management interface ge-1/0/0 remote-loopback
user@PE1# commit and-quit
```

Next, grant the request at X:

```
user@X> configure
user@X# set protocols oam ethernet link-fault-management interface ge-0/0/11.0 negotiation-options
allow-remote-loopback
user@X# commit and-quit
```

Verify that X ge-0/0/11 is in remote loopback mode:

```
user@X> show oam ethernet link-fault-management ge-0/0/11 | match "loopback status"
Remote loopback status: Enabled on local port, Disabled on peer port
```

Finally, verify that the loop is working fine:

```
user@PE1> ping 10.100.1.2 count 1 no-resolve
PING 10.100.1.2 (10.100.1.2): 56 data bytes
36 bytes from 10.100.1.1: Time to live exceeded
Vr HL TOS Len ID Flg off TTL Pro cks Src Dst
4 5 00 0054 516c 0 0000 01 01 5173 10.100.1.1 10.100.1.2
```

```
--- 10.100.1.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

Try It Yourself: Counting Packets at MPLS Snakes

The key is that icmp-tunneling is active, so the time exceeded message is MPLS switched to the tail of the snake. Figure 5.21 shows the forwarding path. To move on, remove the icmp-tunneling knob:

```
user@P> configure
user@P# delete protocols mpls icmp-tunneling
user@P# commit and-quit
```

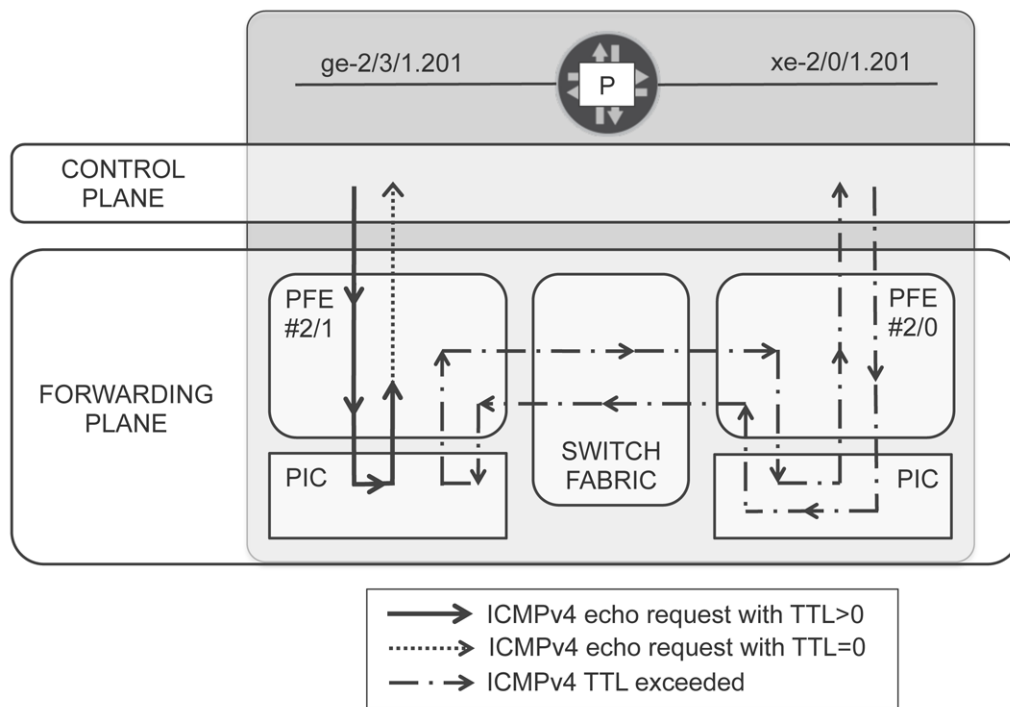


Figure 5.21 Forwarding Path of a Half-duplex IPv4 over MPLS Snake Ping with TTL=1

Try It Yourself: Getting Rid of TTL Exceptions

First, stop the particle accelerator in order to clean the state and start from scratch:

```
user@P> configure
user@P# deactivate interfaces xe-2/1/0 unit 0 family inet filter
user@P# commit and-quit
```

