Juniper Networks | Engineering Simplicity

DAY ONE: INSIDE THE MX 5G

By David Roy

DAY ONE: INSIDE THE MX 5G

Nothing gets network engineers more excited than a packet walkthrough within the most fabled routing platform in the world, the Juniper Networks® MX Series 5G Universal Routing Platform. With its new chassis, new hardware, and new ASICs, David Roy, author of several books on the Juniper MX Series, shows off the box and then gets right into the EA and ZT linecards. Step-by-step, David shows how packets navigate these monster ASICs.

This is the ultimate trip into big iron for those who can appreciate the scalability and programmability of Juniper silicon. Follow the packet and David's CLI command sequences as they illustrate why the MX 5G and Junos® powers networks around the world.

"Have you ever wondered what actually happens to data packets entering your routers? This book gives an excellent and very detailed view of the life of a packet "inside the box." I've been working with Juniper Networks routers for about 15 years now and have never seen anything publicly available that even comes close to this level of detail, yet the book is an easy read and David perfectly guides you through it. Even having years of hands on experience with the MX, I've learned so much! If you really want to understand what is happening with a packet inside the box, this is the book to read."

Melchior Aelmans, Lead Engineer, Cloud Providers, Juniper Networks

"When it comes to an MX PFE packet walkthrough, there is no better person to take you on that journey than Dvid Roy. David has a deep and thorough understanding of the EA and ZT PFEs and shares his knowledge in this exceptionally written book. Juniper's fifth generation of TRIO PFEs introduces key new innovations for engineers who are able to take their knowledge to the next level."

Daniel Hearty, Juniper Ambassador, Principal Engineer, Telent,

IT'S DAY ONE AND YOU HAVE A JOB TO DO, SO LEARN HOW TO:

- Size the MX 5G family of products and discover how to apply them to your needs.
- Gain new skills with MX 5G hardware based on the EA or ZT ASICs.
- Understand the life of unicast, MPLS, and host packets in the MX 5G.
- Perform advanced troubleshooting on the MX 5G Series hardware.

ISBN 978-1-941441-77-0
53500
9 781941 441770

Day One: Inside the MX 5G

by David Roy

Chapter 1: The MX 5G: New Chassis, New Hardware 9

Chapter 2: The MX 5G: Two Powerful ASICs 39

Chapter 3: Follow Some Packets 85

Appendices 156

JUNIPER NETWORKS

Published by Juniper Networks Books
Author: David Roy
Technical Reviewers: Josef Buchsteiner, Daniel Hearty, Ramesh Pillutla
Editor in Chief: Patrick Ames
Copyeditor: Nancy Koerbel
Illustrations: David Roy
TechLibrary Resources: Saheer Karimbayil

Printed in the USA by Vervante Corporation.
Version History: v1, July. 2020
2 3 4 5 6 7 8 9 10
Comments, errata: dayone@juniper.net

About the Author

David Roy is a network support engineer who works for one of the main Service Providers in Europe: Orange. During the last 12 years he was involved in many projects based on IP and MPLS technologies. He is also a focus technical support engineer for the French domestic backbone of Orange. Before that, he was part of a Research and Development team focused on Digital Video Broadcasting and IP over Satellite technologies. He loves to troubleshoot complex routing and switching issues and has spent much time in the lab to reverse engineer different routing platforms such as the Juniper MX series. He wrote the second edition of *MX Series*, an O'Reilly book, and the *Day One* book: *This Week: An Expert Packet Walkthrough on the MX Series 3D*. David is triple JNCIE SP #703, ENT #305, and SEC #144. When he's not diving into the hardware's routers, he plays drums, listens to rock, and savors some nice beers. David can be reached on Twitter @door7302.

Author's Acknowledgments

I would like to thank my wife, Magali, and my two sons, Noan and Timéo, for all their encouragement and support. A very special thank you to Josef Buchsteiner from Juniper Networks for helping me during the project and who was also the main technical reviewer. A great thank you to Daniel Hearty and Ramesh Pillutla for their deep technical review, and to Patrick Ames for his review and assistance.
David Roy, IP/MPLS NOC Engineer, Orange France
JNCIE x3 (SP #703 ; ENT #305 ; SEC #144).

Welcome to Day One

This book is part of the *Day One* library, produced and published by Juniper Networks Books. *Day One* books cover the Junos OS and Juniper Networks network-administration with straightforward explanations, step-by-step instructions, and practical examples that are easy to follow.

- Download a free PDF edition at http://www.juniper.net/dayone
- PDF books are available on the Juniper app: Junos Genius
- Purchase the paper edition at Vervante Corporation (www.vervante.com).

What You Need to Know Before Reading This Book

- You need to be very familiar with the Junos Operating System.
- You need to know the architecture of recent routers such as control plane and forwarding plane separation.
- You need to have some knowledge about network hardware.

After Reading This Book You'll Be Able To:

- Size the MX 5G family of products and discover how to apply them to your needs.
- Gain new skills with MX 5G hardware based on the EA or ZT ASICs.
- Understand the life of unicast, MPLS, and host packets in the MX 5G.
- Perform advanced troubleshooting on the MX 5G Series hardware.

How This Book Is Set Up

- Chapter 1 of this book provides an overview of the MX 5G family. It presents all the routers and their associated linecards.
- Chapter 2 takes a dive inside the hardware. It details how EA and ZT linecards are designed and, step-by-step, how packets are handled and manipulated by these two powerful ASICs.
- Chapter 3 relies on two simple examples of traffic – an MPLS transit traffic and a simple ping flow – to explain with the most details possible how packets are managed by the EA or ZT ASIC.

MX 5G Resources in the Juniper TechLibrary

GENERAL RESOURCES	
Network Management and Monitoring Guide	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/network-management/network-management.html
Routing Protocols Overview	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/config-guide-routing/config-guide-routing-overview.html
Software Installation and Upgrade Guide	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/software-installation-and-upgrade/software-installation-and-upgrade.html
Introducing Junos OS Evolved	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/introducing-evo-guide.html
Multicast Protocols User Guide	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/config-guide-multicast/config-guide-multicast.html
Overview for Junos OS	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/system-basics/junos-overview.html
MPLS Applications User Guide	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/config-guide-mpls-applications/config-guide-mpls-applications.html
High Availability Feature Guide	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/config-guide-high-availability/high-availability.html
Chassis-Level User Guide	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/system-basics/router-chassis.html

HARDWARE (Chassis MX) RESOURCES	
MX240 Universal Routing Platform Hardware Guide	https://www.juniper.net/documentation/en_US/release-independent/junos/information-products/pathway-pages/mx-series/mx240/
MX480 Universal Routing Platform Hardware Guide	https://www.juniper.net/documentation/en_US/release-independent/junos/information-products/pathway-pages/mx-series/mx480/
MX960 Universal Routing Platform Hardware Guide	https://www.juniper.net/documentation/en_US/release-independent/junos/information-products/pathway-pages/mx-series/mx960/

MX2010 Universal Routing Platform Hardware Guide	https://www.juniper.net/documentation/en_US/release-independent/junos/information-products/pathway-pages/mx-series/mx2010/index.html
MX2008 Universal Routing Platform Hardware Guide	https://www.juniper.net/documentation/en_US/release-independent/junos/information-products/pathway-pages/mx-series/mx2008/
MX2020 Universal Routing Platform Hardware Guide	https://www.juniper.net/documentation/en_US/release-independent/junos/information-products/pathway-pages/mx-series/mx2020/index.html

SOFTWARE RESOURCES	
Interface Naming Conventions for Rate Selectability	https://www.juniper.net/documentation/en_US/junos/topics/topic-map/interface-naming-conventions-rate-selectability.html
Configuring Rate Selectability on MX10003 MPC to Enable Different Port Speeds	https://www.juniper.net/documentation/en_US/junos/topics/topic-map/rate-selectability-configuring.html#id-configuring-rate-selectability-on-mx10003-mpc-to-enable-different-port-speeds
Configuring Rate Selectability on MX204 to Enable Different Port Speeds	https://www.juniper.net/documentation/en_US/junos/topics/topic-map/rate-selectability-configuring.html#id-configuring-rate-selectability-on-mx204-to-enable-different-port-speeds
Configuring Rate Selectability on JNP10K-2101 MPC to Enable Different Port Speeds	https://www.juniper.net/documentation/en_US/junos/topics/topic-map/rate-selectability-configuring.html#id-configuring-rate-selectability-on-jnp10k-2101-mpc-to-enable-different-port-speeds
Configuring Rate Selectability on MPC10E-15C-MRATE to Enable Different Port Speeds	https://www.juniper.net/documentation/en_US/junos/topics/topic-map/rate-selectability-configuring.html#id-configuring-rate-selectability-on-mpc10e-15c-mrate-to-enable-different-port-speeds
Configuring Rate Selectability on MPC10E-10C-MRATE to Enable Different Port Speeds	https://www.juniper.net/documentation/en_US/junos/topics/topic-map/rate-selectability-configuring.html#id-configuring-rate-selectability-on-mpc10e-10c-mrate-to-enable-different-port-speeds
Configuring Rate Selectability on the MX2K-MPC11E to Enable Different Port Speeds	https://www.juniper.net/documentation/en_US/junos/topics/topic-map/rate-selectability-configuring.html#id-configuring-rate-selectability-on-mpc11e
Class of Service User Guide (Routers and EX9200 Switches)	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/cos/config-guide-cos.html
CLI User Guide	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/junos-cli/junos-cli.html
CLI Explorer	https://apps.juniper.net/cli-explorer/

HARDWARE RESOURCES	
MX204 Universal Routing Platform Hardware Guide	https://www.juniper.net/documentation/en_US/release-independent/junos/information-products/pathway-pages/mx-series/mx204/index.html
MX10003 Universal Routing Platform Hardware Guide	https://www.juniper.net/documentation/en_US/release-independent/junos/information-products/pathway-pages/mx-series/mx10003/index.html
MX10008 Universal Routing Platform Hardware Guide	https://www.juniper.net/documentation/en_US/release-independent/junos/information-products/pathway-pages/mx-series/mx10008/index.html
MX10016 Universal Routing Platform Hardware Guide	https://www.juniper.net/documentation/en_US/release-independent/junos/information-products/pathway-pages/mx-series/mx10016/index.html
MPC10E-10C-MRATE	https://www.juniper.net/documentation/en_US/release-independent/junos/topics/reference/general/mpc10e-10c-mrate.html
MPC10E-15C-MRATE	https://www.juniper.net/documentation/en_US/release-independent/junos/topics/reference/general/mpc10e-15c-mrate.html
MX2K-MPC11E Modular Port Concentrator	https://www.juniper.net/documentation/en_US/release-independent/junos/topics/reference/general/mpc11e.html
MX10003 MPC (Multi-Rate)	https://www.juniper.net/documentation/en_US/release-independent/junos/topics/reference/general/mpc10003.html
Line card (MX10K-LC2101)	https://www.juniper.net/documentation/en_US/release-independent/junos/topics/reference/general/mx10008-line-card-descripion.html
MX Series 5G Universal Routing Platform Interface Module Reference	https://www.juniper.net/documentation/en_US/release-independent/junos/information-products/pathway-pages/mx-series/mx-module-index.html

Chapter 1

The MX 5G: New Chassis and Hardware

Juniper has recently released a new line of high-speed equipment called the *5G Universal Routing Platforms*. This new line of routers relies on the MX204, the MX10000 Series (aka *MX10K*), and some new MPCs for classic MXs (the MX240, MX480, MX960, MX2008, MX2010, and MX2020). The MX204 is the smallest router of the 5G family: a 1RU fixed platform. The MX10K Series includes the following members:

- MX10003
- MX10008
- MX10016

The 5G Universal Routing Platforms leverage the past innovations implemented on the MX series and specifically rely on the TRIO chipset: a key element of MX routers over the last several years. This new chassis has been completely redesigned to improve power consumption, provide a better cooling environment, and, of course, to handle very high rate interfaces (40Gbps, 100Gbps, 400Gbps, and even more) on a new generation of Line cards.

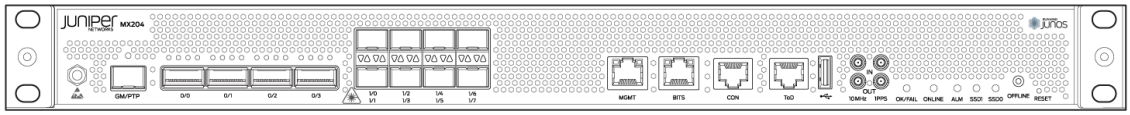
This chapter presents the smallest member of the 5G family: the MX204, then it details the MX10K Series, and finally, as we don't want to forget the classic MXs, it details two new line cards for current MXs in production. Indeed, these two MPCs are the first ones that embed the fifth generation of TRIO ASIC and thus naturally join the 5G family:

- The MPC10e for MX240, MX480, and MX960
- The MPC11e for MX2010 and MX2020

MX204: Small but Powerful

Let's introduce the MX204. This compact router is a fixed-form factor 1RU platform. It provides a capacity of 400Gbps and fully supports all Junos features such as HQoS. As expected, in order to offer this rich feature set the MX204 is not based on a merchant silicon chipset but on the fourth generation of TRIO ASIC whose code name is *EA* (aka Eagle ASIC). Figure 1.1 depicts the front view of the MX204.

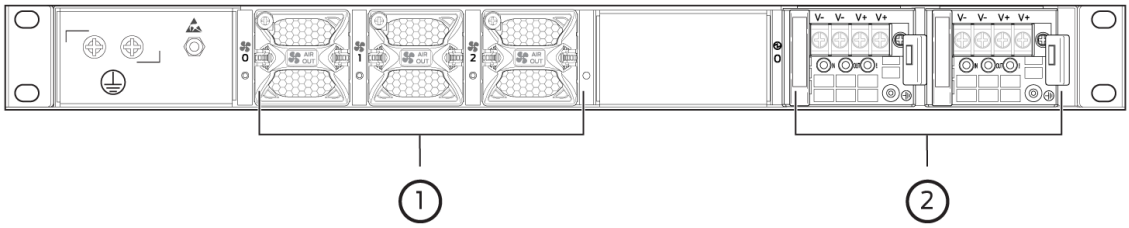
Figure 1.1 MX204 Front View



This router has a single routing engine (RE) that relies on eight cores, 32GB of memory, and a 100GB internal SSD disk. The RE also provides several management interfaces that are accessible from the front of the device. Moreover, you can observe that the MX204 has two blocks of network ports. The first one is a built-in PIC of four multi-rates QSFPP ports, and the second PIC, also fixed, provides eight 10GE SFP+ physical ports. From the rear view in Figure 1.2 we see:

- Fan modules
- Power supply modules (here DC but AC is also supported)

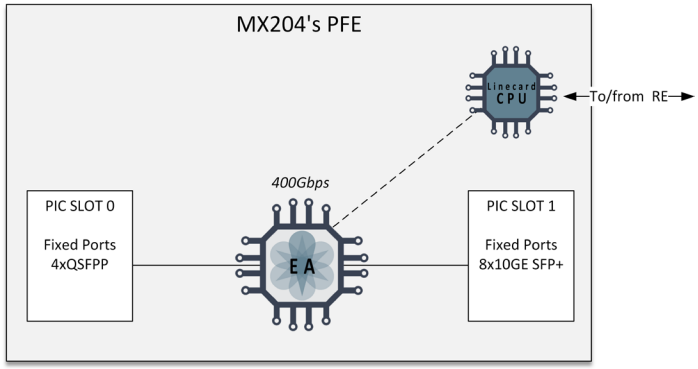
Figure 1.2 MX204 Rear View



As observed, the MX204 provides power and cooling redundancy.

Now let's open the MX204 to see its powerful engine. As mentioned previously, the MX204 is made of one 400Gbps EA ASIC. The two built-in PICs are attached to this single EA. Figure 1.3 shows the internal architecture of the MX204 PFE.

Figure 1.3 MX204 Packet Forwarding Engine



The built-in PIC in slot 0 consists of four QSFPP multi-rate ports where each port may be configured as follows depending on the port profile:

- 1x1GE
- 4x10GE with a breakout cable for 4x10GE lanes
- 1x40GE per QSFPP port
- 1x100GE per QSFPP port

On this PIC, the rate of each port is by default 4x10GE. To override this default behavior you can modify the port profile mode (at the PIC or port level). This is achieved by the following CLI commands:

```
[edit]
door7302@mx204# set chassis fpc 0 pic 0 pic-mode ?
Possible completions:
  100G                 100GE mode
  10G                  10GE mode
  40G                  40GE mode

door7302@mx204# set chassis fpc 0 pic 0 number-of-ports ?
Possible completions:
  <number-of-ports>    Number of physical ports to enable on PIC (1..96)

door7302@mx204# set chassis fpc 0 pic 0 port 0 speed ?
Possible completions:
  100g                 Sets the interface mode to 100Gbps
  10g                  Sets the interface mode to 10Gbps
  40g                  Sets the interface mode to 40Gbps
```

The first command shown allows the configuration of the port profile at the PIC level, which means that all ports attached to the PIC will operate at the same configured speed. The next command is used to handle an oversubscription scenario that's usually used when the port profile at port level is preferred. Finally, the third command shows you how to configure the port profile at port level.

The port mapping of the built-in PIC is depicted by Figure 1.4.

Figure 1.4 4xQSFPP Port Mapping

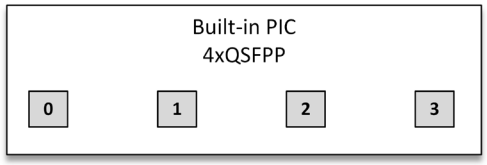


Table 1.1 summarizes the port-naming convention of the built-in PIC in slot 0, depending on the configured port profile mode.

Table 1.1 Interface Naming of the 4xQSFPP PIC

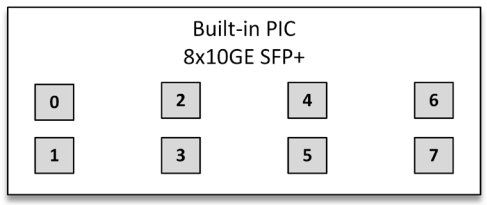
PFE COMPLEX	Port Number	10GE mode	40GE or 100GE mode
EA ASIC 0	0	xe-0/0/0:[0..3]	et-0/0/0
	1	xe-0/0/1:[0..3]	et-0/0/1
	2	xe-0/0/2:[0..3]	et-0/0/2
	3	xe-0/0/3:[0..3]	et-0/0/3

The second fixed PIC in slot 1 consists of eight 10GE SFP+ ports. These ports, as the four QSFPP ports of the PIC 0, also support 1GE port speed. To configure 1GE speed on a given port you need to use this specific command at the interface level:

```
door7302@mx204# set interfaces xe-x/x/x gether-options speed 1g
```

The port mapping of this second PIC is shown on Figure 1.5.

Figure 1.5 8xSFP+ Port Mapping



And Table 1.2 summarizes the interface naming of this 10GE PIC.

Table 1.2 Interface Naming of the 8xSFP+ PIC

PFE COMPLEX	Port Number	10GE mode
EA ASIC 0	0	xe-0/1/0
	1	xe-0/1/1
	2	xe-0/1/2
	3	xe-0/1/3
	4	xe-0/1/4
	5	xe-0/1/5
	6	xe-0/1/6
	7	xe-0/1/7

That's it for the MX204 introduction, next up we move on to the larger 5G routers.

The MX10003: Modular and Compact

The new MX10003 is a compact and modular router, three rack units (3RU) tall, that can provide up to 2.4Tbps of capacity (1.2Tbps per slot) and a large set of routing and switching features as it leverages the TRIO ASIC, one of the most powerful network chipsets in the world.

NOTE The capacity of the MX10003 at its release was 2.4Tbps. As you will see in detail later on, the line card currently available for MX10003 is based on the 4th generation of TRIO ASIC. As in all the MXs, the MX10003 supports the next generation of TRIO ASICs that will provide greatly enhanced slot and per chassis throughput.

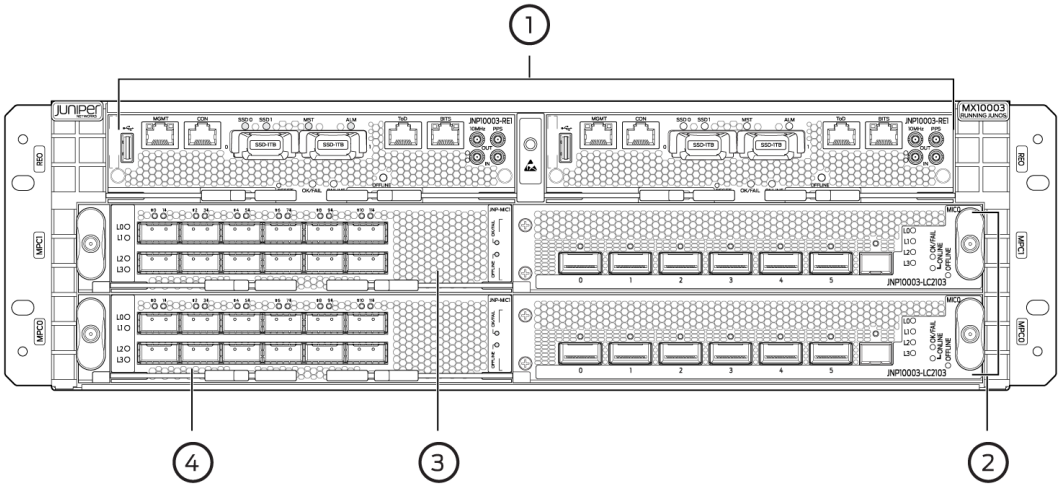
Let's first go *inside* the MX10003 chassis and its several components. Then we'll focus on its 1.2Tbps line card: the LC2103.

The MX10003 Chassis

As mentioned previously, the MX10003 is a compact chassis (3RU), which offers two modular slots for hosting MPCs. Figure 1.6 shows the front view of a fully configured MX10003. As you can see there are:

1. Two Routing and Control Boards (RCBs)
2. Modular Port Connectors (MPCs)
3. An MPC slot with a modular slot for a MIC daughter card
4. A second MPC slot with a modular slot for a MIC daughter card

Figure 1.6 MX10003 Front View

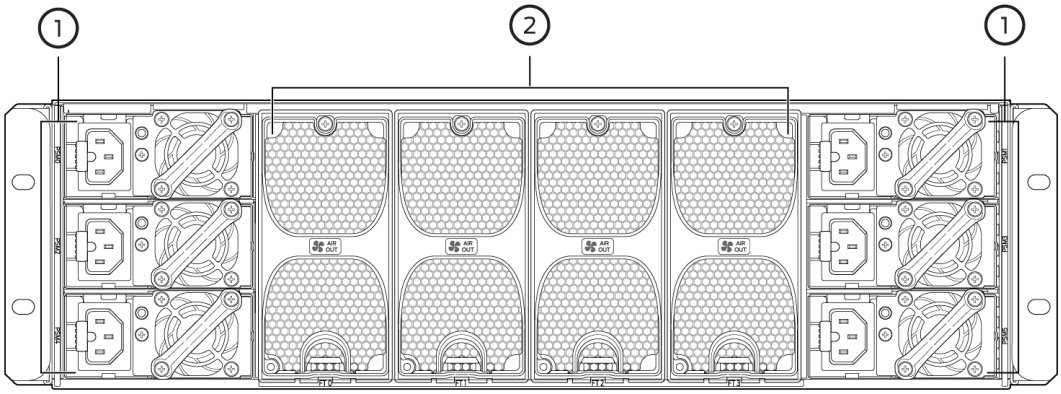


NOTE The RE component actually hosts two separate functions: the routing engine, which runs the Junos OS and the Control Board (CB), which manages all the components of the chassis. Thus, the RE and CB assembled together are called the *RCB*, for Routing and Control Boards.

As detailed later in this chapter, the current MX10003 MPC is made of two MICs: one is a built-in MIC (MIC in slot 0) and the second one is modular (MIC in slot 1). Currently each slot offers 1.2Tbps of capacity but will be upgradeable to 7.2Tbps in the future. The current MPC supports multirate interfaces: 12x100GE, or 18x40GE, or 72x10GE per line card.

Now let's have a look at the rear view of the MX10003, shown in Figure 1.7. There are six power supply modules (PSMs) – three at each end. The MX10003 supports both AC *or* DC PSM. In the middle of the chassis are the four fan modules for the cooling function.

Figure 1.7 MX10003 Rear View



When a chassis is shipped with the two RCBs, the six PSMs, and four fan trays, the system is considered fully redundant meaning that a single point of failure of one hardware component will not cause the entire system to fail. In fact, dual RCBs with nonstop active routing feature (aka NSR) ensures that the control plane will not be impacted in the event of a single RCB failure. At least three AC or DC PSM modules are required to feed the entire chassis. The additional PSMs provide N+1 and N+N power redundancy. Despite the fact the chassis needs the four fan modules to provide optimized front-to-back air-cooling, the failure of one fan is not disruptive. Indeed, the speed of the remaining fans is adjusted to keep the temperature below the threshold that might trigger a power-off of the chassis to save the component's integrity. Table 1.3 depicts which components are hot-removable/insertable and which ones are hot-pluggable.

Table 1.3 FRU Types for MX10003

FRU Type	Hot-insertable	Hot-removable	Hot-pluggable
Master RCB with NSR		X	X
Master RCB without NSR			X
Backup RCB with NSR	X	X	
Backup RCB without NSR	X	X	
MPC	X	X	
MIC (installed in slot 1)	X	X	
PSM	X	X	
Fan module	X	X	
Air Filter unit	X	X	
Optic module	X	X	

NOTE Hot-removable and hot-insertable field replaceable units (FRUs): you can remove and replace these components without powering off the router or disrupting the routing functions. Hot-pluggable FRUs: you can remove and replace these components without powering off the router, but the routing functions of the system are interrupted when the component is removed.

In order to summarize what's been discussed previously, let's issue the `show chassis hardware` on one MX10003 with two RCBs, one MPC in slot 1, three PSMs, and four FANs:

```
door7302@mx10003# show chassis hardware
Hardware inventory:
Item           Version  Part number  Serial number    Description
Chassis                              AF213           JNP10003 [MX10003]
Midplane        REV 01   750-066883   CAGZ6541         Midplane 2
Routing Engine 0         BUILTIN      BUILTIN          RE-S-1600x8
Routing Engine 1         BUILTIN      BUILTIN          RE-S-1600x8
CB 0           REV 10   750-067071   CAGZ9925         Control Board
  Mezz         REV 06   711-066896   CAHA9151         Control Mezz Board
CB 1           REV 06   750-067071   CAGM1872         Control Board
  Mezz         REV 12   711-066896   CAHS7148         Control Mezz Board
FPC 1          REV 08   750-066879   CAHD6676         LC2103
  CPU                   BUILTIN      BUILTIN          SMPC PMB
  PIC 0                 BUILTIN      BUILTIN          6xQSFPP
  PIC 1        REV 05    750-069305   CAGX1008          MIC1-MACSEC
PEM 2          REV 01   740-066938   1HS26500001      JNP-PWR1100-DC
PEM 3          REV 01   740-066938   1HS26500028      JNP-PWR1100-DC
PEM 5          REV 01   740-066938   1HS26490020      JNP-PWR1100-DC
Fan Tray 0      REV 02   760-069329   CAHA9125         JNP FAN 3RU
Fan Tray 1      REV 02   760-069329   CAHA9126         JNP FAN 3RU
Fan Tray 2      REV 02   760-069329   CAHA9131         JNP FAN 3RU
Fan Tray 3      REV 02   760-069329   CAHA9119         JNP FAN 3RU
```

Let's look at the detail in the RCB component, which in this case is an RE-S1600x8. This hardware element, also called the *host subsystem*, provides these functions:

- System control functions such as environmental monitoring
- Routing Layer 2 and Layer 3 protocols
- Communication to all components such as line cards, power, and cooling
- Transparent clocking through BITS or GPS signal interfaces
- Alarm and logging functions

The current RCB consists of an X86-based 8-core CPU. It has 64 GB of DDR4 RAM (expandable to 128 GB) and two slots of 128GB SSD storage.

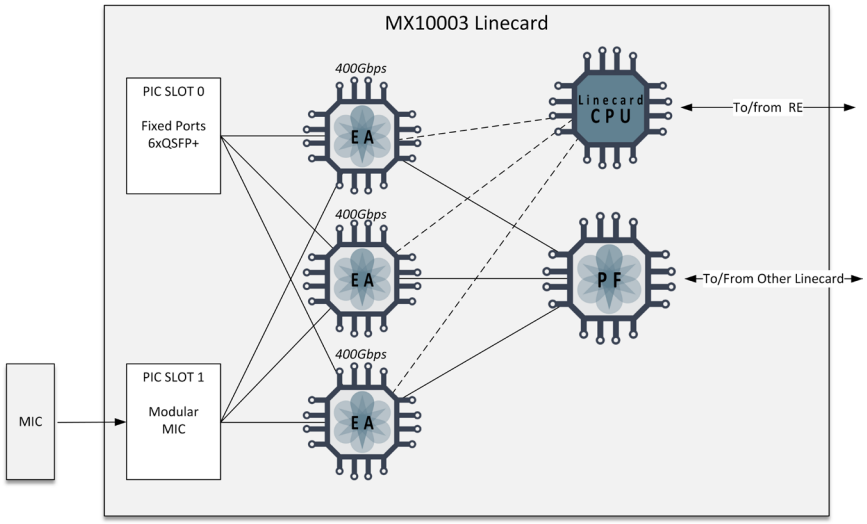
The LC2103 Line Card

The LC2103 line card is the first generation of MPC for the MX10003. This MPC is a mix of a built-in PIC and a modular PIC. Indeed, the PIC in slot 0 is made of six fixed QSFPP multi-rates ports and the PIC in slot 1 consists of a free slot that could host a modular PIC. The current MICs available for this line card are:

- 12xQSFPP without MACSEC
- 12xQSFPP with MACSEC

Those two MICs have multi-rate ports: 10GE, 40GE, and 100GE. You are able to configure the port profile (port speed) per MIC or per port. This line card is built around the fourth generation of TRIO ASIC: the EA ASIC. Each MPC LC2103 includes three PFE complexes – each complex is based on one EA ASIC, which is able to deliver up to 400Gbps bidirectional throughput. Unlike larger MX routers that have dedicated fabric planes (SCB or SFB), the MX10003 is space-optimized; the fabric ASIC (PF ASIC) is thus directly embedded on the line card itself. This fabric ASIC provides switching functionality between the different PFEs (the three complexes) on the MPC (*intra* line card forwarding) and between the different MPCs of the chassis (*inter* line cards forwarding). Figure 1.8 depicts the overview of the LC2103 line card.

Figure 1.8 The LC2103 Internal Architecture



Let's use a PFE command to check the internal composition of this MPC and confirm what is detailed in Figure 1.8:

```
[edit]
SMPC0(mx10003 vty)# show jspec client

ID      Name
  1      MPCS[0]
  3      XR2CHIP[0]
  4      XR2CHIP[2]
  5      XR2CHIP[4]
  6      XR2CHIP[1]
  7      XR2CHIP[3]
  8      XR2CHIP[5]
  9      EACHIP[0]
 10      EACHIP[1]
 11      EACHIP[2]
 12      PF_0
```

As expected, you can see there are three EA ASICs but there are also some other components like the PF chip seen on Figure 1.8 or several XR2 chips, which are actually a High Speed memory controller that each EA relies on to:

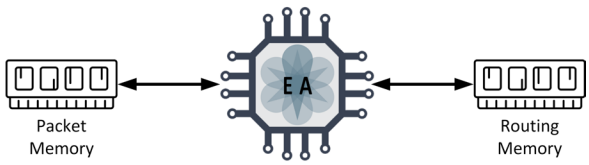
- Hybrid Memory Controller (HMC)
- Store packets in transit: WAN and fabric queuing
- Inline JFlow records

XR2 used for lookup and queuing:

- Routing information: routes, next hop, interface ID, programmed firewall filter, counters, etc.

These functions are stored in several banks of memory, each of them attached directly to an EA ASIC, as shown in Figure 1.9.

Figure 1.9 The EA ASICs and Its Memory Blocks

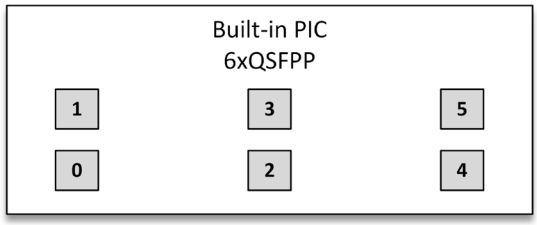


Let's refer back to Figure 1.8 to detail each PIC slot. The built-in PIC in slot 0 consists of six QSFPP multi-rate ports, where each port may be configured as follows depending on the port profile:

- 4x10GE, with a breakout cable, per physical QSFPP port
- 1x40GE per QSFPP port

On the PIC, the rate of each port is by default 4x10GE. To override this default behavior you can modify the port profile mode (at the PIC or port level). The commands to configure the port profile mode are the same as those presented for the MX204. However, there are some restrictions on the LC2103. Both PICs (the built-in one and the modular one) must use the same port profile, they cannot be mixed. In other words, the same port profile mode must be used on both PICs of a given MPC. Having said that, let's see the port mapping of the built-in PIC depicted by Figure 1.10.

Figure 1.10 Built-in PIC Port Mapping – PIC Slot 0



NOTE Remember that to connect 10GE interfaces you need to use a specific QSFPP module with a breakout cable to expand the physical port into four 10GE lanes.

Table 1.4 summarizes the port naming conventions of the built-in PIC depending on the configured port profile mode.

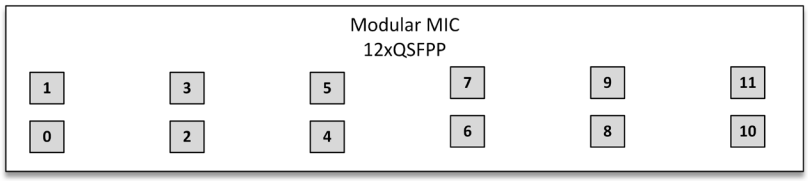
Table 1.4 Interface Naming of the 6xQSFPP PIC

PFE COMPLEX	Port Number	10GE mode	40GE mode
EA ASIC 0	0	xe-x/0/0:[0..3]	et-x/0/0
	1	xe-x/0/1:[0..3]	et-x/0/1
EA ASIC 1	2	xe-x/0/2:[0..3]	et-x/0/2
	3	xe-x/0/3:[0..3]	et-x/0/3
EA ASIC 2	4	xe-x/0/4:[0..3]	et-x/0/4
	5	xe-x/0/5:[0..3]	et-x/0/5

Now let's shift our attention to the modular PIC in slot 1. As mentioned earlier, there's currently one available MIC for this LC2103 line card shipped in two different versions: one with MACSEC and the other without it. The 12xQSFPP MIC might be inserted in slot 1. As a built-in PIC the default port profile mode is 10GE on all ports. You can select your desired port profile mode, and thus the speed of

the ports, by using the previous CLI commands (in this case for `fpc pic slot 1`). The port mapping of this modular interface card is illustrated in Figure 1.11.

Figure 1.11 MIC Port Mapping – PIC Slot 1



And once again, let's list the interface naming depending on the port profile mode in Table 1.5.

Table 1.5 Interface Naming of the 6xQSFPP PIC

PFE COMPLEX	Port Number	10GE mode	40GE or 100GE modes
EA ASIC 0	0	xe-x/1/0:[0..3]	et-x/1/0
	1	xe-x/1/1:[0..3]	et-x/1/1
	2	xe-x/1/2:[0..3]	et-x/1/2
	3	xe-x/1/3:[0..3]	et-x/1/3
EA ASIC 1	4	xe-x/1/4:[0..3]	et-x/1/4
	5	xe-x/1/5:[0..3]	et-x/1/5
	6	xe-x/1/6:[0..3]	et-x/1/6
	7	xe-x/1/7:[0..3]	et-x/1/7
EA ASIC 2	8	xe-x/1/8:[0..3]	et-x/1/8
	9	xe-x/1/9:[0..3]	et-x/1/9
	10	xe-x/1/10:[0..3]	et-x/1/10
	11	xe-x/1/11:[0..3]	et-x/1/11

This concludes our overview of the first new 5G router. Now let's move to two other routers in this new line of 5G routers – the MX10008 and the MX10016. These two big routers are both based on the new Juniper Universal Chassis.

MX10k and the Universal Chassis

The MX10008 and MX10016 are built around the new modular and powerful universal chassis. This chassis is expected to solve some environmental issues by offering a new design without a mid-plane and with a new cooling system. Moreover, the universal chassis, as indicated by its name, is intended not only for MX, but also for PTX, and for QFX as well.

Indeed, there are many components that can be shared between these three device platforms, including chassis, fans, power supplies, RE and control plane, and fabric planes.

Common line cards are in the Juniper product plan, but currently only the line cards are series-specific and cannot be shared between the platforms. Universal chassis is available in two models – the JNP10008, which is a 13RU tall chassis, and the JNP10016, which is a 21RU tall chassis. Both models provide redundancy for all components: RE, power, cooling system, and fabric plane. Figure 1.12 shows the two models in the MX Series family.

Figure 1.12 The 8- and 16-slot Universal Chassis

These new chassis are mid-plane-less. Actually fabric cards (inserted at the rear) are directly connected to line cards (inserted at the front) via orthogonal direct connectors. There are a total of six Switch Fabric Boards (SFBs) on both chassis models. The six fabric cards operate in 5:1 redundancy mode. Nevertheless, the model of fabric cards depends on the size of the chassis.

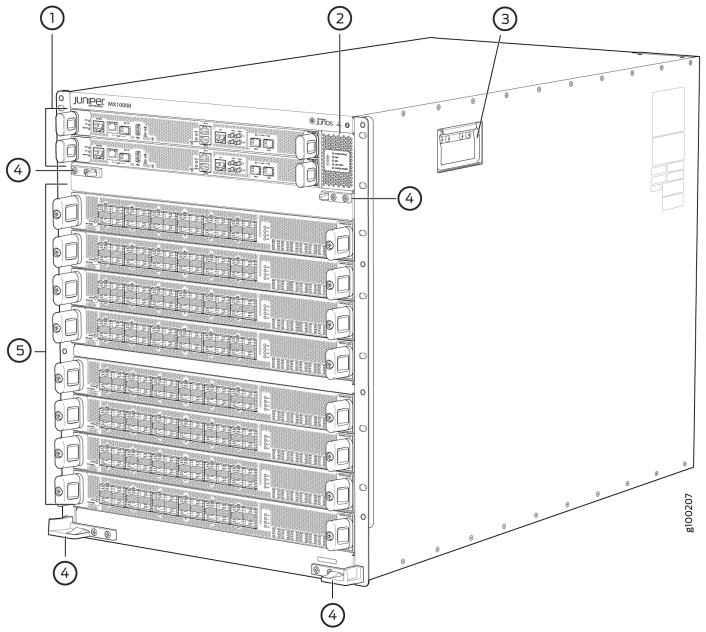
The MX10008 and MX100016 Chassis

The MX10008 and MX10016 offer either eight or sixteen MPC slots, respectively, that are inserted from the front. You can also insert the two RCBs from the front of the chassis. Currently the available universal RE is made of a ten core-based Intel CPUs, a 64GB memory (expandable to 128GB), and two internal SSD disks of 200GB each. Figure 1.13 shows the front view of an MX10008 and is annotated with the following information:

1. Routing and control board (RCB)
2. Status LED panel
3. Handles
4. Installation holes
5. Line card slots 0-7 (numbered top to bottom)

NOTE This is exactly the same for MX10016, except for the number of MPC slots.

Figure 1.13 MX10008 Front View



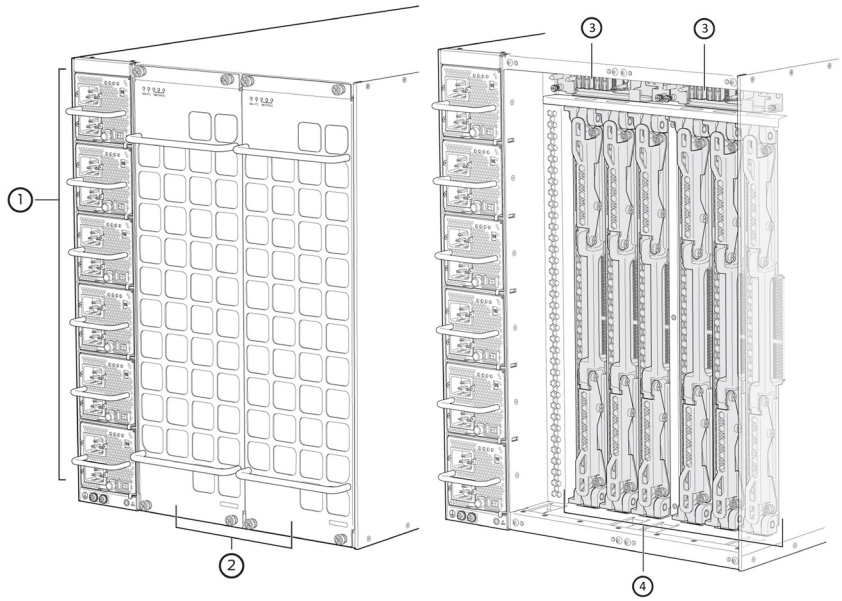
Next is the rear view of the chassis, in which you can observe the following components depicted by Figure 1.14:

1. AC or DC power supplies
2. Fan trays with redundant fans

(When you remove the fan trays you have access to the right view of Figure 1.14):

3. Fan tray controllers
4. The Switch Fabric Board (SFBs)

Figure 1.14 MX10008 Rear View



As the two fan trays are fully redundant, you can remove either fan tray to access the fan tray controllers or SFBs, which are located behind the fans. Remove fan 0 if you want to access to fan tray controller 0 and SFB 0, 1, and 2 or remove fan tray 1 if you want to access to fan tray controller 1 and SFB 3, 4, and 5.

Let's issue a `show chassis hardware` command on both the MX10008 and MX10016 chassis to summarize what's been presented so far.

```
door7302@mx10008# show chassis hardware
Hardware inventory:
Item           Version  Part number  Serial number    Description
Chassis                              DE405           JNP10008 [MX10008]
Midplane       REV 23   750-071974   ACPD4308         Midplane 8
Routing Engine 0        BUILTIN      BUILTIN          RE X10
Routing Engine 1        BUILTIN      BUILTIN          RE X10
CB 0           REV 02   711-065897   CAGY2617         Control Board
CB 1           REV 10   750-079562   CAKK7987         Control Board
FPC 1          REV 09   750-073174   CAJE3408         JNP10K-LC2102
  CPU          REV 02   750-073391   CAHM7947         LC 2101 PMB
  PIC 0                 BUILTIN      BUILTIN          4xQSFP28 MACSEC
  PIC 1                 BUILTIN      BUILTIN          4xQSFP28 MACSEC
  PIC 2                 BUILTIN      BUILTIN          4xQSFP28 MACSEC
  PIC 3                 BUILTIN      BUILTIN          4xQSFP28 MACSEC
  PIC 4                 BUILTIN      BUILTIN          4xQSFP28 MACSEC
  PIC 5                 BUILTIN      BUILTIN          4xQSFP28 MACSEC
FPD Board      REV 07   711-054687   ACPD0440         Front Panel Display
PEM 0          REV 01   740-073147   1EDM6170842      Power Supply DC
PEM 1          REV 01   740-073147   1EDM6170903      Power Supply DC
PEM 2          REV 01   740-073147   1EDM6170819      Power Supply DC
PEM 3          REV 01   740-073147   1EDM6130261      Power Supply DC
PEM 4          REV 01   740-073147   1EDM6170969      Power Supply DC
PEM 5          REV 01   740-073147   1EDM6170686      Power Supply DC
FTC 0          REV 14   750-072657   ACNY9996         Fan Controller 8
FTC 1          REV 13   750-072657   ACNS3200         Fan Controller 8
Fan Tray 0     REV 09   760-072656   ACNV1774         Fan Tray 8
Fan Tray 1     REV 09   760-072656   ACNV1685         Fan Tray 8
SFB 0          REV 25   750-072655   ACPD4573         Switch Fabric (SIB) 8
SFB 1          REV 28   750-072655   ACPK0928         Switch Fabric (SIB) 8
SFB 2          REV 28   750-072655   ACPM5424         Switch Fabric (SIB) 8
SFB 3          REV 28   750-072655   ACPL1841         Switch Fabric (SIB) 8
SFB 4          REV 28   750-072655   ACPL1881         Switch Fabric (SIB) 8
SFB 5          REV 28   750-072655   ACPL1941         Switch Fabric (SIB) 8

door7302@mx10016# show chassis hardware
Hardware inventory:
Item           Version  Part number  Serial number    Description
Chassis                              DL590           JNP10016 [MX10016]
Midplane       REV 24   750-077138   ACPR5157         Midplane 16
Routing Engine 0        BUILTIN      BUILTIN          RE X10 128
Routing Engine 1        BUILTIN      BUILTIN          RE X10 128
CB 0           REV 05   711-065897   CAJD3802         Control Board
CB 1           REV 03   750-079562   CAJS5144         Control Board
FPC 0          REV 04   750-084779   CAKR7034         JNP10K-LC2101
  CPU          REV 05   750-073391   CAKJ2874         LC 2101 PMB
  PIC 0                 BUILTIN      BUILTIN          4xQSFP28 SYNCE
  PIC 1                 BUILTIN      BUILTIN          4xQSFP28 SYNCE
  PIC 2                 BUILTIN      BUILTIN          4xQSFP28 SYNCE
  PIC 3                 BUILTIN      BUILTIN          4xQSFP28 SYNCE
  PIC 4                 BUILTIN      BUILTIN          4xQSFP28 SYNCE
  PIC 5                 BUILTIN      BUILTIN          4xQSFP28 SYNCE
FPC 11         REV 05   750-084779   CAKT4171         JNP10K-LC2101
  CPU          REV 05   750-073391   CAKV2283         LC 2101 PMB
  PIC 0                 BUILTIN      BUILTIN          4xQSFP28 SYNCE
  PIC 1                 BUILTIN      BUILTIN          4xQSFP28 SYNCE
  PIC 2                 BUILTIN      BUILTIN          4xQSFP28 SYNCE
  PIC 3                 BUILTIN      BUILTIN          4xQSFP28 SYNCE
```

```
  PIC 4                 BUILTIN     BUILTIN         4xQSFP28 SYNCE
  PIC 5                 BUILTIN     BUILTIN         4xQSFP28 SYNCE
FPD Board       REV 07   711-054687   ACPS8855        Front Panel Display
PEM 0           REV 01   740-073147   1EDM6171155     Power Supply DC
PEM 1           REV 01   740-073147   1EDM6281575     Power Supply DC
PEM 2           REV 01   740-073147   1EDM6171044     Power Supply DC
PEM 3           REV 01   740-073147   1EDM6281244     Power Supply DC
PEM 4           REV 01   740-073147   1EDM6282093     Power Supply DC
PEM 5           REV 01   740-073147   1EDM6281413     Power Supply DC
PEM 6           REV 01   740-073147   1EDM6171071     Power Supply DC
PEM 7           REV 01   740-073147   1EDM6170709     Power Supply DC
PEM 8           REV 01   740-073147   1EDM6171169     Power Supply DC
PEM 9           REV 01   740-073147   1EDM6170754     Power Supply DC
FTC 0           REV 10   750-050309   ACPE8185        Fan Controller 16
FTC 1           REV 10   750-050309   ACPM2918        Fan Controller 16
Fan Tray 0       REV 10   760-057901   ACPL0546        Fan Tray 16
Fan Tray 1       REV 10   760-077141   ACPV7288        Fan Tray 16
SFB 0           REV 15   750-077140   ACPV3981        Switch Fabric (SIB) 16
SFB 1           REV 15   750-058270   ACPM2808        Switch Fabric (SIB) 16
SFB 2           REV 15   750-077140   ACPV3964        Switch Fabric (SIB) 16
SFB 3           REV 15   750-058270   ACPJ9834        Switch Fabric (SIB) 16
SFB 4           REV 15   750-058270   ACPV3917        Switch Fabric (SIB) 16
SFB 5           REV 15   750-058270   ACPM2804        Switch Fabric (SIB) 16
```

Before reviewing the MX10k line card, Table 1.6 depicts which components are hot-removable/insertable, and which ones are hot-pluggable.

Table 1.6 FRU Types of MX10k

FRU Type	Hot-insertable	Hot-removable	Hot-pluggable
Master RCB with NSR		X	X
Master RCB without NSR			X
Backup RCB with NSR	X	X	
Backup RCB without NSR	X	X	
MPC	X	X	
SFB (Fabric)	X	X	
PSM	X	X	
Fan module	X	X	
Air Filter unit	X	X	
Optic module	X	X	

The LC2101 Line Card

The current line card available for MX10008 and MX10016 is the LC2101. This line card is designed around the fourth generation of TRIO ASIC. The card consists of six pseudo PICs. They're pseudo because the card is monolithic and does not have any modular MIC slot. Each PIC is attached to four ports, thus the LC2101 line card offers 24 multi-rates physical ports. There are two models of the LC2101 line card – one with MACSEC and one without MACSEC. Both models provide a throughput of either 1.44Tbps or 2.4Tbps.

NOTE The difference in the throughput bandwidth is not limited by a hardware component. Actually, this is purely limited by software and therefore the 1.44Tbps line card might be upgradable to 2.4Tbps via a simple CLI command. Just notice that the 1.44Tbps mode requests less power. This is one use case for using the lower bandwidth mode.

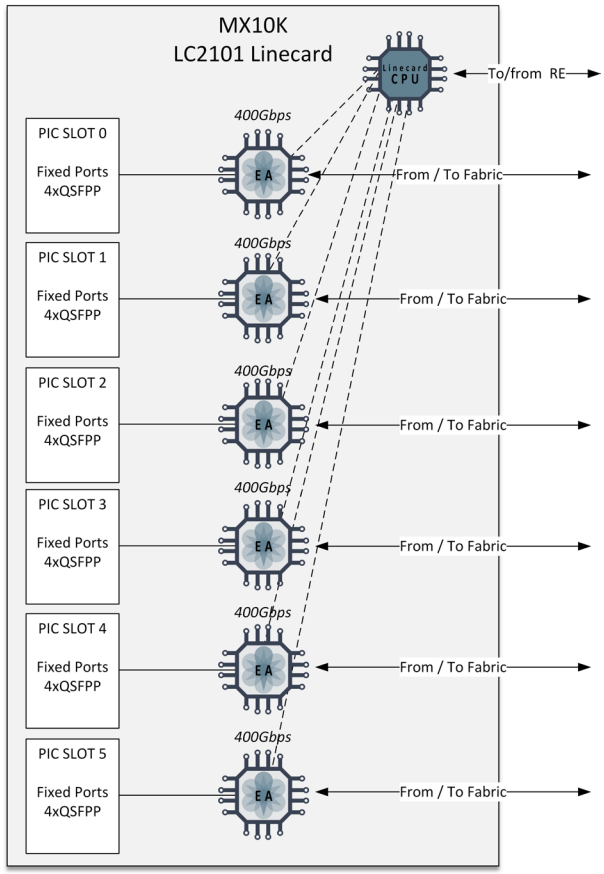
Regardless of the bandwidth mode (1.44 or 2.4Tbps), both models of line cards are made of six EA ASICs. The internal view of the LC2101 is depicted by Figure 1.15. For the 1.44Tbps line card each EA ASIC is rate-limited to 240Gbps instead of 400Gbps when the 2.4Tbps mode is enabled. As observed, each EA ASIC is connected to four multi-rate ports on the WAN side and to the six SFBs on the fabric side. Remember, as well, each EA ASIC is attached to two memory blocks (see Figure 1.9) to handle packets and routing information.

