JUNIPER NETWORKS

Routing

DAY ONE: CGNAT UP AND RUNNING ON THE MX SERIES

The new MS-MPC or MS-MIC service cards for the MX series have advanced processing that supports dynamic NAT or advanced NATing features like PAT, or ALG features such as DPI packet rewrites. It's all here for you to check out and test in your lab.

By Joseph Naughton

DAY ONE: CGNAT UP AND RUNNING ON THE MX SERIES

CGNAT, which is also known as *Large Scale NAT*, is a buzzword for a highly-scalable NAT device that sits between the CPE and a core network. If the device being used is an MX Series, well now, that device is *very* scalable, and it can take your current Network Address Translation usage and truly make it *carrier grade*. It's all in how you set up the MX.

What you need is a JTAC engineer to explain the ins and outs of the MX Series, and that's what Joe Naughton does in this book. He provides the configurations, the feature sets, the application layer gateways, and the syslogs you need to make the MX hum. There's a troubleshooting chapter written as only a JTAC engineer can, as well as a scalable use case that puts some load balancing MX features to the test.

However you define CGNAT it begins with MX.

"This is just the book you need if your current NAT needs are starting to scream at you. It's filled full of useful MX Series insights and even includes a service provider Use Case that puts it all together. This one sits on my desk."

David Roy, IP/MPLS NOC Engineer, Orange France
blogger: junosandme.net

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

- Understand the hardware needed for your network to go carrier grade.
- Understand the different NAT configurations of the MX Series and how they can fit into your network's needs.
- Monitor and manage the MX Series when it is configured in a CGNAT solution.
- Build a working model in your lab for testing and prototyping.

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

Published by Juniper Networks Books

ISBN 978-1-941441-47-3

51600

9 781941 441473

JUNIPER NETWORKS

Day One: CGNAT Up and Running on the MX Series

By Joseph Naughton

JUNIPER NETWORKS

© 2017 by Juniper Networks, Inc. All rights reserved.
Juniper Networks and Junos are registered trademarks of Juniper Networks, Inc. in the United States and other countries. The Juniper Networks Logo and the Junos logo, are trademarks of Juniper Networks, Inc. All other trademarks, service marks, registered trademarks, or registered service marks are the property of their respective owners. Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice.

Published by Juniper Networks Books
Author: Joseph Naughton
Technical Reviewers: Neeraj Gupta, Prakash Channagouda, Vikramadhithya Karamched, Jacopo Pianigiani
Editor in Chief: Patrick Ames
Copyeditor: Nancy Koerbel
Illustrator: Karen Joice

ISBN: 978-1-941441-47-3 (print)
Printed in the USA by Vervante Corporation.

ISBN: 978-1-941441-48-0 (ebook)

Version History: v1, March 2017
2 3 4 5 6 7 8 9 10

http://www.juniper.net/dayone

About the Author
Joseph Naughton has seventeen years experience supporting solutions in the networking industry. He is the Technical Lead in JTAC at Juniper Networks. Prior to supporting the best of breed Mobile Packet Core products, he has supported policy solutions, including SRC and Steel Belted RADIUS, the BRAS line, and in a former life, VPNs, firewalls, and Shiva's Lan Rover products.

Welcome to Day One

This book is part of the *Day One* library, produced and published by Juniper Networks Books.

Day One books were conceived to help you get just the information that you need on day one. The series covers Junos OS and Juniper Networks networking essentials with straightforward explanations, step-by-step instructions, and practical examples that are easy to follow.

The *Day One* library also includes a slightly more comprehensive and longer suite of *This Week* books, whose concepts and test bed examples are more similar to a weeklong seminar.

You can obtain publications from either series in multiple formats:

- Download a free PDF edition at http://www.juniper.net/dayone.
- Get the ebook edition for iPhones and iPads from the iTunes/ iBooks Store. Search for *Juniper Networks Books* or the title of this book.
- Get the ebook edition for any device that runs the Kindle app (Android, Kindle, iPad, PC, or Mac) by opening your device's Kindle app and going to the Kindle Store. Search for *Juniper Networks Books* or the title of this book.
- Purchase the paper edition at either Vervante Corporation (www.vervante.com) for between $12-$28, depending on page length.
- Note that most mobile devices can also view PDF files.