Now start it again, adding counters to measure the TTL of the packets in the loop:

```
user@P> configure
user@P# edit firewall family inet filter MIRROR-INET
user@P# set term MIRROR then next term
user@P# set term TTL-1 from ttl 1
user@P# set term TTL-1 then count TTL-1
user@P# set term TTL-2 from ttl 2
user@P# set term TTL-2 then count TTL-2
user@P# set term TTL-3 from ttl 3
user@P# set term TTL-3 then count TTL-3
user@P# set term TTL-4 from ttl 4
user@P# set term TTL-4 then count TTL-4
user@P# set term TTL-5 from ttl 5
user@P# set term TTL-5 then count TTL-5
user@P# set term TTL-6 from ttl 6
user@P# set term TTL-6 then count TTL-6
user@P# set term TTL-7 from ttl 7
user@P# set term TTL-7 then count TTL-7
user@P# set term TTL-8 from ttl 8
```



```

user@P# set term TTL-8 then count TTL-8
user@P# set term TTL-9 from ttl 9
user@P# set term TTL-9 then count TTL-9
user@P# set term TTL-10 from ttl 10
user@P# set term TTL-10 then count TTL-10
user@P# top
user@P# activate interfaces xe-2/1/0 unit 0 family inet filter
user@P# commit and-quit
commit complete
Exiting configuration mode

```

```

user@P> clear firewall all
user@P> clear interfaces statistics all

```

```

user@P> ping 10.200.1.2 count 1 no-resolve wait 1 ttl 10
PING 10.200.1.2 (10.200.1.2): 56 data bytes

```

```

--- 10.200.1.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss

```

```

user@P> show firewall | match ttl
TTL-1                      32355524208          385184812 /* Growing */
TTL-10                     84                  1
TTL-2                      21504               256
TTL-3                      10752               128
TTL-4                      5376                64
TTL-5                      2688                32
TTL-6                      1344                16
TTL-7                      672                 8
TTL-8                      336                 4
TTL-9                      168                 2

```

```

user@P> show ddos-protection protocols violations
Number of packet types that are being violated: 1
Protocol Packet Bandwidth Arrival Peak Policer bandwidth
group type (pps) rate(pps) rate(pps) violation detected at
ttl aggregate 2000 822742 843456 2012-10-30 12:10:08 CET
Detected on: FPC-2

```

It's easy to stop these Control Plane exceptions. Just discard the TTL=1 packets with the firewall filter: they will still be copied and keep filling the port!

```

user@P> configure
user@P# set firewall family inet filter MIRROR-INET term TTL-1 then discard
user@P# commit and-quit

```

```

user@P> show ddos-protection protocols violations
Number of packet types that are being violated: 1
Protocol Packet Bandwidth Arrival Peak Policer bandwidth
group type (pps) rate(pps) rate(pps) violation detected at
ttl aggregate 2000 0 843456 2012-10-30 12:10:08 CET
Detected on: FPC-2

```

```

user@P> clear ddos-protection protocols ttl states

```

```

user@P> show ddos-protection protocols violations
Number of packet types that are being violated: 0

```

```

user@P> show firewall | match ttl
TTL-1                      3428832204          40819431 /* Growing */
TTL-10                     84                  1
TTL-2                      21504               256

```

TTL-3	10752	128
TTL-4	5376	64
TTL-5	2688	32
TTL-6	1344	16
TTL-7	672	8
TTL-8	336	4
TTL-9	168	2

Try It Yourself: Playing with MPLS Rewrite Rules

You need to change the behavior during the *transit* push:

```
user@P> configure
user@P# set class-of-service rewrite-rules exp CUSTOM-EXP-RW forwarding-class expedited-forwarding
loss-priority low code-point 101
user@P# commit and-quit
```

Appendix

<i>Initial Configuration of the Routers</i>	<i>158</i>
<i>Initial Configuration of the Switch</i>	<i>164</i>
<i>Initial Configuration of the Host</i>	<i>167</i>
<i>Basic Connectivity Tests</i>	<i>167</i>



This Appendix contains the initial configuration of each of the devices included in the book. Only the relevant sections for the test scenarios are displayed. Other generic information like system or management IP addresses is omitted, as it has no influence on the tests.

Initial Configuration of the Routers

PE1 (MX80)

PE1 is a PE-router with two uplinks and two routing-instances: a virtual-router called CE1 that emulates a CE, and a vrf called VRF1.

TIP If you use Logical Systems, do not configure `vrf-table-label` at the VRFs. Use `vt-` interfaces instead. An example is provided at the *Try it Yourself* section of Chapter 1.