Depending on the optical module you use, you can have any of these available interface rates:

- 4x10GE ports with a breakout cable
- 1x40GE port with a 40GE QSFP+ module
- 1x100GE port with a 100GE QSP28 module

As presented previously, when discussing the LC2103 line card of the MX10003 the rate of each port is, by default, 4x10GE. You can override this default configuration by using dedicated CLI commands. The port profile mode is also configurable at the PIC or port level without any restriction.

Figure 1.15 The LC2101 Internal Architecture



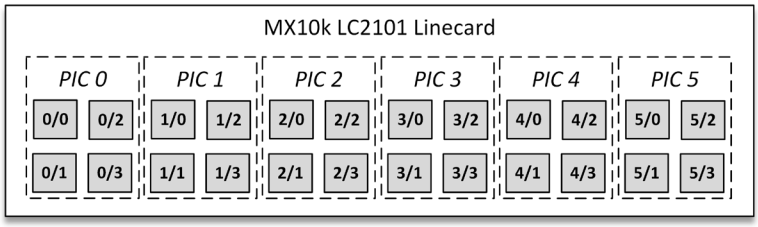
Let's issue a `show jspec client` PFE command to show the ASICs hosted by the MPC LC2101:

```
[edit]
IMPC0(mx10008 vty)# show jspec client

ID      Name
  1      MPCS[0]
  2      XR2CHIP[0]
  3      XR2CHIP[2]
  4      XR2CHIP[4]
  5      XR2CHIP[6]
  6      XR2CHIP[8]
  7      XR2CHIP[10]
  8      XR2CHIP[1]
  9      XR2CHIP[3]
```

```
10      XR2CHIP[5]
11      XR2CHIP[7]
12      XR2CHIP[9]
13      XR2CHIP[11]
14      EACHIP[0]
15      EACHIP[1]
16      EACHIP[2]
17      EACHIP[3]
18      EACHIP[4]
19      EACHIP[5]
```

Finally let's have a look at the port mapping of the LC2101 line card shown in Figure 1.16. The PIC annotation in Figure 1.16 is only there to facilitate the understanding of how physical ports are assigned internally to the EA ASIC. As we mentioned, the LC2101 line card is monolithic without any physical MIC slots.

Figure 1.16 The LC2101 Port Mapping



And, Table 1.7 shows you the port naming convention depending on the port profile configuration.

Table 1.7 Interface Naming of the LC2101 Line Card

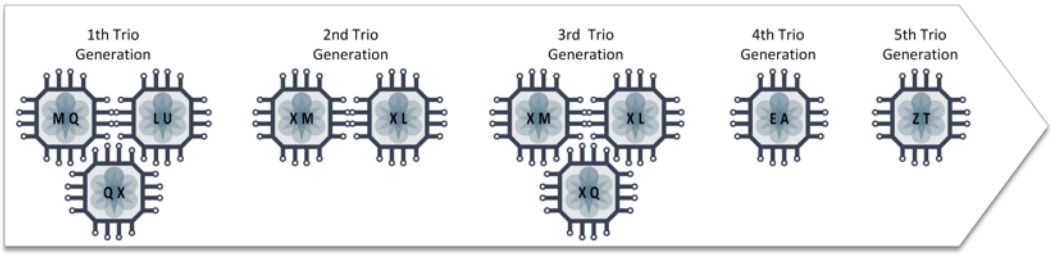
PFE COMPLEX	Port Number	10GE mode	40GE or 100GE modes
EA ASIC 0	0/0	xe-x/0/0:[0..3]	et-x/0/0
	0/1	xe-x/0/1:[0..3]	et-x/0/1
	0/2	xe-x/0/2:[0..3]	et-x/0/2
	0/3	xe-x/0/3:[0..3]	et-x/0/3
EA ASIC 1	1/0	xe-x/1/0:[0..3]	et-x/1/0
	1/1	xe-x/1/1:[0..3]	et-x/1/1
	1/2	xe-x/1/2:[0..3]	et-x/1/2
	1/3	xe-x/1/3:[0..3]	et-x/1/3
EA ASIC 2	2/0	xe-x/2/0:[0..3]	et-x/2/0
	2/1	xe-x/2/1:[0..3]	et-x/2/1
	2/2	xe-x/2/2:[0..3]	et-x/2/2
	2/3	xe-x/2/3:[0..3]	et-x/2/3

EA ASIC 3	3/0	xe-x/3/0:[0..3]	et-x/3/0
	3/1	xe-x/3/1:[0..3]	et-x/3/1
	3/2	xe-x/3/2:[0..3]	et-x/3/2
	3/3	xe-x/3/3:[0..3]	et-x/3/3
EA ASIC 4	4/0	xe-x/4/0:[0..3]	et-x/4/0
	4/1	xe-x/4/1:[0..3]	et-x/4/1
	4/2	xe-x/4/2:[0..3]	et-x/4/2
	4/3	xe-x/4/3:[0..3]	et-x/4/3
EA ASIC 5	5/0	xe-x/5/0:[0..3]	et-x/5/0
	5/1	xe-x/5/1:[0..3]	et-x/5/1
	5/2	xe-x/5/2:[0..3]	et-x/5/2
	5/3	xe-x/5/3:[0..3]	et-x/5/3

The Fifth Generation of ASICs

EA ASIC was the fourth generation of ASIC that was embedded on MPC7e, MPC8e, and MPC9e. The fifth generation of TRIO, recently released by Juniper, with the code name *ZT*, provides a full duplex bandwidth of 500Gbps and it is in the core of the two new MPCs: *MPC10e for MX* and *MPC11e for MX2K*. Figure 1.17 illustrates the TRIO ASIC evolution.

Figure 1.17 The TRIO Evolution



As you can see, since the fourth generation of TRIO all functions, such as queuing, lookup, and rich CoS, are now integrated in a single chipset with no compromise between features and throughput.

ZT brings the support of 400Gbps Ethernet interfaces as well as a new set of advanced inline features such as: Flexible Ethernet or *FlexE*, and a high capacity of IPsec (via a Crypto module), and MACSEC tunnels. Let's go *inside* these new MPCs and take a look.

The MPC10e Line Card

The MPC10e is a new design for the MX240, MX480, or MX960. This line card is delivered in two models: a 1Tbps and a 1.5Tbps line card. Actually the throughput available by this 1.5Tbps line card depends on two factors:

- The version of the MX chassis
- The new fabric plane for MX: SCBE3

Indeed, the first condition to benefit the 1.5Tbps per slot on classic MXs is the new version of the MX chassis. The Premium3 chassis allows the 1.5Tbps per line card while MPC10e will operate at 1Tbps on the Premium2 chassis version (also known as *regular chassis*). The second requirement for the MPC10e on your MX chassis is to upgrade the fabric cards. Just as with previous upgrades, this is achieved by switching the Switch Control Board to the latest version. For MPC10e, the SCBE3 is required to enable the 1.5Tbps per slot.

NOTE There is no need to change power and cooling systems. Indeed, the existing fan trays and power supplies of the MX will work with the MPC10e.

As already mentioned, the MPC10e consists of two models: a 10 multi-rate ports line card named *MPC10e 10C* and a 15 multi-rate ports line card named *MPC10e 15C*. The physical ports are logically grouped in groups of five ports. Each group of five ports is internally attached to a ZT ASIC. Within a given group, all ports can handle QSFP28 modules, and the last port of each group supports the QSFP56-DD module.

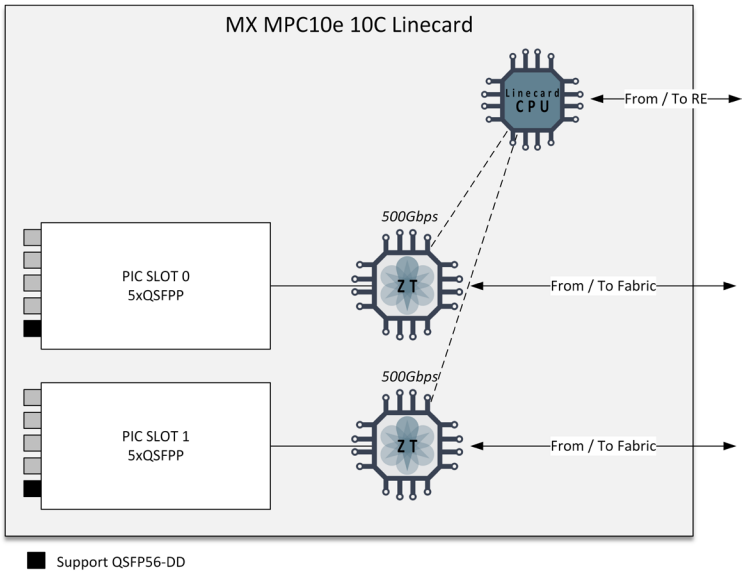
To summarize, each group of five ports might handle these types of link rates:

- 25x10GE, by using a QSFPP-4x10GE module and a breakout cable on each port of the group
- 5x40GE, by using a QSFPP-40GE module on each port of the group
- 5x100GE, by using a QSFP28-100GE module on each port
- 1x400GE, by using a QSFP56-DD module on the dedicated port of the group

NOTE ZT AZIC cannot be oversubscribed.

Figure 1.18 depicts the internal view of the MPC10e 10C, which is made of two ZT ASICs.

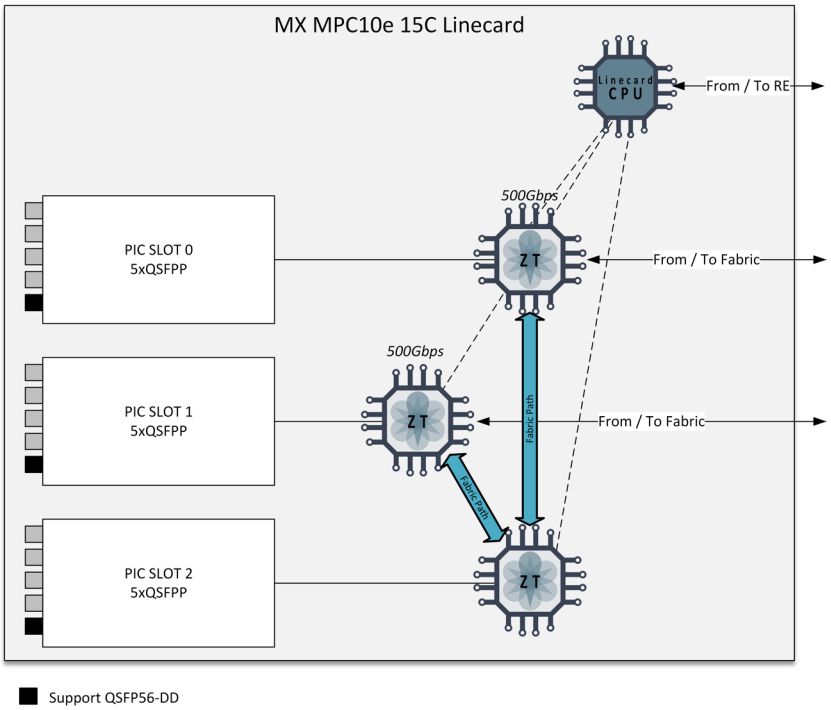
Figure 1.18 MPC10e 10C Internal Architecture



And the next figure, Figure 1.19, shows the internal view of an MPC10e 15C version made of three ZT ASICs. You will have noticed that not all PFEs are connected to the fabric. Indeed, for this model of card, only the ZT0 and ZT1 complexes are attached to the fabric via dedicated SerDes (Serializer/Deserializer), also known as *high-speed links*. The third ZT ASIC uses inter ASIC high-speed links to communicate through the fabric. In this case you can consider that ZT0 and ZT1 play the role of *fabric bridge* for ZT2.

NOTE This specific hardware configuration does not impact any ZT ASIC in terms of performance. It just allows using more PFEs (ZT ASIC) on a given line card due to the limited number of physical links available on the mid plane of the chassis.

Figure 1.19 MPC10e 15C Internal Architecture

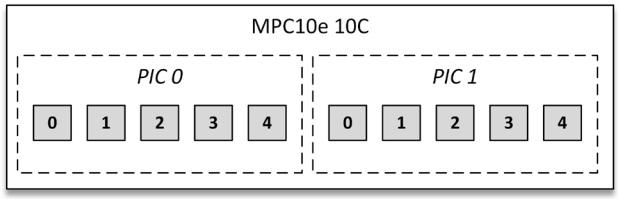


Let's use a PFE command to check the internal composition of this MPC10e 15C and confirm what we see in Figure 1.19:

```
[edit]
NPC11(mx960 vty)# show jspec client

ID      Name
  1      MPCS[0]
  2      ZTCHIP[0]
  3      ZTCHIP[1]
  4      ZTCHIP[2]
```

Now, let's have a look at the port mapping and the port naming convention that depends on the rate configured (port profile mode) of the MPC10e 10C.

Figure 1.20 The MPC10e 10C Port Mapping



As mentioned, only the port 0/4 and 1/4 support QSFP56-DD modules. Now the next table, Table 1.8, lists the port naming convention depending on the port profile configuration.

Table 1.8 Interface Naming of the MPC10e 10C Line Card

PFE COMPLEX	Port Number	10GE mode	40GE or 100GE modes	400GE support
ZT ASIC 0	0/0	xe-x/0/0:[0..3]	et-x/0/0	NA
	0/1	xe-x/0/1:[0..3]	et-x/0/1	NA
	0/2	xe-x/0/2:[0..3]	et-x/0/2	NA
	0/3	xe-x/0/3:[0..3]	et-x/0/3	NA
	0/4	xe-x/0/4:[0..3]	et-x/0/4	et-x/0/4
ZT ASIC 1	1/0	xe-x/1/0:[0..3]	et-x/1/0	NA
	1/1	xe-x/1/1:[0..3]	et-x/1/1	NA
	1/2	xe-x/1/2:[0..3]	et-x/1/2	NA
	1/3	xe-x/1/3:[0..3]	et-x/1/3	NA
	1/4	xe-x/1/4:[0..3]	et-x/1/4	et-x/1/4

Let's do the same for the second model of MPC10e, with 15 ports. Figure 1.21 shows you the port mapping, and once again, only the last port (ports 0/4, ¼, and 2/4) of each PIC supports QSFP56-DD module.

Figure 1.21 The MPC10e 15C Port Mapping

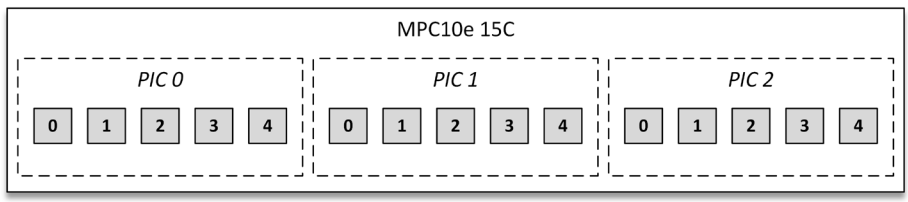


Table 1.9 lists the port naming convention depending on the port profile configuration.

Table 1.9 Interface Naming of the MPC10e 15C line card

PFE COMPLEX	Port Number	10GE mode	40GE or 100GE modes	400GE support
ZT ASIC 0	0/0	xe-x/0/0:[0..3]	et-x/0/0	NA
	0/1	xe-x/0/1:[0..3]	et-x/0/1	NA
	0/2	xe-x/0/2:[0..3]	et-x/0/2	NA
	0/3	xe-x/0/3:[0..3]	et-x/0/3	NA
	0/4	xe-x/0/4:[0..3]	et-x/0/4	et-x/0/4
ZT ASIC 1	1/0	xe-x/1/0:[0..3]	et-x/1/0	NA
	1/1	xe-x/1/1:[0..3]	et-x/1/1	NA
	1/2	xe-x/1/2:[0..3]	et-x/1/2	NA
	1/3	xe-x/1/3:[0..3]	et-x/1/3	NA
	1/4	xe-x/1/4:[0..3]	et-x/1/4	et-x/1/4
ZT ASIC 2	2/0	xe-x/2/0:[0..3]	et-x/2/0	NA
	2/1	xe-x/2/1:[0..3]	et-x/2/1	NA
	2/2	xe-x/2/2:[0..3]	et-x/2/2	NA
	2/3	xe-x/2/3:[0..3]	et-x/2/3	NA
	2/4	xe-x/2/4:[0..3]	et-x/2/4	et-x/2/4

The MPC11e Line Card

The MPC11e line card is designed for the MX2K chassis, which includes the MX2010 and the MX2020. The MPC11e is fully interoperable with other MX2K MPCs such as the MPC6e, MPC8e, or MPC9e.

Like its smaller series member, the MPC10e, the MPC11e line card is made around the ZT ASIC and thus allows 400GE interfaces and benefits of the all the advanced inline features. The MPC11e provides a total capacity of 4Tbps. To leverage this per-slot throughput you need to upgrade the SFBs to the newest release: at present, the SFB version 3. The SFB3 allows 4Tbps per slot and now also supports eight PFE destinations per MPC, while SFB2 supported four PFEs per MPC. Indeed, to provide the 4Tbps of throughput, the MPC11e consists of eight ZT ASICs for a total of 40 physical ports.

Like MPC10e, these 40 physical ports are split into eight groups of five ports. Each group of five ports is internally attached to a ZT ASIC. The first port of each group supports multi-rate, which means that ports 0/0, 1/0, 2/0, 3/0, 4/0, 5/0, 6/0, and 7/0 supports these modules and rates:

- 4x10GE support using QSFPP-4x10GE breakout optics
- 1x40GE support using QSFPP optics
- 1x100GE support using QSFP28 optics
- 1x400GE support using QSFP56 optics

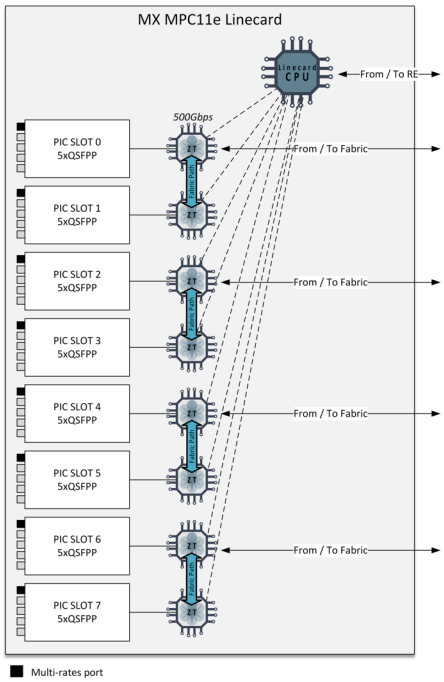
All the other ports support 100GE rate with QSFP28 optic modules. You can retrieve all this information by issuing the following:

```
door7302@mx2020# show chassis pic fpc-slot <fpc-slot> pic-slot <pic-slot>
FPC slot 11, PIC slot 0 information:
[…]

Port speed information:

  Port  PFE     Capable Port Speeds
  0     0      4x10GE, 40GE, 100GE
  1     0      100GE
  2     0      100GE
  3     0      100GE
  4     0      100GE
```

NOTE There is no need to change power and cooling systems. Indeed, the existing fan trays and power supplies of MX2K work with MPC11e.

The internal architecture of the MPC11e is depicted in Figure 1.22.

Figure 1.22 MPC11e Internal Architecture



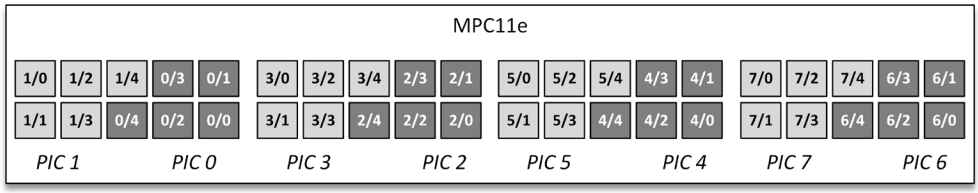
Due to the number of connectors available on the mid plane, not all ZT ASICs (like the MPC10e 15C) have direct high-speed links from/to the fabric. Half of the ZT uses another ZT like a fabric bridge, without impacting the performance and the capacity of ZT ASIC.

Let's check the following PFE command one more time to confirm the internal composition of the MPC11e. As expected, we found our eight ZT ASICs:

```
[edit]
NPC11(mx2020 vty)# show jspec client

ID      Name
  1      ZTCHIP[0]
  2      ZTCHIP[1]
  3      ZTCHIP[2]
  4      ZTCHIP[3]
  5      ZTCHIP[4]
  6      ZTCHIP[5]
  7      ZTCHIP[6]
  8      ZTCHIP[7]
```

The port numbering is a little bit complex on MPC11e. Figure 1.23 helps you to better understand this port numbering:

Figure 1.23 MPC11e Port Mapping



And Table 1.10 provides the port naming convention of the MPC11e depending on the port profile configuration.

Table 1.10 Interface Naming of the MPC11e Line Card

PFE COMPLEX	Port Number	10GE mode	40GE or 100GE modes	400GE support
ZT ASIC 0	0/0	xe-x/0/0:[0..3]	et-x/0/0	et-x/0/0
	0/1	NA	et-x/0/1	NA
	0/2	NA	et-x/0/2	NA
	0/3	NA	et-x/0/3	NA
	0/4	NA	et-x/0/4	NA

ZT ASIC 1	1/0	xe-x/1/0:[0..3]	et-x/1/0	et-x/1/0
	1/1	NA	et-x/1/1	NA
	1/2	NA	et-x/1/2	NA
	1/3	NA	et-x/1/3	NA
	1/4	NA	et-x/1/4	NA
ZT ASIC 2	2/0	xe-x/2/0:[0..3]	et-x/2/0	et-x/2/0
	2/1	NA	et-x/2/1	NA
	2/2	NA	et-x/2/2	NA
	2/3	NA	et-x/2/3	NA
	2/4	NA	et-x/2/4	NA
ZT ASIC 3	3/0	xe-x/3/0:[0..3]	et-x/3/0	et-x/3/0
	3/1	NA	et-x/3/1	NA
	3/2	NA	et-x/3/2	NA
	3/3	NA	et-x/3/3	NA
	3/4	NA	et-x/3/4	NA
ZT ASIC 4	4/0	xe-x/4/0:[0..3]	et-x/4/0	et-x/4/0
	4/1	NA	et-x/4/1	NA
	4/2	NA	et-x/4/2	NA
	4/3	NA	et-x/4/3	NA
	4/4	NA	et-x/4/4	NA
ZT ASIC 5	5/0	xe-x/5/0:[0..3]	et-x/5/0	et-x/5/0
	5/1	NA	et-x/5/1	NA
	5/2	NA	et-x/5/2	NA
	5/3	NA	et-x/5/3	NA
	5/4	NA	et-x/5/4	NA
ZT ASIC 6	6/0	xe-x/6/0:[0..3]	et-x/6/0	et-x/6/0
	6/1	NA	et-x/6/1	NA
	6/2	NA	et-x/6/2	NA
	6/3	NA	et-x/6/3	NA
	6/4	NA	et-x/6/4	NA
ZT ASIC 7	7/0	xe-x/7/0:[0..3]	et-x/7/0	et-x/7/0
	7/1	NA	et-x/7/1	NA
	7/2	NA	et-x/7/2	NA
	7/3	NA	et-x/7/3	NA
	7/4	NA	et-x/7/4	NA

Mixing port speeds when dealing with a high-density linecard may quickly become a nightmare when you have to identify a given port – especially when you need to remotely pilot a technician to plug a fiber to a specific port. But no worries, remember there are some cool CLI commands that help you to identify a specific port or a group of ports assigned to the same speed.

To blink a specific port you can use the following command:

```
door7302@mx2020# request chassis port-led <start|stop> fpc-slot <Slot> pic-slot <Slot> port <Port>
[duration <Time>]
```

To blink all the 10GE ports, use this command:

```
door7302@mx2020# request chassis port-led <start|stop> fpc-slot <Slot> pic-slot <Slot> port all-10g
[duration <Time>]
```

To blink all the 40GE ports, use this command:

```
door7302@mx2020# request chassis port-led <start|stop> fpc-slot <Slot> pic-slot <Slot> port all-40g
[duration <Time>]
```

To blink all the 100GE ports, use this command:

```
door7302@mx2020# request chassis port-led <start|stop> fpc-slot <Slot> pic-slot <Slot> port all-100g
[duration <Time>]
```

And finally, if needed, to blink all the ports use this command:

```
door7302@mx2020# request chassis port-led <start|stop> fpc-slot <Slot> pic-slot <Slot> port all-ports
[duration <Time>]
```

Now, it's time to dive into the EA and ZT ASICs. Chapters 2 and 3 will help you to better understand the internal hardware and software architectures and then better know how host and transit packets are handled by these new chip sets.

Hope you are ready for this adventure—let's go inside the MX5G!

Chapter 2

MX 5G: Two Powerful ASICs

This chapter provides a detailed view of two models of line card: one based on the EA ASIC, and a second one based on the ZT ASIC. As you will see, the functional blocks inside the two ASICs are quite similar, even if the ZT provides more advanced inline features, like the crypto engine, which allows inline MACsec with no performance penalty, than the EA. We will not detail these specific features (like FlexE or Inline IPsec) as the software supporting these advanced features was not yet released when this book was written. We will focus on the classic features such as routing traffic, handling control plane, and transit packets.

MORE? Media Access Control security (MACsec) provides point-to-point security on Ethernet links and is defined by IEEE standard 802.1AE. You can use MACsec in combination with other security protocols, such as IP Security (IPsec), and Secure Sockets Layer (SSL), to provide end-to-end network security. Juniper MPC10E/11E MPCs supports inline MACsec with no throughput/latency penalties. For more see https://www.juniper.net/documentation/en_US/junos/topics/topic-map/understanding_media_access_control_security_qfx_ex.html.

EA and ZT also have a similar architecture, therefore we'll first describe the EA in detail and then describe only the specificities of the ZT in a second part.

In this chapter, for all PFE commands related to MQSS or XQSS blocks, the number after the terms mqss or xqss refers to the Local PFE ID:

```
(mx vty)# show mqss [PFE-ID] xxxx

(mx vty)# show xqss [PFE-ID] xxxx
```

The PFE ID is a number between 0 and 7 depending on the number of PFEs / ASICs of the MPC.

Internal View of EA-based Line Card

Let's start with the EA-based line card. We will not describe a specific model of line card. The MPC7e, MPC8e, MPC9e, LC2101, LC2103, and even the MX204, which are all built around the EA ASIC, work the same way.

Software Architecture

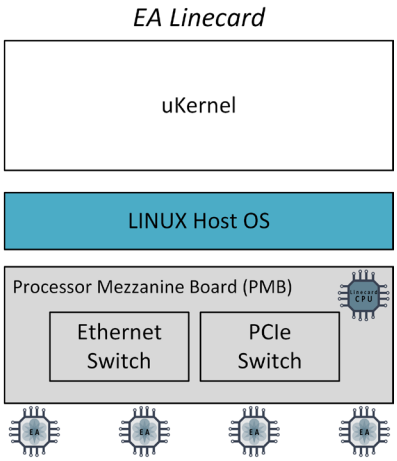
The EA line card software architecture is based on a Linux host OS. The Juniper software, embedded into the MPC, running on top of Linux, is called the *microkernel* (aka *uKernel*). On the EA line card all the PFE software components run on this single 32-bit process. The main tasks managed by the uKernel are:

- ASIC initialization and management
- Host packets handling – DDoS protection at CPU line card level
- NH management / FIB programing
- Distributed protocols support
- Alarm / Error management
- Syslog management

… and many other tasks

Figure 2.1 illustrates the software architecture of a typical EA line card.

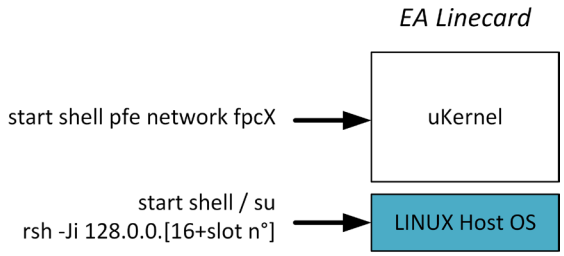
Figure 2.1 Software Architecture of EA Line Card



You can see in Figure 2.1 that the line card hosts a PCIe and an Ethernet switch.

You can connect to both software layers: the uKernel and Linux OS directly from the RE. Figure 2.2 explains how you access the uKernel PFE shell or the Linux shell.

Figure 2.2 Interact with MPC



Ethernet is used by uKernel to communicate with the RE (Master/Backup). The RE usually sends control plane packets to uKernel. After processing them, if needed, the uKernel also sends the control plane packets to the EA via Ethernet, which finally forwards the packets to the WAN interfaces. On the reverse side, when the control plane comes from the WAN ports, the EA ASIC delivers them to the uKernel by using DMA (direct memory access) through a PCIe interface. Those control plane packets are then forwarded to the RE through the Ethernet port. Notice that since Junos 17.4 there is a TurboTX mode, enabled by default, which allows the RE to send some specific host outbound traffic to EA faster (WAN). This mode bypasses the uKernel (outbound direction) and the RE sends its packets directly to the EA (via Ethernet). For your information, this default mode can be turned off with the following command (more detail on this mode in Chapter 3):

```
door7302@mx2020# set chassis turbotx-disable
```

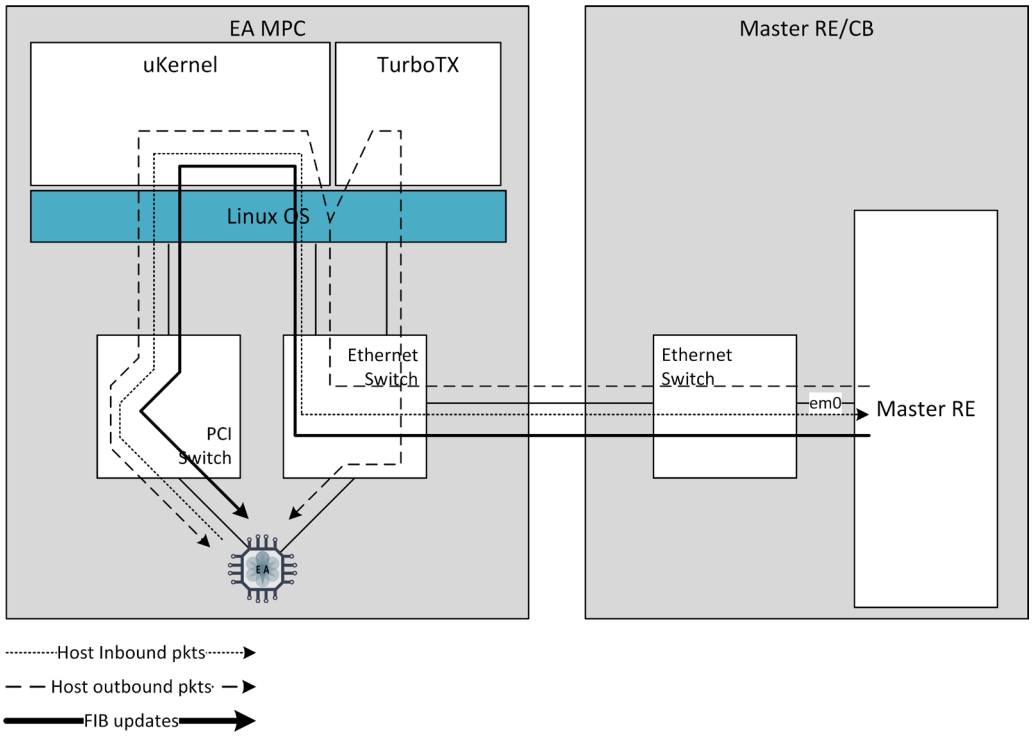
For FIB updates, the RE sends FIB modifications through an IPC (interprocess communication) message. For that, the RE sends its IPC messages via a raw socket over Ethernet to the uKernel. Then uKernel handles IPC and programs the FIB into the EA ASIC via PCIe interface. This useful tool, available at the shell level of the RE, allows you to simulate a uKernel's MPC connection to the Junos kernel (on RE) to receive, like an MPC, the FIB updates. The command is named *rtsockmon* as detailed in the below output:

```
droydavi@mx2020> start shell
% su
root@mx2020:/var/home/remote-su # rtsockmon -tn
       sender   creator  flag    type      op
[16:36:47:715.689] rpd      unknown    PI   route     change  inet 2.0.0.0 tid=0 plen=24 type=user
flags=0x0 nh=dscd nhflags=0x0 nhidx=0 rt_nhiflist = 0 altfwdnhidx=0 filtidx=0 lr_id = 0 featureid=0
```

```
rt_mcast_nhiflist=0 dflags 0x0

[16:36:47:717.023] rpd      self      P    route     change  inet 2.0.0.0 tid=0 plen=24 type=user
flags=0x0 nh=dscd nhflags=0x1 nhidx=34 rt_nhiflist = 0 altfwdnhidx=0 filtidx=0 lr_id = 0 featureid=0
rt_mcast_nhiflist=0 dflags 0x0
```

The Figure 2.3 summarizes the different host flows.

Figure 2.3 Host Traffic Flows



Hardware Architecture

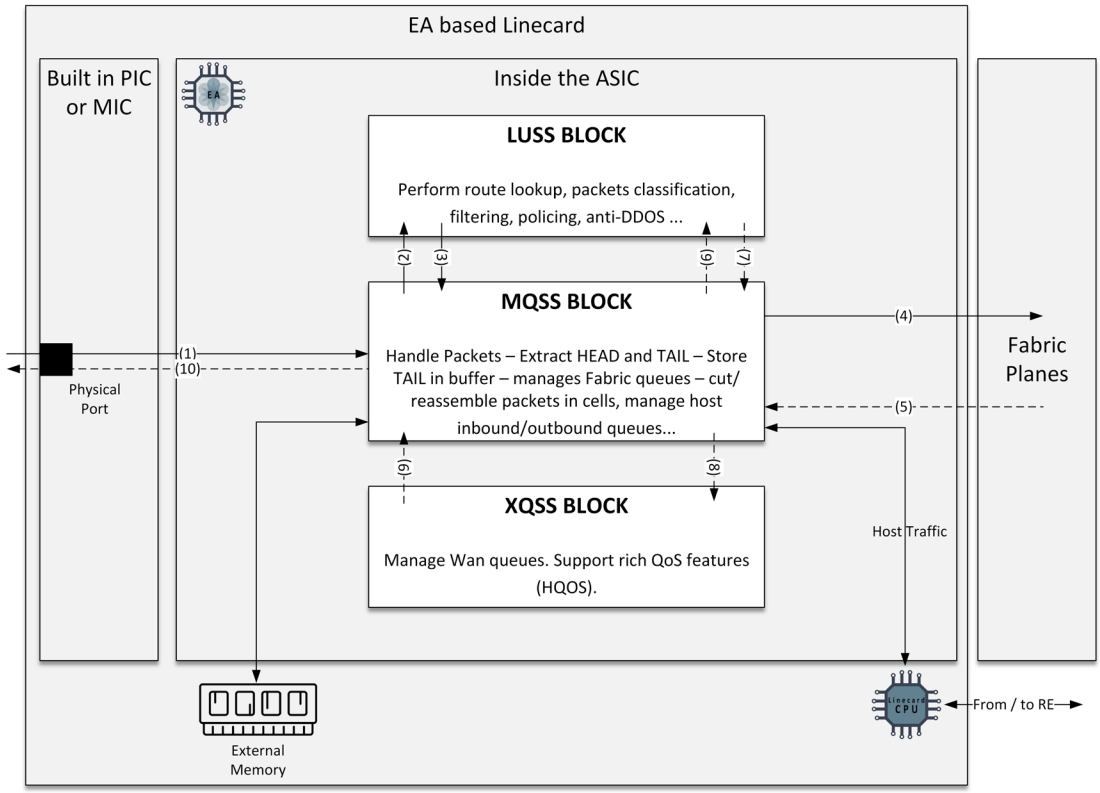
The novelty of the EA ASIC is that the three main components of the classic TRIO architecture – the queuing, lookup, and rich QoS blocks – have been merged into one single ASIC. Nevertheless, if we zoom inside the EA ASIC we'll find these three main functional blocks named like this:

- MQSS block: This is the "Center" and in the queuing block, from previous TRIO generations, it was the MQ or XM ASICs.

- LUSS block: This is the lookup block – from previous TRIO generations it was the LU or XL ASICs.
- XQSS block: And this is the rich QoS block – from previous TRIO generations it was the QX or XQ ASICs.

Figure 2.4 gives you the first view of the internal architecture of the EA ASIC.

Figure 2.4 Functional Blocks Inside the EA



Note again that the features managed by each block in Figure 2.4 are not exhaustive; only the classic features are listed. Given that, what can you notice at first glance?

The MQSS is the interface between the WAN side and the fabric side. Later we'll dive inside the MQSS to have a more precise look at this central block. For now, the important thing to notice is that the host path, meaning *all packets* (Control plane, OAM…) that wish to reach the line card CPU, or the RE, will use the MQSS block.

The EA ASIC is also attached to external memory to store, among other things, the packet during its processing: we called this part of the external memory the *delay bandwidth buffer* or DBB. On each EA ASIC there is a buffer of 75 msecs per physical 10GE/40GE/100GE port.

Before carrying on our walk through the EA, let's do a quick review of data manipulated by the TRIO ASIC family:

- Data Unit or packet HEAD. This is actually a chunk (the first segment) of the real packet. This chunk contains all the packet's headers plus some parts of the payload. On the EA ASIC, the HEAD could have a variable length but has a maximum size of 224 bytes. Actually, if packet size is less than 224 bytes, this entire packet is taken into account (HEAD = entire packet). Otherwise, if packet size is more than 224 bytes, only the first 192 bytes compose the HEAD and the rest of packet's bytes are named TAIL.
- Packet TAIL: The additional bytes corresponding to the data that are not in the HEAD. Those remaining bytes are stored in external memory (DBB partition).
- The Cells: When the entire packet needs to move from one PFE to another PFE through the fabric this packet is split into small cells, which have a fixed size of 64 bytes.

Thus, only the packet HEAD is manipulated by the different blocks of the EA and not the whole packet (except if the packet size is less than 224 bytes). HEAD plays the role of packet descriptor inside the ASIC.

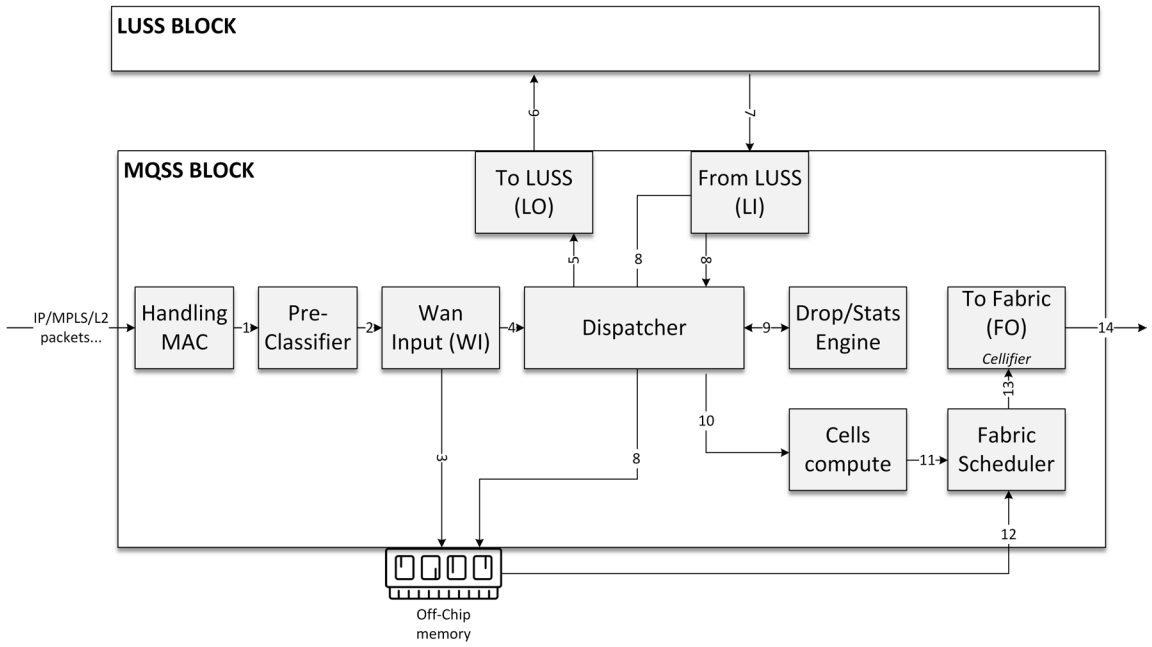
Let's move back to Figure 2.4. The MQSS is also attached to the two other functional blocks: the LUSS for all that concerns packet manipulation (the LUSS is like the brain of the EA); and the XQSS, for all things regarding to WAN QoS: (port-based queuing or hierarchical queuing features are managed by the XQSS). As seen in Figure 2.4, XQSS is only used on the egress direction, meaning from fabric to WAN, unless you request (by configuration) ingress queuing, which is not the default configuration on Junos.

Finally, it's important to recall, as for all the other TRIO generations, that the forwarding of a packet between ports attached to the same EA ASIC doesn't use the fabric path (except in some tunneling use cases where the Anchor PFE is not the same PFE as the ingress PFE – see the Appendix for details).

Ingress Packet Flow

We are now going to zoom inside the MQSS block in order to see some implementation details. MQSS is not a monolithic block. It's also made of several smaller functional sub-blocks. Each of them is in charge of one or several specific/dedicated tasks. Let's dive into MQSS and describe step-by-step how an ingress transit packet, meaning one that's coming from the WAN, is manipulated by the MQSS. Figure 2.5 zooms inside the MQSS.

Figure 2.5 Ingress Traffic Inside the MQSS



When the packet comes into an EA line card, it is first handled by the MAC block of the MQSS. Here, the CRC32 of the Ethernet frame is checked and if it is right, the MAC block removes it from the Ethernet frame. This one is then passed to the Pre-classifier Engine (Step 1). At PFE level you can issue these following commands to retrieve some stats from the MAC block. First let's list all the MAC blocks available:

```
(mx vty)# show mtip-cmac summary
 ID mtip_cmac name          FPC PIC Port Chan ASIC Inst ifd                 (ptr)
--- --------------------- --- --- ---- ---- ---- ---- ------------------ --------
  4 mtip_cmac.3.0.2         3   0    2   0    0   0    et-3/0/2          eb751850
  5 mtip_cmac.3.0.5         3   0    5   0    0   0    et-3/0/5          eb7528d0
  7 mtip_cmac.3.1.2         3   1    2   0    0   0    et-3/1/2          eb751e50
  8 mtip_cmac.3.1.5         3   1    5   0    0   0    et-3/1/5          eb751f10
```

Then, based on the ID information that matches your physical port, issue the second command:

```
(mx vty)# show mtip-cmac 4 statistics
Statistics [port:0]
----------
  aFramesTransmittedOK:                              1834746125434
  aFramesReceivedOK:                                 2466155012772
  aFrameCheckSequenceErrors:                            5199817945
```

```
  aAlignmentErrors:                                          0
  aPAUSEMACCtrlFramesTransmitted:                            0
  aPAUSEMACCtrlFramesReceived:                               0
  aFrameTooLongErrors:                                       0
  aInRangeLengthErrors:                                      0
  VLANTransmittedOK:                                 859788549242
  VLANReceivedOK:                                   2466155012772
  ifOutOctets:                                      2615973002556
  ifInOctets:                                      72872523231284
  ifInUcastPkts:                                    1400030696112
  ifInMulticastPkts:                                1079009218538
  ifInBroadcastPkts:                                  21474836490
  ifInErrors:                                          5199817945
  ifOutErrors:                                               0
  ifOutUcastPkts:                                    803946895624
  ifOutMulticastPkts:                                 73021516382
  ifOutBroadcastPkts:                                 68719483156
  etherStatsDropEvents:                                      0
  etherStatsOctets:                                 73335806763572
  etherStatsPkts:                                   2467059863421
  etherStatsJabbers:                                         0
  etherStatsFragments:                                       0
  etherStatsUndersizePkts:                                   0
  etherStatsOversizePkts:                                    0
  etherStatsPkts64Octets:                             21474842780
  etherStatsPkts65to127Octets:                        12888338147
  etherStatsPkts128to255Octets:                       12884909162
  etherStatsPkts256to511Octets:                       12884907842
  etherStatsPkts512to1023Octets:                    1375161699747
  etherStatsPkts1024to1518Octets:                   1096189627786
  etherStatsPkts1519toMaxOctets:                      12884949285
  etherStatsPkts64OctetsTx:                           73014654851
  etherStatsPkts65to127OctetsTx:                      73021778201
  etherStatsPkts128to255OctetsTx:                     12884903817
  etherStatsPkts256to511OctetsTx:                     12884908800
  etherStatsPkts512to1023OctetsTx:                  799649476582
  etherStatsPkts1024to1518OctetsTx:                   12885278779
  etherStatsPkts1519toMaxOctetsTx:                    12886501684

(mx vty)#)# show mtip-cmac 4 statistics-err
CMAC Error Statistics [port:0]
-------------------------------------------------------
  aFrameCheckSequenceErrors:                           5199817945
  aAlignmentErrors:                                          0
  aFrameTooLongErrors:                                       0
  aInRangeLengthErrors:                                      0
  ifInErrors:                                          5199817945
  ifOutErrors:                                               0
  etherStatsDropEvents:                                      0
  etherStatsJabbers:                                         0
  etherStatsUndersizePkts:                                   0
  etherStatsOversizePkts:                                    0
```

Next step: the pre-classifier engine does a first macro classification based on the packet header. It classifies the packets into two internal ingress streams: one CTRL (aka *Medium*) stream and one BE (aka *Low*) stream. The CTRL stream conveys all packets identified as control plane packets or OAM frames (host or transit) such as ARP, BGP, ICMP, etc. and the BE stream carries all other types of traffic (transit for the most part). This pre-classification avoids control plane packet drops when the PFE (EA) is overloaded: which means the CTRL stream should never be starved.

In order to collect the pre-classifier statistics you need to issue these sets of commands. First of all, list all the pre-classifier instances currently running on a given MPC:

```
(mx vty)# show precl-eng summary
 ID  precl_eng name       FPC PIC ASIC-ID ASIC-INST Port-Group  (ptr)
--- -------------------- ---- --- -------- --------- ----------  --------
  1 MQSS_engine.9.0.60     9   0     60       0          NA       3b421ee0
  2 MQSS_engine.9.0.61     9   0     61       1          NA       3b430b08
```

Then, once you have identified which pre-classifier engine is attached to your given ingress physical port, use the second command related to a specific pre-classifier ID. To identify which pre-classifier is attached to your ingress port you need to refer to the FPC, PIC, and if need be, the ASIC-INST (PFE ID) rows. In this next example we want to check the MAC stats for port et-9/0/3, which is attached to FPC 9, PIC 0, and PFE 0 (aka *ASIC-INST*). So we are attached to pre-classifier 1:

```
(mx vty)# show precl-eng 1 statistics
         stream    Traffic
 port      ID       Class             TX pkts              RX pkts          Dropped pkts
------  --------  ----------         ----------           ----------       --------------
  24      1165        RT          0000000000000000      0000000000000000    0000000000000000
  24      1166        CTRL        0000000021495607      0000000021495607    0000000000000000
  24      1167        BE          0000000222685201      0000000222685201    0000000000000000

  25      1213        RT          0000000000000000      0000000000000000    0000000000000000
  25      1214        CTRL        0000000323743195      0000000323743195    0000000000000000
  25      1215        BE          0000076144622343      0000076144622343    0000000000000000
```

This command displays the statistics for all ports attached to this given pre-classifier engine ID. For each port you'll see the CRTL stream (also named *Medium stream*) and BE stream (also named *Low stream*) statistics.

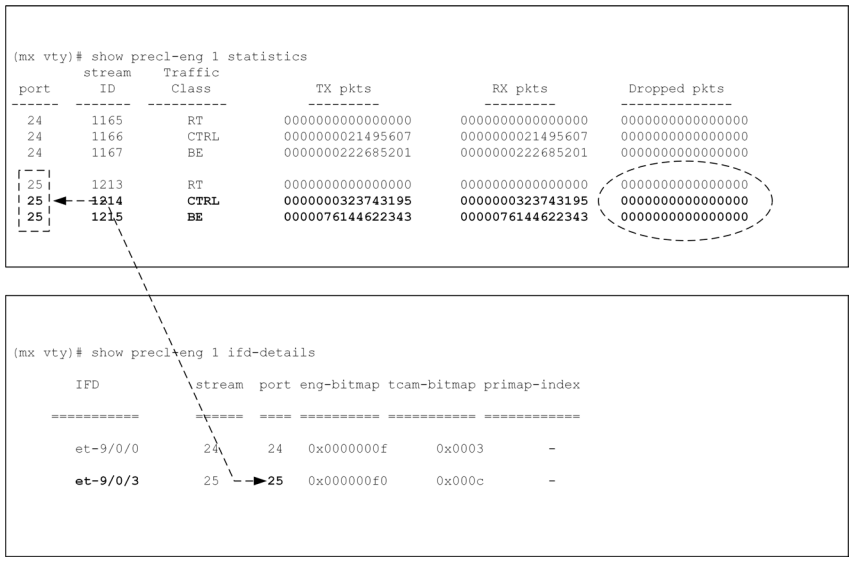
NOTE The RT (Real Time) stream is never used.

You can correlate the above stats with the following command that gives you the assigned MAC port ID for a given physical interface:

```
(mx vty)# (mx vty)#show precl-eng 1 ifd-details
        IFD             stream  port eng-bitmap tcam-bitmap primap-index
     ===========        ======  ==== ========== =========== ============
        et-9/0/0         24      24   0x0000000f      0x0003        -
        et-9/0/3         25      25   0x000000f0      0x000c        -
```

Figure 2.6 will help you determine the relationship between the last two commands.

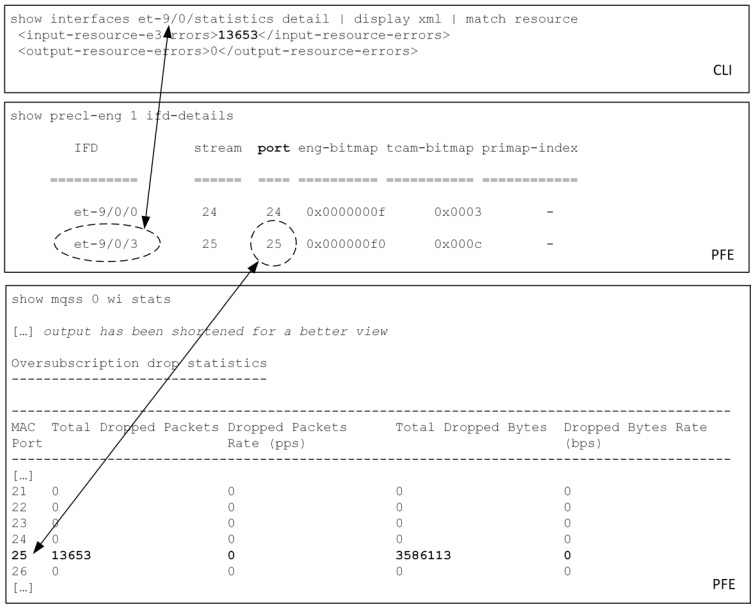
Figure 2.6 IFD and Ingress Stream Mapping for Pre-classifier



Once pre-classified, the Ethernet frame is sent (Step 2) into the right ingress stream (Medium/CTRL or Low/BE) attached to the WAN input (WI) block. WI manages the Ethernet flow control mechanisms and then splits the packets into chunks if needed. "If needed" means if the packet size is more than 224 bytes. As mentioned earlier, EA ASIC plays only with small packets. Thus, if the packet size is less than 224Bytes the EA will handle the entire packet. Otherwise, only the first 192 bytes of the packet will be taken into account and will constitute the packet HEAD (a kind of packet descriptor for the EA ASIC). The rest of the packet (packet TAIL) will be stored into the external memory (Step 3). The WI statistics are provided by the following commands and with the eyes of a support engineer, you should usually have a look at the oversubscription drop statistics. Indeed, when the EA ASIC is overloaded the drops might occur at the WI block level. Figure 2.7 shows sample output. Notice that the MAC Port ID referring to a specific physical port can be found by correlating with one other command (the pre-classifier engine's command mentioned earlier).

NOTE These drops are also reported as interface Input Resource Error (see below).

Figure 2.7 WI Input Drops Due to ASIC Overloading



Then the packet HEAD is supplied to the Dispatcher (Step 4), which creates a packet context inside the ASIC and then passes the packet HEAD to the LO block (Step 5). The LO is actually divided into four LO blocks. The four LO blocks manage the output interface toward the LUSS. You can retrieve some interesting statistics for DRD to LO (DRD means Dispatcher ReorDer) interfaces and LO to LUSS interfaces by issuing these commands:

```
(mx vty)# show mqss 0 drd stats

DRD statistics
--------------

DRD Global Statistics
---------------------

------------------------------------------------------------------------
Counter Name                Total                 Rate per second
------------------------------------------------------------------------
lo_pkt_cnt[0]               66311734494           4070
lo_pkt_cnt[1]               66578729162           4042
lo_pkt_cnt[2]               66444444187           4225
lo_pkt_cnt[3]               66442981009           3958
[..]

(mx vty)# show mqss 0 lo stats
```

```
LO statistics
-------------

-------------------------------------------------------------------------------------------------
LO Block  Parcel Name           Counter Name                Total                 Rate
-------------------------------------------------------------------------------------------------
[..] We kept only interesting counters:
0         M2L_Packet            Parcels sent to LUSS        14779405183           4157 pps
0         M2L_Packet            Parcel bytes sent to LUSS   319077328041          776520 bps
[..]
0         M2L_PacketHead        Parcels sent to LUSS        51541997771           24952 pps
0         M2L_PacketHead        Parcel bytes sent to LUSS   10720735536368        41520200 bps
[..]
0         Error Parcels         Error parcels sent to LUSS  96109340              0 pps
```

NOTE `M2L_Packet` means the entire packet sent from MQSS LO to LUSS – this counter is incrementing when LO processes packets with a size less than 224 bytes. `M2L_PacketHead` means packet HEAD sent from MQSS LO to LUSS – this increments when the size of the packet is more than 224 bytes, and therefore only the packet HEAD has been kept inside the EA (the first 196 bytes).

Inside the LUSS, the packet or packet HEAD is processed: which means that many tasks are performed by the several PPEs (Packet Processor Engine) of LUSS, among others:

- Packet lookup
- Ingress classification
- Ingress filtering, policing
- Ingress rewriting
- Multicast tree replication creation
- Tunnel encap/decap
- Fabric queue assignment if remote PFE, or WAN queue assigned if local PFE
- Control plane packet identification
- Anti-DDoS mechanism

We will cover some of the above-referenced LUSS tasks in more detail in Chapter 3, when we follow a MPLS transit packet and a host packet inside the EA and ZT PFEs.

Now the packet exits the LUSS and goes back to MQSS where it is received by the LI block (Step 7). LI is also made of four LI blocks. In parallel, LUSS has provided information to MQSS such as which fabric queue to use in order to reach the remote PFE (selected after the lookup), the multicast unary tree replication in case of a multicast packet, or if the packet must be dropped. If needed, the command `show mqss <PFE-ID> li stats` gives some similar outputs as the LO one. The packet's

header might be modified by the LUSS, thus the LI re-writes the received packet (rewrites in case of the entire packet) or packet HEAD (add the HEAD to the TAIL) into the external memory (Step 8).

The LI passes the packet or packet HEAD to the dispatcher (Step 8). When the dispatcher decides the packet is eligible to be sent out, it first asks the drop and stats block (Step 9) if the packet needs to be dropped or not. If the answer is *yes*, all the resources/contexts associated to the packet will be freed. If the drop and stats engine determines that the packet should be forwarded to a remote PFE it notifies the dispatcher, which then gives the lead to the cell compute block (Step 10). This block calculates the number of fabric cells required to forward the entire packet through the fabric and finally forwards the information to the Fabric Scheduler (Step 11). Inside the fabric scheduler, based on the fabric queue assigned by LUSS and the fabric CoS configuration, when the enqueued packet or packet HEAD is now eligible to be sent out, the fabric scheduler retrieves the entire packet (remember LI has added the HEAD back after the LUSS processing) from the external memory (Step 12) and sends the entire packet with other internal information in a proprietary header (provided by LUSS) to the FO block (Step 13).

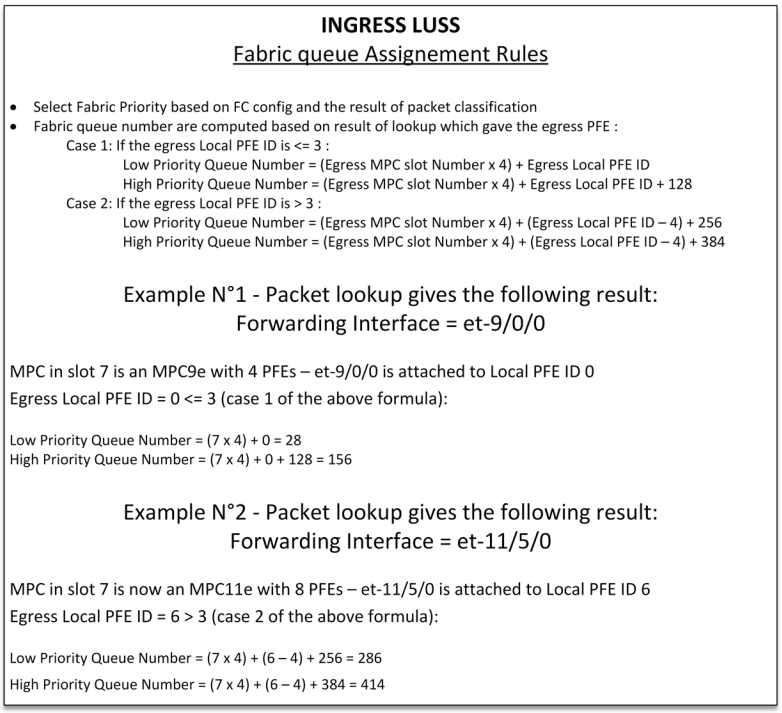
Before moving to the FO block, let's do a review of how the fabric queue numbers are assigned. Each MQSS maintains two separate queues per destination PFE. One high priority queue and one low priority queue. For your information, the high priority queue has 97.5% of guaranteed traffic and the low priority queue has 2.5%.

A packet is assigned to a given queue by the LUSS depending on your CoS configuration. Here, after a sample CoS configuration:

```
door7302@mx2020# show configuration class-of-service
[…]
forwarding-classes {
    class BE queue-num 0 priority low;
    class SILVER queue-num 1 priority high;
    class GOLD queue-num 2 priority high;
    class PREMIUM queue-num 4 priority high;
}
```

In this sample configuration, all traffic classified into the BE forwarding class will be assigned to the low priority fabric queues. All other traffic classified into the SILVER, GOLD, or PREMIUM FC will be conveyed by the high priority fabric queues. Now we know how fabric queue priority is assigned by configuration. We can see how the queue number (queue ID) is calculated by the LUSS. The ingress LUSS, after packet lookup, knows on which egress PFE the packet should be forwarded to be sent out. Each PFE inside a chassis has got a local PFE ID. The local ID depends on the type of MPC. Each MPC has a fixed number of PFEs between 1 and 8. The local PFE ID always starts to 0 and increments by 1 for the next PFE of the MPC. For example, the MPC7e card has two PFEs numbered from 0 to 1 while the MPC11e has eight PFEs numbered from 0 to 7. Based on the local PFE ID and the MPC slot number, which hosts this PFE, the LUSS can compute the fabric queue number. Figure 2.8 details how the fabric queue number is assigned.

Figure 2.8 Fabric Queue Number Assignment



Now you know how to find the fabric queue number and priority of any destination PFE. Let's move back to the fabric scheduler block of the MQSS. You are now able to check traffic statistics for the high and low priority queue of a given egress PFE with the following command. In the example we want to check the two fabric queues used to reach the PFE 0 of the MPC in slot 11. The number 44 means it is a destination MPC in slot 11 with the Local PFE ID 0. This is the low priority fabric queue to reach PFE 0 of MPC in slot 11:

```
(mx vty)# show mqss 0 sched-fab q-node stats 44

Queue statistics (Queue 0044)
-----------------------------

------------------------------------------------------------------------------
Color  Outcome           Counter  Total               Rate
                         Name
------------------------------------------------------------------------------
All    Forwarded (Rule)  Packets  41687833905         99908 pps
                         Bytes    21260794451722      407628536 bps
All    TAIL drops        Packets  0                   0 pps
                         Bytes    0                   0 bps
```

```
0      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
1      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
2      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
3      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
------------------------------------------------------------------------------

Drop structure
--------------
[..]
                         : Queue Depth: 512 bytes
```

The 172 means it is a destination MPC in slot 11 with the Local PFE ID 0 plus the offset of 128. This is the high priority fabric queue to reach PFE 0 of MPC in slot 11:

```
(mx vty)# show mqss 0 sched-fab q-node stats 172

Queue statistics (Queue 0172)
-----------------------------

------------------------------------------------------------------------------
Color  Outcome           Counter  Total               Rate
                         Name
------------------------------------------------------------------------------
All    Forwarded (Rule)  Packets  3643358325          1002 pps
                         Bytes    58297008040         128312 bps
All    TAIL drops        Packets  0                   0 pps
                         Bytes    0                   0 bps
0      WRED drops        Packets  0                   0 pps
                         Bytes    0                   0 bps
1      WRED drops        Packets  0                   0 pps
                         Bytes    0                   0 bps
2      WRED drops        Packets  0                   0 pps
                         Bytes    0                   0 bps
3      WRED drops        Packets  0                   0 pps
                         Bytes    0                   0 bps
------------------------------------------------------------------------------

Drop structure
--------------
[..]
                         : Queue Depth: 0
```

Finally, the FO block (steps 13/14) splits the data (the entire packet with its appended proprietary header) in fixed sized cells. Then it sends a request over the fabric targeting the remote PFE to ask it if the packet can be sent. When it is ready to receive the packet the remote PFE acknowledges by sending back a grant message to the source PFE.

NOTE This mechanism of request/grant allows an overloaded remote PFE to back-press the source PFE. The egress PFE, when overloaded, may throttle the grant messages and thus trigger packet buffering and, if needed, drops on the

ingress PFE (buffering is done in fabric High / Low priority queues). The DBB is also used to buffer packets in transit from one PFE to another one. The EA ASIC offers a buffer of 4 msecs per Fabric queue.

Let's retrieve some FO statistics by issuing the following command. Just have a look at the second tab, the `Counter Group 1`. We will focus on `Counter Group 0` in Chapter 3:

```
(mx vty)# show mqss 0 fo stats

FO statistics
-------------

Counter group 0
[…]
Counter group 1

-------------------------------------------------------------------------------------------------------------
Set Type   Mask  Match Total Requests      Requests per second  Total Grants       Grants per second
Total Cells        Cells per second
-------------------------------------------------------------------------------------------------------------
0  Port   0x3ff 0x0   13631861284632      200150            17772996817814      380284
13631861284634      200152
1  Port   0x3ff 0x1   13631861284686      200158            17772996817818      380284
13631861284686      200158
2  Port   0x3ff 0x2   13631861284688      200161            17772996817815      380281
13631861284688      200161
3  Port   0x3ff 0x3   13631861284690      200165            17772996817827      380283
13631861284693      200168
4  Port   0x3ff 0x4   13631861284692      200169            17772996817825      380294
13631861284694      200171
5  Port   0x3ff 0x5   13631861284698      200177            17772996817833      380294
13631861284698      200176
6  Port   0x3ff 0x6   13631861284696      200176            17772996817850      380299
13631861284697      200175
7  Port   0x3ff 0x7   13631861284702      200183            17772996817833      380298
13631861284702      200183
8  Port   0x3ff 0x8   13631861284705      200189            17772996817843      380306
13631861284705      200189
9  Port   0x3ff 0x9   13631861284707      200188            17772996817863      380306
13631861284707      200186
10 Port   0x3ff 0xa   13631861284704      200182            17772996817844      380310
13631861284704      200180
11 Port   0x3ff 0xb   13631861284703      200176            17772996817866      380309
13631861284703      200176
12 Port   0x3ff 0xc   13631861284699      200173            17772996817868      380312
13631861284699      200173
13 Port   0x3ff 0xd   13631861284694      200173            17772996817871      380315
13631861284694      200173
14 Port   0x3ff 0xe   13631861284693      200173            17772996817877      380326
13631861284693      200173
15 Port   0x3ff 0xf   13631861284694      200173            17772996817893      380329
13631861284694      200173
16 Port   0x3ff 0x10  13631861284696      200175            17772996817898      380325
13631861284696      200175
17 Port   0x3ff 0x11  13631861271421      200178            17772996804498      380319
13631861271421      200178
18 Port   0x3ff 0x12  13631861284695      200176            17772996817903      380324
```

```
13631861284695      200176
19  Port   0x3ff 0x13  13631861271809      200175            17772996804899      380314
13631861271809      200175
20  Port   0x3ff 0x14  13631861272284      200173            17772996805369      380311
13631861272284      200173
21  Port   0x3ff 0x15  13631861284693      200171            17772996817911      380299
13631861284693      200171
22  Port   0x3ff 0x16  13631861284693      200167            17772996817916      380292
13631861284693      200167
23  Port   0x3ff 0x17  13631861284695      200165            17772996817921      380293
13631861284696      200166

Total               327164670794069     4804146           426551923589855     9127307
327164670794078     4804148
```

In this case, we're using a MX2020 with 24 (0 to 23) planes:

```
door7302@mx2020> show chassis fabric plane-location
------------Fabric Plane Locations-------------
Plane 0              Switch Fabric Board 0
Plane 1              Switch Fabric Board 0
Plane 2              Switch Fabric Board 0
Plane 3              Switch Fabric Board 1
Plane 4              Switch Fabric Board 1
Plane 5              Switch Fabric Board 1
Plane 6              Switch Fabric Board 2
Plane 7              Switch Fabric Board 2
Plane 8              Switch Fabric Board 2
Plane 9              Switch Fabric Board 3
Plane 10             Switch Fabric Board 3
Plane 11             Switch Fabric Board 3
Plane 12             Switch Fabric Board 4
Plane 13             Switch Fabric Board 4
Plane 14             Switch Fabric Board 4
Plane 15             Switch Fabric Board 5
Plane 16             Switch Fabric Board 5
Plane 17             Switch Fabric Board 5
Plane 18             Switch Fabric Board 6
Plane 19             Switch Fabric Board 6
Plane 20             Switch Fabric Board 6
Plane 21             Switch Fabric Board 7
Plane 22             Switch Fabric Board 7
Plane 23             Switch Fabric Board 7
```

So FO statistics show you the number of Request/Grant sent/received per plane. Be careful, port number (displays by the *previous* MQSS FO stats command) doesn't mean plane number. For instance, Port 0 doesn't mean Plane 0.

To retrieve which port number is really attached to which plane number you should use the next command. The command requires two parameters:

- The fabric board number that hosts the plane. For a MX2020 chassis it's a value between 0 and 7. Hereafter, we ask for SFB in slot 6.
- The local plane ID. It depends on the number of planes hosted by fabric board. In our case we have three planes per SFB, therefore this value could be 0, 1, or 2. Hereafter we ask for plane 0 (of SFB 6).

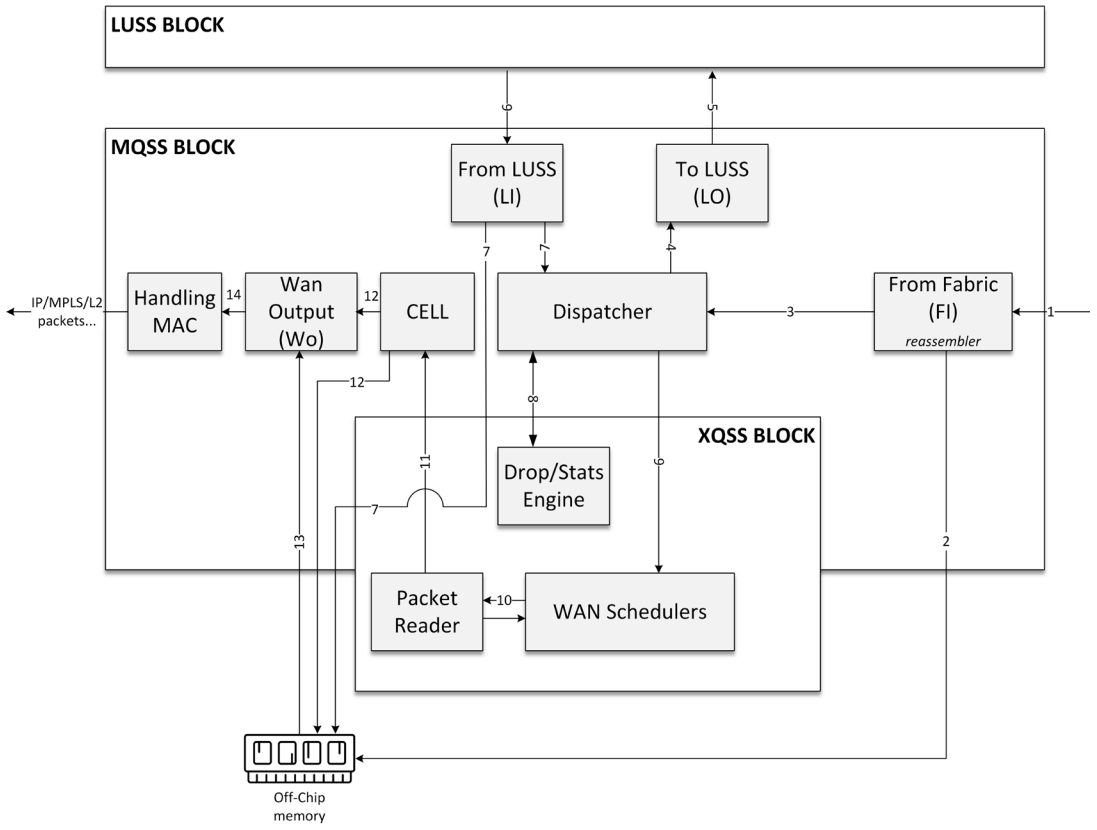
The following output shows us that the plane 0 of SFB 6 (absolute plane number 18) is attached to port 5. Thus, the line of statistics prefixes with port 5 of the show `mqss 0 fo stats` provides statistics regarding traffic sent to plane 18 (plane 0 of SFB 6):

```
(mx vty)#(mx vty)# show fabric link-detail 6 0
 PFE# - asic_name - port - tx_chan - rx_chan
=============================================
    0 - EACHIP(0) -    5 -     57  -     12
    1 - EACHIP(1) -    5 -     57  -     12
    2 - EACHIP(2) -    5 -     57  -     12
    3 - EACHIP(3) -    5 -     57  -     12
```

Egress Packet Flow

The packet has crossed the fabric planes and has reached the egress EA PFE. Figure 2.9 zooms in on the egress PFE. You will notice XQSS is now involved.

Figure 2.9 Inside the Egress EA PFE



The packet comes from the fabric side, into the FI block of the MQSS (Step 1). The FI reassembles the packet from the cells and, as the ingress WI did earlier, it either keeps the entire packet if its size is less than 224 bytes, or keeps only the first 196 bytes (HEAD) and pushes the rest of the packet (TAIL) into the external memory (Step 2).

Some statistics can be collected from the FI block with the following command. The statistics below give you the sum of all the FI streams, meaning all traffic coming from all source PFEs destined to this PFE. We will see how to filter those statistics for a given source PFE, if needed, in Chapter 3.

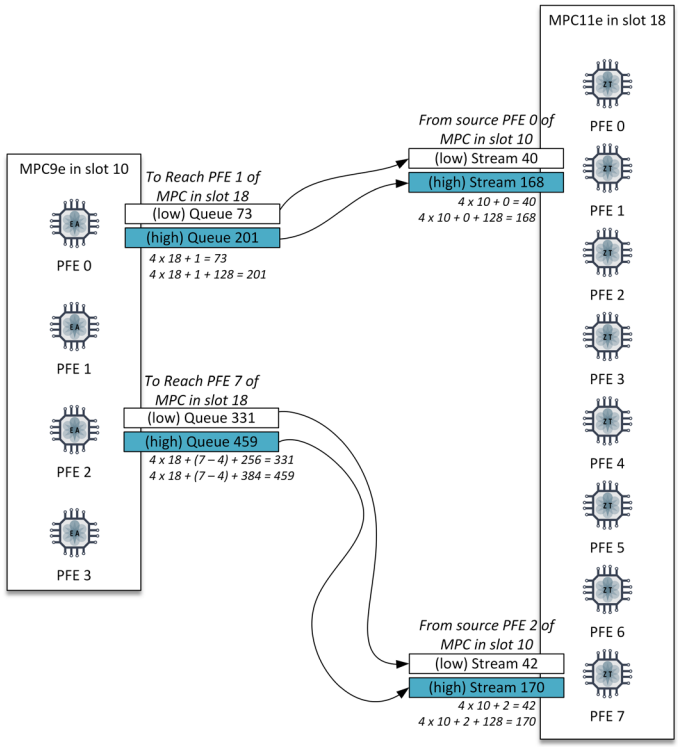
```
(mx vty)# show mqss 0 fi stats

-------------------------------------------------------------------------------------------------
Counter Name                                          Total                Rate
-------------------------------------------------------------------------------------------------
Valid data cells received from FABIO                  1955808696468        816019 cps
Valid fabric grants received from FABIO               1955096652530        816001 grants/second
Valid fabric requests received from FABIO             1955808696477        816004 requests/second
Valid fabric requests sent to FO                      1955808696477        816004 requests/second
Received cells in input block for enabled streams     1955808695145        816004 cps
Received packets in input block for enabled streams   196123577766         116051 pps
Cells dropped in FI memory interface                  9                    0 cps
Packets dropped in FI memory interface                3                    0 pps
Packets sent out of PSV                               196123577779         116051 pps
Error packets sent out of PSV                         0                    0 pps
-------------------------------------------------------------------------------------------------
```

It is important that FI handles packets into precisely separated streams depending on the source PFE that has sent the packets. On the FI side each source PFE has two streams, one high and one low, corresponding to the high and low priority fabric queues described on the ingress side. The way to calculate the attached FI streams of a given source PFE is the same as the formula used to calculate the fabric queue numbers (see Figure 2.8) on the ingress PFE. For instance, the traffic sent by PFE 1 of the MPC in slot 9 will be received by the FI block:

- Low priority stream: 37 = 4 x 9 + 1
- High priority stream: 165 = 4 x 9 + 1 + 128

Figure 2.10 shows you the mapping between fabric queues on the ingress side and fabric streams on the egress side.

Figure 2.10 Fabric Queue/Stream Mapping



Once handled by the FI, the packet or packet HEAD is sent to the dispatcher (Step 3) that, per ingress direction, creates a context for the packet and forwards it to one of the four LO blocks (Step 4). The 5/6 steps are almost the same as those described earlier when the packet was in the ingress PFE. The LUSS does several tasks such as:

- Look up the packet to find the forwarding interface to send out the packet.
- Identify the WAN queue to use based on ingress classification – the forwarding class assigned on the ingress side has been conveyed into the proprietary header with the packet, but it can be overridden by the egress PFE.
- Egress filtering and/or policing.
- Rewrite some fields of the packet.
- Add additional headers (dot1q, MPLS).

Then the packet moves back to MQSS (Step 6). You can use commands similar to those presented for the ingress PFE to retrieve statistics between dispatcher/DRD and LO (Step 4), between LO and the LUSS (Step 5), or LUSS and the LI block (Step 6):

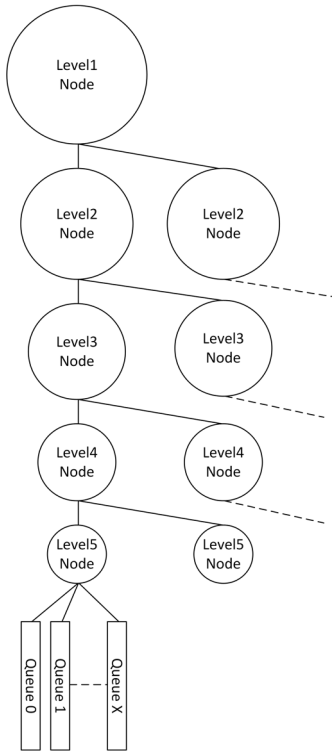
- `show mqss <pfe-id> drd stats`
- `show mqss <pfe-id> lo stats`
- `show mqss <pfe-id> li stats`

The LI block copies the modified packet or packet HEAD to the external memory and then gives it to the Dispatcher. This one asks the drop and stats engine of the XQSS block if packet must be dropped or not (Step 8). If not, the dispatcher sends to the packet HEAD, and when it becomes eligible to be sent out, to the XQSS block (Step 9).

In the XQSS, the packet descriptor is handled by the WAN schedulers block where it is enqueued in the right WAN queue assigned earlier by the LUSS. XQSS supports port and hierarchical queuing.

Let's take a short break here to present how CoS is implemented by the XQSS block. XQSS has been designed to support up to five levels of hierarchical schedulers. Each level is actually a scheduler that manages priorities, shaping rates, and bandwidth allocation. Finally, the last level of scheduler is attached to the WAN queues. Figure 2.11 depicts this concept.

Figure 2.11 Five Levels of the CoS Model

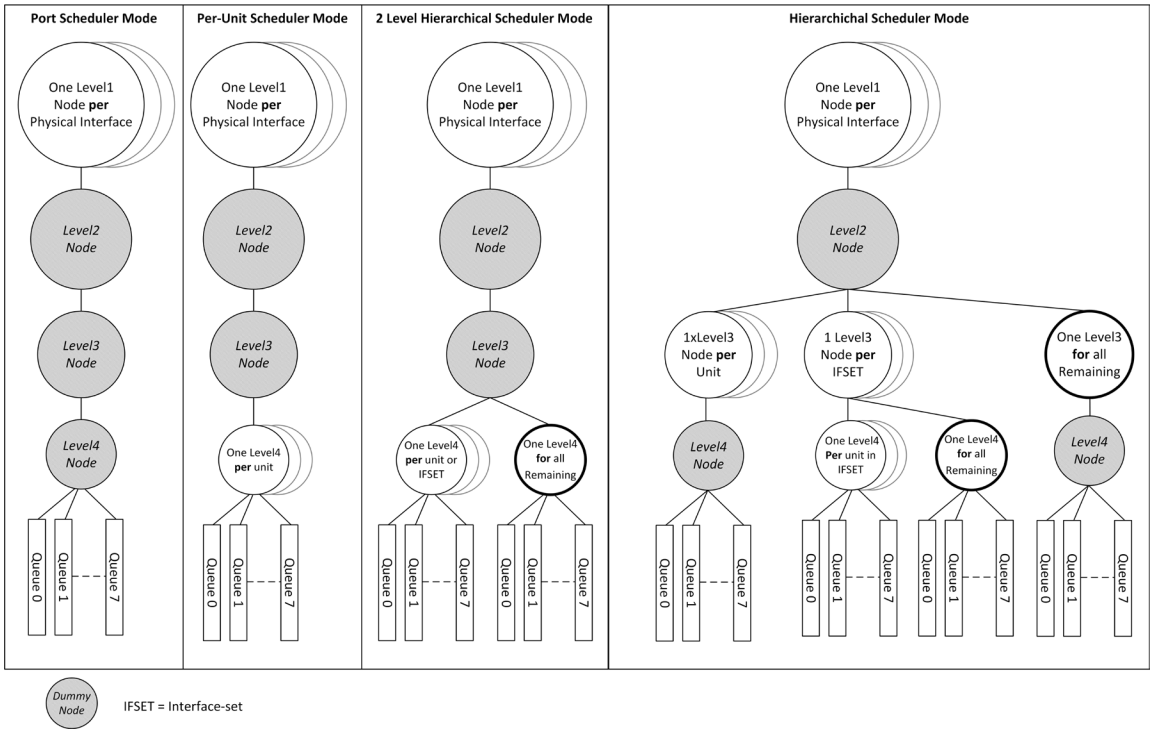


On EA line cards currently only four levels are used:

- A port scheduler CoS model
- A per-unit scheduler CoS model
- A two-level hierarchical scheduler CoS model
- A hierarchical scheduler CoS model

Depending on your CoS configuration, some levels may be unused – so in this case let's talk about *dummy* nodes. Figure 2.12 presents how many levels are involved in the EA's XQSS for each previous CoS model.

Figure 2.12 Supported EA CoS Modes and Levels Mapping



For the port scheduler mode, only the Level 1 node is involved. A maximum of eight dedicated queues is thus attached to each physical port. This is the default mode on Junos.

For the per-unit scheduler mode, both Level 1 and Level 4 nodes are involved. The port scheduling parameters are managed by a Level 1 node and one Level 4 scheduler node is assigned per interface's unit. Each interface's unit has a maximum of eight queues.

For the two levels of hierarchical scheduler mode, as in the previous example, only Level 1 (which manages port scheduling parameters) and Level 4 nodes are involved. But here, a Level 4 scheduler manages either an interface's unit or an interface-set (a group of units). And finally there is also one dedicated Level 4 node for all remaining units or interface-sets (configuration stanza: `output-traffic-control-profile-remaining`). As with the other two modes, each Level 4 node is attached to a maximum of eight queues.

Finally, the hierarchical scheduler mode involves Level 1, Level 3, and Level 4 nodes. Level 1 still manages port scheduling parameters. In this case, each interface has its own Level 3 node scheduler directly attached to a maximum of eight queues (Level 4 is unused). Moreover a separate Level 3 node is created for each interface-set. Each unit of each interface-set is managed by a Level 4 node scheduler. This one is attached to eight queues. A dedicated Level 4 node is also created for the remaining interface's units of the interface-set. Finally, a garbage Level 3 node is also created for all the remaining interface's units and interface-sets. This Level 3 is directly attached to a maximum of eight queues as the Level 4 is a dummy node in this case.

As mentioned earlier, the packet is now enqueued into the XQSS. At the PFE level you can check the WAN queues that are attached to:

- a given physical port: identified by what we called an *IFD*.
- a given interface's unit: identified by what we called an *IFL*.
- or a given interface-set: identified by what we called an *IFLSET*.

To make things simple, let's use the default CoS model (port scheduling mode) on the lab router. Let's check for a given physical port, therefore, et-9/0/0, how WAN queues are programmed into the XQSS.

You should first find the IFD index of your egress physical port. For that, issue the following command. Notice, this command also displays the IFL index of each unit attached to the physical interface:

```
door7302@mx2020# show interfaces et-9/0/0 | match index
  Interface index: 278    <<<<< IFD Index
  Logical interface et-9/0/0.0 (Index 606)    <<<<< IFL Index
```

Now issue the next PFE command on the MPC that hosts the physical interface to retrieve the nodes information:

```
(mx vty)# show cos halp ifd 278
    rich queueing enabled: 1
    Q chip present: 1
IFD name: et-9/0/0   (Index 278) egress information
    XQSS chip id: 0
    XQSS : chip Scheduler: 0
    XQSS chip L1 index: 5
    XQSS chip dummy L2 index: 1989
    XQSS chip dummy L3 index: 5
    XQSS chip dummy L4 index: 2
    Number of queues: 8
    XQSS chip base Q index: 16
```

We have now the Level 1 node scheduler index (5) and the XQSS base queue index (16 means index for queue 0, 17 is the index for queue 1, 18 for queue 2, etc.). With that information you can first display the scheduler parameters attached to our physical interface with the following command:

```
(mx vty)# show xqss 0 sched l1 5     <<< 5 = Level 1 node index retrieved above
L1 node configuration   : 5
        state           : Configured
        child_l2_nodes  : 1
        scheduler_pool  : 0
        gh_max_rate     : 0
        gm_max_rate     : 0
        gl_max_rate     : 0
        eh_max_rate     : 0
        el_max_rate     : 0
        max_rate        : 100000000000
        burst_size      : 67108864 (adjusted burst_size : 67108864)
        byte_adjust     : 22
        cell_mode       : FALSE
        min_pkt_adjust  : 0
```

And finally, check a given WAN queue of a given interface. Hereafter the queue 0 (queue index 16):

```
(mx vty)# show xqss 0 sched queue 16
 Q node configuration   : 16
        state           : Configured
        ISSU priority   : HIGH
        scheduler_pool  : 0
        max_rate        : 100000000000
        g_rate          : 2000000000
        weight          : 5
        burst_size      : 67108864 (adjusted burst_size : 67108864)
        byte_adjust     : 22
        g_priority      : GL
        e_priority      : EL
        cell_mode       : FALSE
```

```
        tail_drop_rule  : 1277
        wred_drop_rule  : 4
        ecn_enable      : FALSE
        queue_mode      : DROP
        stats_enable    : TRUE
        scaling profile : 3
```

The previous commands are provided just for your information, and hopefully they might be helpful during complex troubleshooting sessions. Remember, classic CLI commands usually give enough information such as, for CoS interface: `show interface queue <interface-name> egress` or `show class-of-service interface <interface-name> comprehensive`.

At this point there are several interactions between the WAN scheduler and what we call the *Packet Reader,* which maintains the list of active streams (Step 10). To summarize, when the scheduler decides the packet is eligible to be dequeued, the Packet Reader notifies the cell block (Step 11). It actually provides information to the cell block about the packet's chucks. The cell block sends read requests to external memory (Step 12) in order to fetch the content of the entire packet (HEAD and TAIL). External memory returns the requested data payload to the WO block.

Finally, the WO block sends the entire packet to the MAC block, which computes and then appends the CRC32 to the Ethernet frame before sending out it. The WO statistics are available by using the next example command. Just have a look at the Counter set 0, which gives you the WAN output aggregated statistics (all Ethernet ports attached to the EA ASIC).. We'll see in Chapter 3 how to use Counter set 1 to filter the view on a given port:

```
(mx vty)# show mqss 0 wo stats

WO statistics
-------------

Counter set 0

    Connection number mask   : 0x0
    Connection number match  : 0x0
    Transmitted packets      : 137212310395 (100177 pps)
    Transmitted bytes        : 113707212044499 (401766112 bps)
    Transmitted flits        : 1254813302266 (600325 flits/sec)

Counter set 1
[…]
```

We have now finished our trip inside the EA ASIC.

As you will see in the next part of this chapter, there is much that is similar on the ZT ASICs. But there are also a lot of new PFE commands. Let's go inside.

Internal View of ZT-based Line Card

ZT is based around three main functional blocks: MQSS, LUSS, and XQSS, much like the EA ASIC. However, this ASIC is faster than the EA, requests less power, and embeds more inline features such as inline IPsec support. Finally, one of the most evolved features of ZT-based MPCs is the software running over the line card's CPU: this software has been completely rewritten to offer better performance and flexibility. It also provides a new PFE shell which offers a lot of new interesting commands.

So let's start by presenting this new software architecture in comparison with the EA-based MPC, and then we'll go inside the ZT ASIC to show the differences with the EA.

Software Architecture

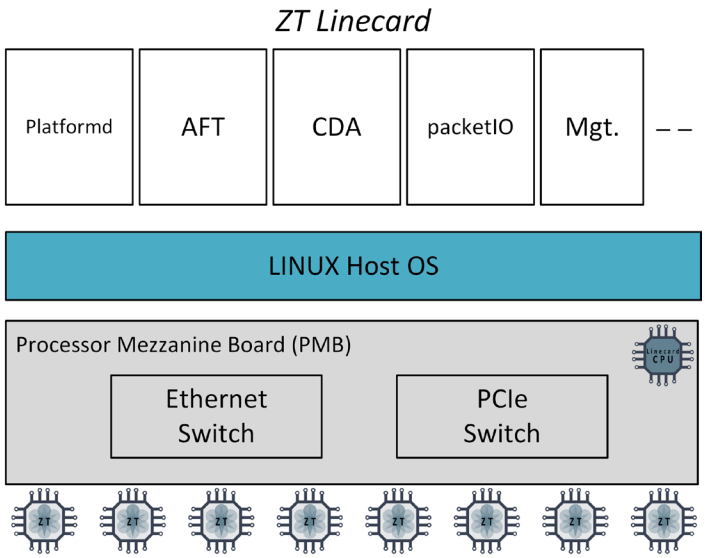
The ZT and EA line cards are both based on the Linux host OS but the Juniper software on top of this OS is totally different for the ZT.

The MPC11e line card is the first unified line card. It reuses the concepts and software architectures of Junos EVO (a complete new generation of Junos not yet widely deployed on the Juniper portfolio). The MPC10e line card, released before the MPC11e, is a kind of hybrid card which does not totally include all the new software enhancements we can find on the MPC11e, but nevertheless it's evolved compared to EA Line cards.

NOTE It's important to mention that all future line cards will be based on EVO concepts like the MPC11e.

As previously mentioned, on the EA line card all the PFE software components run on a monolithic 32-bit process: the uKernel. On the ZT line card, these PFE software components are split into several concurrent 64-bit processes to leverage multi-core CPUs (the MPC10/11e runs an 8-core CPU). Figure 2.13 illustrates the main modules of a ZT line card. As mentioned, the uKernel does the entire job on the EA, while on the ZT line card it runs a lighter version with less functions. On the ZT line card the uKernel is renamed *platformd* because it now manages only platform-related tasks. Figure 2.13 shows you the main software modules.

Figure 2.13 Software Architecture of ZT Line Card

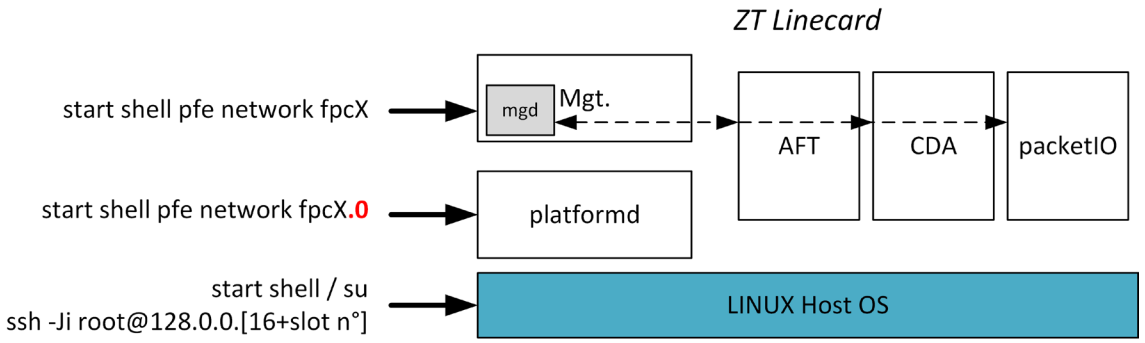


All the software modules on the ZT line card communicate and interact with each other. On the ZT, platformd is responsible for bringing up the ASICs, initializing the hardware components of some others—such as I2C bus, and SERDES interfaces—and collecting some hardware statistics. You can interact with the platformd daemon through a shell (like the classic PFE shell attached to uKernel) to retrieve some PFE statistics or troubleshooting PFE states. Notice that because platformd doesn't manage the same elements, like uKernel did, some classic and well known PFE commands have been removed from the classic PFE shell. But don't worry, as there are similar commands available with the new PFE shell provided by the new software architecture.

All the ASIC management/programing tasks are now done by the CDA module. CDA stands for *Common Driver Architecture* – a new generic ASIC driver framework. It is composed of several processes that manage the ZT chip itself and its three embedded components (MQSS, LUSS, and XQSS). To summarize, CDA is the interface with the ASICs.

The management module (Mgt.) includes several Linux processes (like syslog) and Juniper processes (such as an instance of MGD, the same on the RE, that offers us the new PFE shell). Having MGD embedded in the MPC enables some interesting elements! You can now take advantage of the MGD's flexibility for new PFE CLI commands (such as a pipe match with regex support, or displaying the PFE command output in JSON or XML formats). Figure 2.14 shows you how to access the old PFE shell (attached to platformd) by using the `.0` channel, the new PFE shell (attached to MGD) or the Linux OS.

Figure 2.14 Interact with ZT MPC

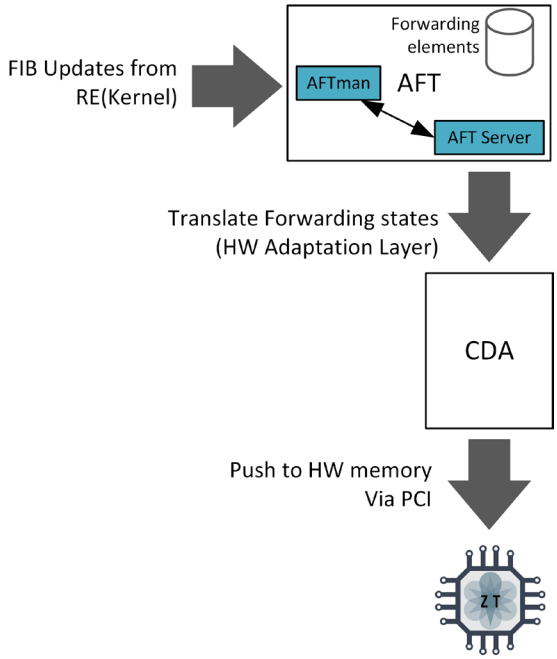


Okay, let's move to the next module: the AFT.

The AFT module, which means *advance forwarding toolkit*, is a new piece of software that manages forwarding states at the MPC level. It provides flexibility to view, manipulate, add, remove, and update forwarding elements. It provides the abstraction layer between the routing/firewalling/QoS configurations and the real forwarding states that reside inside the ASICs.

For example, AFT handles the FIB updates (through IPC) coming from the RE's kernel, then it models this routing information as forwarding elements (called *AFT nodes*), orchestrates an AFT token topology, and finally it communicates with the CDA to push these forwarding states into the hardware. Figure 2.15 illustrates this communication between several modules in a case of routing updates. Later we'll provide some PFE commands to interact with the AFT forwarding elements or to retrieve hardware information from the CDA module.

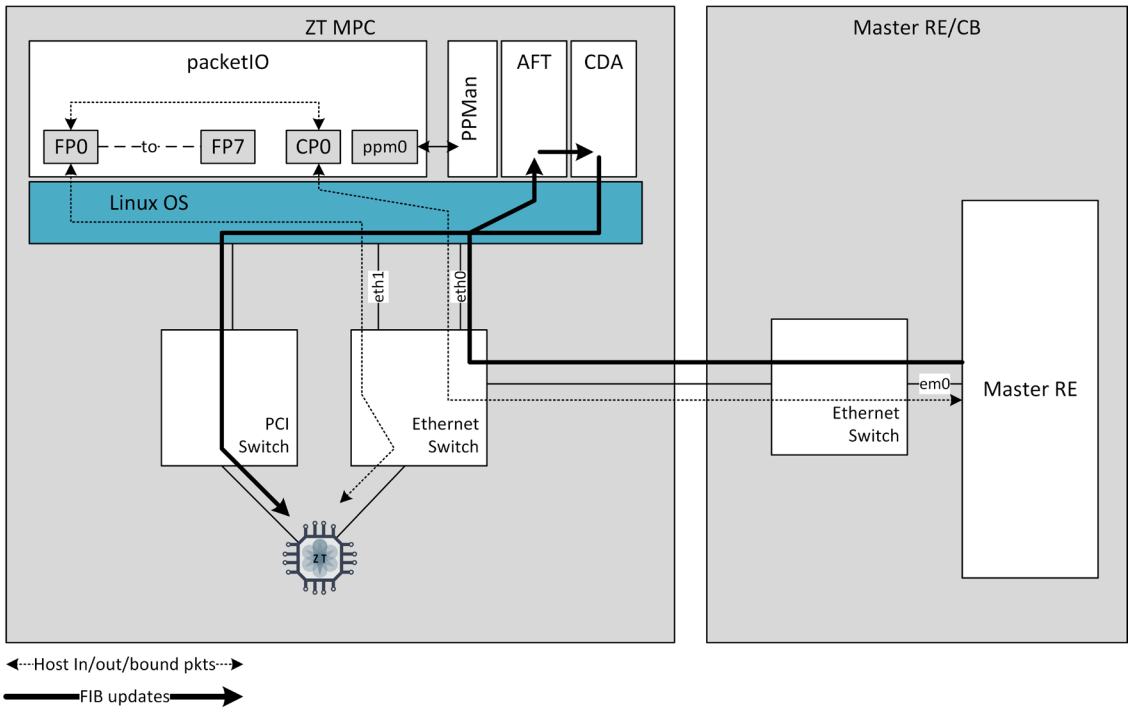
Figure 2.15 Inter-module Communication for Route Updates



In AFT the forwarding state representation is viewed like nodes connected to each other to form trees (the forwarding paths). AFT provides flexible APIs to manipulate nodes and trees. Each node resides in a kind of sandbox and is identified uniquely by an AFT token ID.

PacketIO is also a new module that comes with the ZT line card. It is responsible for transmitting and receiving packets to and from the PFE forwarding ASIC with better performances than on the EA line card. On the ZT line card all the ASICs, the Linux host OS, and the RE are connected together around an Ethernet switch hosted on the MPC. In Figure 2.16 you can see how host packets and FIB updates are handled by the MPC. The packetIO process communicates with ZT ASIC through virtual interfaces called FP (forwarding path). There is one FP interface per ZT ASIC – therefore on MPC10e there are 3 FP and on MPC11e 8 FP interfaces. On the other side packetIO is connected to the RE (em0) through the CP0 virtual interface. Finally there is also a dedicated interface, ppm0, connected to another process, PPMan, responsible for generating and handling inline keepalives of a set of protocols (such as BFD, LACP, etc.). Each virtual FP interface has eight ingress and eight egress queues to handle host traffic coming from and to the ZT ASIC. The CP0 virtual interface has only eight egress queues (by default) to forward traffic to RE.

Figure 2.16 Internal Physical Ethernet Connections



For FIB updates, the path to reach the ASIC is different. The RE sends updates via a socket directly to the AFT module. As mentioned earlier, this module converts these IPC messages into forwarding elements and then notifies the CDA module to program these elements into the ZT ASIC through the PCIe interface.

NOTE There is also a connection between AFT and packetIO. This is needed in case packetIO wants to send a resolve request to the RE via AFT and also to get route information if needed as only AFT know such data.

Okay, we've completed our short break to present the software architecture of the ZT line card. Now let's go back inside ZT's internal view.

Hardware Architecture

As mentioned earlier, the ZT ASIC, like the EA ASIC, is composed of three functional blocks:

- MQSS block: the queuing block
- LUSS block: the lookup block
- XQSS block: the rich QoSblock

The hardware architecture is similar to the EA, although note that the memory, its type, and its placement have been totally redesigned. There are two banks of memory:

- A *large external memory*: This is in charge of storing packets, inline JFlow records, and etc.
- A *smaller but faster and more flexible internal memory*: This is in charge of storing a part of fabric queue buffer and some other data used by the LUSS block.

Globally, a ZT ASIC offers 75msecs of buffer per physical port and 3.8msecs of buffer per fabric queue. Figure 2.17 illustrates the functional hardware architecture of the ZT.