CAUTION Do not add a default IPv4 route at CE1. It's missing on purpose.

```
interfaces {
  ge-1/0/0 {
    vlan-tagging;
    unit 101 {
      vlan-id 101;
      family inet {
        address 10.1.1.2/30;
      }
      family inet6 {
        address fc00::1:2/112;
      }
    }
    unit 111 {
      vlan-id 111;
      family inet {
        address 10.100.1.1/30;
      }
      family iso;
      family mpls;
      family inet6;
    }
  }
  ge-1/0/1 {
    vlan-tagging;
    unit 101 {
      vlan-id 101;
      family inet {
        address 10.1.1.1/30;
      }
      family inet6 {
        address fc00::1:1/112;
      }
    }
    unit 112 {
      vlan-id 112;
      family inet {
        address 10.100.2.1/30;
      }
      family iso;
    }
  }
}
```

```

        family mpls;
        family inet6;
    }
}
lo0 {
    unit 0 {
        family inet {
            address 10.111.1.1/32;
        }
        family iso {
            address 49.0101.1100.1001.00;
        }
    }
}
}
routing-options {
    autonomous-system 65000;
    forwarding-table {
        export LB;
    }
}
}
protocols {
    mpls {
        ipv6-tunneling;
        interface ge-1/0/0.111;
        interface ge-1/0/1.112;
    }
    bgp {
        group IBGP {
            type internal;
            local-address 10.111.1.1;
            family inet-vpn {
                unicast;
            }
        }
    }
    isis {
        level 1 disable;
        interface ge-1/0/0.111 {
            point-to-point;
        }
        interface ge-1/0/1.112 {
            point-to-point;
        }
        interface lo0.0 {
            passive;
        }
    }
    ldp {
        interface ge-1/0/0.111;
        interface ge-1/0/1.112;
    }
    rsvp {
        interface ge-1/0/0.111;
        interface ge-1/0/1.112;
    }
}
}
policy-options {
    policy-statement LB {
        then {
            load-balance per-packet;
        }
    }
}
}

```

```

}
routing-instances {
  CE1 {
    instance-type virtual-router;
    interface ge-1/0/0.101;
    routing-options {
      rib CE1.inet6.0 {
        static {
          route 0::0/0 next-hop fc00::1:1;
        }
      }
    }
  }
  VRF1 {
    instance-type vrf;
    interface ge-1/0/1.101;
    route-distinguisher 65000:1;
    vrf-target target:65000:100;
    vrf-table-label;
  }
}

```

P (MX480)

P is a P-router with two physical and four logical uplinks.

```

interfaces {
  ge-2/3/0 {
    vlan-tagging;
    unit 111 {
      vlan-id 111;
      family inet {
        address 10.100.1.2/30;
      }
      family iso;
      family mpls;
    }
    unit 113 {
      vlan-id 113;
      family inet {
        address 10.100.3.2/30;
      }
      family iso;
      family mpls;
    }
  }
  xe-2/0/0 {
    vlan-tagging;
    unit 112 {
      vlan-id 112;
      family inet {
        address 10.100.2.2/30;
      }
      family iso;
      family mpls;
    }
    unit 114 {
      vlan-id 114;
      family inet {
        address 10.100.4.2/30;
      }
      family iso;
    }
  }
}

```

```

        family mpls;
    }
}
lo0 {
    unit 0 {
        family inet {
            address 10.111.11.11/32;
        }
        family iso {
            address 49.0101.1101.1011.00;
        }
    }
}
}
routing-options {
    forwarding-table {
        export LB;
    }
}
protocols {
    mpls {
        interface ge-2/3/0.111;
        interface ge-2/3/0.113;
        interface xe-2/0/0.112;
        interface xe-2/0/0.114;
    }
    isis {
        level 1 disable;
        interface ge-2/3/0.111 {
            point-to-point;
        }
        interface ge-2/3/0.113 {
            point-to-point;
        }
        interface xe-2/0/0.112 {
            point-to-point;
        }
        interface xe-2/0/0.114 {
            point-to-point;
        }
        interface lo0.0 {
            passive;
        }
    }
}
    ldp {
        interface ge-2/3/0.111;
        interface ge-2/3/0.113;
        interface xe-2/0/0.112;
        interface xe-2/0/0.114;
    }
    rsvp {
        interface ge-2/3/0.111;
        interface ge-2/3/0.113;
        interface xe-2/0/0.112;
        interface xe-2/0/0.114;
    }
}
policy-options {
    policy-statement LB {
        then {
            load-balance per-packet;
        }
    }
}

```

```
}
```

PE2 (MX80)

PE2 is a PE router with two uplinks and two routing-instances: a virtual-router called CE2 that emulates a CE, and a vrf called VRF2.