Figure 2.17 Functional Blocks Inside the ZT

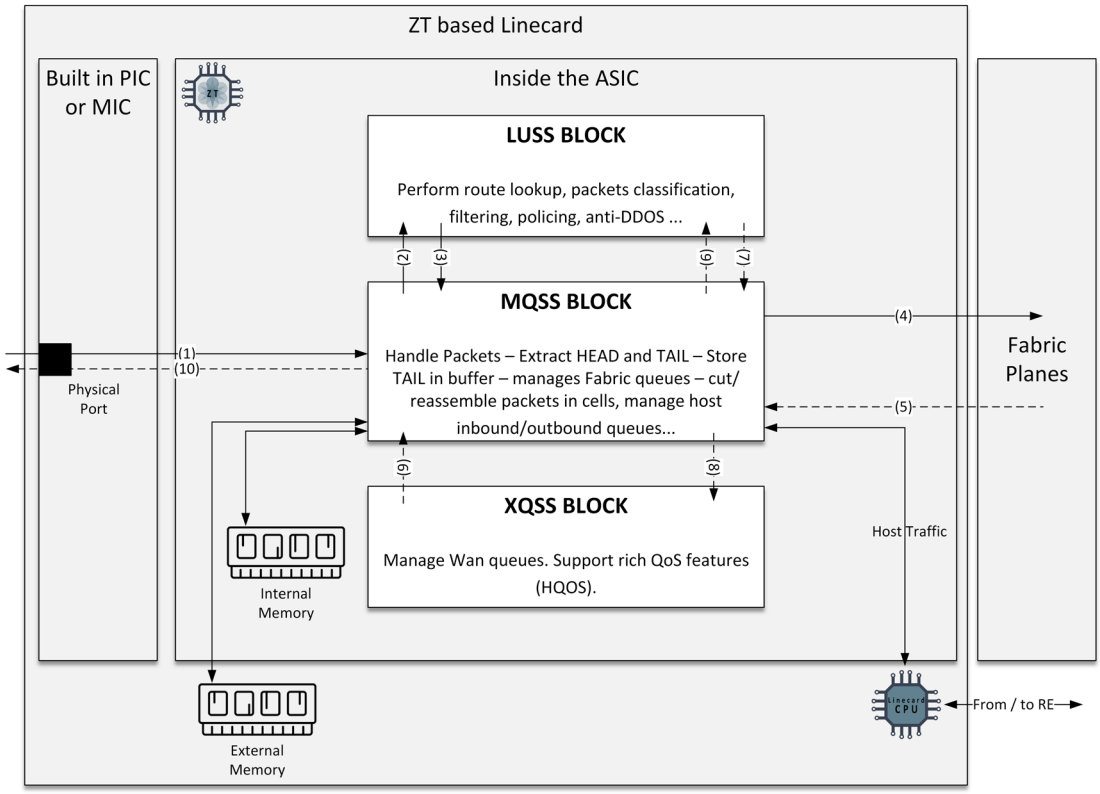
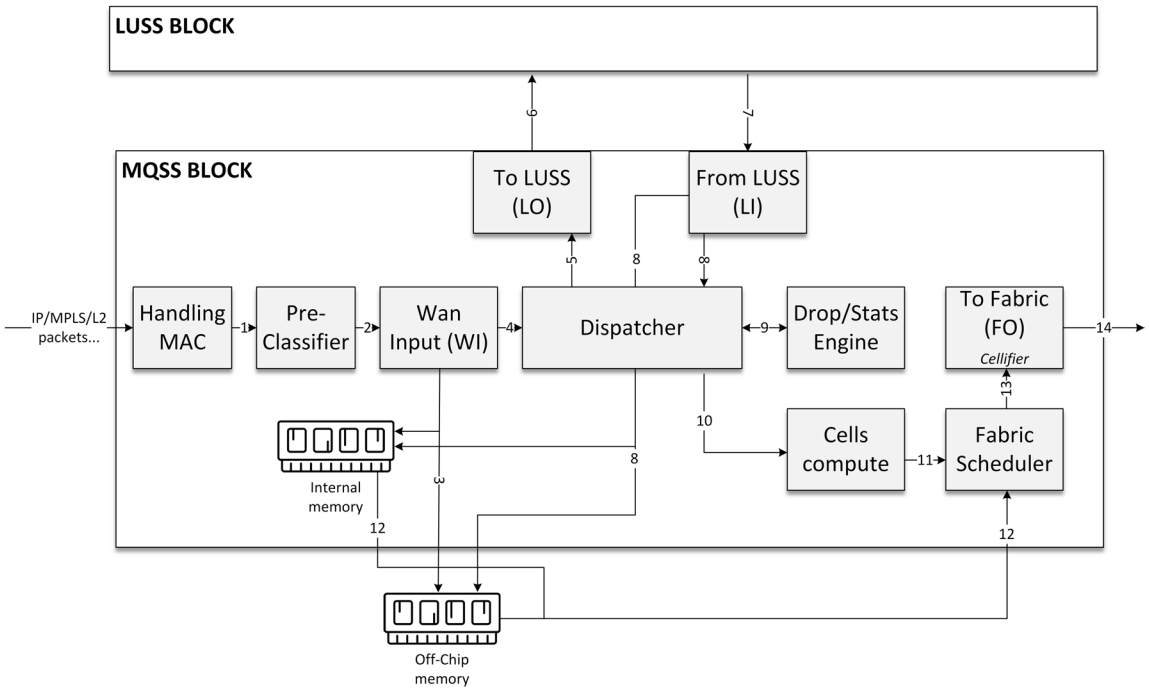


Figure 2.17 is very similar to when the EA ASIC was presented earlier in this chapter. You'll see that there are many similar PFE commands to collect various statistics from the PFE. However, you will also note a number of new commands are now available. Like all the previous generations of TRIO ASIC, ZT still manipulates three kinds of data: packet HEAD, packet TAIL, and cells (presented in detail in the EA walkthough).

Ingress Packet Flow

Let's go inside the ZT MQSS block like we did for the EA. You will notice there may be some redundant explanations, but the aim is to present the equivalent PFE commands on the ZT and to avoid switching between EA explanations, described in previous parts, and the ZT diagrams.

Figure 2.18 Ingress Traffic Inside the ZT MQSS



So far we haven't directly accessed the MAC statistics on the MPC11e line card coming from the MAC block. Nevertheless, you can retrieve some interesting Layer 2 statistics with this new PFE command. Notice we are attached to the new PFE shell. We didn't use the `.0` channel when issuing the `start shell pfe network` command:

```
door7302@mx2020> start shell pfe network fpc11
root@mx2020-fpc11:pfe> show interfaces statistics et-11/0/0
 Traffic Statistics:
   Input Packets         : 6802220021   Output Packets         : 721154175
   Input Bytes           : 3348146303930 Output Bytes          : 346124228574
   Input Multicasts      : 0            Output Multicasts      : 0
   Input Residue Packets : 0            Output Residue Packets : 0

 Traffic Rates:
   Input pps             : 0            Output pps             : 0
   Input bps             : 0            Output bps             : 0

 IPv6 statistics:
   Input Packets         : 29           Output Packets         : 0
   Input Bytes           : 1912         Output Bytes           : 0

 Errors:
   Input errors          : 0            Input framing errors   : 0
   Input queue drops     : 0            Input runts            : 0
   Input giants          : 0            Input discards         : 0
   Input fifo errors     : 0            Input no vc            : 0
   Output errors         : 0            Collisions             : 0
   Carrier transitions   : 43           Output fifo errors     : 0
   Output discards       : 0            Pic link errors        : 0

 Host Path Statistics:
   Input Packets         : 2004559      Output Packets         : 14231160
   Input Bytes           : 169134256    Output Bytes           : 2856768029

 Host Path Rates:
   Input pps             : 0            Output pps             : 0
   Input bps             : 0            Output bps             : 0

 Mac Statistics:               Receive                  Transmit
   Total octets                    3470567269495            359547064725
   Total packets                   6802223446               721422290
   Unicast packets                 5409303618               719843019
   Broadcast packets               313                      289
   Multicast packets               1392919515               1578982
   CRC/Align errors                0                        0
   FIFO errors                     0                        0
   MAC control frames              0                        0
   MAC pause frames                0                        0
   Oversized frames                0
   Jabber frames                   0
   Fragment frames                 0
   VLAN tagged frames              0
   Code violations                 0

 Macsec Statistics:
  Secure Channel transmitted
       Encrypted packets: 0
       Encrypted bytes:   0
       Protected packets: 0
       Protected bytes:   0
  Secure Association transmitted
       Encrypted packets: 0
       Protected packets: 0
```

```
  Secure Channel received
       Accepted packets: 0
       Validated bytes:  0
       Decrypted bytes:  0
  Secure Association received
       Accepted packets: 0
       Validated bytes:  0
       Decrypted bytes:  0

 OutputStream: 1092
 StreamNumber: 560
 StreamToken: 34930
 StreamFlags: 0x04 (ESMC)
        Total Packets               : 6802223415     Total Bytes              : 3443358371535
        Rx0 Packets                 : 0              Rx0 Bytes                : 0
        Rx1 Packets                 : 48711206       Rx1 Bytes                : 12577876459
        Rx2 Packets                 : 6753512209     Rx2 Bytes                : 3430780495076
        Drop0 Packets               : 0              Drop0 Bytes              : 0
        Drop1 Packets               : 0              Drop1 Bytes              : 0
        Drop2 Packets               : 0              Drop2 Bytes              : 0
        Unknown IIF Packets         : 0              Unknown IIF Bytes        : 0
        Checksum Packets            : 0              Checksum Bytes           : 0
        Unknown Proto Packets       : 351            Unknown Proto Bytes      : 38610
        Bad Ucast Mac Packets       : 0              Bad Ucast Mac Bytes      : 0
        Bad Ucast Mac IPv6 Packets  : 0              Bad Ucast Mac IPv6 Bytes : 0
        Bad Smac Packets            : 0              Bad Smac Bytes           : 0
        In STP Drop Packets         : 0              In STP Drop Bytes        : 0
        Out STP Drop Packets        : 0              Out STP Drop Bytes       : 0
        Vlan Check Error Packets    : 0              Vlan Check Error Bytes   : 0
        Frame errors Packets        : 0              Frame errors Bytes       : 0
        Bad IPv4 Hdr Packets        : 0              Bad IPv4 Hdr Bytes       : 0
        Bad IPv4 Len Packets        : 0              Bad IPv4 Len Bytes       : 0
        Bad IPv6 Hdr Packets        : 0              Bad IPv6 Hdr Bytes       : 0
        Bad IPv6 Len Packets        : 0              Bad IPv6 Len Bytes       : 0
        Out MTU Error Packets       : 0              Out MTU Error Bytes      : 0
        L4 Len Error Packets        : 0              L4 Len Error Bytes       : 0
        Error Access Ctrl Packets   : 0              Error Access Ctrl Bytes  : 0
```

On the MPC10e it's a little bit different. Although it's based on the ZT ASIC, remember the MPC10e is a kind of hybrid card between two generations – the old generation based on a monolithic uKernel and the new generation based on Junos EVO architecture. Thus the MPC10e still has access to some MAC statistics via next command. As you can see, we use `fpc10.0`; the `.0` channel means we are connected to the old PFE shell attached to the platformd process (also known as the uKernel):

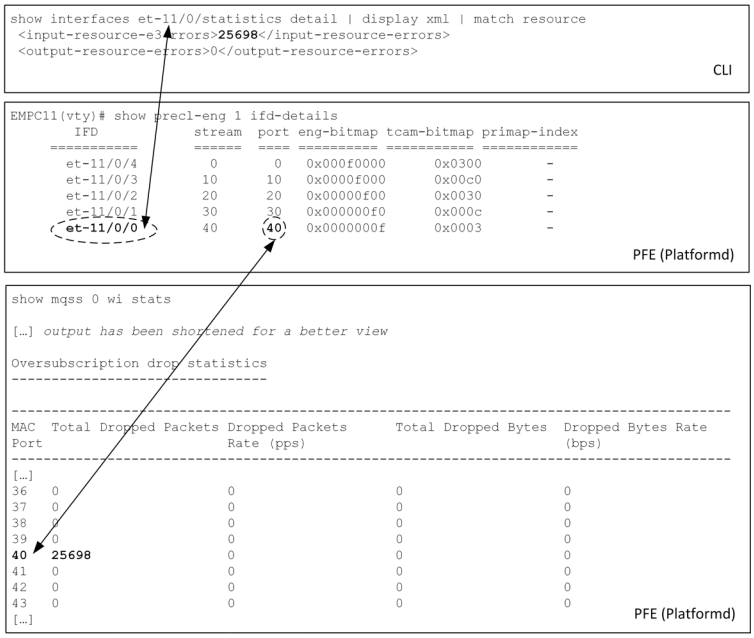
```
door7302@mx2020> start shell pfe network fpc10.0
(mx vty)# show ifd xe-10/0/0:0 mac statistics
Statistics
----------
  aFramesTransmittedOK:                                    5218
  aFramesReceivedOK:                                       5218
  aFrameCheckSequenceErrors:                                  0
  aAlignmentErrors:                                           0
  aPAUSEMACCtrlFramesTransmitted:                             0
  aPAUSEMACCtrlFramesReceived:                                0
```

```
aFrameTooLongErrors:                                0
aInRangeLengthErrors:                               0
VLANTransmittedOK:                                  0
VLANReceivedOK:                                     0
ifOutOctets:                                  1737594
ifInOctets:                                   1737594
ifInUcastPkts:                                      0
ifInMulticastPkts:                               5218
ifInBroadcastPkts:                                  0
ifInErrors:                                         0
ifOutErrors:                                        0
ifOutUcastPkts:                                     0
ifOutMulticastPkts:                              5218
ifOutBroadcastPkts:                                 0
etherStatsDropEvents:                               0
etherStatsOctets:                             1737594
etherStatsPkts:                                  5218
etherStatsJabbers:                                  0
etherStatsFragments:                                0
etherStatsUndersizePkts:                            0
etherStatsOversizePkts:                             0
etherStatsPkts64Octets:                             0
etherStatsPkts65to127Octets:                        0
etherStatsPkts128to255Octets:                       0
etherStatsPkts256to511Octets:                    5218
etherStatsPkts512to1023Octets:                      0
etherStatsPkts1024to1518Octets:                     0
etherStatsPkts1519toMaxOctets:                      0
etherStatsPkts64OctetsTx:                           0
etherStatsPkts65to127OctetsTx:                      0
etherStatsPkts128to255OctetsTx:                     0
etherStatsPkts256to511OctetsTx:                  5218
etherStatsPkts512to1023OctetsTx:                    0
etherStatsPkts1024to1518OctetsTx:                   0
etherStatsPkts1519toMaxOctetsTx:                    0
aMACControlFramesReceived:                          0
aCBFCPAUSEFramesReceived_0:                         0
aCBFCPAUSEFramesReceived_1:                         0
aCBFCPAUSEFramesReceived_2:                         0
aCBFCPAUSEFramesReceived_3:                         0
aCBFCPAUSEFramesReceived_4:                         0
aCBFCPAUSEFramesReceived_5:                         0
aCBFCPAUSEFramesReceived_6:                         0
aCBFCPAUSEFramesReceived_7:                         0
aCBFCPAUSEFramesTransmitted_0:                      0
aCBFCPAUSEFramesTransmitted_1:                      0
aCBFCPAUSEFramesTransmitted_2:                      0
aCBFCPAUSEFramesTransmitted_3:                      0
aCBFCPAUSEFramesTransmitted_4:                      0
aCBFCPAUSEFramesTransmitted_5:                      0
aCBFCPAUSEFramesTransmitted_6:                      0
aCBFCPAUSEFramesTransmitted_7:
```

On the ZT, the pre-classifier engine still does a first pre-classification based on the packet header, and separates the traffic in two different internal streams (CTRL and BE). Currently there are no pre-classifier statistics retrievable on the ZT-based line card. Nevertheless, there is an enhancement scheduled for a next release to add these statistics to be available for the ZT ASIC.

Once pre-classified, the Ethernet frame is sent (Step 2) into the right ingress stream (Medium/CTRL or Low/BE) attached to the WI block. On the ZT, WI also manages the Ethernet flow control mechanisms and then splits the packets into chunks if needed (if the packet size is greater than 224 bytes). The TAIL is stored either in the internal or external memory. The WI command presented for the EA is still valid. You can, like the EA, check oversubscription drops on ZT with the following combination of commands shown in Figure 2.19. In this case we are attached to platformd (the old PFE shell).

Figure 2.19 WI Input Drops Due to ASIC Overloading



Next, the packet HEAD is supplied to the dispatcher (Step 4), which creates a packet context inside the ASIC and then passes the packet HEAD to the LO block (Step 5). The LO block on the ZT is also divided into four LO blocks. The four LO blocks manage the output interface toward the LUSS. You can still retrieve interesting statistics for DRD to LO (DRD means *dispatcher*) interfaces, and LO to LUSS interfaces, by going to the platformd shell and issuing the same commands we used for the EA:

```
door7302@mx2020> start shell pfe network fpc11.0
(mx vty)# show mqss 0 drd stats

DRD statistics
--------------
```

```
DRD Global Statistics
---------------------

------------------------------------------------------------------------
Counter Name                Total                 Rate per second
------------------------------------------------------------------------
lo_pkt_cnt[0]               66311734494           4070
lo_pkt_cnt[1]               66578729162           4042
lo_pkt_cnt[2]               66444444187           4225
lo_pkt_cnt[3]               66442981009           3958
[..]

(mx vty)# show mqss 0 lo stats


LO statistics
-------------

-------------------------------------------------------------------------------------------------
LO Block  Parcel Name           Counter Name                Total                 Rate
-------------------------------------------------------------------------------------------------
[..] We kept only interesting counters:
0         M2L_Packet            Parcels sent to LUSS        14779405183           4157 pps
0         M2L_Packet            Parcel bytes sent to LUSS   319077328041          776520 bps
[..]
0         M2L_PacketHead        Parcels sent to LUSS        51541997771           24952 pps
0         M2L_PacketHead        Parcel bytes sent to LUSS   10720735536368        41520200 bps
[..]
0         Error Parcels         Error parcels sent to LUSS  96109340              0 pps
```

Inside the LUSS packet or packet HEAD things are processed in the same way we've already seen on the EA. The packet exits the LUSS and goes back to the MQSS where it is received by the LI block (Step 7). LI is also made of four LI blocks. In parallel, LUSS has provided information such as the fabric queue to MQSS to use in order to reach the remote PFE (selected after the lookup), and the multicast unary tree replication in case of a multicast packet or in case the packet must be dropped. If needed, issue the `show mqss <PFE-ID> li stats` command in order to retrieve similar outputs of the LO command. The packet's header might be modified by the LUSS, thus the LI re-writes the received packet (re-writes in case of the entire packet) or packet HEAD (appends the HEAD to the TAIL) into the internal or external memory (Step 8).

The LI gives the packet or packet HEAD to the dispatcher (Step 8). When the Dispatcher decides the packet is eligible to be sent out, it first inquires from (Step 9) the drop and stats block whether the packet needs to be dropped or not. If the answer is *yes*, all the resources and contexts associated to the packet will be freed. If the drop and stats engine determines that the packet should be forwarded to a remote PFE it notifies the Dispatcher, which then gives the lead to the cell compute block (Step 10). This block calculates the number of fabric cells required to forward the entire packet through the fabric and finally forwards the information to the fabric scheduler (Step 11).

Based on the fabric queue assigned by LUSS and the fabric CoS configuration inside the fabric scheduler, when the enqueued packet or packet HEAD is eligible to be sent out, the fabric scheduler retrieves the entire packet (remember LI has added back the HEAD after the LUSS processing) from the internal or external memory (Step 12) and sends the entire packet with additional internal information (provided by the LUSS) to the FO block (Step 13).

The fabric stream computation on ZT uses the same formula on the EA. Here is a quick formula reminder:

If the egress Local PFE ID is <= 3:

- Low Priority Queue Number = (Egress MPC slot Number x 4) + Egress Local PFE ID
- High Priority Queue Number = (Egress MPC slot Number x 4) + Egress Local PFE ID + 128

If the egress Local PFE ID is > 3:

- Low Priority Queue Number = (Egress MPC slot Number x 4) + (Egress Local PFE ID – 4) + 256
- High Priority Queue Number = (Egress MPC slot Number x 4) + (Egress Local PFE ID – 4) + 384

On the ZT you can also check traffic statistics for the high and low priority queues of a given egress PFE with the next command sequence. The example checks the two fabric queues used to reach the PFE 0 of the MPC in slot 9. We are attached to the old PFE shell (platformd) for these commands. The `36` means it is a destination MPC in slot 9 with the Local PFE ID 0. This is the low priority fabric queue to reach PFE 0 of MPC in slot 9:

```
(mx vty)# show mqss 0 sched-fab q-node stats 36

Queue statistics (Queue 0036)
-----------------------------

-------------------------------------------------------------------------------
Color  Outcome           Counter  Total                Rate
                         Name
-------------------------------------------------------------------------------
All    Forwarded (Rule)  Packets  8543761366           0 pps
                         Bytes    4377380549158        0 bps
All    TAIL drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
0      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
1      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
2      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
3      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
-------------------------------------------------------------------------------
```

Next, the 164 means it is a destination MPC in slot 9 with the Local PFE ID 0 plus the offset of 128. Here is the High priority fabric queue used to enqueue traffic targeting PFE 0 of MPC in slot 9:

```
(mx vty)# show mqss 0 sched-fab q-node stats 164

------------------------------------------------------------------------------
Color  Outcome           Counter  Total                Rate
                         Name
------------------------------------------------------------------------------
All    Forwarded (Rule)  Packets  1915035095           1016 pps
                         Bytes    49227552773          208832 bps
All    TAIL drops        Packets  178                  0 pps
                         Bytes    2848                 0 bps
0      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
1      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
2      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
3      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
------------------------------------------------------------------------------
```

Finally the FO splits the data (the entire packet plus a proprietary header appended to the packet that conveys additional internal information computed by the LUSS) in fixed-sized cells. Then, it sends a request over the fabric targeting the remote PFE to ask it if packets can be sent. The remote PFE acknowledges when it is ready to receive the packet by sending a grant message back to the source PFE. You can retrieve some FO statistics by issuing the next command, just have a look at the second tab - the `Counter Group 1`:

```
(mx vty)# show mqss 0 fo stats

FO statistics
-------------

Counter group 0
[…]
Counter group 1

------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------
Set Type   Mask  Match Total Requests      Requests per second  Total Grants       Grants per second
Total Cells        Cells per second
------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------
0   Port   0x3ff 0x0   13631861284632      200150       17772996817814 380284
13631861284634      200152
1   Port   0x3ff 0x1   13631861284686      200158       17772996817818 380284
13631861284686      200158
2   Port   0x3ff 0x2   13631861284688      200161       17772996817815 380281
13631861284688      200161
3   Port   0x3ff 0x3   13631861284690      200165       17772996817827 380283
13631861284693      200168
```

```
4  Port  0x3ff 0x4  13631861284692     200169      17772996817825 380294           13631861284694
200171
5  Port  0x3ff 0x5  13631861284698     200177      17772996817833 380294           13631861284698
200176
6  Port  0x3ff 0x6  13631861284696     200176      17772996817850 380299           13631861284697
200175
7  Port  0x3ff 0x7  13631861284702     200183      17772996817833 380298           13631861284702
200183
8  Port  0x3ff 0x8  13631861284705     200189      17772996817843 380306           13631861284705
200189
9  Port  0x3ff 0x9  13631861284707     200188      17772996817863 380306           13631861284707
200186
10 Port  0x3ff 0xa  13631861284704     200182      17772996817844 380310
13631861284704     200180
11 Port  0x3ff 0xb  13631861284703     200176      17772996817866 380309
13631861284703     200176
12 Port  0x3ff 0xc  13631861284699     200173      17772996817868 380312
13631861284699     200173
13 Port  0x3ff 0xd  13631861284694     200173      17772996817871 380315
13631861284694     200173
14 Port  0x3ff 0xe  13631861284693     200173      17772996817877 380326
13631861284693     200173
15 Port  0x3ff 0xf  13631861284694     200173      17772996817893 380329
13631861284694     200173
16 Port  0x3ff 0x10 13631861284696     200175      17772996817898 380325
13631861284696     200175
17 Port  0x3ff 0x11 13631861271421     200178      17772996804498 380319
13631861271421     200178
18 Port  0x3ff 0x12 13631861284695     200176      17772996817903 380324
13631861284695     200176
19 Port  0x3ff 0x13 13631861271809     200175      17772996804899 380314
13631861271809     200175
20 Port  0x3ff 0x14 13631861272284     200173      17772996805369 380311
13631861272284     200173
21 Port  0x3ff 0x15 13631861284693     200171      17772996817911 380299
13631861284693     200171
22 Port  0x3ff 0x16 13631861284693     200167      17772996817916 380292
13631861284693     200167
23 Port  0x3ff 0x17 13631861284695     200165      17772996817921 380293
13631861284696     200166

Total              327164670794069    4804146     42655192358985 9127307          327164670794078
4804148
```

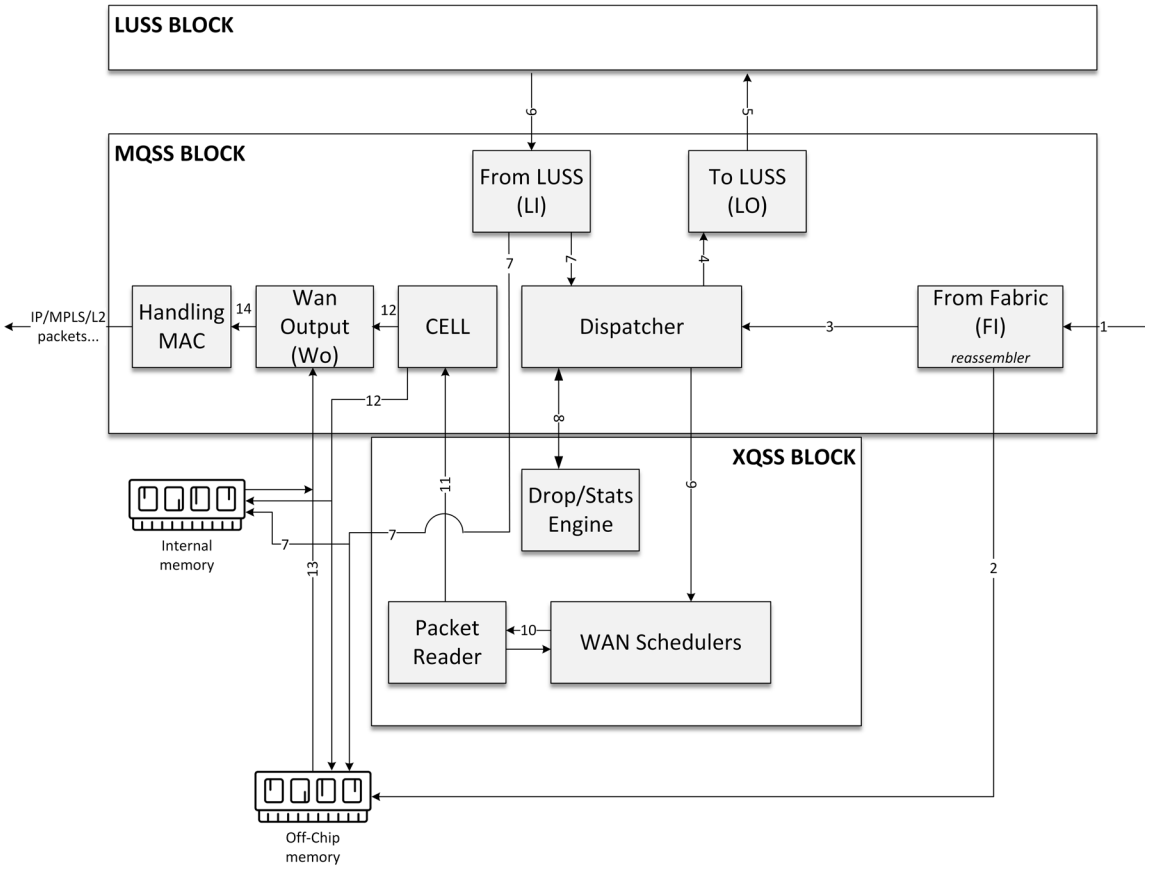
Remember, in our case we're using a MX2020 with 24 (0 to 23) planes, this is why the FO statistics show, for the 24 planes, the number of Request/Grant sent/received per plane. Like on the EA, the port number is not a direct mapping to the plane number so you should issue the next command. It requires the FPC slot number and the local PFE ID. In this next case for the FO statistics of FPC 18 PFE 1, the line prefix with 2 Ports provides statistics of Plane 14, the line with 4 Port with the statistics of Plane 12 , and so on:

```
mx2020-fpc18:pfe> show fabric fo-plane-map fpc 18 pfe 1
FO Port : Plane Number
      2 : 14
      4 : 12
      5 : 13
      8 : 20
[…]
```

Egress Packet Flow

So the packet has crossed the fabric planes and has reached the egress ZT PFE. Figure 2.20 zooms in on the egress PFE. You can see that this diagram is very close to the EA one. There is, however, internal memory only present on the ZT. You can also see that the XQSS is now involved on the egress side.

Figure 2.20 Inside the Egress ZT PFE



The packet enters from the fabric side into the FI block of the MQSS (Step 1). Some statistics can be collected from the ZT's FI block using the next command. The statistics give you the sum of all the FI streams, meaning all traffic coming from all source PFEs destined to this PFE. You'll see in Chapter 3 how to filter those statistics for a given source PFE.

```
(mx vty)# show mqss 0 fi stats

-------------------------------------------------------------------------------------------
Counter Name                                          Total                Rate
-------------------------------------------------------------------------------------------
Valid data cells received from FABIO                  273519730395         24207 cps
Valid fabric grants received from FABIO               121395399826         24201 grants/second
Valid fabric requests received from FABIO             273519730395         24207 requests/second
Valid fabric requests sent to FO                      273519730395         24207 requests/second
Received cells in input block for enabled streams     273519689042         24207 cps
Received packets in input block for enabled streams   70017562934          24073 pps
Cells dropped in FI memory interface                  5                    0 cps
Packets dropped in FI memory interface                1                    0 pps
Packets sent out of PSV                               70017562949          24073 pps
Error packets sent out of PSV                         0                    0 pps
-------------------------------------------------------------------------------------------
```

Remember, on the FI side each source PFE has two input streams: one high stream and one low stream corresponding to the high and low priority fabric queues on the ingress side. The way to calculate the FI streams of a given source PFE is the same as the formula used to calculate the ingress fabric queues.

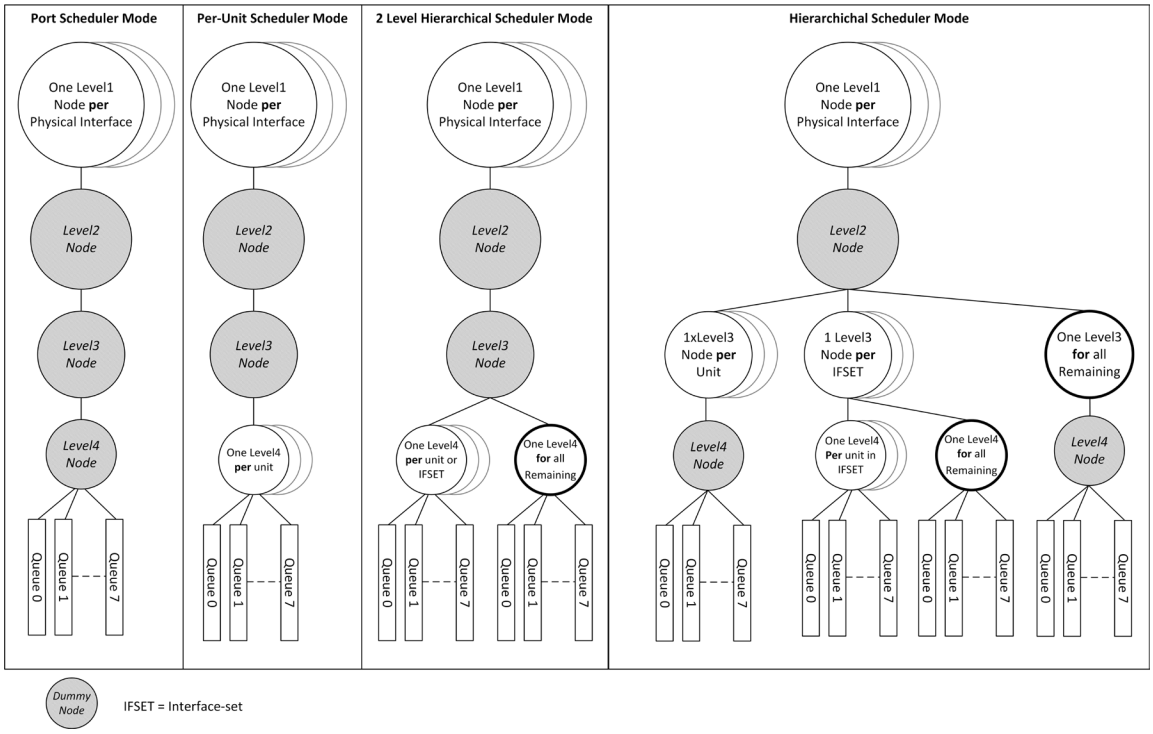
Once handled by the FI, the packet or packet HEAD is sent to the Dispatcher (Step 3) that, per ingress direction, creates a context for the packet and forwards it to one of the four LO blocks (Step 4). Steps 5 and 6 are almost the same as the ones described earlier when the packet was in the ingress PFE. The LUSS does several tasks such as identifying the WAN ouput queue to use based on the ingress classification. Then the packet moves back to the MQSS (Step 6). You can use similar commands presented for the ingress PFE to retrieve statistics between Dispatcher/DRD and LO (Step 4), between LO and the LUSS (Step 5), or LUSS and the LI block (Step 6):

- `show mqss <PFE-ID> drd stats`
- `show mqss <PFE-ID> lo stats`
- `show mqss <PFE-ID> li stats`

The LI block copies the modified packet or packet HEAD to internal or external memory and then gives it to the dispatcher who asks the drop and stats engine of the XQSS block if the packet must be dropped or not (Step 8). If not, the dispatcher sends the packet, when eligible to be sent out, to the XQSS block (Step 9).

In the XQSS, the packet HEAD is handled by the WAN schedulers block where it is enqueued in the right WAN queue assigned earlier by the LUSS. The XQSS supports port and hierarchical queuing. The ZT ASIC also supports the five levels of CoS model. Figure 2.21 reminds you how the CoS modes are modeled inside the ZT ASIC.

Figure 2.21 Supported ZT CoS Modes and Levels Mapping



As mentioned earlier, the packet is now enqueued in the XQSS. All the PFE CoS statistics have been moved to the new PFE shell (the `show cos halp` on platformd (uKernel) doesn't exist anymore). You should first find the IFD index of your egress physical port. For that issue the following command:

```
door7302@mx2020# show interfaces et-11/0/0 | match index
  Interface index: 237     <<<<< IFD Index
```

Now issue the following new PFE command on the MPC that hosts the physical interface to retrieve the AFT token ID attached to the CoS configuration of your given interface:

```
door7302@mx2020> start shell pfe network fpc11
root@mx2020-fpc11:pfe> show class-of-service interface scheduler hierarchy index 237
Interface Schedulers:
 Name               Type    Index    Level   Node-Token      Shaping-Rate
 et-11/0/0          IFD       237    1       34939           100000000000
```

As mentioned earlier, on the ZT line card all is modeled as a forwarding element identified by an AFT Token ID. This is also the case for the CoS parameters. To resolve a specific Token ID, use this specific command:

```
door7302@mx2020> start shell pfe network fpc11
root@ntdib999-fpc11:pfe> show sandbox token 34939
AftNode   : AftCosSched token:34939 group:0 nodeMask:0x1
DebugTxt:NA
ParentToken:NA
SchedMapToken:29976742
StreamToken:34930
MaxLevels:1
Level:0
Enable:1
SchedMode:Nominal
OverheadAcct:24
dbbTimeU:0.00 seconds
GlobalBaseId:0
SchedNodeType:2
Node Index:237
Node Name:et-11/0/0
Parent Name:
Interface Rate:100.00Gbps
DelayBufferRate:100.00Gbps

Table: CoS Scheduler AFT Node
-------------------------------------------
   Rate   |  Priority |   Rate  |  Burst  |
   Type   |   Group   |  (bps)  | Size (B}|
----------+-----------+---------+---------+
Guaranteed|    Nominal|   100.0G|    32.8K|
----------+-----------+---------+---------+
    Excess|    Nominal|   100.0G|      1.0|
    Excess| StrictHigh|   100.0G|      1.0|
    Excess|       High|   100.0G|      1.0|
    Excess|   Med High|   100.0G|      1.0|
    Excess|    Med Low|   100.0G|      1.0|
    Excess|        Low|   100.0G|      1.0|
----------+-----------+---------+---------+
   Maximum|    Nominal|   100.0G|     1.2G|
   Maximum| StrictHigh|      0.0|      0.0|
   Maximum|       High|      0.0|      0.0|
   Maximum|   Med High|      0.0|      0.0|
   Maximum|    Med Low|      0.0|      0.0|
   Maximum|        Low|      0.0|      0.0|
-------------------------------------------


JnhHandle : JnhHandleCosSchedNode Jnh:0x0 PfeInst:0 Paddr:0x0 Vaddr:0x0

CoS Scheduler Node:

      PFE Instance : 0
           L1 Index : 7
           L2 Index : 7
           L3 Index : 7
           L4 Index : 8
Enhanced Priority Mode : 0
Table: Queue Configuration
```

```
---------------------------------------------------------------------------------------------------
  Index | Shaping-Rate | Transmit-Rate |  Burst | Weight | G-Priority | E-Priority | Tail-Rule | WRED-Rule |
--------+--------------+---------------+--------+--------+------------+------------+-----------+-----------+
   64   |    100.0G    |      2.0G     | 1.2G   |    5   |     GL     |     EL     |   1278    |     0     |
   65   |    100.0G    |      6.0G     | 1.2G   |   15   |     GL     |     EL     |   1255    |     0     |
   66   |    100.0G    |     32.0G     | 1.2G   |   80   |     GL     |     EL     |   1255    |     0     |
   67   |    100.0G    |      0.0      | 1.2G   |    1   |     GH     |     EH     |   1087    |     0     |
   68   |    100.0G    |     50.0G     | 1.2G   |   50   |     GH     |     EH     |   1278    |     0     |
   69   |    100.0G    |     10.0G     | 1.2G   |   50   |     GM     |     EH     |   1087    |     0     |
   70   |    100.0G    |      0.0      | 1.2G   |    1   |     GL     |     EL     |   279     |     0     |
   71   |    100.0G    |      0.0      | 1.2G   |    1   |     GL     |     EL     |   279     |     0     |
--------+--------------+---------------+--------+--------+------------+------------+-----------+-----------+

Queue Statistics:

  PFE Instance : 0
                                   Transmitted                                                     Dropped
                         Bytes                          Packets                         Bytes                          Packets
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------
 Queue:0    370666928769(          0 bps)    704331542(          0 pps)           0(          0 bps)           0(          0
pps)
 Queue:1             0(          0 bps)            0(          0 pps)          0(          0 bps)          0(          0 pps)
 Queue:2             0(          0 bps)            0(          0 pps)          0(          0 bps)          0(          0 pps)
 Queue:3     2844662397(          0 bps)     11089454(          0 pps)           0(          0 bps)           0(          0
pps)
 Queue:4             0(          0 bps)            0(          0 pps)          0(          0 bps)          0(          0 pps)
 Queue:5      629934723(          0 bps)      6313366(          0 pps)           0(          0 bps)           0(          0
pps)
 Queue:6             0(          0 bps)            0(          0 pps)          0(          0 bps)          0(          0 pps)
 Queue:7             0(          0 bps)            0(          0 pps)          0(          0 bps)          0(          0 pps)
Previous nodes count: 3
Next nodes count    : 2
Entries count       : 0
```

You can also retrieve the per queue statistics of a given physical port at the PFE level by issuing this command:

```
door7302@mx2020> start shell pfe network fpc11
mx2020-fpc11:pfe> show class-of-service interface queue-stats index 237
Physical interface : et-11/0/0 (Interface index:  237, Egress queues:  8)
Queue: 0
        Queued Packets                :        704331691                 0  pps
        Queued Bytes                  :     370666941285                 0  bps
        Transmitted  Packet           :        704331691                 0  pps
        Transmitted Bytes             :     370666941285                 0  bps
        Tail-dropped Packets          :                0                 0  pps
        RL-dropped Packets            :                0                 0  pps
        RL-dropped Bytes              :                0                 0  bps
        RED-dropped Packets           :                0                 0  pps
          Low                         :                0                 0  pps
          Medium-low                  :                0                 0  pps
          Medium-high                 :                0                 0  pps
          High                        :                0                 0  pps
        RED-dropped Bytes             :                0                 0  bps
          Low                         :                0                 0  bps
          Medium-low                  :                0                 0  bps
          Medium-high                 :                0                 0  bps
```

```
          High                          :               0               0  bps
        Queue-depth bytes               :
          Average                       :               0
          Current                       :               0
          Peak                          :            1518
          Maximum                       :       316669952show mqss 0 wo stats

WO statistics
-------------
```

At this point there are several interactions between the WAN scheduler and what we called the *Packet Reader* that maintain the list of active streams (Step 10). To summarize, when the WAN scheduler decides the packet is eligible to be dequeued, the Packet Reader notifies the cell block (Step 11). It actually provides information to the cell block about the packet's chucks. The cell block sends Read requests to external or internal memory (Step 12) in order to fetch the content of the entire packet (HEAD and TAIL). External or internal memory returns the requested data payload to the WO block.

Finally the WO block sends the entire packet to the MAC block, which computes and then appends the CRC32 to the Ethernet frame before sending it out.

The WO statistics are available by using the next command. Just have a look at the `Counter set 0`, which gives you the WAN output aggregated statistics (all Ethernet ports attached to the EA ASIC). We'll see in the Chapter 3 how to use `Counter set 1` to filter the view on a given port:

```
door7302@mx2020> start shell pfe network fpc11.0
(mx vty)# show mqss 0 wo stats

WO statistics
-------------

Counter set 0

    Connection number mask   : 0x0
    Connection number match  : 0x0
    Transmitted packets      : 137212310395 (100177 pps)
    Transmitted bytes        : 113707212044499 (401766112 bps)
    Transmitted flits        : 1254813302266 (600325 flits/sec)

Counter set 1
[…]
```

This last command concludes Chapter 2. You need most of the commands presented here in Chapter 3. Indeed, Chapter 3 takes two specific use cases to illustrate how to troubleshoot the EA or the ZT PFEs.

Chapter 3

Follow the Packets

This chapter reviews and extends what was presented in Chapter 2. It uses two simple examples of following traffic to do that:

- The first is MPLS transit traffic.
- The second is a simple OAM traffic: our classic friend, a ping echo/reply.

During this deep dive we refer to the Local PFE ID/index and the Global PFE ID/index. As mentioned previously, the Local PFE ID is the local index assigned, on a given MPC, to the PFEs. This Local PFE ID is a number between 0 and 7 depending on the number of PFEs in the MPC.

The global PFE ID/index is computed as:

- MPC slot number * 4 + Local PFE ID (when Local ID is <= 3)
- 80 + MPC slot number * 4 + Local PFE ID (when Local ID is > 3)

For example, our MPC11e is installed in slot 11 with eight PFEs:

- The Local PFE ID are assigned from 0 to 7
- The Global PFE ID for local PFEs 0 to 3 are: 44 (Slot 11 * 4 + Local ID (0)), 45, 46, and 47
- The Global PFE ID for local PFEs 4 to 7 are: 124 (80 + Slot 11 * 4 + Local ID (0)), 125, 126, and 127

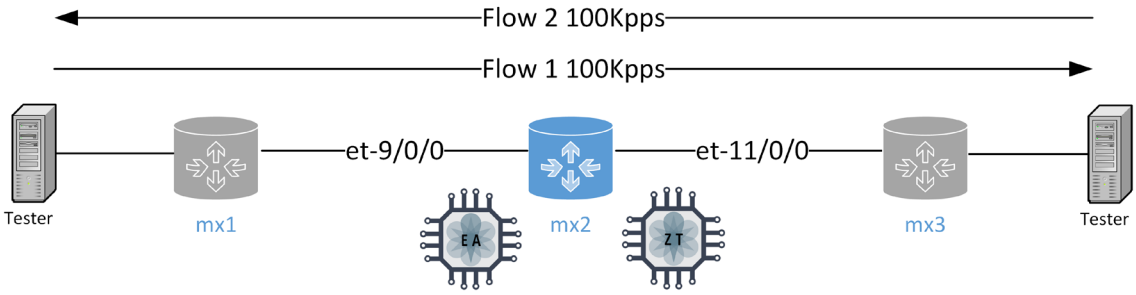
One last point to mention: just remember that when we issue the `start shell pfe network fpcX` command for the ZT ASIC we will be attached to the new PFE shell, the one managed by MGD. When we issue the other `start shell pfe network fpcX.0` command we will be connecting to the old PFE shell, managed by platformd also known as uKernel on the EA ASIC.

Okay, now we're ready to begin. Let's start with following MPLS traffic.

MPLS in Transit in the EA and ZT

Figure 3.1 illustrates the network topology we'll use throughout this section. As observed, it's a very simple topology with three routers involved: mx1, mx2, mx3. We will focus on mx2, a MX2020, with two line cards, one a MPC9e and one a MPC11e. The physical port et-9/0/0 is attached to MPC9e in slot 9 and the et-11/0/0 to the MPC11e in slot 11. Junos 20.1 is on all routers.

Figure 3.1 Simple MPLS Network Topology



Both interfaces have ISIS and MPLS LDP protocols enabled.

A question you may have is: Why LDP and not SR MPLS?

We preferred using LDP in order to have a SWAP of labels with two distinct values. Usually the SWAP still occurs with SR MPLS and global range, but with the same label value. It will be easier to check where the SWAP occurs inside the PFEs with LDP (but it would be exactly the same with RSVP or SR MPLS).

There are two unidirectional streams configured. Each stream has a fixed throughput: 100Kpps. The first one enters in an EA ASIC (MPC9e) and is forwarded to a ZT ASIC (MPC11e). The second one is the reverse: coming from the ZT and destined to the EA ASIC. At each step we'll track both streams to show you PFE commands on both ASICs. Sometimes the command will be the same as was seen in Chapter 2, and sometimes not.

First, let's go inside the mpls.0 routing table to track pure MPLS transit flows:

```
door7302@mx2020> show route table mpls.0
mpls.0: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

2223               *[LDP/9] 00:00:46, metric 1
                    >  to 192.168.1.2 via et-9/0/0.0, Swap 339376
2221               *[LDP/9] 00:00:15, metric 1
                    >  to 192.168.2.2 via et-11/0/0.0, Swap 299792
```

As shown above, this is as simple as it gets! There are only two entries: one for each stream. The flow 1 labeled with 2221 is received by et-9/0/0.0. The label is swapped to 299792 and forwarded to et-11/0/0.0. The flow 2 labeled with 2223 is received by et-11/0/0.0 and forwarded to et-9/0/0.0 after the label is swapped for 339376. You can use the same command except with the `detail` option and have a look at the next-hop index to retrieve the value, which we will need later:

```
door7302@mx2020> show route table mpls.0 detail | match "entry|index"
2223 (1 entry, 1 announced)
                Next hop type: Router, Next hop index: 543
2221 (1 entry, 1 announced)
                Next hop type: Router, Next hop index: 550
```

The MPLS forwarding table gives us the same information:

```
door7302@mx2020> show route forwarding-table table default family mpls
Routing table: default.mpls
MPLS:
Destination   Type RtRef Next hop           Type Index    NhRef Netif
2223          user     0 192.168.1.2       Swap 339376     543     2 et-9/0/0.0
2221          user     0 192.168.2.2       Swap 299792     550     2 et-11/0/0.0
```

Keep in mind this LFIB view is the LFIB from the point of view of the RE, more precisely, from the RE RPD's module named KRT(*KRT* is the module within RPD which is the interface to the Kernel). This means it is the LFIB's view before pushing it to the PFE. You can also retrieve similar information by using the hidden CLI `KRT` command, which gives you the translation between a next-hop ID and its related data: `show krt next-hop <NH-TYPE> index <NH-INDEX>`. The NH-TYPE is the "*router*" whose information was provided by the previous `show route table mpls.0 detail`:

```
door7302@mx2020> show krt next-hop router index 543
        Next-hop type: Router
        Index: 543
        Address: 0x5214f1c
        Reference count 2
        Kernel Table Id 0
                Next hop: 192.168.1.2 via et-9/0/0.0
                Label operation: Swap 339376
                Load balance label: Label 339376: None;
                Label element ptr: 0x4fc2b40
                Label parent element ptr: 0x0
                Label element references: 1
                Label element child references: 0
                Label element lsp id: 0
                Session Id: 0x176c
        Flags: explicit-add on-nhid-tree
```

The information is pushed via IPC messages to the MPCs, where they are converted into FIB states and finally installed into ASIC's memory. As mentioned earlier, you can use the RE shell command `rtsockmon` to simulate an MPC connection and see the RIB to FIB activity. Let's do it and simulate a network event: a simple flap of et-9/0/0:

```
door7302@mx2020> start shell
% su
Password:
mx2020:/var/home/lab # rtsockmon –t
[…]
nexthop   delete  inet addr=192.168.1.2 nh=ucst flags=0x85 uflags=0x0 idx=543 ifidx=396 filteridx=0
tid=0 lr_id=0   infotype = 0 fwdnhidx = 0 fwdnhtype = 0 nh_adders=0x0000000000000000 subnh=542 MPLS
Data: opcode=5, flags=0x0, selfid=805311035, parentid=0, mtu=0, lbl_count=1, policer_count=0
label[0]: label=339376, proto=3, lb_attr=None
nexthop   add    inet addr=192.168.1.2 nh=ucst flags=0x85 uflags=0x0 idx=543 ifidx=396 filteridx=0
tid=0 lr_id=0   infotype = 0 fwdnhidx = 0 fwdnhtype = 0 subnh=542 MPLS Data: opcode=5, flags=0x0,
selfid=805311037, parentid=0, mtu=0, lbl_count=1, policer_count=0 label[0]: label=339376, proto=3,
lb_attr=None
[…]
```

You can see that next-hop ID 543 was first deleted when the interface et-9/0/0 went down and then added back when the link came back up. You can see all the KRT information pushed to MPCs due to network events.

Don't forget to use this awesome command that can help you in case of ECMP or LAG to find the right forwarding interface. Based on well-known IP stack fields, or a packet hexa dump, the next command specifies which output interface will be used to forward a given stream (the `jsim` PFE command is no longer necessary to achieve that):

```
door7302@mx2020> show forwarding-options load-balance ?
Possible completions:
  destination-address  Destination IP address
  destination-port     Destination port
  family               Layer 3 family
  ingress-interface    Ingress Logical Interface
  packet-dump          Raw packet dump in hex without '0x'
  source-address       Source IP address
  source-port          Source port
  tos                  Type of Service field
  transport-protocol   Transport layer protocol
```

Now it's time to go inside the PFE!

The aim is to track both flow (flow 1 and flow 2) inside the EA and ZT. At each step we'll try to provide the PFE commands used on both ASICs. Remember, flow 1 comes in from mx2 with the label 2221 and is handled by EA ASIC, while flow 2 comes in from mx2 with label 2223 and is handled by a ZT ASIC.

MPLS Packet on the Ingress PFE

Before tracking the MPLS transit packets let's first check some routing stuff at the PFE level, meaning the MPLS FIB, this time with the PFE point of view. On the EA ASIC you can use this next PFE command, which is very similar to what's in classic CLI operational mode:

```
(mx vty)# show route mpls table default
MPLS Route Table 0, default.0, 0x800c8:
Destination    Type   ID    NhRef
-------------  -----  ----- -----
default   Discard    50     1
2221      Unicast   550     1 et-11/0/0.0
2223      Unicast   543     1 et-9/0/0.0
```

On the ZT the equivalent command is:

```
door7302@mx2020> start shell pfe network fpc11
mx2020-fpc11:pfe> show route proto mpls name default

Index Destination                      NH Id     NH Type   NH Token
----- -------------------------------- --------- --------- ---------
0     default                          50        Discard   1477
0     2221                             550       Unicast   1453283
0     2223                             543       Unicast   1453327
```

Take a few moments to ponder the power to have MGD at the PFE level on ZT-based line cards.

You can actually display the PFE command outputs in JSON or XML like we did, for a long time, at the CLI level:

```
mx2020-fpc11:pfe> show route proto mpls name default | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/20.1I0/junos">
    <route-entry-information junos:style="brief" >
[…]
          <entries>
            <proto>mpls</proto>
            <index>0</index>
            <prefix></prefix>
            <destination>2221</destination>
            <length>20</length>
            <nexthop>550</nexthop>
            <nexthop-type>Unicast</nexthop-type>
            <nhToken>1453283</nhToken>
          </entries>
[…]
    </route-entry-information>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>
```

So cool, isn't it?

Now, if you want more detail about a specific next hop installed in PFE on the EA line card, we can issue the next command. Suppose you are attached to the ingress PFE where the packets come in. Therefore, on the EA line card (MPC in slot 9) you want to check in detail the entry about the next-hop index 550. This one is attached to the forwarding entry with the label value equal to 2221. Remember, this is flow 1 (in Figure 3.1), which is received by the MPC in slot 9 and forwarded to MPC in slot 11.

Let's issue the command for next-hop index 550 (the output has been truncated for the sake of this book):

```
 (mx vty)# show nhdb id 550 extensive
   ID     Type     Interface    Next Hop Addr    Protocol      Encap    MTU            Flags  PFE
internal Flags
------  --------  -------------  ---------------  ----------  ------------  ----    550   Unicast
et-11/0/0.0    -                  MPLS     Ethernet  1488
[…]
Topo-link:
[pfe-0]: 0x11c000000002262c
   ModifyNH: Subcode=SetNH-Token(7),Desc=0x0,Data=0x2262c,NextNH=0
 (pfeDest:44, TokenIdMode:0/  , VC memberId:0, token:0x226/550)
[…]
```

What can we see? The egress physical port et-11/0/0, which is a port that is not attached to local PFE. It means that packets entering with label 2221 should be forwarded through the fabric to another PFE. You can see the topology table gives us the selected egress PFE. In our case the PFE destination (aka pfeDest) is 44 (11*4+0). This is the global PFE index: the egress port is attached to PFE 0 of MPC in slot 11.

There is a similar PFE command on the ZT. Next, let's check the detail about next-hop index 543 attached to incoming label 2223:

```
door7302@mx2020> start shell pfe network fpc11
mx2020-fpc11:pfe> show nh detail index 543
Nexthop Info:

NH Index                    : 543
NH Type                     : Unicast
NH Proto                    : tag
NH Flags                    : 0x1
IF Name                     : et-9/0/0.0
Prefix                      : 789120
NH Token Id                 : 1453327
NH Route Table Id           : 0
Sgid                        : 0
[…]
Platform Info
-------------
  FabricToken: 18446744073709551615
  EgressToken: 18446744073709551615
  IngressFeatures:
```

```
Container token: 1453327
#1 SetNhToken tokens:
 Mask : 0x0
 [  SetNhToken:1453326  ]
[…]
```

You need a second command to display the content of a specific AFT token. In our example, we want to display the information about the `SetNhToken 1453326`. So use the helpful command here:

```
door7302@mx2020> start shell pfe network fpc11
mx2020-fpc11:pfe> show sandbox token 1453326
AftNode   : AftTrioFwd token:1453326 group:0 nodeMask:0xffffffffffffffff nextHopId:543 coreNumber:36
nexthopIdMode:0 linkIdValid:0 childLinkId:0 isDefaultLink:0 cwPresent:0
JnhHandle : JnhHandleFwd Jnh:0x11c0000000021f24 PfeInst:ff Paddr:0x0 Vaddr:0x0 AppType:NH
JNH decode:
ModifyNH: Subcode=SetNH-Token(7),Desc=0x0,Data=0x21f24,NextNH=0
 (pfeDest:36, TokenIdMode:0/  , VC memberId:0, token:0x21f/543)
[…]
```

Now you can see the selected egress PFE (pfeDest), which is for label 2223 the global PFE index 36 (actually this is local PFE 0 of MPC in slot 9: 9*4+0).

Great! We have checked the LFIB consistency on each ingress MPC and for both flows. Now, let's track the packets themselves.

In Chapter 2 we saw that incoming packets are first handled by the WI block (we didn't check their pre-classification engine stats). As already mentioned, by default WI statistics shows aggregated stats – the input traffic sum of all physical ports attached to the WI. On the EA line card the command is:

```
(mx vty)# show mqss 0 wi stats
WI statistics
-------------
[...]
Total incoming statistics
-------------------------
----------------------------------------------------------------
Counter Name      Total                 Rate
----------------------------------------------------------------
Received Packets  23682709482           500044 pps
Received Bytes    12027207901532        2008145200 bps
Flushed Packets   0                     0 pps
----------------------------------------------------------------
[...]
Tracked stream statistics
-------------------------
-------------------------------------------------------------------------------------------------------------------
-----------------------------------------
Track Stream Stream Total Packets       Packets Rate       Total Bytes        Bytes Rate         Total
EOPE         EOPE Rate
     Mask   Match                  (pps)                              (bps)           (pps)
-------------------------------------------------------------------------------------------------------------------
-----------------------------------------
0    0xff   0x8f   14535685879        500037            7413198556434       408151104          0
0
```

```
[...]
47   0x0   0x0   23682709462        500045            12027207891332      408149280          0
0
```

And for ZT:

```
door7302@mx2020> start shell pfe network fpc11.0
(mx vty)# show mqss 0 wi  stats

WI statistics
-------------
[...]
Total incoming statistics
-------------------------
----------------------------------------------------------------
Counter Name      Total                 Rate
----------------------------------------------------------------
Received Packets  117422462371          299977 pps
Received Bytes    59950705219198        1211095920 bps
Flushed Packets   0                     0 pps
----------------------------------------------------------------
[...]
Tracked stream statistics
-------------------------
------------------------------------------------------------------------------------------------------------------
-------------------------------------------
Track Stream Stream Total Packets       Packets Rate       Total Bytes        Bytes Rate         Total
EOPE         EOPE Rate
     Mask   Match                   (pps)                                 (bps)     (pps)
------------------------------------------------------------------------------------------------------------------
-------------------------------------------
0    0x0   0x0   117422462332       299975             59950705199152      411087704          0
0
[...]
49   0x0   0x0   117422462356       299975             59950705211488      411087704          0
0
```

If you have a look at `Total Incoming Statistics`, you should observe the total packet received by the WI block. At the end of the output you can see a specific table called `Tracked Stream Statistics`. These specific counters (47 for EA, and 49 for ZT) are configurable. By default they also count all packets. It is possible to temporarily configure a specific counter to count packets received on a given port attached to the WI block of a given PFE.

To do that you should first find the incoming stream number attached to your ingress interface, which conveys the traffic. For that, issue this command on the EA (the last `0` at the end means ingress direction):

```
(mx vty)# show mqss 0 ifd list 0
Ingress IFD list
----------------
------------------------------------------------------------------------
IFD name           IFD index    PHY stream    LUSS SID    Traffic Class
------------------------------------------------------------------------
et-9/0/0           439          1165          560         0 (High)
et-9/0/0           439          1166          560         1 (Medium)
```

```
et-9/0/0           439          1167          560          2 (Low)
et-9/0/3           455          1213          764          0 (High)
et-9/0/3           455          1214          764          1 (Medium)
et-9/0/3           455          1215          764          2 (Low)
et-9/0/0           439          1278          560          3 (Drop)
et-9/0/3           455          1278          764          3 (Drop)
------------------------------------------------------------------------
```

And on ZT the command is the same as before:

```
door7302@mx2020> start shell pfe network fpc11.0
(mx vty)# show mqss 0 ifd list 0
Ingress IFD list
----------------
------------------------------------------------------------------------
IFD name           IFD index    PHY stream    LUSS SID    Traffic Class
------------------------------------------------------------------------
et-11/0/0          535          1165          560          0 (High)
et-11/0/0          535          1166          560          1 (Medium)
et-11/0/0          535          1167          560          2 (Low)
et-11/0/1          536          1197          696          0 (High)
et-11/0/1          536          1198          696          1 (Medium)
et-11/0/1          536          1199          696          2 (Low)
et-11/0/2          537          1213          764          0 (High)
et-11/0/2          537          1214          764          1 (Medium)
et-11/0/2          537          1215          764          2 (Low)
et-11/0/3          538          1229          832          0 (High)
et-11/0/3          538          1230          832          1 (Medium)
et-11/0/3          538          1231          832          2 (Low)
et-11/0/4          539          1245          900          0 (High)
et-11/0/4          539          1246          900          1 (Medium)
et-11/0/4          539          1247          900          2 (Low)
et-11/0/3          538          1275          832          3 (Drop)
et-11/0/4          539          1275          900          3 (Drop)
et-11/0/1          536          1276          696          3 (Drop)
et-11/0/2          537          1276          764          3 (Drop)
et-11/0/0          535          1277          560          3 (Drop)
```

Remember, only low and medium streams are used. The pre-classifier of the EA or ZT has assigned the stream low to our MPLS flows because they have not been identified as control plane or OAM traffic but rather as transit traffic. Therefore the WI stream ID is 1167 for both incoming interfaces.

Now to configure the WI counter. For EA use the following command:

```
(mx vty)# test mqss 0 wi stats stream 0 0 143
```

What does `0 0 143` mean?

The first 0 means WAN port group 0 – it's always 0. The next 0 is the counter ID. As noted, on the EA you have 48 counters available (0 to 47) and for ZT you have 50 counters (0 to 49). Here we use the counter index 0 for our stats. Finally, the last number is the incoming stream connection. This value is derived from the incoming stream number (retrieved previously – for us it was 1167) from which we subtract 1024 (1167-1024 = 143). Then we can again issue the previous `show mqss 0 wi stats` command to display the `Tracked stream statistics` table:

```
(mx vty)# show mqss 0 wi stats
[...]

Tracked stream statistics
-------------------------
-------------------------------------------------------------------------------------------
Track Stream Stream Total Packets Packets Rate         Total Bytes          Bytes Rate
      Mask   Match                pps)                                     (bps)
-------------------------------------------------------------------------------------------
0     0xff   0x8f   951165       100009               485093702            408033136
```

We refind our 100K pps, which comes into et-9/0/0. You can clear the counter config by issuing this last command:

```
(mx vty)# test mqss 0 wi stats default 0 0
```

Let's do the same on ZT. Actually, it's exactly the same set of commands:

```
test mqss 0 wi stats stream 0 0 143
show mqss 0 wi stats
[...]
Tracked stream statistics
-------------------------
-------------------------------------------------------------------------
Track Stream Stream Total Packets        Packets Rate         Total Bytes
      Mask   Match                       (pps)
-------------------------------------------------------------------------
0     0xff   0x8f   498787               100037               256376518
test mqss 0 wi stats default 0 0
```

Our two flows are handled well by the WI block. The next step you can do is capture a transit packet before and after its processing by the LUSS block. Some cool PFE commands allow us to capture a packet (the packet HEAD) inside a PFE. If you want to focus on a specific type of traffic you have the ability to specify a hexadecimal pattern of eight bytes maximum in length. This specific hexadecimal pattern is used as a packet filtering condition: it means PFE should find this pattern inside the packet to capture it. Eight bytes is small but usually enough to identify a specific flow. You can use these eight bytes to filter:

- A specific Ether type
- A specific MPLS label
- An IPv4 address
- A couple of IPv4 source/destination addresses
- A part of an IPv6 address
- A source or destination MAC address

It is important to conclude that research is done by default on the 32 first bytes of the packet (internal packet header included). If you want to research beyond that, you should provide an offset to look further into the packet: usually if you can't capture your packet you can try using the `offset` option to analyze beyond the 32 first bytes: later we'll see some examples where offset is required.

What if you want to filter on the ingress side?

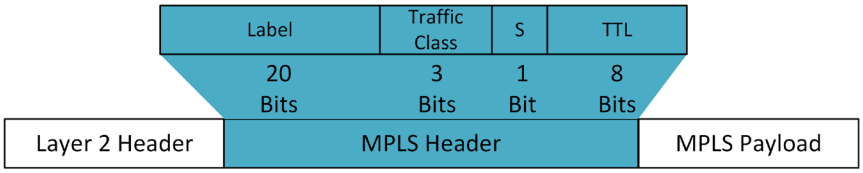
In this case capture packets:

On the EA – MPLS packet (EtherType 0x8847) with label 2221 (this is flow 1)

On the ZT - MPLS packet (EtherType 0x8847) with label 2223 (this is flow 2)

Let's have a quick look at the MPLS header in Figure 3.2.

Figure 3.2 MPLS Header



The label is encoded in 20 bits followed by the traffic class field (3 bits) and the S bit. Therefore, for our cases both flows are best effort traffic, which means for us the traffic class field is set to 0. The S bit is set to 1 for labels 2221 and 2223, as they are the last labels of the stack. The TTL field is variable; we don't want to filter on it. Finally, we will filter on the first 24 bits of the MPLS header plus the EtherType. For flow 1 it will give us the following hexadecimal pattern:

```
0x8847 + 008ad hex(2221)  + 000 ( TC) + 1 (S) = 0x8847008ad1
```

And for flow 2:

```
0x8847 + 008af hex(2223)  + 000 ( TC) + 1 (S) = 0x8847008af1
```

Now we're ready to capture our ingress packet. Let's start on the EA ASIC. First of all, enable packet capture on the right PFE (`jnh 0` means `PFE 0`):

```
(mx vty)# test jnh 0 packet-via-dmem enable
```

Next, capture packets that match our hexadecimal pattern 8847008ad1. To do so, always set the 0x3 option, which means capturing all types (packet and packet HEAD). We don't need the offset option – if needed replace `<Offset>` by a value such as 32:

```
(mx vty)# test jnh 0 packet-via-dmem capture 0x3 8847008ad1 <Offset>
```

Wait a little and then stop the capture by using the 0x0 option:

```
(mx vty)# test jnh 0 packet-via-dmem capture 0x0
```

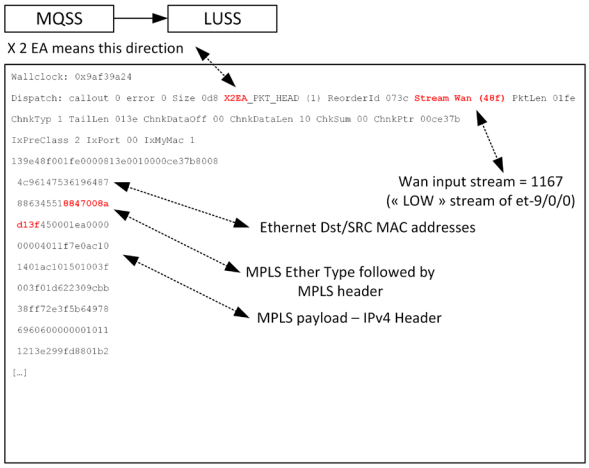
Finally, dump the capture's buffer. Don't be scared about the amount of data provided – the buffer size is by default quite large and allows capturing a bunch of packets:

```
(mx vty)# test jnh 0 packet-via-dmem decode
Wallclock: 0x9af39a24
Dispatch: callout 0 error 0 Size 0d8 X2EA_PKT_HEAD (1) ReorderId 073c Stream Wan (48f) PktLen 01fe
ChnkTyp 1 TailLen 013e ChnkDataOff 00 ChnkDataLen 10 ChkSum 00 ChnkPtr 00ce37b
IxPreClass 2 IxPort 00 IxMyMac 1
139e48f001fe0000813e0010000ce37b8008
 […]
Wallclock: 0x9af3a190
Reorder: EA2X_REORDER_SEND_TERMINATE (d) HasTail 1 Stat 0
ReorderId 073c Color 0 Qop ENQUEUE (2) Qsys FAB (1) Queue 0002c
Frag 0 RefCnt 0 OptLbk 0 NumTailHoles 00
ChnkTyp 1 TailLen 013e ChnkDataOff 00 ChnkDataLen 10 ChkSum 00 ChnkPtr 00ce37b
PType MPLS (4) SubType 2 PfeMaskF 0 OrigLabel 1 SkipSvc 0 IIF 0018c
FC 00 DP 0 SkipSample 0 SkipPM 0 L2Type None (0) IsMcast 0 SrcChas 1 MemId 0
PfeNum 2c PfeMaskE 0 FabHdrVer 1
RwBuf 00000000 SvcCtx 0 ExtHdr 0 PfeMaskE2 0 Token 000226
d873c050002c000005b0d00f0600d7d0d9044000813e0000813e0010000ce37b4220018c0001058100000000000000226
 […]
```

For each captured packet you have two dumps. Dumps with the same "ReorderId" are part of the same capture.

The first dump gives you the packet information before LUSS processing, meaning when the MQSS sends the packet to the LUSS. The second dump gives you the packet information after it was processed by the LUSS – actually when it comes back in the MQSS. Figure 3.3 helps us to illustrate these very useful dumps

Figure 3.3 How to Decode Ingress Packet Capture on the EA



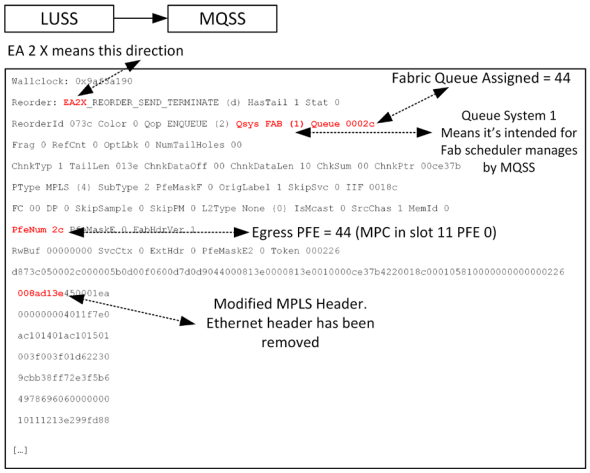


Figure 3.3 shows you that there is so much relevant pieces of information provided by the packet capture, such as:

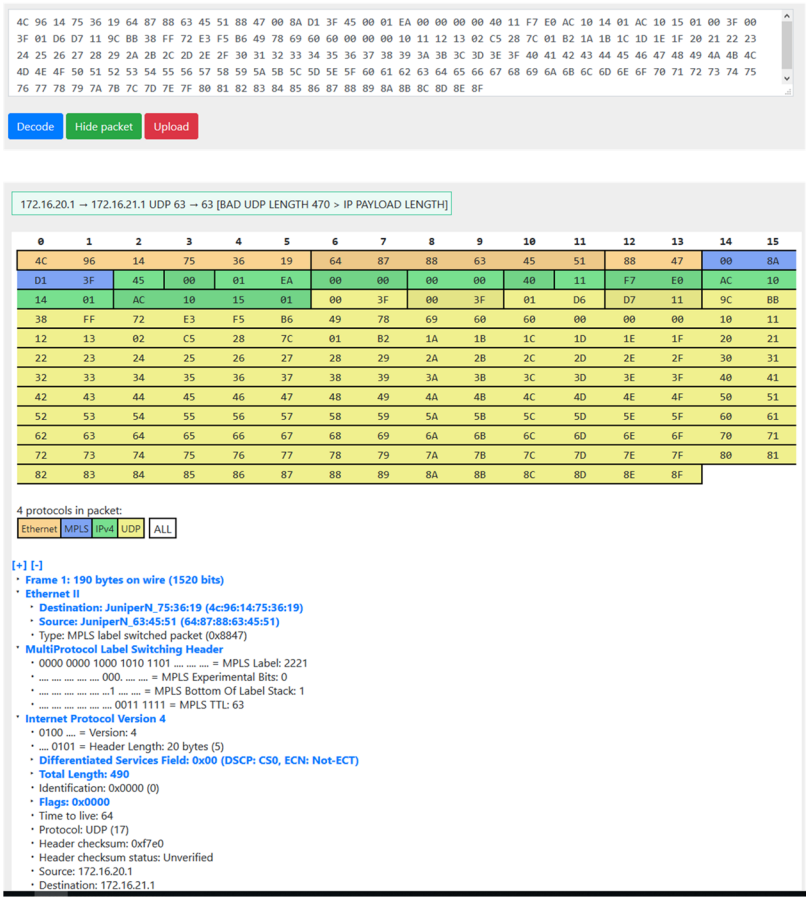
- Ethernet packet header plus its payload before processing.
- Packet header plus its payload after processing. We can see that the Ethernet header has been removed by the ingress LUSS – the MPLS TTL has been decremented (TTL 3f became 3e) but the label has not yet been swapped (assuming this action will be done on the egress side – we'll see this later).
- We also discovered which fabric queue has been assigned and which egress PFE the lookup has selected. This is for flow 1 the Fabric queue 44 – low priority queue to reach global PFE ID 44 (PFE 0 hosted by MPC in slot 11).

At the end of your troubleshooting, don't forget to disable packet capture:

```
(mx vty)# test jnh 0 packet-via-dmem disable
```

If needed, you can copy the hexadecimal string of the captured packet and paste it on site such as: https://hpd.gasmi.net/. This helps you to decode packets like tcpdump or wireshark. See Figure 3.4:

Figure 3.4 Decode Transit Packets



Let's do the same on ZT. Enable packet capture on the right PFE (inst 0 means PFE 0):

```
door7302@mx2020> start shell pfe network fpc11
mx2020-fpc11:pfe> test jnh packet-via-dmem inst 0 enable
```

Then capture packets that match our hexadecimal pattern 8847008af1. Always set the `0x3` option, which means capturing all types (packet and packet HEAD). We don't need the offset option :

```
mx2020-fpc11:pfe> test jnh packet-via-dmem-capture inst 0 parcel-type-mask 0x3 match-string 8847008af1
offset <Offset>
```

Wait a little and then stop the capture:

```
mx2020-fpc11:pfe> test jnh packet-via-dmem-capture inst 0 parcel-type-mask 0x0
```

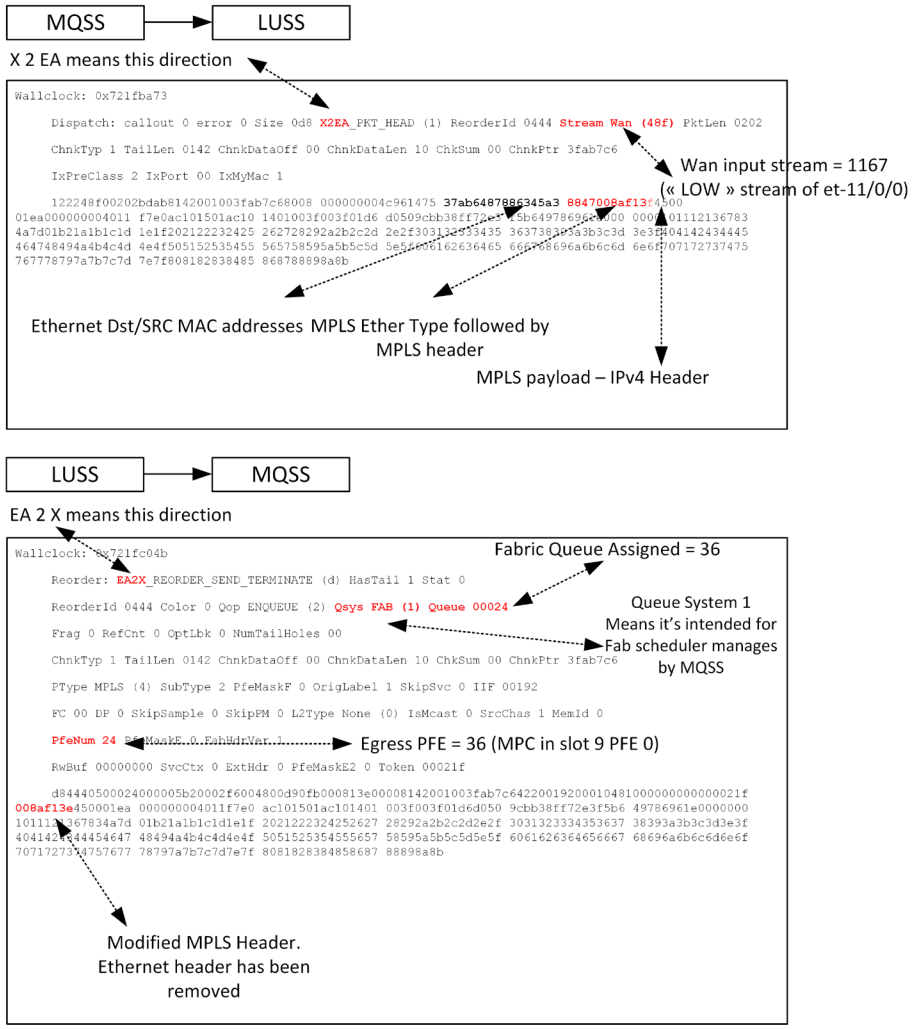
And finally, dump the buffer of capture. Once again, don't be scared about the amount of data displayed:

```
mx2020-fpc11:pfe> test jnh packet-via-dmem-dump inst 0
     Wallclock: 0x721fba73
     Dispatch: callout 0 error 0 Size 0d8 X2EA_PKT_
HEAD (1) ReorderId 0444 Stream Wan (48f) PktLen 0202
     ChnkTyp 1 TailLen 0142 ChnkDataOff 00 ChnkDataLen 10 ChkSum 00 ChnkPtr 3fab7c6
     IxPreClass 2 IxPort 00 IxMyMac 1
    122248f00202bdab8142001003fab7c68008 000000004c961475 37ab6487886345a3 8847008af13f4500
01ea000000004011 f7e0ac101501ac10 1401003f003f01d6 d0509cbb38ff72e3 f5b649786961e000 0000101112136783
4a7d01b21a1b1c1d 1e1f202122232425 262728292a2b2c2d 2e2f303132333435 363738393a3b3c3d 3e3f404142434445
464748494a4b4c4d 4e4f505152535455 565758595a5b5c5d 5e5f606162636465 666768696a6b6c6d 6e6f707172737475
767778797a7b7c7d 7e7f808182838485 868788898a8b
     Wallclock: 0x721fc04b
     Reorder: EA2X_REORDER_SEND_TERMINATE (d) HasTail 1 Stat 0
     ReorderId 0444 Color 0 Qop ENQUEUE (2) Qsys FAB (1) Queue 00024
     Frag 0 RefCnt 0 OptLbk 0 NumTailHoles 00
     ChnkTyp 1 TailLen 0142 ChnkDataOff 00 ChnkDataLen 10 ChkSum 00 ChnkPtr 3fab7c6
     PType MPLS (4) SubType 2 PfeMaskF 0 OrigLabel 1 SkipSvc 0 IIF 00192
     FC 00 DP 0 SkipSample 0 SkipPM 0 L2Type None (0) IsMcast 0 SrcChas 1 MemId 0
     PfeNum 24 PfeMaskE 0 FabHdrVer 1
     RwBuf 00000000 SvcCtx 0 ExtHdr 0 PfeMaskE2 0 Token 00021f
    d84440500024000005b20002f6004800d90fb000813e00008142001003fab7c64220019200010481000000000000021f
008af13e450001ea 000000004011f7e0 ac101501ac101401 003f003f01d6d050 9cbb38ff72e3f5b6 49786961e0000000
1011121367834a7d 01b21a1b1c1d1e1f 2021222324252627 28292a2b2c2d2e2f 3031323334353637 38393a3b3c3d3e3f
4041424344454647 48494a4b4c4d4e4f 5051525354555657 58595a5b5c5d5e5f 6061626364656667 68696a6b6c6d6e6f
7071727374757677 78797a7b7c7d7e7f 8081828384858687 88898a8b
```

Also on the ZT, for each captured packet you have two dumps: one before processing (MQSS to LUSS) and the second after processing (LUSS to MQSS). Figure 3.5 helps illustrate this tricky new output. As for the EA, the dump on the ZT shows you similar information such as:

- Ethernet packet header plus its payload before processing.
- Packet header plus its payload after processing.
- We also retrieved which fabric queue has been assigned and which egress PFE the lookup has selected. This is for flow 2 the Fabric queue 36 – low priority queue to reach global PFE ID 36 (PFE 0 hosted by MPC in slot 9).

Figure 3.5 How to Decode Ingress Packet Capture On ZT



At the end of your troubleshooting on ZT, don't forget to disable packet capture:

```
mx2020-fpc11:pfe> test jnh packet-via-dmem inst 0 disable
```

Now we know how the packet has been manipulated by the ingress PFE and we also know which fabric queue has been assigned for each flow. Remember you can deduce the fabric queue used based on the outgoing interface and your CoS configuration: if needed, refer to the formula on Figure 2.8. So let's check fabric queue statistics. On the EA ASIC we want to check fabric queue 44, which is the Low priority queue to reach PFE 0 of the MPC in slot 11:

```
(mx vty)# show mqss 0 sched-fab q-node stats 44

Queue statistics (Queue 0044)
-----------------------------

------------------------------------------------------------------------------
Color  Outcome           Counter  Total                Rate
                         Name
------------------------------------------------------------------------------
All    Forwarded (Rule)  Packets  25997467162          100032 pps
                         Bytes    13263582199401       408130816 bps
All    TAIL drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
0      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
1      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
2      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
3      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
------------------------------------------------------------------------------
```

You can see that all of our packets of flow 1 are forwarded well to MPC in slot 11. No ingress fabric drop occurred: good!

Let's do the same on the ZT line card. Here, you want to check how flow 2 is forwarded to PFE 0 of MPC in slot 9. Issue the same command as used on the EA, but here we display statistics for fabric queue 36 (PFE 0 of MPC in slot 9):

```
door7302@mx2020> start shell pfe network fpc11.0
(mx vty)# show mqss 0 sched-fab q-node stats 36
Queue statistics (Queue 0036)
-----------------------------

------------------------------------------------------------------------------
Color  Outcome           Counter  Total                Rate
                         Name
------------------------------------------------------------------------------
All    Forwarded (Rule)  Packets  19542826070          99995 pps
                         Bytes    9971715252564        407980808 bps
All    TAIL drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
0      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
1      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
2      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
3      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
------------------------------------------------------------------------------
```

As expected, our 100Kpps are forwarded without any ingress fabric drops. Before leaving the ingress PFE and moving to the egress side let's have a look at the FO block of the ingress PFE. Earlier you saw the command to give you aggregated statistics. On EA it was:

```
(mx vty)# show mqss 0 fo stats
FO statistics
-------------
Counter group 0
------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------
Set Type   Mask  Match Total Packets       Packets per second   Total Bytes        Bits per second
Total Cells        Cells per second
------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------
0  Stream 0x0  0x0  11647589          124054            4825432776         411190080
77383308          824249
1  Stream 0x0  0x0  19057649          124054            7895323948         411190080
126613774         824255
2  Stream 0x0  0x0  19057649          124054            7895324458         411190080
126613784         824257
```

The `Counter group 0` table allows you to configure specific FO counters like we did for WI. By default, this table gives the aggregated statistics (sum of all remote PFE) for each counter. To enable a specific counter use this specific `test` command. Here we want to check stats for remote stream 44:

```
(mx vty)# test mqss 0 fo stats resource 0 0 0 44
```

What does 0 0 0 44 mean?

- The first 0 is Counter group 0.
- The second 0 is the counter number we want to use – here, counter 0. There are 36 counters available.
- The third 0 means we want to collect fabric stream statistics .
- 44 is the stream we want to collect statistics: LOW fabric stream of PFE 0 in slot 11.

Once the counter is set, issue back the command to show FO statistics and have a look at `Counter Group 0` and `counter ID 0`. You can see our 100Kpps flows and its equivalent number of cells sent over the fabric to PFE 0/MPC slot 11:

```
(mx vty)# show mqss 0 fo stats
FO statistics
-------------
Counter group 0
------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------
Set Type   Mask  Match Total Packets       Packets per second   Total Bytes        Bits per second
Total Cells        Cells per second
------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------
0  Stream 0x3ff 0x2c  23242778          100101            11853816780        408412488
185942224         800816
```

You can reset a specific counter to its default configuration by using the next command below: `0 0` means counter 0 of counter group 0:

```
(mx vty)# test mqss 0 wi stats default 0 0
```

On the ZT, it's exactly the same command to configure a FO counter for a given fabric stream. Can you recall the list of commands without their output?

- `Start shell PFE network fpc11.0`
- `Test mqss 0 fo stats resource 0 0 0 36`

It's `36` because we want to collect statistics for the LOW fabric stream of remote PFE 0 hosted by MPC in slot 9.

- `Show mqss 0 fo stats`
- `Test mqss 0 fo stats default 0 0`

Okay, it's time to move on to the egress PFE. For flow 1 we will dive into MPC in slot 11 (on the ZT ASIC) and for flow 2 we will move to MPC in slot 9 (on the EA ASIC).

MPLS Packet on Egress PFE

Now we're going to do almost the same thing we did on the ingress PFE but in reverse. The first thing to check on the egress PFE is the FIB's programing. On the EA we reuse the following command to retrieve next-hop information:

```
(mx vty)# show route mpls table default
MPLS Route Table 0, default.0, 0x800c8:
Destination    Type   ID    NhRef
-------------  -----  ----- -----
default   Discard    50     1
2221      Unicast   550     1 et-11/0/0.0
2223      Unicast   543     1 et-9/0/0.0
```

Then we're going to see the details of the next-hop 543, which is attached to flow 2 (ingress label 2223). This output shows us that the Layer 2 header is pre-computed (source, destination, MAC addresses). There is also what we call a JHH word pointer, which provides in this case information about label manipulation:

```
(mx vty)# show nhdb id 543 extensive
   ID      Type      Interface    Next Hop Addr    Protocol       Encap
-----  --------  -------------  ----------------  ----------  ------------
  543   Unicast  et-9/0/0.0     -                      MPLS      Ethernet

[...]

Topo-link:
[pfe-0]: 0x11c0000000021f24
    ModifyNH: Subcode=SetNH-Token(7),Desc=0x0,Data=0x21f24,NextNH=0
 (pfeDest:36, TokenIdMode:0/  , VC memberId:0, token:0x21f/543)

[...]
NH Egress/Fabric:
        Feature List: NH
           [pfe-0]: 0x087b30f800100000;
          f_mask:0x82008000000000; c_mask:0xe000000000000000; f_num:27; c_num:3,  inst:0x0
        Idx#8     labels:
```

```
           [pfe-0]: 0x121fffffe1c52db0    << JNH word: Label pointer information

        Idx#14    counter:
           [pfe-0]: 0x2bfffffc8c000d00

        Idx#24      ucast:
           [pfe-0]: 0x129926fac007c8c1
PFE:0
Encap-ptr chain:
----------------
Encapsulation Pointer (0xeabd5b70) data:
    Encap-ptr-type:ether-da
[...]
   Ether-DA Details:
         Dest MAC:64:87:88:63:45:51     <<<<<<<<<<< Pre-computed Destination Mac
         InnerTPID(0x0)
[...]
Encapsulation Pointer (0x3208b4d8) data:
    Encap-ptr-type:ether-sa
[...]
   Ether-SA Details:
        Source MAC:4c:96:14:75:36:19  <<<<<<<<<<< Pre-computed Source Mac
         OuterTPID(0x0)
[...]
```

The JHN Word 0x121fffffe1c52db0 can be decoded with the following command:

```
(mx vty)# show jnh 0 decode 0x121fffffe1c52db0
ModifyNH: Subcode=SetLabel(8),Desc=0xffffff,Data=0x1c52db0,NextNH=0
 (label:0x52db0/339376, no_ttl_prop:0, fixed_exp_valid:0, fixed_exp:7, el flags 0)
```

And here we see label 2223 is swapped to value 339376, as expected. Let's do the same thing on the ZT line card. First of all, call back the command to display the MPLS routing table:

```
mx2020-fpc11:pfe> start shell pfe network fpc11
mx2020-fpc11:pfe> show route proto mpls name default

Index Destination                     NH Id     NH Type   NH Token
----- ------------------------------- --------- --------- ---------
0     default                         50        Discard   1477
0     2221                            550       Unicast   1453283
0     2223                            543       Unicast   1453327
```

Then check the next-hop 550 related to flow 1. We retrieve, like on the EA, the pre-computed Layer 2 header and the label value for the swap operation:

```
mx2020-fpc11:pfe> start shell pfe network fpc11
mx2020-fpc11:pfe> show nh detail index 550
Nexthop Info:

NH Index                      : 550
NH Type                       : Unicast
NH Proto                      : tag
NH Flags                      : 0x1
IF Name                       : et-11/0/0.0     <<< egress interface
Prefix                        : 789120
NH Token Id                   : 1453283
NH Route Table Id             : 0
```

```
Sgid                          : 0
OIF Index                     : 402
Underlying IFL                : .local..0 (0)
Session Id                    : 5997
Num Tags                      : 1
Label                         : 0x49310eff (299792)lbFlags: 0      <<<< Swap Label
MTU                           : 0
L2 Length                     : 12
L2 Data                       : 00:00:01:92:80:00:64:87:88:63:45:a3 <<<<
                        Precomputed Ethernet Header (Dst/Src Mac Adresses)
[...]
Container token: 1453283
#1 SetNhToken tokens:
 Mask : 0x0
 [  SetNhToken:1453282  ]

  EgressFeatures:

Container token: 1453288
#0 PushLabels tokens:
 Mask : 0x0
 [  PushLabels:1453285  ]  <<<<<< Here it's the token ID not the label
#1 StatsCounter tokens:
 Mask : 0x1
 [  StatsCounter:1453287  ]
#5 UcastEncap tokens:
 Mask : 0x0
 [  UcastEncap:1453286  ]

mx2020-fpc11:pfe> show sandbox token 1453285     << Entry Token for Label
                                                        Manipulation
AftNode   : AftList token:1453285 group:0 nodeMask:0x1 nodes: {1453284,} <<<<<<<
                                                             Next Token 1453284

mx2020-fpc11:pfe> show sandbox token 1453284     << Resolve next token
                                                      1453284
AftNode   : AftEncap token:1453284 group:0 nodeMask:0x1 name: label keys: {{ field:packet.mpls.label,
data:Value 299792,     <<<< New Label Value (SWAP)
JnhHandle : JnhHandleEncapLabel Jnh:0x121fffffe1c49310 PfeInst:0 Paddr:0x0 Vaddr:0x0 AppType:NH
```

After checking the consistency of the LFIB on both egress line cards you can first see statistics about the amount of traffic received by the FI block coming from a specific source PFE.

You can configure a specific counter for the FI block much like with the FO block. On EA (MPC in slot 9) we want to see how many packets are received from source PFE 44 (the PFE 0 of MPC in slot 11). This is actually flow 2 that comes from MPC in slot 11 and has crossed the fabric to reach MPC in slot 9. Configuring an FI counter is very similar to what we did for FO. Let's first do it on the EA line card:

```
(mx vty)# test mqss 0 fi stats stream 44
```

Notice here we are on the egress PFE. The above command requests to specify the incoming stream from the egress's point of view. The 44 means the traffic comes from PFE 0 in slot 11. Refer to Figure 2.10 if you need a quick refresher about the fabric queue/stream logic.

Now, issue the following command. We again find our 100Kpps. No drop is identified on the FI side. All seems fine at this waypoint:

```
(mx vty)# show mqss 0 fi stats
FI statistics
-------------
Stream number mask  : 0x3ff
Stream number match : 0x2c
-------------------------------------------------------------------------------------------------
Counter Name                                          Total               Rate
-------------------------------------------------------------------------------------------------
Valid data cells received from FABIO                  1991474             824209 cps
Valid fabric grants received from FABIO               1991418             824201 grants/second
Valid fabric requests received from FABIO             1991473             824209 requests/second
Valid fabric requests sent to FO                      1991473             824209 requests/second
Received cells in input block for enabled streams     1932968             800008 cps
Received packets in input block for enabled streams   241620              100000 pps
Cells dropped in FI memory interface                  0                   0 cps
Packets dropped in FI memory interface                0                   0 pps
Packets sent out of PSV                               241620              100000 pps
Error packets sent out of PSV                         0                   0 pps
-------------------------------------------------------------------------------------------------
```

Don't forget to reset the FI counter to its default configuration:

```
(mx vty)# test mqss 0 fi stats default
```

This is exactly the same set of commands as on the ZT. For the next example we just change the fabric stream number to `36`, which is attached to source PFE 0 in slot 9:

```
door7302@mx2020> start shell pfe network fpc11.0
(mx vty)# test mqss 0 fi stats stream 36

(mx vty)# show mqss 0 fi stats

FI statistics
-------------
Stream number mask  : 0x3ff
Stream number match : 0x24
-------------------------------------------------------------------------------------------------
Counter Name                                          Total               Rate
-------------------------------------------------------------------------------------------------
Valid data cells received from FABIO                  3161238             824347 cps
Valid fabric grants received from FABIO               3161176             824344 grants/second
Valid fabric requests received from FABIO             3161237             824347 requests/second
Valid fabric requests sent to FO                      3161237             824347 requests/second
Received cells in input block for enabled streams     3068358             800134 cps
Received packets in input block for enabled streams   383545              100017 pps
Cells dropped in FI memory interface                  0                   0 cps
Packets dropped in FI memory interface                0                   0 pps
Packets sent out of PSV                               383544              100016 pps
Error packets sent out of PSV                         0                   0 pps
-------------------------------------------------------------------------------------------------
(mx vty)# test mqss 0 fi stats default
```

Our 100Kpps of flow 1 are there, as well, with no drop. It's time to capture our packet on the egress side – meaning from fabric to WAN. As we saw earlier on the ingress PFE, the commands will display the packet before and after its processing

by the egress LUSS. We saw that the label swap operation didn't occur on the ingress PFE and also the Layer 2 Ethernet header has been removed by the ingress LUSS. Therefore, the hexadecimal pattern we will use to filter our capture will be a little bit different.

Indeed, we could not match on the MPLS ether type 0x8847, as it has been stripped. We decided to only match based on the label value (`+ TC` and `S` flags as these 4 bits are part of a byte shared with the label value).

For flow 1 it will give us the following hexadecimal pattern:

```
008ad hex(2221)  + 000 ( TC) + 1 (S) = 0x008ad1
```

And for flow 2 we will have:

```
008af hex(2223)  + 000 ( TC) + 1 (S) = 0x008af1
```

Now we're ready to capture our egress packet. Let's start on the EA ASIC. First of all, enable packet capture on the right PFE (`jnh 0` means PFE 0) like we did on the ingress:

```
(mx vty)# test jnh 0 packet-via-dmem enable
```

Then capture packets that match our hexadecimal pattern 8847008af1 (flow 2 is sent out by EA line card.) Always set the `0x3` option, which means capturing all types (packet and packet HEAD). No offset is needed:

```
(mx vty)# test jnh 0 packet-via-dmem capture 0x3 008ad1 <Offset>
```

Wait some time and then stop the capture:

```
(mx vty)# test jnh 0 packet-via-dmem capture 0x0
```

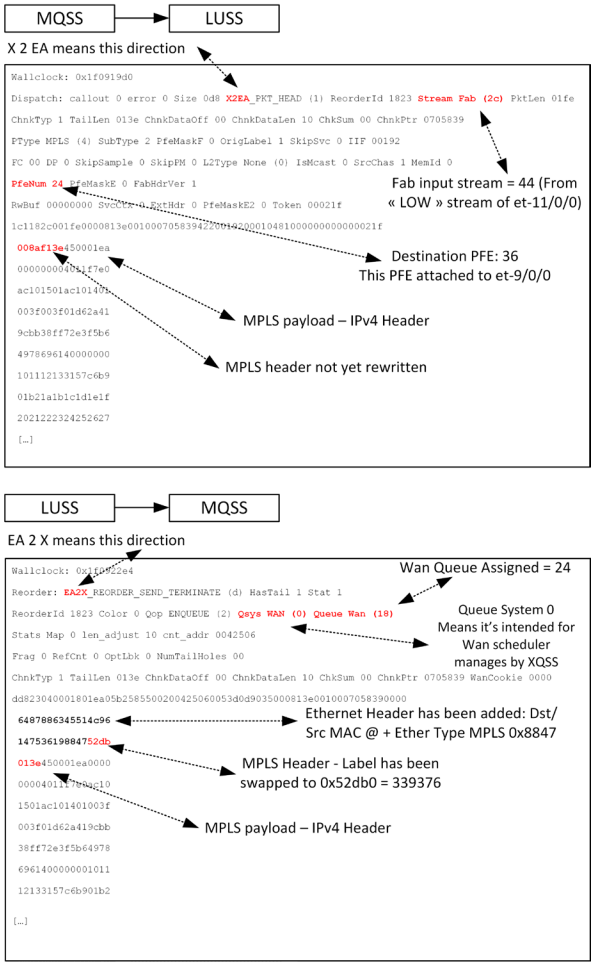
And finally, dump the buffer of captured packets, and we'll have a look at just a couple:

```
(mx vty)# test jnh 0 packet-via-dmem decode
Wallclock: 0x1f0919d0
Dispatch: callout 0 error 0 Size 0d8 X2EA_PKT_HEAD (1) ReorderId 1823 Stream Fab (2c) PktLen 01fe
ChnkTyp 1 TailLen 013e ChnkDataOff 00 ChnkDataLen 10 ChkSum 00 ChnkPtr 0705839
PType MPLS (4) SubType 2 PfeMaskF 0 OrigLabel 1 SkipSvc 0 IIF 00192
FC 00 DP 0 SkipSample 0 SkipPM 0 L2Type None (0) IsMcast 0 SrcChas 1 MemId 0
PfeNum 24 PfeMaskE 0 FabHdrVer 1
RwBuf 00000000 SvcCtx 0 ExtHdr 0 PfeMaskE2 0 Token 00021f
1c1182c001fe0000813e00100070583942200192000104810000000000000021f
 008af13e450001ea
 000000004011f7e0

Wallclock: 0x1f0922e4
Reorder: EA2X_REORDER_SEND_TERMINATE (d) HasTail 1 Stat 1
ReorderId 1823 Color 0 Qop ENQUEUE (2) Qsys WAN (0) Queue Wan (18)
Stats Map 0 len_adjust 10 cnt_addr 0042506
Frag 0 RefCnt 0 OptLbk 0 NumTailHoles 00
ChnkTyp 1 TailLen 013e ChnkDataOff 00 ChnkDataLen 10 ChkSum 00 ChnkPtr 0705839 WanCookie 0000
dd823040001801ea05b2585500200425060053d0d9035000813e0010007058390000
 6487886345514c96
 14753619884752db
 013e450001ea0000
 00004011f7e0ac10
 […]
```

As the ingress captures, for each captured packet you have two dumps (`Wallclock`). The first dump gives you the packet information before LUSS processing, meaning when the MQSS sends the packet to the LUSS. The second dump gives you the packet after LUSS when it comes back in the MQSS.

Before its processing by the LUSS we saw that the packet is tagged as coming from the fabric (stream fab is displayed – here it's 0x2C = 44 – it comes from PFE 0 of MPC in slot 11 as expected). After processing, we see that the LUSS has assigned a WAN queue 0x18 (24) – if you check the detail of the egress packet in hexadecimal you should see that the new Layer 2 header has been added and the MPLS label value has been swapped. Figure 3.6 helps you to decode these outputs.

Figure 3.6 How to Decode Egress Packet Capture On the EA



At the end of your troubleshooting, don't forget to disable the packet capture:

```
(mx vty)# test jnh 0 packet-via-dmem disable
```

Let's do the same on the ZT. First of all, enable packet capture on the right egress PFE (`inst 0` means PFE 0):

```
door7302@mx2020> start shell pfe network fpc11
mx2020-fpc11:pfe> test jnh packet-via-dmem inst 0 enable
```

Then capture packets that match our hexadecimal pattern 8847008ad1 (Flow 1 is sent out by the ZT line card.) Always set the `0x3` option, which means capturing all types (packet and packet HEAD):

```
mx2020-fpc11:pfe> test jnh packet-via-dmem-capture inst 0 parcel-type-mask 0x3 match-
string 008ad1 offset <Offset>
```

Wait some time and then stop the capture:

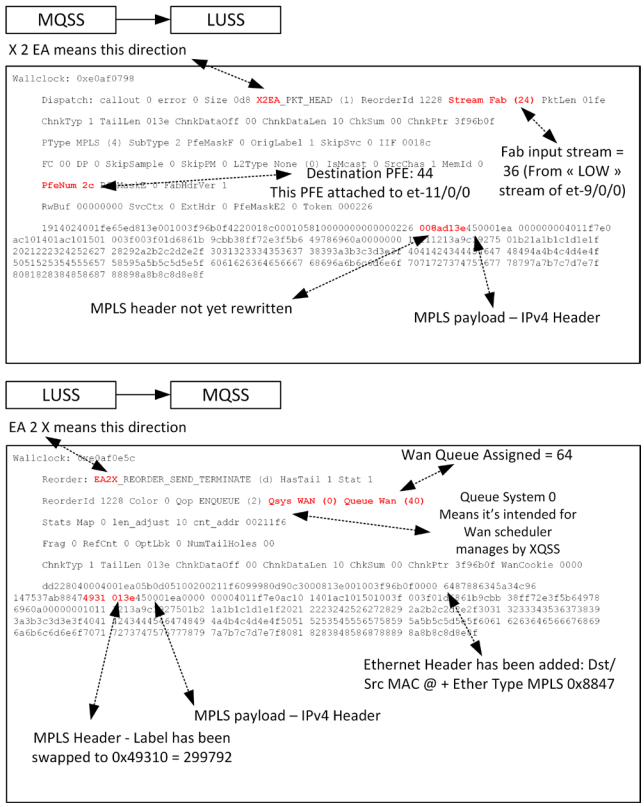
```
mx2020-fpc11:pfe> test jnh packet-via-dmem-capture inst 0 parcel-type-mask 0x0
```

And finally, dump the capture's buffer. Once again, don't be scared about the amount of data displayed:

```
mx2020-fpc11:pfe> test jnh packet-via-dmem-dump inst 0
Wallclock: 0xe0af0798
     Dispatch: callout 0 error 0 Size 0d8 X2EA_PKT_
HEAD (1) ReorderId 1228 Stream Fab (24) PktLen 01fe
     ChnkTyp 1 TailLen 013e ChnkDataOff 00 ChnkDataLen 10 ChkSum 00 ChnkPtr 3f96b0f
     PType MPLS (4) SubType 2 PfeMaskF 0 OrigLabel 1 SkipSvc 0 IIF 0018c
     FC 00 DP 0 SkipSample 0 SkipPM 0 L2Type None (0) IsMcast 0 SrcChas 1 MemId 0
     PfeNum 2c PfeMaskE 0 FabHdrVer 1
     RwBuf 00000000 SvcCtx 0 ExtHdr 0 PfeMaskE2 0 Token 000226
    1914024001fe65ed813e001003f96b0f4220018c0001058100000000000226 008ad13e450001ea
000000004011f7e0 ac101401ac101501 003f003f01d6861b 9cbb38ff72e3f5b6 49786960a0000000 10111213a9c19275
01b21a1b1c1d1e1f […]

Wallclock: 0xe0af0e5c
     Reorder: EA2X_REORDER_SEND_TERMINATE (d) HasTail 1 Stat 1
     ReorderId 1228 Color 0 Qop ENQUEUE (2) Qsys WAN (0) Queue Wan (40)
     Stats Map 0 len_adjust 10 cnt_addr 00211f6
     Frag 0 RefCnt 0 OptLbk 0 NumTailHoles 00
     ChnkTyp 1 TailLen 013e ChnkDataOff 00 ChnkDataLen 10 ChkSum 00 ChnkPtr 3f96b0f WanCookie 0000
    dd228040004001ea05b0d05100200211f6099980d90c3000813e001003f96b0f0000 6487886345a34c96
147537ab88474931 013e450001ea0000 00004011f7e0ac10 […]'
```

And once again, for each captured packet on the egress side you have two dumps: the one before processing: MQSS to LUSS, and the second one after processing: LUSS to MQSS. Figure 3.7 helps you to understand the output.