CAUTION Do not add a default IPv4 route at CE2. It's missing on purpose.

```
interfaces {
  ge-1/0/0 {
    vlan-tagging;
    unit 102 {
      vlan-id 102;
      family inet {
        address 10.2.2.2/30;
      }
      family inet6 {
        address fc00::2:2/112;
      }
    }
    unit 113 {
      vlan-id 113;
      family inet {
        address 10.100.3.1/30;
      }
      family iso;
      family mpls;
      family inet6;
    }
  }
  ge-1/0/1 {
    vlan-tagging;
    unit 102 {
      vlan-id 102;
      family inet {
        address 10.2.2.1/30;
      }
      family inet6 {
        address fc00::2:1/112;
      }
    }
    unit 114 {
      vlan-id 114;
      family inet {
        address 10.100.4.1/30;
      }
      family iso;
      family mpls;
      family inet6;
    }
  }
  lo0 {
    unit 0 {
      family inet {
        address 10.111.2.2/32;
        filter {
          input BLOCK-ICMP;
        }
      }
    }
  }
}
```



```

    }
    family iso {
        address 49.0101.1100.2002.00;
    }
}
}
}
routing-options {
    autonomous-system 65000;
    forwarding-table {
        export LB;
    }
}
protocols {
    mpls {
        ipv6-tunneling;
        interface ge-1/0/0.113;
        interface ge-1/0/1.114;
    }
    bgp {
        group IBGP {
            type internal;
            local-address 10.111.2.2;
            family inet-vpn {
                unicast;
            }
        }
    }
    isis {
        level 1 disable;
        interface ge-1/0/0.113 {
            point-to-point;
        }
        interface ge-1/0/1.114 {
            point-to-point;
        }
        interface lo0.0 {
            passive;
        }
    }
    ldp {
        interface ge-1/0/0.113;
        interface ge-1/0/1.114;
    }
    rsvp {
        interface ge-1/0/0.113;
        interface ge-1/0/1.114;
    }
}
policy-options {
    policy-statement LB {
        then {
            load-balance per-packet;
        }
    }
}
}
firewall {
    family inet {
        filter BLOCK-ICMP {
            term ICMP {
                from protocol icmp;
                then reject;
            }
        }
    }
}

```

```

        term REST {
            then accept;
        }
    }
}
routing-instances {
    CE2 {
        instance-type virtual-router;
        interface ge-1/0/0.102;
        routing-options {
            rib CE2.inet6.0 {
                static {
                    route ::/0 next-hop fc00::2:1;
                }
            }
        }
    }
    VRF2 {
        instance-type vrf;
        interface ge-1/0/1.102;
        route-distinguisher 65000:1;
        vrf-target target:65000:100;
        vrf-table-label;
    }
}

```

Initial Configuration of the Switch

X (EX4200)

The switch performs a double role. First, it allows for flexible connections between routers, and second, it mirrors traffic towards the host.

```

interfaces {
    ge-0/0/10 {
        mtu 1600;
        unit 0 {
            family ethernet-switching {
                port-mode access;
                vlan {
                    members v120;
                }
            }
        }
    }
    ge-0/0/11 {
        mtu 1600;
        unit 0 {
            family ethernet-switching {
                port-mode trunk;
                vlan {
                    members [ v101 v111 ];
                }
                filter {
                    input NO-ARP;
                }
            }
        }
    }
}

```

```

ge-0/0/12 {
    mtu 1600;
    unit 0 {
        family ethernet-switching {
            port-mode trunk;
            vlan {
                members [ v111 v113 ];
            }
        }
    }
}
ge-0/0/13 {
    mtu 1600;
    unit 0 {
        family ethernet-switching {
            port-mode trunk;
            vlan {
                members [ v113 v102 ];
            }
        }
    }
}
ge-0/0/14 {
    mtu 1600;
    unit 0 {
        family ethernet-switching {
            port-mode trunk;
            vlan {
                members [ v101 v112 ];
            }
            filter {
                input NO-ARP;
            }
        }
    }
}
ge-0/0/15 {
    mtu 1600;
    unit 0 {
        family ethernet-switching {
            port-mode trunk;
            vlan {
                members [ v114 v102 ];
            }
        }
    }
}
xe-0/1/0 {
    mtu 1600;
    unit 0 {
        family ethernet-switching {
            port-mode trunk;
            vlan {
                members [ v112 v114 ];
            }
        }
    }
}
}
firewall {
    family ethernet-switching {
        filter NO-ARP {
            term NO-ARP {