Figure 3.7 How to Decode Packet Egress Capture on the ZT



At the end of your troubleshooting on the egress ZT, don't forget to disable packet capture:

```
mx2020-fpc11:pfe> test jnh packet-via-dmem inst 0 disable
```

We know how the packet has been manipulated by the egress PFE (Layer 2 added Label swap) and we also know which WAN queue has been assigned for each flow. Remember, on the EA and the ZT that WAN queues are managed by the XQSS block. So the `queue 24` will enqueue packets of flow 2 on the EA XQSS and `queue 60` will be used on the ZT XQSS for flow 1. You can retrieve XQSS statistics for a given queue on the EA by issuing this command:

```
(mx vty)# show xqss 0 sched queue 24 local-stats
Queue:24
        Forwarded pkts : 26319795124       100007        pps
        Forwarded bytes: 14002112664864    425633320     bps
        Dropped pkts   : 0                 0             pps
        Dropped bytes  : 0                 0             bps
```

You can see that our 100Kpps of flow 2 are there. You can also check CoS statistics of your given egress interface with the `show cos halp` command. Here queue 0 of et-9/0/0 conveys the 100Kpps. Actually, this queue 0 of et-9/0/0 is the absolute queue number 24 on the XQSS (see previous command):

```
(mx vty)# show cos halp ifd queue-stats et-9/0/0
IFD index: 439 Queue: 0
  Last Enqueue time     : No packets
Queued              :
  Packets           :    26336604909         100024 pps
  Bytes             : 14011055457530       53213200 Bps
Transmitted         :
  Packets           :    26336604909         100024 pps
  Bytes             : 14011055457530       53213200 Bps
  Tail-dropped pkts :              0              0 pps
  Tail-dropped bytes:              0              0 Bps
  RED-dropped pkts  :
    Low             :              0              0 pps
    Medium-low      :              0              0 pps
    Medium-high     :              0              0 pps
    High            :              0              0 pps
  RED-dropped bytes :
    Low             :              0              0 Bps
    Medium-low      :              0              0 Bps
    Medium-high     :              0              0 Bps
    High            :              0              0 Bps
  RL-dropped pkts   :              0              0 pps
  RL-dropped bytes  :              0              0 Bps
  Queue depths      :
    Average         :              0
    Current         :            510
    Peak till now   :           1530
    Maximum         :      316669952
[…]
```

On the ZT, you can use a new set of commands to display interface statistics at the PFE level. You should first retrieve the IFD index of your physical port:

```
door7302@mx2020> start shell pfe network fpc11
mx2020-fpc11:pfe> show interfaces et-11/0/0 | match index
 Name: et-11/0/0     Index: 535
```

And call next the following command:

```
mx2020-fpc11:pfe> show class-of-service interface queue-stats index 535
Physical interface : et-11/0/0 (Interface index:  535, Egress queues:  8)
Queue: 0
        Queued Packets                  :     26370165892          100002  pps
        Queued Bytes                    :  14028401532285       425604704  bps
        Transmitted  Packet             :     26370165892          100002  pps
        Transmitted Bytes               :  14028401532285       425604704  bps
        Tail-dropped Packets            :               0               0  pps
        RL-dropped Packets              :               0               0  pps
        RL-dropped Bytes                :               0               0  bps
        RED-dropped Packets             :               0               0  pps
          Low                           :               0               0  pps
          Medium-low                    :               0               0  pps
          Medium-high                   :               0               0  pps
          High                          :               0               0  pps
```

```
        RED-dropped Bytes               :               0               0  bps
          Low                           :               0               0  bps
          Medium-low                    :               0               0  bps
          Medium-high                   :               0               0  bps
          High                          :               0               0  bps
        Queue-depth bytes               :
          Average                       :               0
          Current                       :               0
          Peak                          :            1530
          Maximum                       :       316669952
```

Finally, the last thing we can check on the egress PFE is the WO statistics. On the EA and ZT this is the same PFE command:

```
(mx vty)# show mqss 0 wo stats
WO statistics
-------------
Counter set 0

    Connection number mask   : 0x0
    Connection number match  : 0x0
    Transmitted packets      : 33977564640 (500057 pps)
    Transmitted bytes        : 17279896388620 (2008221760 bps)
    Transmitted flits        : 203348335667 (3000309 flits/sec)

Counter set 1

    Connection number mask   : 0x1f
    Connection number match  : 0x1d
    Transmitted packets      : 5131 (0 pps)
    Transmitted bytes        : 811958 (0 bps)
    Transmitted flits        : 10276 (0 flits/sec)
```

The counter 0 gives the aggregate statistics. This means it provides output traffic stats for physical ports attached to the MQSS instance. We can configure a specific counter to filter output traffic attached to a specific port. To do that, use the `test` command one more time. But before that you need to find the connection ID allocated to your given port. For that, issue the command either on the EA or the ZT (the commands below are similar on both ASICs):

```
(mx vty)# show mqss 0 wo wan-conn-entry

WAN connection entries
----------------------

------------------------------------------------------
Connection    Allocated    Port Speed    IFD Name
------------------------------------------------------
0             No           -             -
1             No           -             -
2             No           -             -
3             No           -             -
4             No           -             -
5             No           -             -
```

```
6             No          -             -
7             No          -             -
8             No          -             -
9             No          -             -
10            No          -             -
11            No          -             -
12            No          -             -
13            No          -             -
14            No          -             -
15            No          -             -
16            No          -             -
17            No          -             -
18            No          -             -
19            No          -             -
20            No          -             -
21            No          -             -
22            No          -             -
23            No          -             -
24            Yes         100 GE        et-9/0/0
25            Yes         100 GE        et-9/0/3
-----------------------------------------------------
```

The connection ID of our et-9/0/0 interface is 24. Now we can configure the WO counter 1 to display output statistics of et-9/0/0:

```
(mx vty)# test mqss 0 wo stats conn 1 24
```

And issue back the command to display WO statistics:

```
(mx vty)# show mqss 0 wo stats

WO statistics
-------------
Counter set 0

    Connection number mask   : 0x0
    Connection number match  : 0x0
    Transmitted packets      : 34020270536 (100027 pps)
    Transmitted bytes        : 17301675780631 (408100088 bps)
    Transmitted flits        : 203604564340 (600148 flits/sec)

Counter set 1

    Connection number mask   : 0x1f
    Connection number match  : 0x18
    Transmitted packets      : 663939 (100021 pps)
    Transmitted bytes        : 338608369 (408098408 bps)
    Transmitted flits        : 3983628 (600144 flits/sec)
```

At the end, reset the counter configuration to its default value:

```
(mx vty)# test mqss 0 wo stats default
```

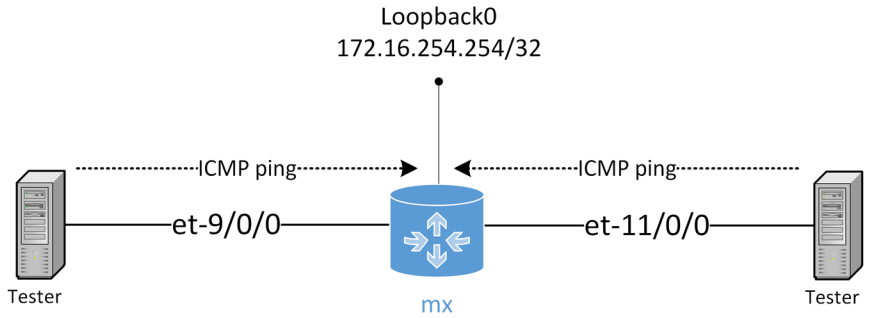
This last command ends the first part of this chapter. We can now move to part two, where we will try to better understand how the control plane/OAM traffic is handled and managed by the two ASICs.

A Host Packet Targets the MX

Figure 3.8 shows you the network topology we are going to use during this second part of the chapter. We are using a tester to generate ICMP echo request packets. The tester is connected to two interfaces of the MX: one is attached to an MPC9e in slot 9 and the second one to an MPC11e in slot 11.

On each interface, the tester sends 200pps of ICMP echo-request targeting the loopback0 of the MX: 172.16.254.254.

Figure 3.8 Network Topology to Track Host Traffic



As we did during the first part of this chapter, we will provide the point of view of the EA and the point of view of the ZT at each step. Let's go inside!

Tracking Host Inbound Traffic

On both line cards, the ICMP echo-request packets are first handled by the MAC block of the EA or ZT. Then the pre-classifier engine of both ASICs performs a short packet analysis, and based on well-known packet headers, it determines if the packet is a control plane/OAM packet or not. At this point, the EA or ZT are not able to know if the packet is a packet in transit or a packet trying to reach the router itself (only the look-up step, done by the LUSS, will give this information).

Nevertheless, this pre-classification should detect our traffic as OAM and therefore classify it into the CTRL/medium WI input stream. On the EA we have access to the pre-classifier statistics (not yet on the ZT as mentioned earlier). We confirm with the following two commands and our 200pps are put into the right input stream:

```
(mx vty)# show precl-eng summary
 ID  precl_eng name        FPC PIC ASIC-ID ASIC-INST Port-Group  (ptr)
--- -------------------- ---- --- -------- --------- ----------  --------
  1 MQSS_engine.9.0.60      9   0      60       0          NA       f6f023d0
  2 MQSS_engine.9.0.61      9   0      61       1          NA       f6f12a90
```

```
(mx vty)# show precl-eng 1 statistics
         stream    Traffic
 port      ID       Class             TX pkts               RX pkts          Dropped pkts
------  -------  ----------          ---------           ---------        --------------
  24      1165        RT          0000000000000000      0000000000000000    0000000000000000
  24     1166       CTRL       0000000013872090     0000000013872090    000000000000000 <<<< It's
incrementing - these are our ICMP echo-request
  24     1167       BE         0000066904792122     0000066904792122    0000000000000000
```

Then, packets are processed as usual (for instance: HEAD and TAIL are extracted if needed) by the MQSS block until they are forwarded to the LUSS. On the LUSS the lookup engine detects the packets are targeting the router: we name this type of traffic *host inbound*. From the EA or ZT's point of view the host inbound traffic is a kind of packet `exception`.

Exception statistics are available by issuing this CLI command:

```
door7302@mx2020> show pfe statistics exceptions fpc <FPC-SLOT>
Slot 11

     PFE State Invalid
     ----------------------
     sw error                          DISC( 64)           12        768

     Routing
     ----------------------
     control pkt punt via nh           PUNT( 34)        10853    1246951
     host route                        PUNT( 32)      1505273  741629096

     PFE State Invalid
     ----------------------
     sw error                          DISC( 64)           12        768

     Routing
     ----------------------
     control pkt punt via nh           PUNT( 34)        10567    1183534
     host route                        PUNT( 32)         3484     243866
```

The command above gives you aggregated statistics, which means for all PFEs of a given MPC. If you want to see the exception statistics of a specific PFE more precisely you must issue the following PFE command on the EA (`0` means PFE 0 in our case – the `terse` option allows you to display only non-zero values). Moreover, before issuing the command, you can clear `exceptions` statistics – this is what we do below. Notice some `exception` counters cannot be cleared – this is why you can see some warnings (just don't take these warnings into account).

In our case the ping packets are counted as host route exceptions:

```
(mx vty)# clear jnh 0 exceptions
Relative counters cannot be cleared
[…]
(mx vty)# sho jnh 0 exceptions terse
```

```
Reason                              Type           Packets      Bytes
=================================================================
Routing
----------------------
control pkt punt via nh             PUNT( 34)            1         80
host route                          PUNT( 32)          836     410744
```

Similar commands can be used on the ZT (`inst 0` means PFE 0):

```
door7302@mx2020> start shell pfe network fpc11
mx2020-fpc11:pfe> clear jnh exceptions inst 0
mx2020-fpc11:pfe> show jnh exceptions inst 0 level terse

     Routing
     ----------------------
     control pkt punt via nh            PUNT( 34)           14       1524
     host route                         PUNT( 32)         1716     845566
```

As you can see from the previous command outputs, there is a type and an ID between the parenthesis `( )` associated with each exception. There are actually two types: PUNT and DISCARD. PUNT means the packet might be handled by the upper level (the line card CPU or the RE). DISCARD means the packet matching the exception must be silently discarded by the ASIC. The associated ID uniquely identifies the exception.

Let's take a short break to have a better look at the exceptions. Punted packets can usually be captured by the classic `monitor traffic interface` command since those packets usually reach the host. This is the case of our ping packets:

```
door7302@mx2020> monitor traffic interface et-9/0/0 matching "icmp[icmptype]==icmp-echo" no-resolve
verbose output suppressed, use <detail> or <extensive> for full protocol decode
Address resolution is OFF.
Listening on et-9/0/0, capture size 96 bytes

18:03:05.959314  In IP 192.168.3.1 > 172.16.254.254: ICMP echo request, id 0, seq 0, length 474
```

But with DISCARD exceptions, the previous command will give you nothing because drops occurred inside the ASIC. Sometimes it can be interesting to view which packet is considered as an exception to discard. The name of the exception usually gives you the first information about the nature of the packet, but if we need more, how to proceed?

Junos offers us a way to capture exception traffic quite easily: only exceptions with a type DISCARD can be captured. To illustrate this feature let's apply a firewall filter on the loopback0 that drops all ICMP traffic:

```
door7302@mx2020> show configuration firewall family inet filter PROTECT-RE
term 1 {
    from {
        protocol icmp;
    }
    then {
        discard;
    }
}
term 2 {
```

```
    then accept;
}
```

First on the EA, you can now see packets are no longer seen as host route but as firewall discard. You can also see the type of the exception is DISCARD with the ID 67:

```
(mx vty)# show jnh 0 exceptions terse

Reason                              Type           Packets      Bytes
=================================================================

Firewall
-----------------------
firewall discard                    DISC( 67)         5822    2876068

Routing
-----------------------
control pkt punt via nh             PUNT( 34)           52       4846
host route                          PUNT( 32)        23450   11416404
```

To capture packets of this specific exception you should use this set of commands (using the ID and type previously retrieved):

```
(mx vty)# debug jnh exceptions-trace
(mx vty)# debug jnh exceptions 67 discard
```

When you think your packet has been dropped (by calling back `show jnh X exception` periodically), you can issue the next command to display the captured packets. Hopefully you have good eyes, because the dump is in hexadecimal. Notice there is an internal header attached to your packet – just look at your IP packet by looking for the well-known IPv4 signature `45 xx xx`:

```
(mx vty)# show jnh exceptions-trace
 [7185] jnh_exception_packet_trace:1456 ###############
 [7186] jnh_exception_packet_trace:1461 [iif:496,code/info:195 D(firewall discard)/0x0,score:(0x0),p
type:2/0,orig_ptype:2,offset:14,orig_offset:14,len:508,l2iif:0,oif:0,BD 0,l2-off=0,token=1048575 ]
 [7187] jnh_exception_packet_trace:1485 0x00: 20 00 c3 00 00 00 01 f0 00 0e 01 fc 80 00 00 20
 [7188] jnh_exception_packet_trace:1485 0x10: 0e 00 00 40 01 00 0f ff ff 00 00 00 00 00 00 4c
 [7189] jnh_exception_packet_trace:1485 0x20: 96 14 75 36 19 64 87 88 63 45 51 08 00 45 00 01
 [7190] jnh_exception_packet_trace:1485 0x30: ee 00 00 00 00 3f 01 0b 57 c0 a8 03 01 ac 10 fe
 [7191] jnh_exception_packet_trace:1485 0x40: fe 08 00 b2 9b 00 00 00 00 9c bb 38 ff 72 e3 f5
 [7192] jnh_exception_packet_trace:1485 0x50: b6 49 78 69 60 c0 00 00 00 da da da da 7b 4e 78
 [7193] jnh_exception_packet_trace:1485 0x60: 75 01 b6 da da da da da da da da da da da da da
 [7194] jnh_exception_packet_trace:1485 0x70: da da da da da da da da da da da da da da da da
 [7195] jnh_exception_packet_trace:1456 ###############
```

Don't forget to disable the packet capture at the end of your troubleshooting session:

```
(mx vty)# undebug jnh exceptions-trace
```

If you want to do a similar capture on the ZT, first of all, check as we did earlier the `exception` statistics to confirm that our pings are now seen as `firewall discard exceptions`:

```
door7302@mx2020> start shell pfe network fpc11
mx2020-fpc11:pfe> show jnh exceptions inst 0 level terse

     Firewall
     ----------------------
     firewall discard                  DISC( 67)         1653     816582

     Routing
     ----------------------
     control pkt punt via nh           PUNT( 34)           12       1267
     host route                        PUNT( 32)            6        402
```

Sounds good, the `firewall discard` is continuously incrementing. Now you can enable packet capture for this specific exception:

```
mx2020-fpc11:pfe> debug jnh exceptions state enable inst 0 exception 67 type DISCARD
```

Next, enable the dump of these packets into the syslog of the MPC (this command is available since Junos 20.2):

```
mx2020-fpc11:pfe> set host-path ports punts trace enable
```

And finally, issue the command to see the dumped packets:

```
mx2020-fpc11:pfe> show syslog
packetio[18339]: Trace.
    CPU Hdr
    -----------------
    ptype:          2
    sub type:       0
    stream type:    1
    reason:         195
    punt:           268
    drop reason:    67
    token:          1048575
    input Ifl:      414
    l2 input Ifl:   0
    output Ifl:     0
    l2len:          508
    l2Offset        0
    l3Offset        14
    origL3Type      2
    origL3Offset    14
    ddosProto       0x4001
    fwdClass        0
    vbfFlowId       0
    hostOrig:       0
    pktLen:         0
    addInfo:        0x0
    :
    Packet inIf:fp0 outIf:cp0 type:Init net:Trace Pkt Len: 494 pos: 0
    45 00 01 ee 00 00 00 00 3f 01 0a 57 c0 a8 04 01
    ac 10 fe fe 08 00 1e 97 00 00 00 00 9c bb 38 ff
    72 e3 f5 b6 49 78 69 61 80 00 00 00 da da da da
    1c 6e ab 59 01 b6 da da da da da da da da da da
    da da da da da da da da da da da da da da da da
```

The good thing on the ZT is that the internal header is decoded and only the captured packet is displayed in hexadecimal. Like on the EA, at the end you must disable both the packet capture and the syslog dump with the following two commands:

```
mx2020-fpc11:pfe> debug jnh exceptions state disable inst 0 exception 67 type DISCARD
mx2020-fpc11:pfe> set host-path ports punts trace disable
```

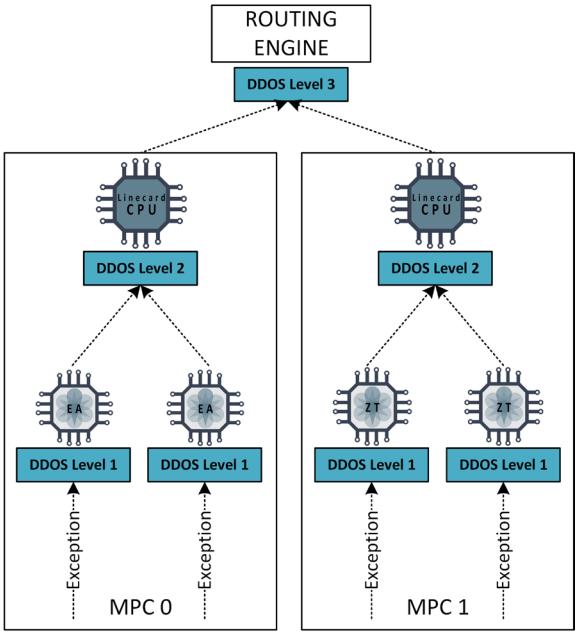
Remove the firewall filter applied on the lo0 and let's move on!

Still inside the LUSS, this host inbound traffic that might be dropped or rate-limited by any input firewall filter configured either on the lo0 or on the physical interfaces. Of course, the packets should match the firewall family.

If the packets are accepted, they are processed by a first level of DDoS protection (still inside the LUSS). Remember there are three levels of DDoS protection on the TRIO. Figure 3.9 provides a quick refresher on how the distributed protection is managed on TRIO line cards.

Therefore, if needed, a new rate limiting is performed by the LUSS's DDoS protection feature depending on either default policer values or your own `set system ddos-protection xxx` configuration.

Figure 3.9 Distributed DDoS Protection



To check DDoS statistics occurring at the ASIC level, you can use these PFE commands on the EA. You should supply the name of the protocol for which you wish to see the stats (`?` displays all protocols supported by the DDoS protection feature).

Now let's check ICMP statistics:

```
(mx vty)# show ddos policer icmp stats
DDOS Policer Statistics:
                                                                                arrival   pass  # of
 idx prot    group        proto  pol   scfd      loc       pass     drop     rate   rate flows
 ---  ---  --------  -----------  ---  -----  -------  -------- --------  ------ ------ -----
  77  900     icmp    aggregate   on normal    UKERN     3296726        0     200    200     0
                                            PFE-0:0     3296726        0     200    200     0
                                            PFE-1:0           0        0       0      0     0
                                            PFE-2:0           0        0       0      0     0
                                            PFE-3:0           0        0       0      0     0
```

With this command, you can see statistics of the two first levels of DDoS protection. Remember that Level 1 is managed by the ASIC itself – here the EA. Level 1 statistics are given by the lines starting with the prefix PFE. In our case we have four PFEs, so four EA ASICs on this line card; this is why we have four lines of statistics. We can confirm our 200pps of ICMP have been accepted by the EA (rate column) and forwarded to Level 2. The Level 2 of DDoS protection on the EA line card is managed by the uKernel process (the uKernel line gives you the statistics of the second level). We see the 200pps are also accepted by Level 2 and therefore forwarded to the RE where Level 3 DDoS protection will occur.

If you wish to retrieve the configuration of a given protocol you can use this second command. On the EA the default ICMP policer is configured at 20Kpps (same configuration is set for the three levels):

```
(mx vty)# show ddos policer configuration icmp
DDOS Policer Configuration:

                                        UKERN-Config   PFE-Config

 idx prot        group        proto on Pri  rate burst   rate burst
 ---  ---  ------------ ------------ -- -- ------ -----  ------ -----
  77  900         icmp    aggregate  Y Hi  20000 20000  20000 20000
```

Let's do the same for the ZT. On the ZT you should use a combination of three commands, but first let's list all supported protocols (named `Group`):

```
door7302@mx2020> start shell pfe network fpc11
mx2020-fpc11:pfe> show ddos groups
Ddos Proto Groups:

Proto Group              Group Id
    host-path            0
    resolve              1
    filter-action        2
    Dynamic-Vlan         3
    PPP                  4
    PPPoE                5
    DHCPv4               6
```

```
    DHCPv6                7
    Virtual-Chassis       8
    ICMP                  9
    IGMP                  10
    OSPF                  11
[…] Output has been truncated.
```

Once you have identified your protocol you can issue the second command to list the policer(s) available for this specific protocol. For this exercise it's ICMP, but pay attention here. The following command is case sensitive and there is an 's' at the end of the word `policers` (actually there is the same command without an `s` and we are going to use it after this one). Notice at the same time you can see the `policers` configuration:

```
door7302@mx2020> start shell pfe network fpc11
mx2020-fpc11:pfe> show ddos policers ICMP
Host Path Ddos Policers:

Group        Policer     Id     BWidth  Burst  BWScale BTScale RcvTime P  Q
ICMP         aggregate   0x0900 20000   20000  100     100     300     2  0
```

And finally, for a given group, `ICMP` for us, and policer name, `aggregate` for us, you can retrieve statistics.

Here the command used is `policer` without the s:

```
door7302@mx2020> start shell pfe network fpc11
mx2020-fpc11:pfe> show ddos policer ICMP aggregate
Ddos Proto:

Group           Policer         Id     BWidth  Burst  BWScale BTScale RcvTime P  Q
ICMP            aggregate       0x0900 20000   20000  100     100     300     2  0

Loc    Total Rcvd Pkts  Total Drops  Policer Drops  Flow drops  Other drops  [Rate(pps) Max Rate
PktIo  3453803          0            0              0           0            200        200
Pfe:0  3453967          0            0              0           0            200        201
Pfe:1  0                0            0              0           0            0          0
Pfe:2  0                0            0              0           0            0          0
Pfe:3  0                0            0              0           0            0          0
Pfe:4  0                0            0              0           0            0          0
Pfe:5  0                0            0              0           0            0          0
Pfe:6  0                0            0              0           0            0          0
Pfe:7  0                0            0              0           0            0          0
Sum    3453967          0            0              0           0            200        201

Violations:
Loc    State   Start                    Last                     Count
PktIo  ok      ---                      ---                      0
Pfe:0  ok      ---                      ---                      0
Pfe:1  ok      ---                      ---                      0
Pfe:2  ok      ---                      ---                      0
Pfe:3  ok      ---                      ---                      0
Pfe:4  ok      ---                      ---                      0
Pfe:5  ok      ---                      ---                      0
Pfe:6  ok      ---                      ---                      0
Pfe:7  ok      ---                      ---                      0
Sum    ok      ---                      ---                      0
```

You can also see the two first levels of DDoS protection statistics. Level 1 (lines starting with `Pfe:`) shows the 200pps are also accepted by the the ZT ASIC of PFE 0 and then forwarded to the upper level. On the ZT line card, the second level of DDoS protection is managed by the packetIO module. This is why the Level 2 statistics line starts with the statement `PktIo`. Once again, the 200pps, which do not violate the default ICMP policer (20Kpps), are accepted and forwarded to the Level 3 hosted by the RE.

If packets are accepted by the Level 1 DDoS protection, they are forwarded back to the MQSS with some internal information such as:

- The type of exception
- The WAN queue assigned to enqueue host inbound traffic
- The DDoS protocol ID (will be used by other levels of DDoS protection). The DDoS ID, in our ICMP case, is 0x900. Looking back at the previous output you can see this value under the `Id` column.

The interface to reach the router's processor is visible, for the MQSS, just like a classic WAN output interface. As said, the LUSS assigned a WAN output queue to host inbound traffic. There are actually eight queues for the internal host interface (like a classic physical port has). To retrieve information about the eight queues attached to the internal host interface you can use the following set of commands on the EA. First, retrieve the output stream index attached to the host interface: look for Host in the Type column.

For our line card the WAN output stream attached to the host interface is 1088. The last `1` means egress direction:

```
(mx vty)# show mqss 0 phy-stream list 1

Egress PHY stream list
----------------------

------------------------------------------------------------------------
Stream    Type        Enabled    NIC    PIC    Connection    L1 Node
Number                           Slot   Slot   Number
------------------------------------------------------------------------
[…]
1087      Loopback    No         255    255    29            255
1088      Host        Yes        255    255    28            0
[…]
```

With this stream ID you can retrieve the eight queue indexes. Here we have queue ID 1008 for Queue 0, 1009 for queue 1, up to 1015 for queue 7:

```
(mx vty)# show mqss 0 phy-stream 1088 1

Egress PHY stream structure
---------------------------
```

```
Stream number                : 1088
Stream type                  : Host
[...]
Queues                       : 1008..1015
Number of Queues             : 8
Stream weight                : 1
```

You can also retrieve similar information by issuing this next command, where the last `0` means the PFE 0. It can be different if you want to check the same information on a different PFE:

```
(mx vty)# show cos halp stream-sched-nodes 0
======================================
[…]
======================================
Stream id: 1088     <<< Host Wan output stream
    L1 index     : 0
    L2 index     : 1984
    L3 index     : 0
    L4 index     : 126
    Base Q index : 1008 <<< First queue
======================================
[…]
```

Finally, you can now collect queue statistics with the next command. Here we display the statistics about queue 0 (absolute queue ID 1008) of the host interface. We are lucky this is the queue that conveys our 200 pps of echo-request. You can see that the host queues are also managed by the XQSS block, just like a classic physical port:

```
(mx vty)# show xqss 0 sched queue 1008 local-stats
Queue:1008
        Forwarded pkts : 666476           200          pps
        Forwarded bytes: 357629164        862536       bps
        Dropped pkts   : 0                0            pps
        Dropped bytes  : 0                0            bps
```

Let's try to do the same on the ZT, which is quite different. There is a virtual interface named `.punt` which represents the internal host interface. Let's first retrieve information about this specific interface:

```
door7302@mx2020> start shell pfe network fpc11
mx2020-fpc11:pfe> show interfaces .punt
 Name: .punt         Index: 49157  IflCount: 1      Type: 0          Weight: 1

 CfgState:     Up    OverallState: Down             Slot: 11         PfeInst: 255
 Local:        Yes   IsAggregate:  No               MTU:  0          PfeId:   0
 LinkState:    Down  Macsec:       Disabled         Pic:  255        PicPort: 65535
 VlanEnabled:  False                                StatsMgr Map: Present
 ChannelCount: 0                                    IfdSpeed:     1000000000
[…]
```

Note the `index` of this interface: here it is `49157`. With this index you can issue a second command to display queue statistics (you might remember we used this command in part one of this chapter to collect output statistics for physical interfaces).

As shown here, we find our 200pps conveyed by queue 0:

```
mx2020-fpc11:pfe> show class-of-service interface queue-stats index 49157
Physical interface : .punt (Interface index:  49157, Egress queues:  8)
Queue: 0
        Queued Packets                  :         5207847             200  pps
        Queued Bytes                    :      3025743683          929544  bps
        Transmitted  Packet             :         5207847             200  pps
        Transmitted Bytes               :      3025743683          929544  bps
        Tail-dropped Packets            :               0               0  pps
        RL-dropped Packets              :               0               0  pps
        RL-dropped Bytes                :               0               0  bps
        RED-dropped Packets             :               0               0  pps
          Low                           :               0               0  pps
          Medium-low                    :               0               0  pps
          Medium-high                   :               0               0  pps
          High                          :               0               0  pps
        RED-dropped Bytes               :               0               0  bps
[…]
Queue: 1
        Queued Packets                  :           57700               3  pps
        Queued Bytes                    :         8451806            3328  bps
[…]
```

There are two other cool commands on the ZT to collect similar information. The first command allows you to retrieve the AFT Token ID associated to the configuration of the scheduler attached to the host interface (we re-use the IFD of the .punt interface as a parameter):

```
mx2020-fpc11:pfe> show class-of-service interface scheduler hierarchy index 49157
Interface Schedulers:
 Name               Type    Index    Level   Node-Token
 .punt              IFD     49157    1       1071
```

And once the AFT token ID is translated with the second command (which is used next), you can see both the scheduler configuration and the queue statistics (there's a preference for this last command):

```
mx2020-fpc11:pfe> show sandbox token 1071
[…]
Node Index:49157
Node Name:.punt
Parent Name:
Interface Rate:1.00Gbps
DelayBufferRate:1.00Gbps

Table: CoS Scheduler AFT Node
-----------------------------------------
   Rate   |  Priority |   Rate  |  Burst  |
   Type   |   Group   |  (bps)  | Size (B)|
----------+-----------+---------+---------+
Guaranteed|    Nominal|     1.0G|    32.8K|
----------+-----------+---------+---------+
    Excess|    Nominal|      1.0|      1.0|
    Excess| StrictHigh|      1.0|      1.0|
    Excess|       High|      1.0|      1.0|
    Excess|   Med High|      1.0|      1.0|
    Excess|    Med Low|      1.0|      1.0|
```

```
   Excess|        Low|      1.0|      1.0|
----------+-----------+---------+---------+
  Maximum|    Nominal|     1.0G|    12.5M|
  Maximum| StrictHigh|      0.0|      0.0|
  Maximum|       High|      0.0|      0.0|
  Maximum|   Med High|      0.0|      0.0|
  Maximum|    Med Low|      0.0|      0.0|
  Maximum|        Low|      0.0|      0.0|
-------------------------------------------
[…]
CoS Scheduler Node:
[…]
Enhanced Priority Mode : 0
Table: Queue Configuration
----------------------------------------------------------------------------------------------------
  Index | Shaping-Rate | Transmit-Rate |  Burst | Weight | G-Priority | E-Priority | Tail-Rule |
WRED-Rule |
--------+--------------+---------------+--------+--------+------------+------------+-----------+
   8    |    1.0G      |     0.0       | 32.8K |  60   |     GL     |     EL     |    639    |    0    |
   9    |    1.0G      |     0.0       | 32.8K |  20   |     GL     |     EH     |    639    |    0    |
  10    |    1.0G      |     0.0       | 32.8K |  127  |     GL     |     EL     |    639    |    0    |
  11    |    1.0G      |     0.0       | 32.8K |  127  |     GL     |     EH     |    639    |    0    |
  12    |    1.0G      |     0.0       | 32.8K |  20   |     GL     |     EL     |    639    |    0    |
  13    |    1.0G      |     0.0       | 32.8K |  10   |     GL     |     EL     |    639    |    0    |
  14    |    1.0G      |     0.0       | 32.8K |   1   |     GL     |     EL     |    639    |    0    |
  15    |    1.0G      |     0.0       | 32.8K |  30   |     GL     |     EL     |    639    |    0    |
--------+--------------+---------------+--------+--------+------------+------------+-----------+
Queue Statistics:
  PFE Instance : 0
                             Transmitted                                              Dropped
                    Bytes                          Packets                    Bytes      Packets
----------------------------------------------------------------------------------------------------
-----------------------------------
 Queue:0     3064034488(     929768 bps)      5273752(       201 pps)          0(          0 bps)
0(         0 pps)
 Queue:1        8550945(        968 bps)        58389(         2 pps)          0(          0 bps)
0(         0 pps)
 Queue:2           6650(          0 bps)           50(         0 pps)          0(          0 bps)
0(         0 pps)
 Queue:3        5478047(       1648 bps)        27951(         2 pps)          0(          0 bps)
0(         0 pps)
 Queue:4              0(          0 bps)            0(         0 pps)          0(          0 bps)
0(         0 pps)
 Queue:5            417(          0 bps)            3(         0 pps)          0(          0 bps)
0(         0 pps)
 Queue:6            143(          0 bps)            1(         0 pps)          0(          0 bps)
0(         0 pps)
 Queue:7              0(          0 bps)            0(         0 pps)          0(          0 bps)
0(         0 pps)
[…]
```

If you wish to know exactly which queue is assigned to the inbound traffic of a specific host, you can issue this command on the EA line card and have a look at the "q#" row:

```
(mx vty)# show ddos asic punt-proto-maps
PUNT exceptions directly mapped to DDOS proto:
```

```
code PUNT name                      group proto         pid q# bwidth  burst
---- --------------------      --------- ------      ---- -- ------ ------
   1 PUNT_TTL                         ttl aggregate    3c00  5   2000  10000
   3 PUNT_REDIRECT               redirect aggregate    3e00  0   2000  10000
   5 PUNT_FAB_OUT_PROBE_PKT    fab-probe aggregate    5700  0  20000  20000
   7 PUNT_MAC_FWD_TYPE_HOST     mac-host aggregate    4100  2  20000  20000
   8 PUNT_TUNNEL_FRAGMENT       tun-frag aggregate    4200  0   2000  10000
[...]
```

And for ZT line card issue this similar command and focus on the "Q" row as well:

```
door7302@mx2020> start shell pfe network fpc11
mx2020-fpc11:pfe> show ddos all-policers
Ddos Policers:

Group         Policer       Id    BWidth  Burst  BWScale BTScale RcvTime P  Q  Rcvd Packets  Drops
Rate(pps) States
host-path      aggregate      0x0000 25000   25000  100    100    300    0  0  0           0           0
ok
resolve       aggregate       0x0100 5000    10000  100    100    300    1  0  0           0           0
ok
resolve       other           0x0101 2000    2000   100    100    300    0  6  0           0           0
ok
resolve       ucast-v4        0x0102 3000    5000   100    100    300    0  6  0           0           0
ok
resolve       mcast-v4        0x0103 3000    5000   100    100    300    0  6  0           0           0
ok
[...]
```

There is also another way to retrieve this information. Indeed you can capture a host inbound packet after the LUSS processing. We explained the packet capture procedure during part one of this chapter, but let's do it again for our specific ping packet. We have to find a pattern inside the packet to filter our capture. So let's use the IPv4 source address followed by IPv4 destination address as the pattern to use for filtering: 0xC0A80301AC10FEFE.

But while doing this on the EA something strange happened!

The packet we captured seems to be coming from the fabric. The output says that our packet was received by the fabric stream 164 (a4), and if you take a look at the hexadecimal dump you should see your matching pattern, but the IPv4 header seems to be cut.

Actually our packets are made of 512 bytes. Therefore as the size is more than 224bbytes, the packet we captured is the HEAD. When the packet is made of a HEAD and a TAIL, the internal header is bigger than when there is no TAIL (this means there are more internal fields appended before the packet). Remember that the packet capture feature looks in the first 32 bytes by default. Thus, if the offset is not correct, since the local memory of the LUSS is not cleared out, you'd think you would get the packets you look for — but it's a trap. In this case we captured a self-fabric probe packet (see the Appendix). If you see the statement `PType CTRL (1)` you can deduce that you captured internal packets/keepalives:

```
(mx vty)# test jnh 0 packet-via-dmem enable
(mx vty)# test jnh 0 packet-via-dmem capture 0x3 C0A80301AC10FEFE
(mx vty)# test jnh 0 packet-via-dmem capture 0x0
(mx vty)# test jnh 0 packet-via-dmem decode

Wallclock: 0x8d6d90ef <<<< Before LUSS Processing
Dispatch: callout 0 error 0 Size 0d8 X2EA_PKT_HEAD (1) ReorderId 172a Stream Fab (a4) PktLen 0200
ChnkTyp 1 TailLen 0140 ChnkDataOff 00 ChnkDataLen 10 ChkSum 00 ChnkPtr 06b1b7b
PType CTRL (1) SubType f PfeMaskF 0 OrigLabel 0 SkipSvc 0 IIF 00000
FC 00 DP 0 SkipSample 0 SkipPM 0 L2Type None (0) IsMcast 0 SrcChas 0 MemId 0
PfeNum 00 PfeMaskE 0 FabHdrVer 1
RwBuf 00000000 SvcCtx 0 ExtHdr 0 PfeMaskE2 0 Token 000000
1b950a4002000000814000010006b1b7b1f0000000000000100000000000000000
 010b57c0a80301ac <<< Some parts of the IPv4 header is missing
 10fefe08002c9800
 0000009cbb38ff72
 e3f5b649786960c0
 000000dadadadabc
 2cbd9a01b6dadada
Wallclock: 0x8d6d9479 <<<< After LUSS Processing
Reorder: EA2X_REORDER_SEND_TERMINATE (d) HasTail 1 Stat 0
ReorderId 172a Color 0 Qop DROP (1) Qsys WAN (0) Queue Drop (0)
Frag 0 RefCnt 0 OptLbk 0 NumTailHoles 00
ChnkTyp 1 TailLen 0140 ChnkDataOff 00 ChnkDataLen 10 ChkSum 00 ChnkPtr 06b1b7b
d972a0200000000005b79050040146d15d00200000000000814000010006b1b7b
 00
```

No worries. We mentioned earlier that you can set an offset to look for after the first 32 bytes. Let's do it:

```
(mx vty)# test jnh 0 packet-via-dmem enable
(mx vty)# test jnh 0 packet-via-dmem capture 0x3 C0A80301AC10FEFE 32
(mx vty)# test jnh 0 packet-via-dmem capture 0x0
(mx vty)# test jnh 0 packet-via-dmem decode
Wallclock: 0x20f548f5 <<<< Before LUSS Processing
Dispatch: callout 0 error 0 Size 0d8 X2EA_PKT_HEAD (1) ReorderId 09d8 Stream Wan (48e) PktLen 01fe
ChnkTyp 1 TailLen 013e ChnkDataOff 00 ChnkDataLen 10 ChkSum 00 ChnkPtr 014d947
IxPreClass 1 IxPort 00 IxMyMac 1
14ec48e001fe0000813e00100014d9474008
 4c96147536196487
 8863455108004500
 01ee000000003f01
Wallclock: 0x20f5a58b <<<< After LUSS Processing
Reorder: EA2X_REORDER_SEND_TERMINATE (d) HasTail 1 Stat 0
ReorderId 09d8 Color 0 Qop ENQUEUE (2) Qsys WAN (0) Queue Host (3f0)
Frag 0 RefCnt 0 OptLbk 0 NumTailHoles 00
ChnkTyp 1 TailLen 013e ChnkDataOff 00 ChnkDataLen 10 ChkSum 00 ChnkPtr 014d947
PType IPV4 (2) Subtype 0 Score 00 Reason 20 AddInfo 00203 IIF 001f0
L2Off 00 L3Off 0e PktLen 01fc StreamType 1 LUid 0 L2iif 00000 OrigPType IPV4 (2) OrigL3Off 0e BDid 0000
DdosProto 0900 FC 00 DP 0 TokenIsOIF 0 Token 000000 FlowId 000000 EgrIIF 00000
```

Looks good this time!

You can see that the packet comes from the input stream 1166 (0x48E) – the CTRL Stream of et-9/0/0/ – the same that was found at the beginning part two in this chapter. After LUSS processing we can finally find what we are looking for: the WAN output queue assigned to this host inbound traffic is 1008 (0x3F8) – this matches with what we saw previously. Don't forget to disable packet capture:

```
(mx vty)# test jnh 0 packet-via-dmem disable
```

If needed you can do it the same thing on the ZT. The pattern to match is a little bit different because the source address is different on this side: C0A80401AC10FEFE

```
mx2020-fpc11:pfe> test jnh packet-via-dmem inst 0 enable
mx2020-fpc11:pfe> test jnh packet-via-dmem-capture inst 0 parcel-type-mask 0x3 match-
string C0A80401AC10FEFE offset 32
mx2020-fpc11:pfe> test jnh packet-via-dmem-capture inst 0 parcel-type-mask 0x0
mx2020-fpc11:pfe> test jnh packet-via-dmem-dump inst 0
```

As usual, disable the packet capture at the end:

```
mx2020-fpc11:pfe> test jnh packet-via-dmem inst 0 disable
```

Now the host packets are dequeued from the WAN queue managed by the XQSS and come back to the MQSS. The WO block puts together a packet HEAD and TAIL (if needed) to rebuild the entire packet. On the EA line card the WO block forwards the packets plus an internal header to the uKernel through the PCIe interface. This track is managed by the TOE (Traffic Offload Engine), a piece of hardware inside the TRIO ASIC that does many things – on the EA it is used to deliver host inbound packets to the uKernel, for instance.

You can, if needed, collect statistics from the TOE. The TOE manages eight streams. For each stream you've got both RX and TX statistics. Pay attention here: RX means packets received by the TOE from the MQSS – those packets will then be sent to the uKernel. TX means packets sent to the MQSS – those packets have been received by the uKernel. So in our case, regarding host inbound traffic, we should have a look at the RX statistics of stream 0 (mapping of the Queue 0):

```
(mx vty)# show toe pfe 0 mqss 0 toe-inst 0 packet-stats stream 0
Stream 0: halt flag is NOT set
   TX Packets
MQSS TOE pfe 0 asic 0 toe 0 mailbox register 12 contains 0x0004b3e9
[…]
   TX Rates:
      packets per second: 1
      descriptors per second: 1
      bytes per second: 81
      descriptors completed since last count:   94
   TX Errors:
[…]
MQSS TOE pfe 0 asic 0 toe 0 mailbox register 11 contains 0x01d28ba8
[…]
   RX Rates:  <<< FROM MQSS and then forwarded to uKERNEL
      packets per second: 202 <<<<< our 200pps pings
      descriptors per second: 202
      bytes per second: 107981
      completed since last count:  9550
   RX Errors:
[…]
```

On the ZT this process is totally different (even if the TOE is still there – it is used, today, to collect some ASIC's statistics, for instance). There is a dedicated internal Ethernet port to send/receive host traffic and this port is used to forward the host inbound traffic to the packetIO module. If you need to collect statistics about the WO stream attached to this given Ethernet port you can follow the next procedure. First retrieve the connection number of the WO stream attached to the internal GE port. Just look for the GE type. Actually there are two internal Ethernet ports per ZT – only the first one is used today. So, for us, the WO stream attached to the internal host port is 1089 and its connection number is 50, as shown here:

```
door7302@mx2020> start shell pfe network fpc11.0
(mx vty)# show mqss 0 phy-stream list 1
Egress PHY stream list
----------------------
------------------------------------------------------------------------
Stream    Type        Enabled    NIC    PIC    Connection    L1 Node
Number                           Slot   Slot   Number
------------------------------------------------------------------------
[...]
1089      GE          Yes        255    255    50            0      <<<< Internal GE port
1090      GE          Yes        255    255    51            255
1091      Crypto      Yes        255    255    53            2
1092      WAN         Yes        11     0      40            4
1100      WAN         Yes        11     0      30            5
[...]
```

With this information you can configure a WO counter for this specific connection number. The `0 50` means counter 0 – connection number 50:

```
(mx vty)# test mqss 0 wo stats conn 0 50
```

Finally, issue the well-known command to display WO statistics and have a look at the counter 0:

```
(mx vty)# show mqss 0 wo stats
WO statistics
-------------
Counter set 0
    Connection number mask   : 0x3f
    Connection number match  : 0x32
    Transmitted packets      : 14766 (203 pps)      <<<< our pings
    Transmitted bytes        : 8162781 (899168 bps)
```

Reset your WO counterback at end:

```
(mx vty)# test mqss 0 wo stats default
```

It's time to leave the ASIC and move to the uKernel (for the EA) or packetIO (for the ZT).

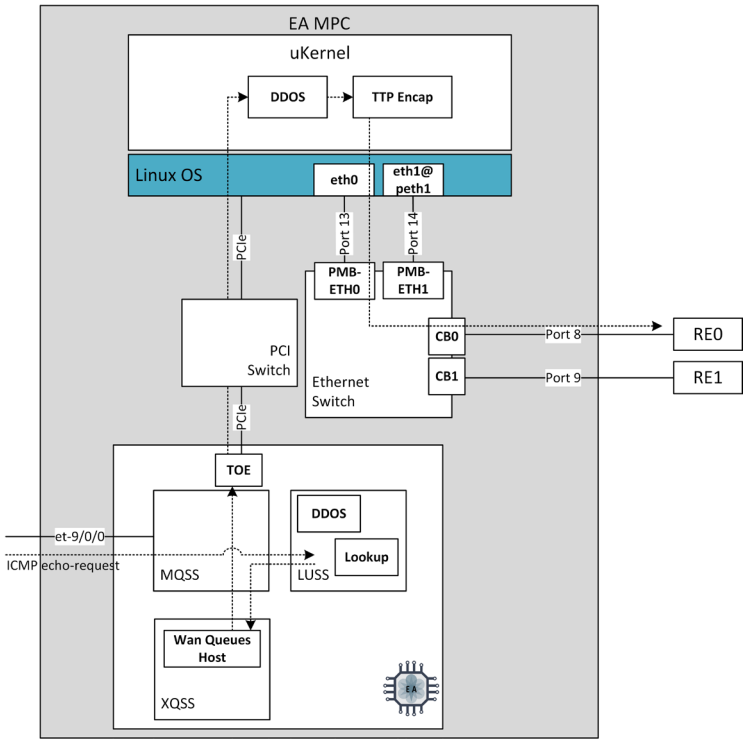
On the EA line card the host traffic coming from the WAN is received on a PCIe interface. It then processes by several threads of the uKernel process. We already mentioned that the inbound traffic is rate-limited by the second level of DDoS

protection. The traffic that is accepted is then encapsulated into a proprietary tunneling protocol named TTP (Trivial Tunneling Protocol). On the EA you can retrieve statistics of the TTP thread by issuing this command:

```
(mx vty)# show ttp statistics
TTP Statistics:
                  Receive    Transmit
                ----------  ----------
 L2 Packets         33850           0
 L3 Packets      21071434           0
 Drops                 0           0
 Netwk Fail            0           0
 Queue Drops           0           0
 Unknown               0           0
 Coalesce              0           0
 Coalesce Fail         0           0

TTP Transmit Statistics:
                  Queue 0     Queue 1     Queue 2    Queue 3
                ----------  ----------  ----------  ----------
 L2 Packets            0           0           0           0
 L3 Packets            0           0           0           0

TTP Receive Statistics:
                  Control        High      Medium         Low     Discard
                ----------  ----------  ----------  ----------  ----------
 L2 Packets            0       21822       12028           0           0
 L3 Packets            0       39261    21032173 << Pings  0           0
 Drops                 0           0           0           0           0
 Queue Drops           0           0           0           0           0
 Unknown               0           0           0           0           0
 Coalesce              0           0           0           0           0
 Coalesce Fail         0           0           0           0           0
```

Just have a look at the information marked as `Receive`. Indeed, like on TOE, `Receive` means packets received by the TTP thread, processed (encapsulated), and forwarded to the RE. Notice, the TTP receive thread has got four queues (control, high, medium, and low). You can also check if there are discards at this level. The traffic, once encapsulated, is pushed to the RE via one Ethernet port attached to the Ethernet Switch, embedded on the line card. The Figure 3.10 illustrates the trip made by the host inbound packets into the EA line card until they reach the RE.

Figure 3.10 Host Inbound Processing On the EA



As you can see in Figure 3.10, the Linux OS is connected to the Ethernet Switch by two Ethernet ports (one per RE). Let's go to the Linux OS of the line card and issue two Linux ip commands:

```
door7302@mx2020> start shell
% su
Password: xxx
mx2020:/var/home/remote-su # rsh -Ji 128.0.0.25

mx2020-fpc9:~# ip link 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode
DEFAULT group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT group default
qlen 1000
    link/ether 02:00:00:00:00:19 brd ff:ff:ff:ff:ff:ff
3: peth1: <BROADCAST,MULTICAST,NOARP,UP,LOWER_UP> mtu 9600 qdisc mq state UP mode DEFAULT group
default qlen 1000
    link/ether 00:00:00:00:00:ff brd ff:ff:ff:ff:ff:ff
4: sit0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN mode DEFAULT group default qlen 1
    link/sit 0.0.0.0 brd 0.0.0.0
5: eth1@peth1: <BROADCAST,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT
group default qlen 1000
    link/ether 02:00:00:00:00:19 brd ff:ff:ff:ff:ff:ff
6: eth_asic@peth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9600 qdisc noqueue state UP mode DEFAULT
group default qlen 1000
    link/ether 00:00:00:00:00:fe brd ff:ff:ff:ff:ff:ff
```

```
mx2020-fpc9:~# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 02:00:00:00:00:19 brd ff:ff:ff:ff:ff:ff
    inet 128.0.0.25/2 brd 191.255.255.255 scope global eth0
       valid_lft forever preferred_lft forever
3: peth1: <BROADCAST,MULTICAST,NOARP,UP,LOWER_UP> mtu 9600 qdisc mq state UP group default qlen 1000
    link/ether 00:00:00:00:00:ff brd ff:ff:ff:ff:ff:ff
4: sit0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1
    link/sit 0.0.0.0 brd 0.0.0.0
5: eth1@peth1: <BROADCAST,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
qlen 1000
    link/ether 02:00:00:00:00:19 brd ff:ff:ff:ff:ff:ff
6: eth_asic@peth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9600 qdisc noqueue state UP group default
qlen 1000
    link/ether 00:00:00:00:00:fe brd ff:ff:ff:ff:ff:ff
```

You can see there are two physical ports:

- eth0
- eth1 renamed peth1: p stands for *physical*
  - peth1 is shared for two usages. There are two sub-interfaces:
  - eth1@peth1 (alias eth1): which is actually similar to eth0
  - eth_asic@peth1 (alias eth_asic): which is the logical interface (VLAN) used to push control plane traffic from the RE to the ASIC (this is the host outbound case and we'll cover it during the next section of this chapter).

The internal IP address (here 128.0.0.25) is used to established sockets with the RE and it is configured either on eth0 or eth1@peth1, depending on if the master routing engine is RE0 or RE1. If RE0 is the master then the IP address will be assigned to eth0: this is the case, here. Eth0 and eth1@peth1 are physically connected to the interface PMB-ETH0 and PMB-ETH1 of the Ethernet Switch. As observed, both REs are also remotely connected to this switch by one Ethernet port: the CB0 port attached to RE0 and CB1 to RE1. Remember there is also an Ethernet switch embedded on the RE/CB. The information related to this RE Ethernet switch is accessible by issuing these following commands:

- `show chassis ethernet-switch`
- `show chassis ethernet-switch statistics <port>`

If needed, you can also collect statistics of the embedded Ethernet switch on the EA line card. Let's issue the following command to display how physical ports of the Ethernet Switch are connected:

```
(mx vty)# show mesw ports
GE-Port   | Link  | Speed | Auto-Neg | Port-map
-----------------------------------------------
0-0        UP       1000      Disabled     EA2-0
0-1        UP       1000      Disabled     EA2-1
0-2        UP       1000      Disabled     EA3-0
0-3        UP       1000      Disabled     EA3-1
0-4        UP       1000      Disabled     EA0-0
0-5        UP       1000      Disabled     EA0-1
0-6        UP       1000      Disabled     EA1-0
0-7        UP       1000      Disabled     EA1-1
0-8        UP       1000      Disabled     CB0  << to RE0
0-9        UP       1000      Disabled     CB1  << to RE1
0-10       UP       1000      Disabled     PTP-1588
0-11       DOWN     1000      Disabled     MIC0
0-12       DOWN     1000      Disabled     MIC1
0-13       UP       1000      Disabled     PMB-ETH0  << to eth0
0-14       UP       1000      Disabled     PMB-ETH1  << to eth1@peth1
0-15       DOWN     1000      Disabled     N/C
```

We refind the four ports, already mentioned, attached to the Linux OS and remotely to both REs. But, as observed, you can see there are also two Ethernet ports directly connected to each EA ASIC. The first port (EAx-0) is used for the PTP synchronization. The second port (EAx-1) is used to convey host outbound traffic – more detail about it later. You can also display the VLAN assignment by using this second command and thus deduce which port can communicate with other ports:

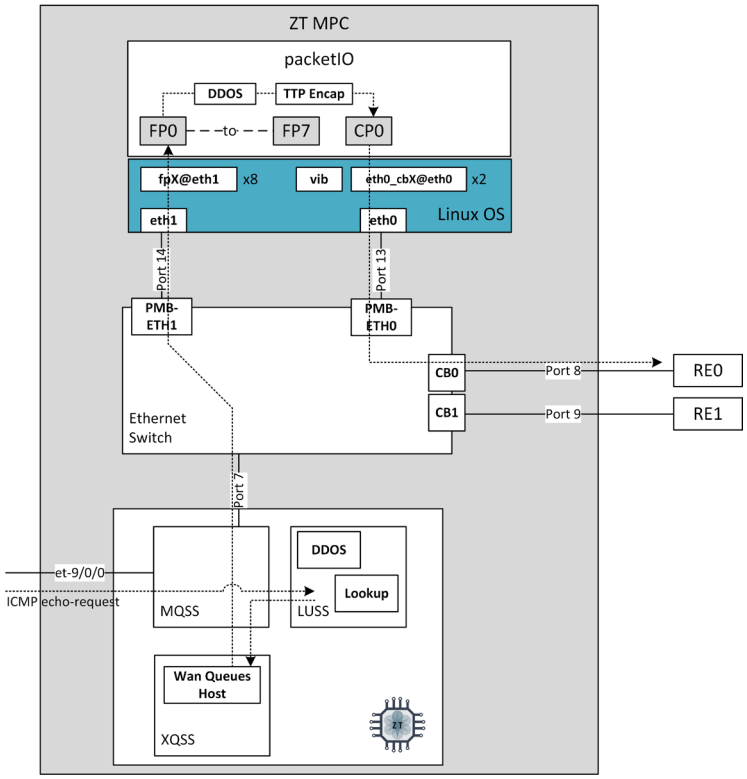
```
(mx vty)# show mesw port_vlan 0
MESW port VLAN assignments
Port        VID
---------------
0           3
1           2
2           3
3           2
4           3
5           2
6           3
7           2
8           1
9           2
10          3
11          3
12          3
13          1
14          2
15          1
```

Finally, you can retrieve per port statistics by issuing the next command. The first number means the switch number, which is always 0. The second number is the port number. Let's display statistics of the PMB-ETH0 port (port 13):

```
(mx vty)# show mesw statistics 0 13
Traffic statistics
             34229350 good octets received
            134562896 good octets sent
 Packet statistics
          0 Undersize packets received
          0 Oversize packets received
          0 Jabber packets received
 Frame statistics
          0 MAC Transmit errors
     259317 MAC good frames received
    1551776 MAC Good frame sent
          0 MAC bad frames received
          0 bad octets received
     259305 Broadcast frames received
    1551775 Broadcast frames sent
         10 Multicast frames received
          0 Multicast frames sent
    1033403 Frames of 64 byte size
         12 Frames of 65 to 127 byte size
     777677 Frames of 128 to 255 byte size
          0 Frames of 256 to 511 byte size
          0 Frames of 512 to 1023 byte size
          0 Frames of size 1024 and more
          0 Excessive collisions
          0 Unrecognized MAC control frames received
          0 Flow control frames sent
          0 Good flow control messages received
          0 Drop events
          0 Fragemnts received
 Other errors
          0 MAC Receive error
          0 CRC errors
          0 Collisions in MAC
          0 Late collisions in MAC
```

On the ZT line card, we leave the ASIC by using the Ethernet port attached to the ZT. The physical and logical internal connectivity is quite different on the ZT. Figure 3.11 illustrates this.

Figure 3.11 Host Inbound Processing On ZT



On the ZT, there are also two physical interfaces, eth0 and eth1, but on this type of line card the eth0 is dedicated to communicate with the upper level (the REs), and the eth1 to the lower level (the ZT ASICs). Let's again issue the two `ip` commands on the Linux OS of the ZT line card:

```
door7302@mx2020> start shell
% su
Password: xxx
mx2020:/var/home/lab # ssh -Ji root@128.0.0.27
mx2020-fpc11:/# ip link
[…]
2: eth0: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT group
default qlen 1000
    link/ether 02:00:00:00:00:1b brd ff:ff:ff:ff:ff:ff
3: eth1: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 9600 qdisc mq state UP mode DEFAULT group
default qlen 1000
    link/ether 00:00:00:00:00:ff brd ff:ff:ff:ff:ff:ff
[…]
8: vib: <BROADCAST,MULTICAST,NOARP,PROMISC,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT
group default qlen 1000
```

```
    link/ether 02:00:00:00:00:1b brd ff:ff:ff:ff:ff:ff
9: eth0_cb0@eth0: <BROADCAST,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT
group default qlen 1000
    link/ether 02:00:00:00:00:1b brd ff:ff:ff:ff:ff:ff
10: eth0_cb1@eth0: <BROADCAST,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue master vib state UP
mode DEFAULT group default qlen 1000
    link/ether 02:00:00:00:00:1b brd ff:ff:ff:ff:ff:ff
11: eth_fp0@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9600 qdisc noqueue state UP mode DEFAULT group
default qlen 1000
    link/ether 00:00:00:00:05:fe brd ff:ff:ff:ff:ff:ff
12: eth_fp1@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9600 qdisc noqueue state UP mode DEFAULT group
default qlen 1000
    link/ether 00:00:00:00:04:fe brd ff:ff:ff:ff:ff:ff
[…]

mx2020-fpc11:/# ip addr
[…]
2: eth0: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 02:00:00:00:00:1b brd ff:ff:ff:ff:ff:ff
    inet6 fe80::ff:fe00:1b/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 9600 qdisc mq state UP group default qlen 1000
    link/ether 00:00:00:00:00:ff brd ff:ff:ff:ff:ff:ff
    inet6 fe80::ff:fe00:1b/64 scope link
       valid_lft forever preferred_lft forever
[…]
8: vib: <BROADCAST,MULTICAST,NOARP,PROMISC,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
qlen 1000
    link/ether 02:00:00:00:00:1b brd ff:ff:ff:ff:ff:ff
    inet 128.0.0.27/2 scope global vib
       valid_lft forever preferred_lft forever
9: eth0_cb0@eth0: <BROADCAST,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
default qlen 1000
    link/ether 02:00:00:00:00:1b brd ff:ff:ff:ff:ff:ff
10: eth0_cb1@eth0: <BROADCAST,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue master vib state UP
group default qlen 1000
    link/ether 02:00:00:00:00:1b brd ff:ff:ff:ff:ff:ff
11: eth_fp0@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9600 qdisc noqueue state UP group default qlen
1000
    link/ether 00:00:00:00:05:fe brd ff:ff:ff:ff:ff:ff
12: eth_fp1@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9600 qdisc noqueue state UP group default qlen
1000
    link/ether 00:00:00:00:04:fe brd ff:ff:ff:ff:ff:ff
[…]
```

Actually there are two sub interfaces, eth0_cb0 and eth0_cb1, and they are dedicated, respectively, to communicate with RE0 and RE1. Only one sub interface is used by packetIO depending on which RE is the master. There is also a virtual interface named `vib` that hosts the IP address of the MPC to communicate with the REs. Depending on which RE is primary, the vib interface is bounded to the eth0_cbX interface. Hereafter, RE0 is primary, so eth0_cb0 is currently used and therefore bounded to vib virtual Layer 3 port:

```
mx2020-fpc11:/# brctl show vib
bridge name     bridge id               STP enabled     interfaces
vib             8000.02000000001b       no              eth0_cb1
```

On the other side, depending on the number of ZT ASICs, there is one sub interface (physically attached to eth1) used by packetIO to communicate with each ZT ASIC: those interfaces are named like that: eth_fp0, eth_fp1 … until eth_fp7 (on MPC11e).

The ZT MPC provides some cool packetIOs statistics. You can first use this command to see statistics of all packetIO interfaces:

```
door7302@mx2020> start shell pfe network fpc11
mx2020-fpc11:pfe> show host-path ports
Name               State      Tx-Packets    Tx-Rate(pps)  Rx-Packets    Rx-Rate(pps)  Drop-
Packets  Drop-
Rate(                                                                                    pps)
=============================================================================================
fp7                READY      112232        3             30564        3             0       0
fp6                READY      21747         1             21788        1             0       0
ppm0               READY      55895         1             56107        1             0       0
pktin0             READY      0             0             81272        0             0       0
am0                READY      0             0             0            0             0       0
cp0                READY      930967        201           938639       202           0       0
fp0                READY      908022        201           898920       201           0       0
fp1                READY      21747         1             21762        1             0       0
fp2                READY      72368         1             65404        1             0       0
fp3                READY      21747         1             21860        1             0       0
fp4                READY      21747         1             21831        1             0       0
fp5                READY      70319         1             79284        1             0       0
```

This command shows us that fp0 sub interface is currently receiving 200pps (these are our ping echo-request coming from ZT ASIC 0 (PFE 0)). There is also a nice python script available on the Linux host OS the line card that allows you to monitor in real time the above statistics (like monitor "monitor interface traffic"):

```
door7302@mx2020> start shell
% su
Password:
# ssh -Ji root@128.0.0.27
Last login: Wed May 20 15:21:43 2020 from 128.0.0.1
--- JUNOS 20.1R1.3-EVO Linux (none) 4.8.28-WR2.2.1_standard-
gcf18df4 #1 SMP PREEMPT Tue Jan 28 19:10:46 PST 2020 x86_64 x86_64 x86_64 GNU/Linux
mx2020-fpc11:/# monitor_interface_traffic.py
```

These packets have been processed by packetIO modules (DDoS protection Level 2, TTP encapulation) and forwarded to the RE by using the CP0 interface (200pps in the TX direction). The CP0 is linked to the logical interface eth0_cb0 as the master RE is currently RE0. Eth0_cb0 is itself attached to physical port eth0.

Remember, each FP interface has eight ingress/egress queues and CP0 has only eight egress queues. You can display detailed statistics of FP or CP interfaces.

For our given FP interface, you will see the 200pps ping-echo requests are received by the TX queue 4. The RX 200pps displayed are actually the echo-reply coming from the RE (we will cover that later):

```
door7302@mx2020> start shell pfe network fpc11
mx2020-fpc11:pfe> show host-path ports fp0
Host Path Port
    Name   : fp0
[…]
Packet Stats:
Name                  Tx-Packets    Tx-Rate(pps)      Rx-Packets    Rx-Rate(pps)
================================================================================
If                    1027899       201               1018823       202
Transport             1027899       201               1018823       202
Io                    1027899       201               1020650       202
Io-Prio-0             942738        200<<egress Q     0             0
Io-Prio-1             0             0                 0             0
Io-Prio-2             0             0                 0             0
Io-Prio-3             62820         0                 0             0
Io-Prio-4             0             0                 950302        200<<ingress Q
Io-Prio-5             0             0                 23            0
Io-Prio-6             0             0                 51277         2
Io-Prio-7             22341         1                 17221         0
```

Now issue the same command for the CP0 port. As mentioned earlier, the CP0 interface performs only egress queuing. Hereafter, we will see the 200pps of ping echo-requests processed by packetIO, enqueued in TX queue 4, and finally forwarded to RE:

```
mx2020-fpc11:pfe> show host-path ports cp0
Host Path Port
    Name   : cp0
[…]
Packet Stats:
Name                  Tx-Packets    Tx-Rate(pps)      Rx-Packets    Rx-Rate(pps)
================================================================================
If                    1097239       201               1104877       201
Transport             1097239       201               1104877       201
Io                    1097239       201               1104877       201
Io-Prio-0             66            0                 0             0
Io-Prio-1             0             0                 0             0
Io-Prio-2             0             0                 0             0
Io-Prio-3             0             0                 0             0
Io-Prio-4             1008559       200<<egress Q     0             0
Io-Prio-5             72            0                 0             0
Io-Prio-6             88542         1                 0             0
Io-Prio-7             0             0                 0             0
```

Finally, it is also possible on the ZT line card to collect information about the embedded Ethernet Switch. These statistics are managed by the platformd module; therefore you have to connect to the old shell:

```
door7302@mx2020> start shell pfe network fpc11.0
(mx vty)# show esw ports
Dev/Port         Mode          Link   Speed  Duplex  Loopback Mode  Port-map
---------  -----------------  -----  -----  ------  -------------  ---------
0/0          1000_BaseX       Up      1G     Full    None           ZT7
0/1          1000_BaseX       Up      1G     Full    None           ZT6
0/2          1000_BaseX       Up      1G     Full    None           ZT5
0/3          1000_BaseX       Up      1G     Full    None           ZT4
```

```
0/4           1000_BaseX      Up     1G     Full    None           ZT3
0/5           1000_BaseX      Up     1G     Full    None           ZT2
0/6           1000_BaseX      Up     1G     Full    None           ZT1
0/7           1000_BaseX      Up     1G     Full    None           ZT0
0/8           1000_BaseX      Up     1G     Full    None           CB0
0/9           1000_BaseX      Up     1G     Full    None           CB1
0/10             n/a         Down    n/a    Full    N/A            N/C
0/11             n/a         Down    n/a    Full    N/A            N/C
0/12             n/a         Down    n/a    Full    N/A            N/C
0/13              KR          Up     10G    Full    None           PMB-ETH0
0/14              KR          Up     10G    Full    None           PMB-ETH1
0/15             n/a         Down    n/a    Full    N/A            N/C
```

To retrieve statistics of a given port, issue the second command. The first number means the switch number, which is always `0`. The second number is the port number:

```
(mx vty)# show esw statistics 0 13
Good Octets Received:             780595144
Bad Octets Received:              0
MAC Transmit Error:               0
BRDC Packets Received:            46174
MC Packets Received:              0
Size 64:                          135
Size 65 to 127:                   849229
Size 128 to 255:                  342789
Size 256 to 511:                  68884
Size 512 to 1023:                 2145479
Size 1024 to 1518:                0
Size 1519 to Max:                 0
Good Octets Sent:                 1842085313
Excessive Collision:              0
MC Packets Sent:                  0
BRDC Packets Sent:                271584
FC Sent:                          0
Good FC Received:                 0
Drop Events:                      0
Undersize Packets:                0
Fragments Packets:                0
Oversize Packets:                 0
Jabber Packets:                   0
MAC RX Error Packets Received:    0
Bad CRC:                          0
Collisions:                       0
Late Collision:                   0
FC Received:                      0
Good UC Packets Received:         1631327
Good UC Packets Sent:             2270946
Multiple Packets Sent:            0
Deferred Packets Sent:            0
```

Both our streams of 200pps have successfully passed the DDoS protection levels 1 and 2, respectively, managed by the ASIC and the line card's CPU. Now those two flows arrive on the RE (through the TTP tunnel).

Remember TTP is used between the MPC and the RE to encapsulated control plane/OAM packets. The TTP header allows conveying the internal piece of

information collected, in particular, during LUSS processing. For instance, the TTP header carries the type of packet and the DDoS protocol identifier ,which will be used by the third DDoS protection level.

Is it possible to decode the TTP header?

Actually yes! Let's first identify the internal port used by your RE to communicate with the MPCs.

The setup in this chapter uses an MX2020 with the RE-MX2000-1800X4 model. On this type of RE, TTP traffic coming from MPC is received on the em0 internal Ethernet port..

How do I know that?

First you need to refer to this document: https://www.juniper.net/documentation/en_US/release-independent/junos/topics/reference/general/routing-engine-m-mx-t-series-support-by-chassis.html .

For all models of routers and all models of REs available it lists the name(s) of the internal interfaces (have a look at the Internal Ethernet Interface column). In our case, the output table mentions two internal ports: em0, em1.

To find which port is used to communicate with the MPCs, you just have to issue this hidden command (from the classic Junos CLI):

```
door7302@mx2020> show tnp addresses | match "IF|fpc"
   Name                TNPaddr   MAC address    IF     MTU E H R
fpc6                    0x16 02:00:00:00:00:16 em0    1500 4 0 3
fpc9                    0x19 02:00:00:00:00:19 em0    1500 4 0 3
fpc11                   0x1b 02:00:00:00:00:1b em0    1500 2 0 3
fpc18                   0x22 02:00:00:00:00:22 em0    1500 2 0 3
```

Have a look at the IF column: for us this is the em0, which should receive (and send) TTP traffic coming from (to) the MPCs.

The last step before decoding the TTP: we must find the IP address of the MPCs.

We already said that each MPC has an internal IP address within the range 128.0.0.0/2. To deduce the IP address assigned to a given MPC, follow this formula:

- The IP of an MPC = 128.0.0.[16 + MPC slot number]
- The master RE always has the IP 128.0.0.1.

Now, we have everything we need to capture and decode, for example, TTP traffic coming from the MPC in slot 9. The MPC in slot 9 has the IP address 128.0.0.25 (16+9). We now use the classic `monitor traffic interface` command to capture the TTP traffic coming from MPC in slot 9:

```
door7302@mx2020> monitor traffic interface em0 matching "ip src 128.0.0.25" layer2-headers detail
In IP (tos 0x0, ttl  64, id 33897, offset 0, flags [DF], proto: TTP (84), length: 542) 128.0.0.25 >
128.0.0.1: TTP, type L3-rx (3), ifl_input 324, pri medium (3), length 502, proto ipv4 (2), hint(s)
[none] (0x00008010), queue 0, nh_index 884
          TTP TLV's:
          Num :: Type - Length - Value
          ----------------------------
            1 ::   2 -   4   - 0x9 0x0 0x0 0x0
          ----------------------------
          ifd_mediatype Ethernet (1), ifl_encaps Ethernet (14), cookie-len 0, payload IP
       -----payload packet-----
       IP (tos 0x0, ttl  63, id 0, offset 0, flags [none], proto: ICMP (1), length: 494) 192.168.3.1 >
172.16.254.254: ICMP echo request, id 0, seq 0, length 474
```

It's magic! Junos's tcpdump decodes the TTP. We can retrieve, as mentioned, the type of traffic, the protocol, the DDoS protection ID (0x900 = ICMP aggregate), and finally, our ICMP echo-request packet (tunneled).

Once decapsulated by the master RE, the packets are handled by Level 3 of DDoS protection (packets had been already identified as ICMP aggregate traffic by the LUSS – with information conveyed in the TTP header):

```
door7302@mx2020> show ddos-protection protocols icmp statistics terse
Packet types: 1, Received traffic: 1, Currently violated: 0

Protocol    Packet      Received        Dropped        Rate     Violation State
group       type        (packets)       (packets)      (pps)    counts
icmp        aggregate   3423801         0              400      0         ok
```

You can see that both streams of 200pps each are well accepted by the third level of policing, as we didn't violate (the sum of both flows = 400pps) the threshold of 20Kpps.

This last command ends the explanation about how host inbound traffic is managed by the EA and ZT Line cards. We are now moving to the last segment in following the RE's answers to our 400pps echo-requests.

Tracking Host Outbound Traffic

It's time for the IP stack of the RE to reply to the echo-requests.

The echo-reply packets are built by the RE kernel and we can easily see them by using the well-known `monitor traffic interface` command:

```
door7302@mx2020> monitor traffic interface et-9/0/0 no-resolve matching "icmp[icmptype]==icmp-
echoreply" layer2-headers
Listening on et-9/0/0, capture size 96 bytes
Out 4c:96:14:75:36:19 > 64:87:88:63:45:51, ethertype IPv4 (0x0800), length 74: truncated-ip - 434
bytes missing! 172.16.254.254 > 192.168.3.1: ICMP echo reply, id 0, seq 0, length 474
Out 4c:96:14:75:36:19 > 64:87:88:63:45:51, ethertype IPv4 (0x0800), length 74: truncated-ip - 434
bytes missing! 172.16.254.254 > 192.168.3.1: ICMP echo reply, id 0, seq 0, length 474
Out 4c:96:14:75:36:19 > 64:87:88:63:45:51, ethertype IPv4 (0x0800), length 74: truncated-ip - 434
bytes missing! 172.16.254.254 > 192.168.3.1: ICMP echo reply, id 0, seq 0, length 474
```

You can notice the Layer 2 header is already computed by the RE. Those echo-replies are then pushed to the right MPC (found by the RE after a kernel lookup based on its local RIB) on which the packets must be sent out. The prior command output showed us the packets before their TTP encapsulation. If you want to see the TTP encapsulation you must capture traffic on the em0, as we did earlier. Let's do it. Here only traffic sent to the MPC in slot 11 (the IP address 128.0.0.27 (16+11)):

```
door7302@mx2020> monitor traffic interface em0 matching "ip dst 128.0.0.27" detail layer2-headers
Out 02:01:01:00:00:05 > 02:00:00:00:00:1b, ethertype IPv4 (0x0800), length 570: (tos 0x0, ttl  64, id
11889, offset 0, flags [none], proto: TTP (84), length: 556) 128.0.0.1 > 128.0.0.27: TTP, type L2-tx
(2), ifd_output 228, pri unknown (0), length 516, proto unkwn (0), hint(s) [no key lookup]
(0x00009009), queue 0, nh_index 0
            TTP TLV's:
            Num :: Type - Length - Value
            ----------------------------
              1 ::   21 -   4    - 0x0 0x0 0x1 0xf4
            ----------------------------
            ifd_mediatype Ethernet (1), ifl_encaps Ethernet (14), cookie-len 0, payload ETHER
        -----payload packet-----
      4c:96:14:75:37:ab > 64:87:88:63:45:a3, ethertype IPv4 (0x0800), length 516: (tos 0x0, ttl  64,
id 11887, offset 0, flags [none], proto: ICMP (1), length: 494) 172.16.254.254 > 192.168.4.1: ICMP echo
reply, id 0, seq 0, length 474
```

As shown in the output, the echo-reply is totally embedded (Layer 2 included) into the TTP packet. The TTP provides some pre-computed information to the lower layers, especially the egress interface (IFD and the IFL (provided through the TLV)) and the queue assigned (this is the relative WAN queue number: 0 to 7). Here queue 0 is assigned. Remember host outbound queue assignment is something configurable and there are two ways to modify the default host queue assignment, either:

- With the knob `set class-of-service host-outbound-traffic`
- By using an egress firewall filter applied to the loopback0.

We are going to illustrate the modification of the host queue assignment by using the first knob that allows us to globally modify the host outbound queue. First of all, let's have a look back at our CoS configuration:

```
{master}[edit class-of-service]
door7302@mx2020# show
[…]
forwarding-classes {
    class BEST_EFFORT queue-num 0 priority low;
    class SILVER queue-num 1 priority high;
    class GOLD queue-num 2 priority high;
    class PREMIUM queue-num 4 priority high;
    class HOST_GEN queue-num 3 priority high;
    class CONTROL_PLANE queue-num 5 priority high;
}
[…]
```

We have six forwarding-classes. The aim of our test is to force the host outbound traffic to use the FC HOST_GEN – queue number 3. Let's commit this piece of config:

```
door7302@mx2020> edit exclusive
{master}[edit]
door7302@mx2020# set class-of-service host-outbound-traffic forwarding-class HOST_GEN
door7302@mx2020# commit and-quit
```

Now, let's issue back the previous `monitor traffic interface` command and check back at the TTP header:

```
monitor traffic interface em0 matching "ip dst 128.0.0.27" detail layer2-headers
Out 02:01:01:00:00:05 > 02:00:00:00:00:1b, ethertype IPv4 (0x0800), length 570: (tos 0x0, ttl  64, id
9305, offset 0, flags [none], proto: TTP (84), length: 556) 128.0.0.1 > 128.0.0.27: TTP, type L2-tx
(2), ifd_output 228, pri unknown (0), length 516, proto unkwn (0), hint(s) [no key lookup]
(0x10009009), queue 3, nh_index 0
          TTP TLV's:
            Num :: Type - Length - Value
            ------------------------------
              1 ::   21 -   4    - 0x0 0x0 0x1 0xf4
            ------------------------------
            ifd_mediatype Ethernet (1), ifl_encaps Ethernet (14), cookie-len 0, payload ETHER
       -----payload packet-----
      4c:96:14:75:37:ab > 64:87:88:63:45:a3, ethertype IPv4 (0x0800), length 516: (tos 0x0, ttl  64,
id 9303, offset 0, flags [none], proto: ICMP (1), length: 494) 172.16.254.254 > 192.168.4.1: ICMP echo
reply, id 0, seq 0, length 474
```

As you can now see, the queue field of the TTP header has the value 3, which is, based on our CoS configuration, the HOST_GEN forwarding class.

Let's keep this CoS configuration enabled for the rest of this section.

On the EA line card, the TTP traffic, transmitted by the RE, is handled in two different ways depending on the configuration. By default, and since Junos 17.4, an enhanced mode named *TurboTX* is available and allows handling high rates of control plane packets generated by the RE. Before this feature was introduced, the TTP handling and decapsulation was performed by a dedicated thread of the uKernel. Remember that uKernel is a single mono-core process.

To enhance the MPC's performances, a new dedicated process has been developed for transmitting more control plane packets per second. Turbo-TX is using the second CPU core on the line card, which is mostly idle. This way it can achieve ~80kpps transmit side. In addition, with a raw socket filter (BPF filter) the TurboTX feature allows bypassing the uKernel for the management of some TTP packets (usually L2 packets directly generated by the RE). The TurboTX process removes the TTP header, adds another internal header understandable by the EA ASICs, and finally forwards the control plane traffic to a dedicated Ethernet port attached to the EA ASIC. Actually it's a sub-interface named eth_asic attached physically to the eth1 port.

When TurboTX is not available, or disabled by using the `set chassis turbotx-disable` command, the TTP header is handled and removed by the uKernel, which then forwards the control plane packets (with additional info) to the EA ASIC through the PCIe interface. Without TurboTX, packets coming in on the ASIC from the PCIe interface are received by the TOE block.

On the EA, with TurboTX enabled, you can collect statistics of the TurboTX process by issuing the following command (have a look at the second part of the output regarding TurboTX):

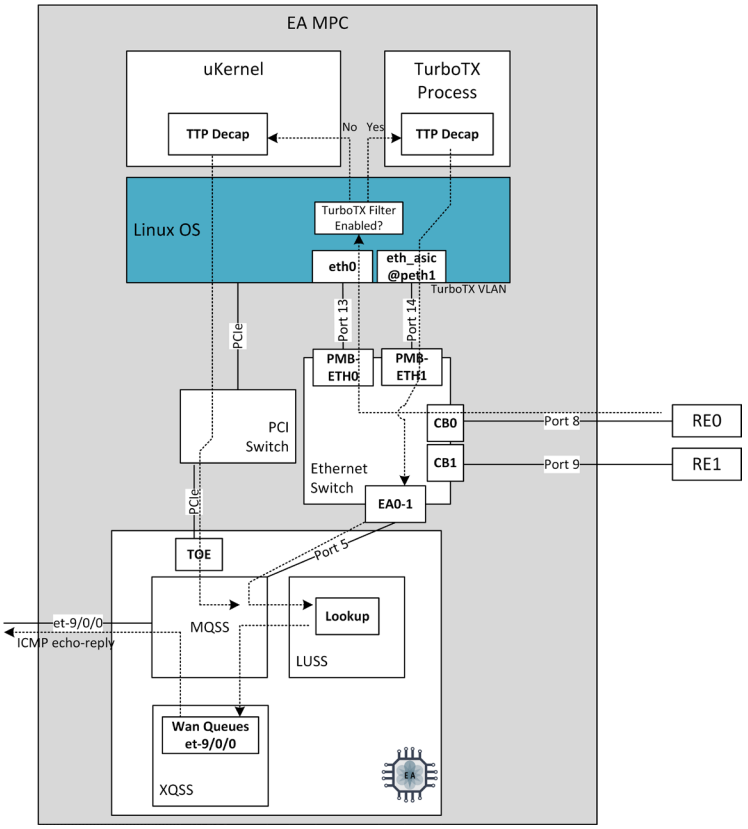
```
(mx vty)# show ttp statistics
[…]
Turbotx Stats:
TTP Receive Stats:
--------------
                      Num Recv  3063964 (200/sec) <ping with TTP header from RE
                 Num Recv Errors        0 ( 0 /sec)
                Num Flush Writes        0 ( 0 /sec)
                 Num Flush Reads        0 ( 0 /sec)


TTP Transmit Stats:
---------------
                Num Xform Fails        0 ( 0 /sec)
                      Num Sends  3063964 ( 200 /sec) <ping without TTP to EA
                 Num Send Fails        0 ( 0 /sec)
                      Num flush  2964777 ( 190 /sec)
                 Num Parse Fail        0 ( 0 /sec)
       Num Platform Xform Fails        0 ( 0 /sec)
           Num Invalid Instance        0 ( 0 /sec)
     Num Ukern Reroute Packets        0 ( 0 /sec)

TX Ring Stats:
--------------
        Queued Packet  : 0
        Flush Success  : 2964777
        Flush Fail     : 0
        Flush Retry    : 0

Buffer Pool Stats:
------------------
        Num Buf Allocs : 3065051
        Num Buf Frees  : 3063964

State Manager:
--------------
        Init OK        : yes
        DB PID         : 963
        Poller Errors  : 0

Total Stats:
------------
        Receives       : 3063964
        Receive Drops  : 0
        Send           : 3063964
        Send Drops     : 0
```

The same command can be used when TurboTX is disabled to check the TTP statistics received and handled by the TTP thread of the uKernel. You can also check when TurboTX is not enabled in the TOE statistics. Remember TX stats on TOE means received from upper layer (via PCIe) and transmitted internally to the ASIC. Let's issue the following command after disabling the TurboTX feature:

```
(mx vty)# show toe pfe 0 mqss 0 toe-inst 0 packet-stats
Stream 0: halt flag is NOT set
   TX Packets
MQSS TOE pfe 0 asic 0 toe 0 mailbox register 12 contains 0x000d843b
[…]
   TX Rates:
      packets per second: 202 <<< from uKernel and TX to MQSS
      descriptors per second: 202
      bytes per second: 105140
      descriptors completed since last count:   266
   TX Errors:
[…]
```

Let's enable the TurboTX feature back and have a look at Figure 3.12, which summarizes what we discussed before and gives you an overview of the end of the trip for the host outbound traffic inside the egress EA ASIC.

Figure 3.12 Host Outbound Processing On the EA



If needed you can again collect statistics on the Ethernet switch by using these two commands:

- `show mesw ports`
- `show mesw statistics <port-number>`

On the ZT line card, it's quite different. Since the software that manages all host traffic has been completely rethought (packetIO), there is no need for the TurboTX feature. Indeed, TTP packets coming from the RE are received by the CP0 interface, processed by the packetIO module (TTP decap, adding internal header), and then they are enqueued in one queue of the FPx interface attached to the right ZT ASIC, and finally forwarded to the ZT ASIC via the dedicated Ethernet port. Once again you can check packetIO statistics by issuing the following commands:

```
door7302@mx2020> start shell pfe network fpc11
mx2020-fpc11:pfe> show host-path ports
Name               State      Tx-Packets    Tx-Rate(pps)  Rx-Packets    Rx-Rate(pps)  Drop-
Packets  Drop-Rate(pps)
================================================================================================
fp7                READY      470563        3             207474       3             4       0
fp6                READY      107928        1             107969       1             0       0
ppm0               READY      194360        1             204942       1             0       0
pktin0             READY      0             0             262654       0             0       0
am0                READY      0             0             0            0             0       0
cp0                READY      8068891       201           7808200      202<from RE   0       0
fp0                READY      7846013       202<to ASIC   8026174      201           0       0
fp1                READY      107928        1             107943       1             0       0
fp2                READY      196399        1             258101       1             0       0
fp3                READY      107928        1             108041       1             0       0
fp4                READY      107928        1             108012       1             0       0
fp5                READY      186381        1             195454       1             0       0
```

We can display detailed statistics for the CP0 interface. Notice there is no ingress queuing on CP0 interface – this is why all TTP traffic from the RE are aggregated in one RX counter:

```
mx2020-fpc11:pfe> show host-path ports cp0
Host Path Port
    Name   : cp0
Packet Stats:
```

```
Name                  Tx-Packets    Tx-Rate(pps)      Rx-Packets    Rx-Rate(pps)
================================================================================
If                    8100523       201               7839810       201
Transport             8100523       201               7839810       201
Io                    8100523       201               7839810       201 <from RE
Io-Prio-0             8229          0                 0             0
Io-Prio-1             0             0                 0             0
Io-Prio-2             0             0                 0             0
Io-Prio-3             0             0                 0             0
Io-Prio-4             7732894       200               0             0
Io-Prio-5             288           0                 0             0
Io-Prio-6             359112        1                 0             0
```

And then, we can do the same for FP0 interface. As seen, the host outbound packets are enqueued in queue number 3 of the FP0 interface and finally forwarded to the ZT ASIC 0:

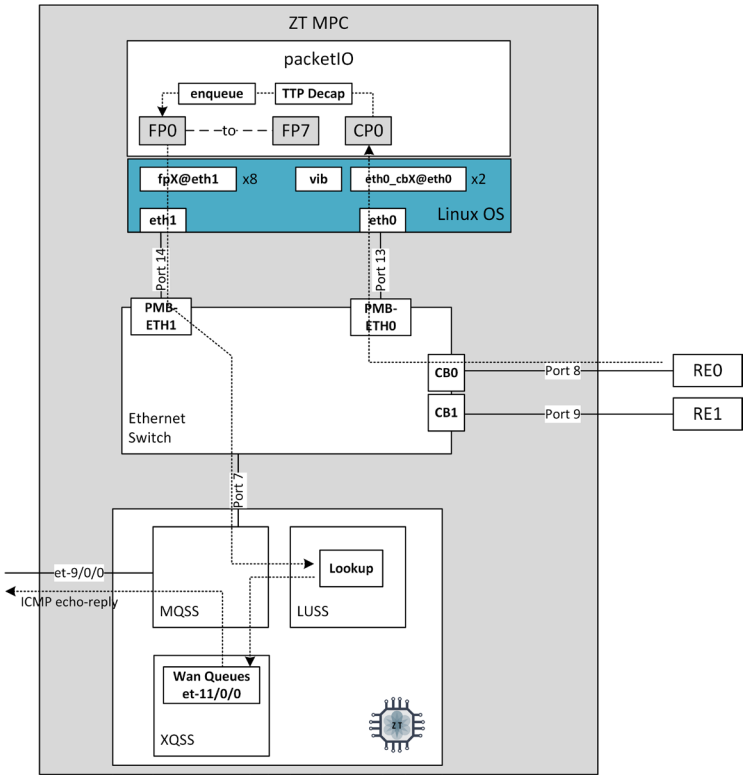
```
mx2020-fpc11:pfe> show host-path ports fp0
Host Path Port
    Name   : fp0
Packet Stats:
Name                  Tx-Packets    Tx-Rate(pps)      Rx-Packets    Rx-Rate(pps)
================================================================================
If                    7901509       201               8081700       201
Transport             7901509       201               8081700       201
Io                    7901508       201               8156582       201
Io-Prio-0             3742658       0                 0             0
Io-Prio-1             0             0                 77            0
Io-Prio-2             0             0                 0             0
Io-Prio-3             4050647       200< to ZT Asic   0             0
Io-Prio-4             0             0                 7736021       200
Io-Prio-5             0             0                 104           0
Io-Prio-6             0             0                 290511        1
Io-Prio-7             108204        1                 54987         0
```

Like on the EA, on the ZT line card you can issue the following set of commands to retrieve statistics of the Ethernet switch embedded on the line card (use the old shell for those commands):

- `show esw ports`
- `show esw statistics <port-number>`

Figure 3.13 provides a schematic view of how the host outbound packets are handled by the ZT line card after we began our trip a few chapters ago.

Figure 3.13 Host Outbound Processing On the ZT



Now we are back again inside the ASIC: either the EA or the ZT. On the EA, depending if TurboTX is enabled or not, the traffic coming from the RE is handled by one of these WI input streams (the last `0` means ingress):

```
(mx vty)# show mqss 0 phy-stream list 0

Ingress PHY stream list
-----------------------

------------------------------------------------------------------------
Stream    Type        Enabled    NIC    PIC    Connection    Traffic Class
Number                           Slot   Slot   Number
------------------------------------------------------------------------
[…]
1088      Host < TurbotTX disabled
1089      GE <unsed   Yes        255    255    26            4 (Invalid)
1090      GE < TurboTX enabled
[…]
```

Indeed, when TurboTX is disabled, the traffic received by the TOE is served to the MQSS through the WI input stream with the type `Host (1088)`. When it is enabled, the traffic is received by the second dedicated Ethernet port attached to the EA ASIC. Therefore, look for the second stream with type `GE`. Here it's the WI stream 1090. We left TurboTX enabled in our case. You can, as you now know how to do it, configure a specific WI input counter for the stream 1090. Let's configure it:

```
(mx vty)# test mqss 0 wi stats stream 0 0 66
```

The `0 0 66` means the first 0 is the WAN port group 0 – it's usually 0. The next 0 is the counter ID. As noted, on the EA you have 48 counters available (0 to 47). Here we use the counter index 0 for our statistics. Finally the last number is the incoming stream connection. This value is derived from the incoming stream number (retrieved previously – for us this is 1090), from which we subtract 1024 (1090-1024 = 66). Now call the next command and have a look at counter 0, and we should discover our echo-replies:

```
(mx vty)# show mqss 0 wi stats
WI statistics
-------------
[…]
Oversubscription drop statistics
[…]
Tracked stream statistics
-------------------------
Track Stream Stream Total Packets       Packets Rate       Total Bytes        Bytes Rate         Total
EOPE         EOPE Rate
     Mask   Match                  (pps)                              (bps)
(pps)
-------------------------------------------------------------------------------------------------------
---------
0    0xff   0x42   896379            208 <<< echo-reply   467373164          866704            0
```

Of course, as always, don't forget to reset the counter 0:

```
(mx vty)# test mqss 0 wi stats default 0 0
```

On the ZT there is only one choice, because the host outbound traffic is always received by the internal GE port. Look for the first GE type and note the WI input stream value (for us it's 1089):

```
door7302@mx2020> start shell pfe network fpc11.0
(mx vty)# show mqss 0 phy-stream list 0
Ingress PHY stream list
-----------------------
-------------------------------------------------------------------------
Stream    Type        Enabled    NIC    PIC    Connection    Traffic Class
Number                           Slot   Slot   Number
-------------------------------------------------------------------------
[…]
1089      GE <from pktIO    Yes        255    255    50            4 (Invalid)
1090      GE <unsed         Yes        255    255    51            4 (Invalid)
1091      Crypto            Yes        255    255    53            0 (High)
1165      WAN               Yes        11     0      40            0 (High)
[…]
```

Create a WI counter for this stream (65 = 1089-1024):

```
(mx vty)# test mqss 0 wi stats stream 0 0 65
```

And display the WI statistics:

```
(mx vty)# show mqss 0 wi stats
WI statistics
-------------
[…]
Tracked stream statistics
-----------------------------------------------------------------
Track Stream Stream Total Packets       Packets Rate       Total Bytes        Bytes Rate         Total
EOPE         EOPE Rate
     Mask   Match                  (pps)                              (bps)
(pps)
-----------------------------------------------------------------------------------------------------
0    0xff   0x41   51035             202< echo-reply     27491121           869128            0
```

And reset the default counter configuration at the end of your troubleshooting:

```
(mx vty)# test mqss 0 wi stats default 0 0
```

We are almost at the end of our trip.

Received by the WI block, the host outbound traffic follows its path until it reaches the LUSS. A new packet manipulation is performed inside the LUSS. Once again we can capture the packet to see the state of the packet before and after the LUSS processing. We decided to filter the capture based on the IPv4 source address followed by IPv4 destination in hexadecimal: 0xAC10FEFEC0A80301.

As you can see, next we use an offset of 48 bytes. *Why?* We found it by dichotomy. Usually you start with no offset, and if you capture nothing or an internal liveness packet marked as followed (`PType CTRL (1)`) you could progressively increment the offset by eight bytes, and so on.

This is what we did until we didn't capture a ptype CTRL packet:

```
(mx vty)# test jnh 0 packet-via-dmem enable
(mx vty)# test jnh 0 packet-via-
dmem capture capture 0x3 AC10FEFEC0A80301 32      <<< Offset 32 was not the good value
(mx vty)# test jnh 0 packet-via-dmem capture 0x0
(mx vty)# test jnh 0 packet-via-dmem decode
Wallclock: 0x88414a0d
[…]
PType CTRL (1) SubType f     <<< Means it's a Fabric probe not our packet
[…]
```

Let's plug in another offset value:

```
(mx vty)# test jnh 0 packet-via-dmem capture 0x3 AC10FEFEC0A80301 48
(mx vty)# test jnh 0 packet-via-dmem capture 0x0
(mx vty)# test jnh 0 packet-via-dmem decode
PFE 0 Parcel Dump:
Wallclock: 0xfc7502fc      <<< MQSS to LUSS
Dispatch: callout 0 error 0 Size 0d8 X2EA_PKT_HEAD (1) ReorderId 09df Stream GE (442) << WI inputStream
```

```
ChnkTyp 1 TailLen 015a ChnkDataOff 00 ChnkDataLen 10 ChkSum 00 ChnkPtr 0b798a1
DMAC 020000000050 SMAC 020000000060 Etype 0800
PType HOST (8) < no more CTRL pkt Subtype 0 Score 00 Reason 00 AddInfo 000c0 IIF 00001
L2Off 80 L3Off 00 PktLen 2200 StreamType 0 LUid 0 L2iif 30000 OrigPType UNK (0) OrigL3Off 64 BDid 8788
DdosProto 6345 FC 14 DP 1 TokenIsOIF 0 Token 0c9614 FlowId 3a9b0c EgrIIF 08004
```

You can see the output confirms:

```
420021a0000815a001000b798a10200000000500200000000600800800000000c000001800022000300
00006487886345514c9614753619080045

 0001ee0520000040
 010537ac10fefec0
 a803010000601600
 0000009cbb38ff72
 e3f5b64978696000
 000000dadadada36
 ebd75e01b6dadada
 dadadadadadadada
[...]
Wallclock: 0xfc750a18   <<< LUSS to MQSS
Reorder: EA2X_REORDER_SEND_TERMINATE (d) HasTail 1 Stat 1
ReorderId 09df Color 0 Qop ENQUEUE (2) Qsys WAN (0) Queue Wan (13) <<<< WAN output Queue
Stats Map 0 len_adjust 02 cnt_addr 0040146
Frag 0 RefCnt 0 OptLbk 0 NumTailHoles 00
ChnkTyp 1 TailLen 015a ChnkDataOff 00 ChnkDataLen 10 ChkSum 00 ChnkPtr 0b798a1 WanCookie 0000
dc9df0400013030105b7500d000404014601084020000000815a001000b798a10000
 6487886345514c96
 1475361908004500
 01ee052000004001
 0537ac10fefec0a8
 0301000060160000
 00009cbb38ff72e3
 f5b6497869600000
 0000dadadada36eb
 d75e01b6dadadada
 dadadadadadadada
[...]
```

Don't forget to disable capture:

```
(mx vty)# test jnh 0 packet-via-dmem disable
```

Let's analyze that last capture. We can see that the packet before LUSS processing is identified as coming from the WI input stream 0x442 – 1090 (the stream we mentioned earlier attached to the internal GE port connected to the EA – remember TurboTX is enabled). After the LUSS processing, the packet comes back into the MQSS with some additional information such as the WAN output queue to use to enqueue the packet. In our case this is the WAN queue ID: 0x13 (queue 19). The queue 19 is the absolute queue, and it is managed by the XQSS, which is actually the queue 3 (FC HOST_GEN) of the egress port et-9/0/0.

How did we deduce that absolute queue 19 is equal to queue 3?

We already know that the packet should be sent out via the et-9/0/0 interface (see prior discussion on the TTP header – field ifd_output). The IFD of et-9/0/0 is 311:

```
door7302@mx2020> show interfaces et-9/0/0| match index
  Interface index: 311, SNMP ifIndex: 1000
  Logical interface et-9/0/0.0 (Index 324) (SNMP ifIndex 1008)
```

With this IFD index and the following command we can display the CoS parameter attached to the physical port et-9/0/0:

```
(mx vty)# show cos halp ifd 311
------------------------------------------------------------------------------
    rich queueing enabled: 1
    Q chip present: 1
IFD name: et-9/0/0   (Index 311) egress information
    XQSS chip id: 0
    XQSS : chip Scheduler: 0
    XQSS chip L1 index: 5
    XQSS chip dummy L2 index: 1989
    XQSS chip dummy L3 index: 5
    XQSS chip dummy L4 index: 2
    Number of queues: 8
    XQSS chip base Q index: 16
Queue    State        Max       Guaranteed   Burst  Weight Priorities Drop-Rules  Scaling-profile
Index                 rate         rate      size           G    E   Wred  Tail       ID
------ ----------- ---------- ------------ ------- ------ ---------- ----------  ---------------
    16  Configured 100000000000   2000000000 67108864      5   GL    EL    4  1277         3
    17  Configured 100000000000   6000000000 67108864     15   GL    EL    4  1254         3
    18  Configured 100000000000  32000000000 67108864     80   GL    EL    4  1254         3
    19  Configured 100000000000     Disabled 67108864      1   GH    EH    4  1086         1
    20  Configured 100000000000  50000000000 67108864     50   GH    EH    4  1277         1
    21  Configured 100000000000  10000000000 67108864     50   GM    EH    4  1086         2
    22  Configured 100000000000            0 67108864      1   GL    EL    0   278         3
    23  Configured 100000000000            0 67108864      1   GL    EL    0   278         3
```

As seen above, the absolute base queue ID of the et-9/0/0 is the queue ID 16. This means that queue 0 is mapped to queue 16, queue 1 is mapped to queue 17, and so on. Thus, the queue 3 of the forwarding class HOST_GEN is actually the absolute queue ID 19: 0x13 – the information provided by the packet capture.