```

```

        from {
            ether-type arp;
            vlan v101;
        }
        then discard;
    }
    term REST {
        then accept;
    }
}

}

}

ethernet-switching-options {
    analyzer test-server {
        input {
            ingress {
                vlan v101;
                vlan v102;
                vlan v111;
                vlan v112;
                vlan v113;
                vlan v114;
            }
        }
        output {
            vlan {
                v120 {
                    no-tag;
                }
            }
        }
    }
}

}

vlangs {
    v101 {
        vlan-id 101;
    }
    v102 {
        vlan-id 102;
    }
    v111 {
        vlan-id 111;
    }
    v112 {
        vlan-id 112;
    }
    v113 {
        vlan-id 113;
    }
    v114 {
        vlan-id 114;
    }
    v120 {
        vlan-id 120;
    }
}

}

```

NOTE With the no-tag knob, mirrored packets will only have the original VLAN tag (101-102,111-114), but not the VLAN 120 tag. Without the knob, there is a double VLAN stack. You can safely ignore commit warning messages related to this mirroring configuration.

Initial Configuration of the Host

H (FreeBSD)

The host just needs an IP address configured at the interface where tcpdump is run.

```
[root@H ~]# ifconfig bce1 10.100.5.1 netmask 255.255.255.252
[root@H ~]#
```

Basic Connectivity Tests

PE1, P, and PE2 are connected to each other in Autonomous System 65000, with IS-IS and LDP running in the core links. Later in Chapter 1, there will be a Multi-protocol IBGP session between PE1 and PE2, initially used to exchange `inet-vpn` unicast prefixes of the VRFs.

Check one-hop IP connectivity is fine at the backbone, by executing at P:

```
user@P> ping 10.100.1.1 count 1
user@P> ping 10.100.2.1 count 1
user@P> ping 10.100.3.1 count 1
user@P> ping 10.100.4.1 count 1
```

Check one-hop IP connectivity is broken at the VRF1-CE1 and VRF2-CE2 links:

```
user@PE1> ping 10.1.1.1 routing-instance CE1 count 1
user@PE2> ping 10.2.2.1 routing-instance CE2 count 1
```

IMPORTANT

The later two pings are expected to fail at this stage.

Verify that everything is fine at the ISIS and LDP level. The following commands are executed on PE1, you can do similar checks for P and PE2, too:

```
user@PE1> show isis adjacency
Interface      System      L State      Hold (secs) SNPA
ge-1/0/0.111   P           2 Up          26
ge-1/0/1.112   P           2 Up          21

user@PE1> show ldp neighbor
Address        Interface      Label space ID      Hold time
10.100.1.2     ge-1/0/0.111   10.111.2.2:0        11
10.100.2.2     ge-1/0/1.112   10.111.2.2:0        14

user@PE1> show ldp session
Address        State      Connection      Hold time
10.111.2.2     Operational Open            28
```

No BGP sessions are established, since you did not configure any neighbors yet (that is expected):

```
user@PE1> show bgp summary
Groups: 0 Peers: 0 Down peers: 0
Table      Tot Paths  Act Paths Suppressed  History Damp State  Pending
bgp.13vpn.0
           0         0         0         0         0         0
```

```
user@PE1> show route table CE1
```

```
CE1.inet.0: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

```

10.1.1.0/30      *[Direct/0] 02:35:31
                  > via ge-1/0/0.101
10.1.1.2/32      *[Local/0] 02:35:51
                  Local via ge-1/0/0.101

CE1.inet6.0: 5 destinations, 5 routes (5 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

::/0            *[Static/5] 00:08:02
                  > to fc00::1:1 via ge-1/0/0.101
fc00::1:0/112    *[Direct/0] 00:22:04
                  > via ge-1/0/0.101
fc00::1:2/128    *[Local/0] 00:22:04
                  Local via ge-1/0/0.101
fe80::/64        *[Direct/0] 00:22:04
                  > via ge-1/0/0.101
fe80::5e5e:ab00:650a:c360/128
                  *[Local/0] 00:22:04
                  Local via ge-1/0/0.101

user@PE1> show route table VRF1

VRF1.inet.0: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.1.1.0/30      *[Direct/0] 01:35:06
                  > via ge-1/0/1.101
10.1.1.1/32      *[Local/0] 01:35:06
                  Local via ge-1/0/1.101

VRF1.inet6.0: 4 destinations, 4 routes (4 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

fc00::1:0/112    *[Direct/0] 00:22:33
                  > via ge-1/0/1.101
fc00::1:1/128    *[Local/0] 00:22:33
                  Local via ge-1/0/1.101
fe80::/64        *[Direct/0] 00:22:33
                  > via ge-1/0/1.101
fe80::5e5e:ab00:650a:c361/128
                  *[Local/0] 00:22:33
                  Local via ge-1/0/1.101

```