Finally let's issue the next command to retrieve statistics of the absolute queue 19:

```
(mx vty)# show xqss 0 sched queue 19 local-stats
Queue:19
        Forwarded pkts : 8629661          200          pps
        Forwarded bytes: 4578894829       852216       bps
        Dropped pkts   : 0                0            pps
        Dropped bytes  : 0                0            bps
```

Once dequeued the host outbound packets move back from XQSS to MQSS, are handled by the WO block, which merges HEAD and TAIL of the packets (if needed) and forwards them to the MAC block and, bye-bye, our echo-replies leave our router.

Hey! 200pps of ping echo-replies are still in the ZT ASIC. Indeed, we've stopped our trip on the ZT side in the WI block where we found that host outbound packets coming from packetIO were handled by the WI input stream 1089. As on the

EA, packets are split into HEAD and TAIL (if needed) and take their trip inside the MQSS until reaching the LUSS. We can perform a similar packet capture on the ZT based also on the IPv4 source and destination addresses. The hexadecimal pattern used as filter is therefore: 0xAC10FEFEC0A80401. As observed, we also use an offset of 48 bytes (again, we found this value by dichotomy):

```
door7302@mx2020> start shell pfe network fpc11
mx2020-fpc11:pfe> test jnh packet-via-dmem inst 0 enable
mx2020-fpc11:pfe> test jnh packet-via-dmem-capture inst 0 parcel-type-mask 0x3 match-
string AC10FEFEC0A80401 offset 48
mx2020-fpc11:pfe> test jnh packet-via-dmem-capture inst 0 parcel-type-mask 0x0
mx2020-fpc11:pfe> test jnh packet-via-dmem-dump inst 0
PFE 0 Parcel Dump:
     Wallclock: 0x0c6136e6     << MQSS to LUSS
     Dispatch: callout 0 error 0 Size 0d8 X2EA_PKT_
HEAD (1) ReorderId 0998 Stream GE (441) << WI stream (GE)
     ChnkTyp 1 TailLen 015e ChnkDataOff 00 ChnkDataLen 10 ChkSum 00 ChnkPtr 3fae477
     DMAC fe0000000000 SMAC 0000000000fe VLAN 81000004 Etype 0800
    14cc4410021e23c8815e001003fae477fe0000000000000000000fe810000040800 800000e40c080001
00000001f4000000 6487886345a34c96 147537ab08004500 01eeb40c00004001 554aac10fefec0a8 04010000f6e20000
00009cbb38ff72e3 f5b649786961a000 0000dadadadacabe 0cbd01b6dadadada dadadadadadadada dadadadadadadada
dadadadadadadada dadadadadadadada dadadadadadadada dadadadadadadada dadadadadadadada dadadadadadadada
dadadadadadadada dadadadadadadada dadadadadada
Wallclock: 0x0c613c1a     << LUSS to MQSS
    Reorder: EA2X_REORDER_SEND_TERMINATE (d) HasTail 1 Stat 1
    ReorderId 0998 Color 0 Qop ENQUEUE (2) Qsys WAN (0) Queue InSrvc (2b).  <<< WAN output Queue
    Stats Map 0 len_adjust 02 cnt_addr 002123e
    Frag 0 RefCnt 0 OptLbk 0 NumTailHoles 00
    ChnkTyp 1 TailLen 015e ChnkDataOff 00 ChnkDataLen 10 ChkSum 00 ChnkPtr 3fae477
    Magic 0000 Tun 0 IngQ 1 RefCnt 0 OptLbk 0 TailHndl 886345a34c961475
    IxPreClass 0 IxPort 17 IxMyMac 1
dc998040002b040105b1c84b900402123e01c8000001f400815e001003fae47700006487886345a34c96147537ab
0800450001eeb40c 00004001554aac10 fefec0a804010000 f6e2000000009cbb 38ff72e3f5b64978 6961a0000000dada
dadacabe0cbd01b6 dadadadadadadada dadadadadadadada dadadadadadadada dadadadadadadada dadadadadadadada
dadadadadadadada dadadadadadadada dadadadadadadada dadadadadadadada dadadadadadadada dadadadadadadada
dada

mx2020-fpc11:pfe> test jnh packet-via-dmem inst 0 disable
```

As expected, the host outbound packet is identified as coming from the WI input stream 0x441 (stream 1089 – already found earlier). The second part of the capture informs us on which WAN output queue of the XQSS the packet will be enqueued: this is the absolute queue ID 0x2b – queue ID 43. Retrieve the IFD index of the et-11/0/0, here it's IFD 228:

```
door7302@mx2020> show interfaces et-11/0/0 | match index
  Interface index: 228, SNMP ifIndex: 617
  Logical interface et-11/0/0.0 (Index 500) (SNMP ifIndex 850)
```

Using the command below, you can retrieve the Token ID attached to the CoS parameters/statistics of the et-11/0/0:

```
door7302@mx2020> start shell pfe network fpc11
mx2020-fpc11:pfe> show class-of-service interface scheduler hierarchy index 228
Interface Schedulers:
```

```
 Name            Type    Index    Level   Node-Token     Shaping-Rate      Guaranteed-Rate   Delay-
Buffer-Rate    Excess-Rate
 et-11/0/0        IFD      228    1      3157          100000000000        100000000000
100000000000        100000000000
```

Finally, let's resolve the AFT Token IDin order to display the CoS configuration and statistics (those stats come from XQSS):

```
mx2020-fpc11:pfe> show sandbox token 3157
[…]
Node Index:228
Node Name:et-11/0/0
Parent Name:
Interface Rate:100.00Gbps
DelayBufferRate:100.00Gbps
[…]
CoS Scheduler Node:
      PFE Instance : 0
           L1 Index : 4
           L2 Index : 4
           L3 Index : 4
           L4 Index : 5
Enhanced Priority Mode : 0
Table: Queue Configuration
------------------------------------------------------------------------------------------------------
--------
  Index | Shaping-Rate | Transmit-Rate |  Burst | Weight | G-Priority | E-Priority | Tail-Rule |
WRED-Rule |
--------+--------------+---------------+--------+--------+------------+------------+-----------+----
-------+
   40   |   100.0G     |     2.0G      |  1.2G  |   5    |    GL      |    EL      |   1278    |    0     |
   41   |   100.0G     |     6.0G      |  1.2G  |   15   |    GL      |    EL      |   1255    |    0     |
   42   |   100.0G     |    32.0G      |  1.2G  |   80   |    GL      |    EL      |   1255    |    0     |
   43   |   100.0G     |     0.0       |  1.2G  |   1    |    GH      |    EH      |   1087    |    0     |
   44   |   100.0G     |    50.0G      |  1.2G  |   50   |    GH      |    EH      |   1278    |    0     |
   45   |   100.0G     |    10.0G      |  1.2G  |   50   |    GM      |    EH      |   1087    |    0     |
   46   |   100.0G     |     0.0       |  1.2G  |   1    |    GL      |    EL      |   279     |    0     |
   47   |   100.0G     |     0.0       |  1.2G  |   1    |    GL      |    EL      |   279     |    0     |
--------+--------------+---------------+--------+--------+------------+------------+-----------+----
-------+

Queue Statistics:

  PFE Instance : 0
                          Transmitted                                                  Dropped
             Bytes                        Packets             Bytes          Packets
------------------------------------------------------------------------------------------------------
-----------------------------------
 Queue:0    41885064578(          0 bps)     78737147(          0 pps)           0(          0 bps)
0(          0 pps)
 Queue:1             0(          0 bps)           0(          0 pps)           0(          0 bps)
0(          0 pps)
 Queue:2             0(          0 bps)           0(          0 pps)           0(          0 bps)
0(          0 pps)
 Queue:3     4796032280(     851504 bps)      9081386(        202 pps)           0(          0 bps)
0(          0 pps)
 Queue:4             0(          0 bps)           0(          0 pps)           0(          0 bps)
```

```
0(         0 pps)
 Queue:5       9055577(         0 bps)       45092(         0 pps)          0(         0 bps)
0(         0 pps)
 Queue:6             0(         0 bps)          0(         0 pps)          0(         0 bps)
0(         0 pps)
 Queue:7             0(         0 bps)          0(         0 pps)          0(         0 bps)
0(         0 pps)
```

With this last command you'll see the absolute egress queue indexes attached to port et-11/0/0 and the queues' statistics at the same time.

We see that absolute queue index 43 is actually queue 3 (HOST_GEN forwarding class). The queue 3 conveys, as expected, our 200 pps of echo-reply. Those packets, once dequeued, return to the MQSS (from XQSS) and finally leave the router after they were handled by the WO and MAC blocks of the ZT ASIC.

This is the end of our trip inside EA and ZT ASIC. If you are still alive and interested, there are two more little topics explained quickly in the Appendixes:

- How Junos uses internal keepalives to detect internal HW/SW failures – we talked briefly about PFE lLiveness, host loopback probes, and fabric probes earlier in this chapter. The first part of the Appendix should provide more detailfor you about those concepts.
- And the second topic, how tunnels are managed on the EA and ZT. It's a short review that reuses the commands we already used to analyze a specific tunneled traffic: a GRE tunnel.

Appendices

EA/ZT Data Path Health Check

During our packet capture troubleshooting sessions performed in Chapter 3, we mentioned that sometimes we can catch internal packets. Those internal packets, which are generated by the line card itself, even if there is no data plane traffic, allow the system to detect internal failures and to response to these failure by isolating and trying to restore the piece of hardware in the faulty state.

There are actually three kinds of self-generated packets:

- Host data path health check: Sent every second by either the uKernel on the EA or packetIO on the ZT line card.
- Fabric self-probe: Sent every 50ms by each LUSS over the fabric to itself.
- PFE liveness: Sent every 1ms by each LUSS toward each remote operational PFE.

Let's take an MX2020 with two line cards: one MPC9e in slot 9 and one MPC11e in slot 11 and put this MX2020 alone in the dark, meaning put on it a factory default configuration, shut down all physical ports, and use a console port to manage it.

Why are we doing this?

You don't want to be polluted by any control plane packets such as ARP, or even management protocols like SSH.. You will see that even in this configuration there is still life inside our MX2020.

Host Data Path Check

The host data path check mechanism allows detecting failure along the host path. Every second an echo packet is generated by the uKernel or the packetIO process depending on the type of line card.

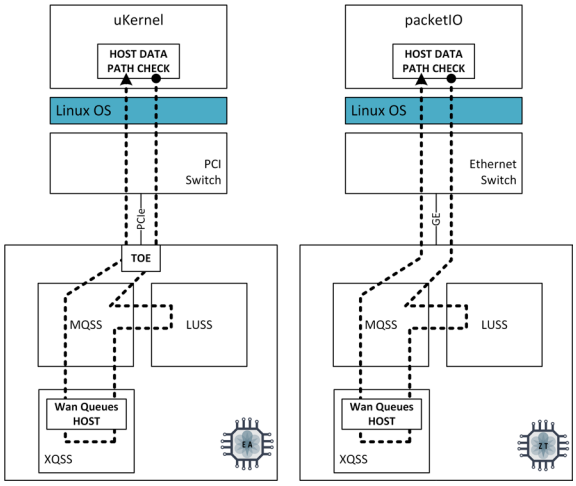
On the EA, it is sent to the ASIC via the PCIe interface and it is received by the TOE and delivered to the MQSS, which forwards it to the LUSS. The packet is destined to loop back to the uKernel. After LUSS processing, the packet comes back into the MQSS, then it is enqueued in the host WAN queue (into XQSS), and finally it goes back to the uKernel via the PCIe interface.

When the host path check module detects five consecutive packet losses, it triggers what we call a *PFE Wedge condition*: the host path is isolated.

On the ZT line card it's almost the same mechanism except that echo packets are sent and received by the packetIO through the internal port attached to the ZT.

Figure A.1 illustrates this on both types of line card.

Figure A.1 Host Data Path Health Check



Let's check the TOE statistics on the PFE 0 of the MPC9e. As observed there is one pps received and sent by the TOE: this is our host data path check packet:

```
(mx vty)# show toe pfe 0 mqss 0 toe-inst 0 packet-stats stream 0
Stream 0: halt flag is NOT set
   TX Packets
MQSS TOE pfe 0 asic 0 toe 0 mailbox register 12 contains 0x00148133
      accepted:        0000000001343795
   TX Rates:
      packets per second: 1
[…]
```

```
MQSS TOE pfe 0 asic 0 toe 0 mailbox register 11 contains 0x0231b1d2
   RX Packets:
      accepted:        0000000036811218
   RX Rates:
      packets per second: 1
[…]
```

On the ZT line card you can issue the following PFE command to see that every FP interface of packetIO is sent and received 1pps. We neglected to mention that the uKernel and the packetIO check the host path of every ASICs of the MPC:

```
mx2020-fpc11:pfe> show host-path ports | match "Name|fp"
Name               State      Tx-Packets    Tx-Rate(pps)  Rx-Packets    Rx-Rate(pps)  Drop-
Packets  Drop-Rate(pps)
fp7                READY      150216        1             151404        1             0       0
fp6                READY      56512         1             56686         1             0       0
fp0                READY      10353425      1             10355643      1             0       0
fp1                READY      56512         1             56572         1             0       0
fp2                READY      56512         1             56530         1             0       0
fp3                READY      56512         1             56526         1             0       0
fp4                READY      56512         1             56524         1             0       0
fp5                READY      56512         1             56523         1             0       0
```

Moreover, on ZT line card, you can also check if "Wedge" has been triggered by issuing this following command:

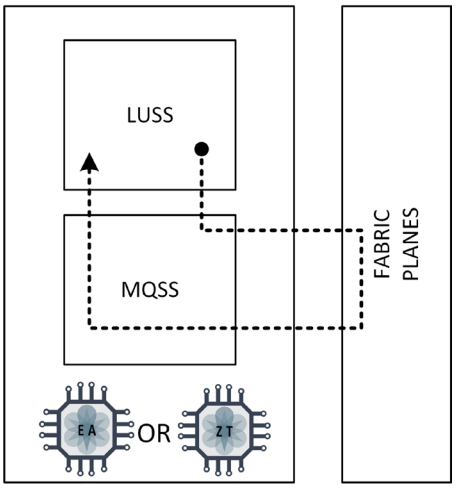
```
door7302@mx2020> start shell pfe network fpc11
mx2020-fpc11:pfe> show host-path app wedge-detect pfe-status
+------------+---------+
| PFE Number | Status  |
+------------+---------+
| 07         | Online  |
| 06         | Online  |
| 05         | Online  |
| 04         | Online  |
| 00         | Online  |
| 01         | Online  |
| 02         | Online  |
| 03         | Online  |
+------------+---------+
```

Fabric Self Probe / PFE Liveness

The next two mechanisms allow checking the forwarding path. They are more aggressive in terms of pps to quickly detect any forwarding issue in order to isolate and try to restore the faulty path.

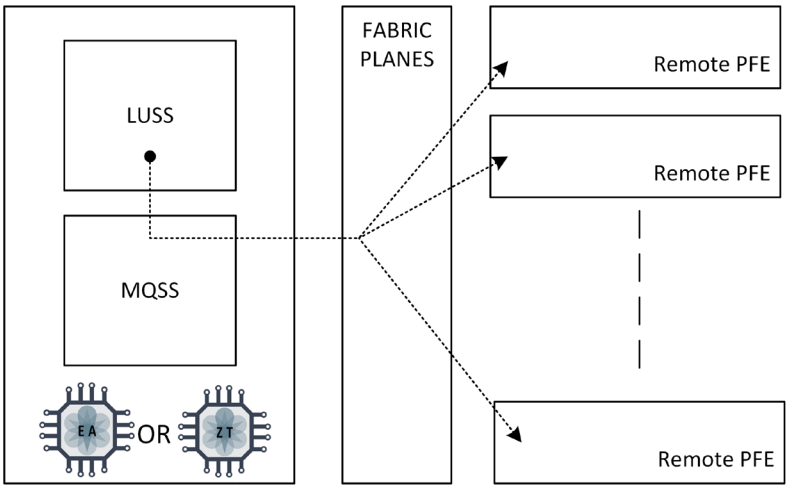
Fabric self probes (or self pings) are sent by the LUSS to itself, but through the fabric. This single packet is big enough to be split as there are enough fabric cells to cover all the fabric planes available. The self ping is sent every 50ms (20pps rate). These self pings are conveyed in the high fabric queue. Figure A.2 illustrates the mechanism.

Figure A.2 *Fabric Self Ping*



The last mechanism is named *PFE Liveness*. It allows a given PFE to test every remote destination (remote PFE). In this case, the LUSS generates for each remote PFE a unidirectional packet every 1ms (1000pps rate). The remote PFE detects any fabric blackholing or source PFE issue by checking the sequence number of received packets. These PFE liveness packets are conveyed in the high fabric queue of each PFE. Figure A.3 illustrates the mechanism.

Figure A.3 *PFE Liveness*



Let's check the Fabric queue statistics on PFE 0 of the MPC9e and have a specific look at the high fabric queue to reach the remote PFE 0 hosted by the MPC in slot 11. We specify the queue 172, which is the fabric queue ID to reach PFE 0 in slot 11 (128 + (4 * 11 + 0)):

```
(mx vty)# show mqss 0 sched-fab q-node stats 172

Queue statistics (Queue 0172)
-----------------------------

-------------------------------------------------------------------------------
Color  Outcome           Counter  Total               Rate
                         Name
-------------------------------------------------------------------------------
All    Forwarded (Rule)  Packets  258417654           1000 pps
                         Bytes    4136499889          128192 bps
```

As expected we see the 1000pps PFE Liveness stream to test the PFE0-slot11 destination. Issue the same command but for the high fabric queue attached to itself: 164 (128 + (4*9 +0)):

```
(mx vty)# show mqss 0 sched-fab q-node stats 164

Queue statistics (Queue 0164)
-----------------------------

-------------------------------------------------------------------------------
Color  Outcome           Counter  Total               Rate
                         Name
-------------------------------------------------------------------------------
All    Forwarded (Rule)  Packets  264423857           1020 pps
                         Bytes    6798895444          210336 bps
```

Interesting, here we see 1020pps. This is actually the sum of the two data path health check mechanisms: the PFE liveness 1000pps sent to itself through the fabric and the fabric self ping at 20pps.

Handling Tunnels on the EA and ZT

The EA and ZT support Inline Tunneling such as GRE (gr- interface), or Logical-Tunnel (lt- interface).

NOTE Starting with Junos 19.3R1, Junos also supports flexible tunnels interfaces. With this specific feature you don't care about tunnel PIC location and bandwidth, redundancy question, and number of IFL restrictions, as all is handled inside. It's typically used for VXLAN but now for all other sort of tunnel encapsulation features as well. More information can be retrieved here: https://www.juniper.net/documentation/en_US/junos/topics/concept/flexible_tunnel_interfaces_overview.html.

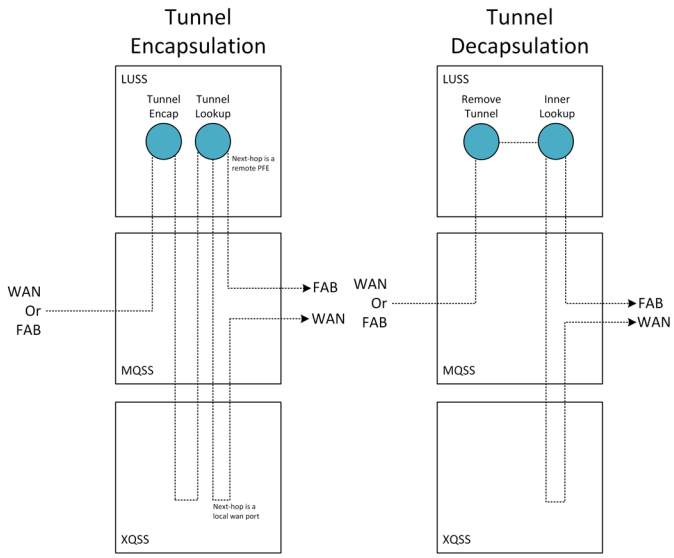
The tunnel feature can be enabled per PFE. The configuration statement is the following:

```
door7302@mx2020# set chassis fpc <fpc-slot> pic <pic-slot> tunnel-services bandwidth ?
Possible completions:
  <bandwidth>          Bandwidth reserved for tunnel service
  100g                 100 gigabits per second
  10g                  10 gigabits per second
  1g                   1 gigabit per second
  200g                 200 gigabits per second
  20g                  20 gigabits per second
  300g                 300 gigabits per second
  30g                  30 gigabits per second
  400g                 400 gigabits per second
  40g                  40 gigabits per second
  50g                  50 gigabits per second
  60g                  60 gigabits per second
  70g                  70 gigabits per second
  80g                  80 gigabits per second
  90g                  90 gigabits per second
```

The bandwidth option allows reserving a part of the ASIC/PFE bandwidth for the tunnel processing. The tunnel encapsulation and decapsulation are performed by the LUSS block. Notice that the PFE in charge of the tunnel service is not always the PFE attached to the ingress or egress ports. Actually, the tunnel service might be managed by any PFE of the router. This is one will be reachable through the fabric by other PFEs if they need to perform tunneling processing. The placement of the tunnel service usually depends on your architecture (current load of the ingress or egress PFEs), the required latency, and if you wish to save or not save fabric bandwidth. Figure A.4 illustrates how encapsulation and decapsulation are done by the tunnel service PFE. As you can see, encapsulation requests more PFE capacity as there is a double LUSS circulation. The first circulation into LUSS is for adding the tunnel header, and then the tunneled traffic loops back to LUSS a second time (by using a loopback internal stream) in order to perform the tunnel lookup. The decapsulation process is simpler; in one round the LUSS removes the tunnel header and performs inner header lookup.

As seen previously, traffic might come from the WAN interface or fabric and might be sent out also to WAN or fabric.

Figure A.4 Tunnel Encapsulation/Decapsulation On the EA or ZT



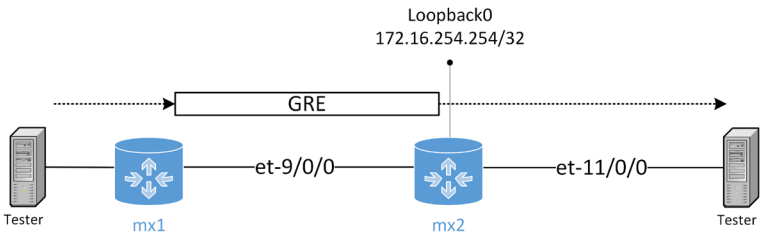
Let's consider two examples to illustrate these functionalities:

- Inline GRE decapsulation
- Simple use of logical tunnel interface

Inline GRE Decapsulation

The first use case is depicted by Figure A.5. There are two MXs. We have set up a GRE tunnel between MX1 and MX2 and a unidirectional 100Kpps stream is sent by a tester. MX1 encapsulates the traffic while MX2 decapsulates it. The MPC in slot 9 is an EA line card (MPC9e) and the one in slot 11 is an MPC11e (ZT based). MX2 has two physical ports, both attached to PFE 0 of each MPC.

Figure A.5 GRE Topology

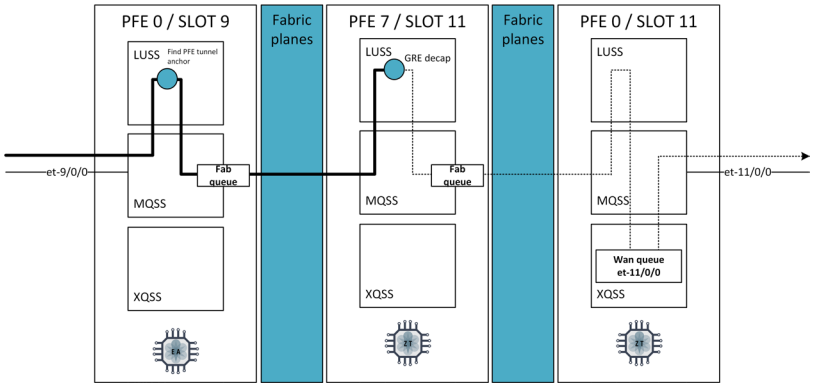


Let's focus on the MX2. The tunnel service is configured on the ZT line card – especially on PFE 7: a different PFE of ingress or egress PFEs. The tunnel configuration is the following:

```
chassis {
    fpc 11 {
        pic 7 {
            tunnel-services {
                bandwidth 10g;
            }
        }
    }
}
interfaces {
    gr-11/7/0 {
        unit 0 {
            tunnel {
                source 172.16.254.254;
                destination 192.168.3.1;
            }
            family inet {
                address 192.167.1.1/30;
            }
        }
    }
```

Figure A.6 illustrates how the GRE tunnel is handled by the MX2.

Figure A.6 Inline GRE Decapsulation



Let's try to figure out how GRE decapsulation is performed. You can do a first packet capture on the ingress EA PFE (refer to Chapter 3 if you need the details of the commands to perform the EA or ZT packet capture – here we only show the result of the capture).

The ingress PFE receives the tunneled traffic and the packet capture gives us this output:

```
Wallclock: 0x76bdc118
Dispatch: callout 0 error 0 Size 06e X2EA_PKT (0) ReorderId 0327 Stream Wan (48f)
IxPreClass 2 IxPort 00 IxMyMac 1
0193c8f08008
 4c96147536196487
 8863455108004500
 0052000000003f2f
 0cc5c0a80301ac10
 fefe000008004500
[…]
Wallclock: 0x76bdc9b6
Reorder: EA2X_REORDER_SEND_TERMINATE (d) HasTail 0 Stat 0
ReorderId 0327 Color 0 Qop ENQUEUE (2) Qsys FAB (1) Queue 0012f <<< Low Fab Queue to reach PFE 7 / Slot
11 (303)
PType IPV4 (2) SubType 0 PfeMaskF 0 OrigLabel 0 SkipSvc 0 IIF 00142
FC 00 DP 0 SkipSample 0 SkipPM 0 L2Type None (0) IsMcast 0 SrcChas 1 MemId 0
PfeNum 7f PfeMaskE 0 FabHdrVer 1
RwBuf 00000000 SvcCtx 0 ExtHdr 0 PfeMaskE2 0 Token 000272
d0327050012ff85405b4d84d4da48000d12000014200010fe10000000000000272
 4500005200000000
 3e2f0dc5c0a80301
 ac10fefe00000800
 4500003a00000000
 4011b40902000001
[…]
```

The ingress LUSS identified the traffic as having reached the end of the tunnel and therefore the traffic must be decapsulated. It knows that tunnel service PFE is the PFE 7 in slot 11. This is why we see the packet as marked to be forwarded to fabric

queue 303 (0x12f) after the LUSS processing. This is actually the low fabric queue to reach PFE 7 in slot 11. If you check this specific queue on the ingress EA PFE you'll discover our 100Kpps of tunneled traffic:

```
(mx vty)# show mqss 0 sched-fab q-node stats 303

Queue statistics (Queue 0303)
-----------------------------

------------------------------------------------------------------------------
Color  Outcome           Counter  Total               Rate
                         Name
------------------------------------------------------------------------------
All    Forwarded (Rule)  Packets  5934626143          100055 pps
                         Bytes    1118819211658       78443400 bps
All    TAIL drops        Packets  0                   0 pps
                         Bytes    0                   0 bps
0      WRED drops        Packets  0                   0 pps
                         Bytes    0                   0 bps
1      WRED drops        Packets  0                   0 pps
                         Bytes    0                   0 bps
2      WRED drops        Packets  0                   0 pps
                         Bytes    0                   0 bps
3      WRED drops        Packets  0                   0 pps
                         Bytes    0                   0 bps
------------------------------------------------------------------------------
```

Now, let's move on to the MPC11 and especially PFE 7. We can perform another packet capture. Here we see that the packet came from the PFE 0 in slot 9 (stream fab = 36 (0x24)) and the outer IP and GRE headers are still there before the LUSS (tunnel anchor PFE) processes it. After LUSS processing, we see that the outer IP + GRE headers have been removed and a lookup of the inner IP header gave the fabric stream 44 (0x2C) as next hop, which is actually the fabric queue to reach the egress PFE. This is the PFE 0 of this MPC in slot 11 (where it is attached the egress port et-11/0/0):

```
Wallclock: 0xb42a2783
     Dispatch: callout 0 error 0 Size 070 X2EA_PKT (0) ReorderId 1212 Stream Fab (24) UdpCsum 8b4b
     PType IPV4 (2) SubType 0 PfeMaskF 0 OrigLabel 0 SkipSvc 0 IIF 00142
     FC 00 DP 0 SkipSample 0 SkipPM 0 L2Type None (0) IsMcast 0 SrcChas 1 MemId 0
     PfeNum 7f PfeMaskE 0 FabHdrVer 1
     RwBuf 00000000 SvcCtx 0 ExtHdr 0 PfeMaskE2 0 Token 000272
    090902408b4b200001420001 0fe10000000000000272 4500005200000000 3e2f0dc5c0a80301 ac10fefe00000800
<<< outer IP + GRE header 4500003a00000000 4011b40902000001 c0a80401003f003f 00268dba9cbb38ff
72e3f5b649786960 4000000010111213 7c07351700021a1b 8c31

Wallclock: 0xb42a2ec3
     Reorder: EA2X_REORDER_SEND_TERMINATE (d) HasTail 0 Stat 0
     ReorderId 1212 Color 0 Qop ENQUEUE (2) Qsys FAB (1) Queue 0002c
     PType IPV4 (2) SubType 0 PfeMaskF 0 OrigLabel 0 SkipSvc 0 IIF 0015a
     FC 00 DP 0 SkipSample 0 SkipPM 0 L2Type None (0) IsMcast 0 SrcChas 1 MemId 0
     PfeNum 2c PfeMaskE 0 FabHdrVer 1
    RwBuf 00000000 SvcCtx 0 ExtHdr 0 PfeMaskE2 0 Token 0002b0
d1212050002cf86e05b3804e4a4e0004d12000015a000105810000000000002b0 <<< No more Tunnel >>>
4500003a00000000 3f11b50902000001 c0a80401003f003f 00268dba9cbb38ff 72e3f5b649786960 4000000010111213
7c07351700021a1b 8c31
```

A double check of the fabric queue 44 on PFE 7 shows us our 100Kpps of decapsulated traffic:

```
(mx vty)# show mqss 7 sched-fab q-node stats 44

Queue statistics (Queue 0044)
-----------------------------

------------------------------------------------------------------------------
Color  Outcome           Counter  Total               Rate
                         Name
------------------------------------------------------------------------------
All    Forwarded (Rule)  Packets  5903103845          100865 pps
                         Bytes    1111434879150       59712456 bps
All    TAIL drops        Packets  0                   0 pps
                         Bytes    0                   0 bps
0      WRED drops        Packets  0                   0 pps
                         Bytes    0                   0 bps
1      WRED drops        Packets  0                   0 pps
                         Bytes    0                   0 bps
2      WRED drops        Packets  0                   0 pps
                         Bytes    0                   0 bps
3      WRED drops        Packets  0                   0 pps
                         Bytes    0                   0 bps
------------------------------------------------------------------------------
```

Finally, we can launch a third packet capture: this time on PFE 0 of the MPC11. It's actually a classic lookup result performed by the egress PFE. We first see that all of the packet came from the PFE 7 in slot 11 (stream fab 303 – 0x12f)). Then we can retrieve the WAN output queue ID assigned to the packet: the queue 40 (0x28). You can notice the Ethernet header has been computed and appended:

```
Wallclock: 0xcdf97460
     Dispatch: callout 0 error 0 Size 058 X2EA_PKT (0) ReorderId 189c Stream Fab (12f) UdpCsum 9d55
     PType IPV4 (2) SubType 0 PfeMaskF 0 OrigLabel 0 SkipSvc 0 IIF 0015a
     FC 00 DP 0 SkipSample 0 SkipPM 0 L2Type None (0) IsMcast 0 SrcChas 1 MemId 0
     PfeNum 2c PfeMaskE 0 FabHdrVer 1
     RwBuf 00000000 SvcCtx 0 ExtHdr 0 PfeMaskE2 0 Token 0002b0
    0c4e12f09d552000015a0001058100000000000002b0 4500003a00000000 3f11b50902000001 c0a80401003f003f
0026d9959cbb38ff 72e3f5b649786960 a000000010111213 89907bb200021a1b 8c31

Wallclock: 0xcdf9798a
    Reorder: EA2X_REORDER_SEND_TERMINATE (d) HasTail 0 Stat 1
     ReorderId 189c Color 0 Qop ENQUEUE (2) Qsys WAN (0) Queue InSrvc (28)
     Stats Map 0 len_adjust 10 cnt_addr 00211e0
     Magic 0000 Tun 0 IngQ 1 RefCnt 0 OptLbk 0 TailHndl 886345a34c961475
     IxPreClass 0 IxPort 17 IxMyMac 1
    d589c0400028f85605b2d01300200211e00000 6487886345a34c96147537ab 08004500003a0000 00003f11b5090200
0001c0a80401003f 003f0026d9959cbb 38ff72e3f5b64978 6960a00000001011 121389907bb20002 1a1b8c31
```

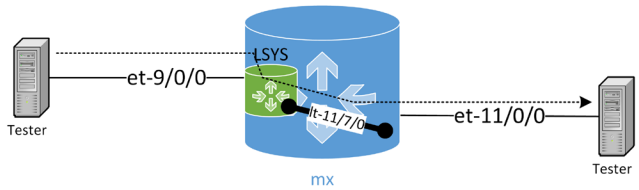
Just for fun, we can also check the CoS configuration and statistics of the egress port et-11/0/0 based on its IFD (330). We find our 100Kpps enqueued in queue 0 which is, based on the queue configuration info below, the absolute queue 40 (the same queue ID the packet capture output gave us earlier):

```
mx2020-fpc11:pfe> show class-of-service interface scheduler hierarchy index 330
Interface Schedulers:
 Name               Type    Index    Level   Node-Token
 et-11/0/0          IFD       330    1       3223
mx2020-fpc11:pfe> show sandbox token 3223
[…]
Table: Queue Configuration
-------------------------------------------------------------------------------------------------
  Index | Shaping-Rate | Transmit-Rate |  Burst | Weight | G-Priority | E-Priority | Tail-Rule |
WRED-Rule |
--------+--------------+---------------+--------+--------+------------+------------+-----------+-
   40   |   100.0G    |     2.0G     | 1.2G |   5   |    GL    |    EL    |   1278   |    0    |
   41   |   100.0G    |     6.0G     | 1.2G |  15   |    GL    |    EL    |   1255   |    0    |
   42   |   100.0G    |    32.0G     | 1.2G |  80   |    GL    |    EL    |   1255   |    0    |
   43   |   100.0G    |    0.0       | 1.2G |   1   |    GH    |    EH    |   1087   |    0    |
   44   |   100.0G    |    50.0G     | 1.2G |  50   |    GH    |    EH    |   1278   |    0    |
   45   |   100.0G    |    10.0G     | 1.2G |  50   |    GM    |    EH    |   1087   |    0    |
   46   |   100.0G    |    0.0       | 1.2G |   1   |    GL    |    EL    |   279    |    0    |
   47   |   100.0G    |    0.0       | 1.2G |   1   |    GL    |    EL    |   279    |    0    |
--------+--------------+---------------+--------+--------+------------+------------+-----------+----
-------+
Queue Statistics:
  PFE Instance : 0
                            Transmitted                                              Dropped
                    Bytes                        Packets                      Bytes
Packets
----------------------------------------------------------------------------------------------------
-----------------------------------
 Queue:0   1254425766564(   76806120 bps)   6007099073(    100008 pps)          0(          0 bps)
0(         0 pps)
 Queue:1            0(          0 bps)          0(          0 pps)          0(          0 bps)
0(         0 pps)
 Queue:2            0(          0 bps)          0(          0 pps)          0(          0 bps)
0(         0 pps)
 Queue:3     7286416711(       912 bps)     13762594(         2 pps)          0(          0 bps)
0(         0 pps)
 Queue:4            0(          0 bps)          0(          0 pps)          0(          0 bps)
0(         0 pps)
 Queue:5            0(          0 bps)          0(          0 pps)          0(          0 bps)
0(         0 pps)
 Queue:6            0(          0 bps)          0(          0 pps)          0(          0 bps)
0(         0 pps)
 Queue:7            0(          0 bps)          0(          0 pps)          0(          0 bps)
0(         0 pps)
```

Simple Use of Logical Tunnel Interface

This is a second use case to illustrate the packet recirculation case inside the tunnel service PFE. This time we rely on a simple topology using a logical system. Figure A.7 shows you the topology.

Figure A.7 *Logical System Topology*



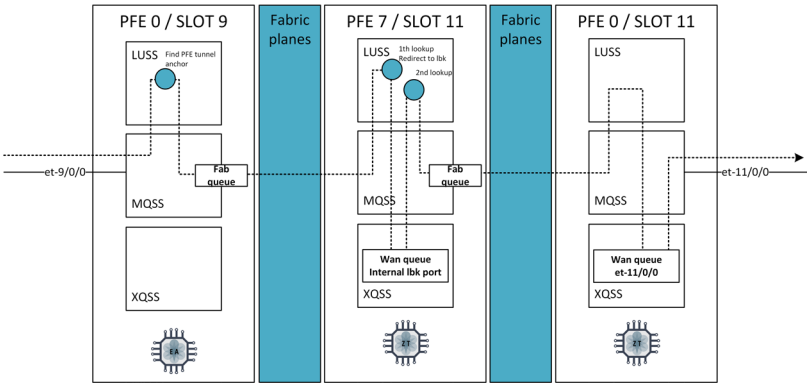
As observed, the real router instance and the logical system both have physical ports connected to them. Both instances are connected together with a logical tunnel interface: lt-11/7.0. The tester sends a 100Kpps stream, which is first routed by the logical system instance, then followed by the default routing instance. Once again, we use the PFE 7 of the MPC 11 to achieve the role of tunnel service PFE. The MX2 configuration is the following:

```
chassis {
    fpc 11 {
        pic 7 {
            tunnel-services {
                bandwidth 10g;
            }
        }
    }
}
interfaces {
    et-11/0/0 {
        unit 0 {
            family inet {
                address 192.168.2.1/30;
            }
            family iso;
        }
    }
    lt-11/7/0 {
        unit 2 {
            encapsulation ethernet;
            peer-unit 1;
            family inet {
                address 172.16.0.2/30;
            }
            family iso;
        }
    }
}
logical-systems {
```

```
    LSYS-RTR {
        interfaces {
            et-9/0/0 {
                unit 0 {
                    family inet {
                        address 192.168.1.1/30;
                    }
                    family iso;
                }
            }
            lt-11/7/0 {
                unit 1 {
                    encapsulation ethernet;
                    peer-unit 2;
                    family inet {
                        address 172.16.0.1/30;
                    }
                    family iso;
                }
            }
        }
}
```

Now the aim is to to one more time find out how forwarding is done with a set of PFE commands in this kind of topology.. You should be able to build this kind of data flow diagram in Figure A.8.

Figure A.8 *Logical Tunnel PFE Handling*



The first step is to perform a packet capture on the PFE 0 / slot 9. The first LUSS identified the next hop as the `lt` interface. Therefore, it forwards the traffic to the tunnel service PFE. It uses the low fabric queue for that to reach the PFE 7 / slot 11 (FAB queue 303 – 0x12f)):

```
Wallclock: 0xfee800b5
Dispatch: callout 0 error 0 Size 06e X2EA_PKT (0) ReorderId 0682 Stream Wan (48f)
IxPreClass 2 IxPort 00 IxMyMac 1
034148f08008
 4c96147536196487
```

```
 8863455108004500
 0052000000003f11
 f972ac101484ac10
 1584003f003f003e
[...]
Wallclock: 0xfee80a69
Reorder: EA2X_REORDER_SEND_TERMINATE (d) HasTail 0 Stat 0
ReorderId 0682 Color 0 Qop ENQUEUE (2) Qsys FAB (1) Queue 0012f
PType IPV4 (2) SubType 0 PfeMaskF 0 OrigLabel 0 SkipSvc 0 IIF 00142
FC 00 DP 0 SkipSample 0 SkipPM 0 L2Type None (0) IsMcast 0 SrcChas 1 MemId 0
PfeNum 7f PfeMaskE 0 FabHdrVer 1
RwBuf 00000000 SvcCtx 0 ExtHdr 0 PfeMaskE2 0 Token 000311
d0682050012ff85405b4084850280000d12000014200010fe10000000000000311
 4500005200000000
 3e11fa72ac101484
 ac101584003f003f
 003e0ca69cbb38ff
[...]
```

A quick check of the fabric queue statistics confirms this:

```
(mx vty)# sho mqss 0 sched-fab q-node stats 303
Queue statistics (Queue 0303)
------------------------------
--------------------------------------------------------------------------------
Color  Outcome           Counter  Total                Rate
                         Name
--------------------------------------------------------------------------------
All    Forwarded (Rule)  Packets  6254418747           100140 pps
                         Bytes    1150158886850        78510072 bps
All    TAIL drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
0      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
1      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
2      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
3      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
--------------------------------------------------------------------------------
```

Let's move to PFE 7 / slot 11 and perform and second packet capture. This time it's more complex. Actually we see the double circulation. The first round inside the tunnel service LUSS gives this:

```
    Wallclock: 0x8566a6a0
    Dispatch: callout 0 error 0 Size 070 X2EA_PKT (0) ReorderId 10a1 Stream Fab (24) UdpCsum 4ea5
    PType IPV4 (2) SubType 0 PfeMaskF 0 OrigLabel 0 SkipSvc 0 IIF 00142
    FC 00 DP 0 SkipSample 0 SkipPM 0 L2Type None (0) IsMcast 0 SrcChas 1 MemId 0
    PfeNum 7f PfeMaskE 0 FabHdrVer 1
    RwBuf 00000000 SvcCtx 0 ExtHdr 0 PfeMaskE2 0 Token 000311
    085082404ea52000014200010fe10000000000000311 4500005200000000 3e11fa10ac1014b5 ac1015b5003f003f
003ec1259cbb38ff 72e3f5b649786960 e000000010111213 06dd86dc001a1a1b 1c1d1e1f20212223 2425262728292a2b
2c2d2e2f30313233 4a20
    Wallclock: 0x8566ac12
    Reorder: EA2X_REORDER_SEND_TERMINATE (d) HasTail 0 Stat 1
    ReorderId 10a1 Color 0 Qop ENQUEUE (2) Qsys WAN (0) Queue Wan (68)
    Stats Map 0 len_adjust 20 cnt_addr 00214c8 WanCookie dea8
```

```
    d50a10400068f86605b2c803d0400214c8dea8 0000000000000000 7000030000000200 4e961475385b4e96
1475385a08004500 0052000000003e11 fa10ac1014b5ac10 15b5003f003f003e c1259cbb38ff72e3 f5b649786960e000
00001011121306dd 86dc001a1a1b1c1d 1e1f202122232425 262728292a2b2c2d 2e2f303132334a20
```

We confirm that traffic came from the PFE 0 / Slot 9 because it was received by fabric stream 36 (0x24). The first lookup shows us that the traffic should now be forwarded to the WAN queue 104 (0x68). Strange, isn't it, because there is no physical port attached to this PFE. Actually it's an internal stream called *loopback*. To retrieve the statistics of this specific queue, you first need to find the AFT token ID assigned to the CoS configuration of the lt-11/7/0 interface:

```
mx2020-fpc11:pfe> show class-of-service interface scheduler brief | match  lt-11/7/0
Interface Schedulers:
 Name             Index        Mode           Levels    Node-Token
[...]
 lt-11/7/0          478        Port             1          7376
```

Then, if you resolve this token ID (7376), you should see the CoS queue configuration and statistics of the lt-11/7/0. You will observe that the queue 0 conveys the 100Kpps – which is, based on the queue configuration table, the absolute queue index 104 (the queue provided by the first LUSS lookup).

```
mx2020-fpc11:pfe> show sandbox token 7376
Table: Queue Configuration
----------------------------------------------------------------------------------------------------
--------
  Index | Shaping-Rate | Transmit-Rate |  Burst | Weight | G-Priority | E-Priority | Tail-Rule |
WRED-Rule |
--------+--------------+---------------+--------+--------+------------+------------+-----------+----
-------+
   104  |    10.0G     |     2.5G     | 125.0M |   25   |     GL     |     EL     |    1037   |     0    |
   105  |    10.0G     |     2.5G     | 125.0M |   25   |     GL     |     EL     |    1037   |     0    |
   106  |    10.0G     |     2.5G     | 125.0M |   25   |     GL     |     EL     |    1037   |     0    |
   107  |    10.0G     |     2.5G     | 125.0M |   25   |     GL     |     EL     |    1037   |     0    |
   108  |    10.0G     |     0.0      | 125.0M |   1    |     GL     |     EL     |    279    |     0    |
   109  |    10.0G     |     0.0      | 125.0M |   1    |     GL     |     EL     |    279    |     0    |
   110  |    10.0G     |     0.0      | 125.0M |   1    |     GL     |     EL     |    279    |     0    |
   111  |    10.0G     |     0.0      | 125.0M |   1    |     GL     |     EL     |    279    |     0    |
--------+--------------+---------------+--------+--------+------------+------------+-----------+----
-------+

Queue Statistics:
  PFE Instance : 7
                            Transmitted                                                  Dropped
                  Bytes                    Packets                    Bytes
Packets
----------------------------------------------------------------------------------------------------
-----------------------------------
 Queue:0    23383348456(    83189320 bps)    224839889(     99988 pps)         0(         0 bps)
0(        0 pps)
 Queue:1            0(          0 bps)          0(          0 pps)          0(          0 bps)
0(        0 pps)
 Queue:2            0(          0 bps)          0(          0 pps)          0(          0 bps)
0(        0 pps)
 Queue:3        189968(          0 bps)       1248(          0 pps)          0(          0 bps)
```

```
0(         0 pps)
 Queue:4             0(         0 bps)          0(         0 pps)          0(         0 bps)
0(         0 pps)
 Queue:5            62(         0 bps)          1(         0 pps)          0(         0 bps)
0(         0 pps)
 Queue:6             0(         0 bps)          0(         0 pps)          0(         0 bps)
0(         0 pps)
 Queue:7             0(         0 bps)          0(         0 pps)          0(         0 bps)
0(         0 pps)
```

This traffic dequeued from the XQSS is reinjected (as it comes from a classic WAN interface) into the MQSS for a second round. The second part of the packet capture output gives us information about this second lookup:

```
    Wallclock: 0xe62045f6
    Dispatch: callout 0 error 0 Size 080 X2EA_PKT (0) ReorderId 0390 Stream Lbk (402) UdpCsum efaa
    Magic dea Tun 1 IngQ 0 RefCnt 0 OptLbk 0 TailHndl 0000000000000000
    PType BRIDGE (7) SkipSvc 0 Probe 0 LM 0 CCC 0 Channel 0003 FC 00 DP 0 Hash 00000 L2Type Ether (2)
vrf_lpbk 0
    01c84020efaadea8000000000000000070000300000000200 4e961475385b4e96 1475385a08004500
0052000000003e11 facaac101458ac10 1558003f003f003e 805b9cbb38ff72e3 f5b6497869600000 000010111213ce84
e0b9001a1a1b1c1d 1e1f202122232425 262728292a2b2c2d 2e2f303132334a20
    Wallclock: 0xe6204cc6
    Reorder: EA2X_REORDER_SEND_TERMINATE (d) HasTail 0 Stat 0
    ReorderId 0390 Color 0 Qop ENQUEUE (2) Qsys FAB (1) Queue 0002c
    PType IPV4 (2) SubType 0 PfeMaskF 0 OrigLabel 0 SkipSvc 0 IIF 001f2
    FC 00 DP 0 SkipSample 0 SkipPM 0 L2Type None (0) IsMcast 0 SrcChas 1 MemId 0
    PfeNum 2c PfeMaskE 0 FabHdrVer 1
    RwBuf 00000000 SvcCtx 0 ExtHdr 0 PfeMaskE2 0 Token 0002b0
    d0390050002c780005b1c81300030000002000001f2000105810000000000002b0 4500005200000000
3d11fbcaac101458 ac101558003f003f 003e805b9cbb38ff 72e3f5b649786960 0000000010111213 ce84e0b9001a1a1b
1c1d1e1f20212223 2425262728292a2b 2c2d2e2f30313233 4a20
```

As seen here, the second lookup shows that the packet comes from the WI internal loopback stream 1026 (0x402) and must be forwarded to the fabric queue 44 (0x2c). You can check, if needed, the WI stream 1026 (the last `0` means ingress), which is considered as loopback:

```
(mx vty)# show mqss 7 phy-stream list 0

Ingress PHY stream list
-----------------------
-------------------------------------------------------------------------
Stream    Type        Enabled    NIC    PIC    Connection    Traffic Class
Number                           Slot   Slot   Number
-------------------------------------------------------------------------
[…]
1026      Loopback    Yes        11     7      54            0 (High)
[…]
```

We could also enable a WI counter to see if we received the reinjected traffic coming from the XQSS (WAN queue 104). Remember, the last number `2` is the stream number value – 1024 (here: 1026 - 1024 = 2)

```
(mx vty)# test mqss 7 wi stats stream 0 0 2
(mx vty)# show mqss 7 wi stats
```

```
WI statistics
-------------
Tracked stream statistics
-------------------------
------------------------------------------------------------------------------
Track Stream Stream Total Packets        Packets Rate         Total Bytes
      Mask   Match                      (pps)
------------------------------------------------------------------------------
0     0xff   0x2    539346              99984                61485421
```

Finally, before leaving the PFE 7, let's check the fabric queue 44 (0x2c) which is assigned to PFE 0 / Slot 11:

```
(mx vty)# show mqss 7 sched-fab q-node stats 44

Queue statistics (Queue 0044)
-----------------------------
------------------------------------------------------------------------------
Color  Outcome           Counter  Total                Rate
                         Name
------------------------------------------------------------------------------
All    Forwarded (Rule)  Packets  6284740952           100331 pps
                         Bytes    1145278412220        78660232 bps
All    TAIL drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
0      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
1      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
2      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
3      WRED drops        Packets  0                    0 pps
                         Bytes    0                    0 bps
```

At the end, the traffic will reach the PFE 0 / slot 11 and after a last lookup of the LUSS PFE 0 / slot 11 it will leave the router through the et-11/0/0 port.