What You Need to Know Before Reading This Book

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

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

- You are familiar with and versed in using the Junos CLI.
- You have a basic understanding of IPv4 and IPv6.
- You have access to a lab with at least one MX Series router, one Ethernet switch (with port mirroring capability), and one server or workstation. It is ideal if your MX Series has MS-MPC or MS-MIC service cards.

What You Will Learn by Reading This Book

After reading this book you will be able to:

- Understand the hardware needed for your network to go carrier grade.
- Understand the different NAT configurations of the MX Series and how they can fit into your network's needs.
- Monitor and manage the MX Series when it is configured in a CGNAT solution.
- Build a working model in your lab for testing and prototyping.

MORE? It's highly recommended you go through the technical documentation and the minimum requirements to get a sense of CGNAT and the Junos OS before you jump into this book. The Juniper technical documentation on CGNAT can be found here: https://www.juniper.net/documentation/en_US/junos14.1/topics/topic-map/nat-junos-cgn-implementations.html.

Preface

This book is not meant to be a network design book or to serve as training material for Network Address Translation (NAT), or how NAT can be generically applied in one's network – instead you can use this book to educate yourself on how the MX Series NAT solution and features can fit your operational NATing needs.

By using an imaginary regional service provider called Massachusetts Telcom or MassT for short as a model, this book allows you to see how the MX Series can be used as a very powerful and flexible NATing solution. You can follow along as MassT sets up several different types of NAT scenarios using its MX Series to fit its needs.

Most readers of this book will understand the majority of NAT terms and acronyms, since many of the terms Juniper uses are generic, but some terms are unique to the Junos OS. Over the next several pages this book will explain the basics you need to know before you jump into Chapter 1.

Acronyms used in this *Day One* book include:

- PAT: Port Address Translation
- NAT: Network Address Translation
- PBA: Port Block Allocation
- EIM: End Point Independent Mapping
- EIF: End Point Independent Filtering
- ALGs: Application Layer Gateways
- AMS: Aggregated Multi Service

IMPORTANT

Throughout this book the author uses "MX" as an abbreviation for the Juniper Networks MX Series 3D Universal Edge Router. Given the complexity of the text, the author hopes this modest abbreviation will aid in the book's readability. View all of Juniper Networks routing platforms, and the complete MX Series family, at: https://www.juniper.net/us/en/products-services/routing/mx-series/.

Carrier Grade NAT

So, what is Carrier Grade NAT, aka CGNAT, as opposed to plain NAT? CGNAT, also known as Large Scale NAT, is just a buzzword for a highly scalable NAT device that sits between the CPE and a core network. If the box being used as a NATing box is an MX Series, it is very scalable, so if you are using NAT on the MX, consider yourself using CGNAT!

Let's lay out a list of some of the actual NAT technologies that comprise the CGNAT buzzword that will be configured in this book:

- *NAT 44* is IPv4 only. NAT 44 is truly traditional NAT and has been used to fight off IPv4 starvation until IPv6 is fully adopted in every facet of the network. NAT44 can be used to hide the subscriber's true IP address for security reasons or simply to deal with getting subscriber traffic from a private network onto a public network.
- *NAT 66* is IPv6 only. NAT 66 is the IPv6 world's version of NAT44.
- *NAT 46* is a one-to-one NAT mapping translating a private IPv4 to an IPv6 address so that an IPv4 host can communicate with an IPv6 host/server.
- *NAT 64* is used to assign IPv6 IP addresses to the client premise while allowing the NATing router to handle translation to IPv4 network hosts when a DNS64 server is used.
- *Destination NAT,* or dNAT, is used often to hide the real IP addresses of servers from the public network. DNAT is used to translate the destination address versus the source address.

Some readers may know these different NAT technology types as existing in the more generic terms (yet another level of classification) of Static NAT and Dynamic NAT. So, let's also clear up what this book will consider as Static NAT and what it means by Dynamic NAT:

- Static NAT happens when the private address of the end user maps to the same NAT'd address every time they have to traverse the MX as a NATing device. Static NAT requires an equal-sized NAT pool based on the range of source-IP addresses you define as being the private host range(s). If the range of potential private addresses that can be NAT'd is 100, then the NAT pool needs to be at least 100 in size.

- Dynamic NAT means you will get a random NAT'd address each time you traverse the NATing device. The NATing device does not need to define an equal-sized NAT pool in regard to the number of potential private source IP addresses that will reflect your client subscriber's range.

As you read through this book and the different configurations around these different NAT types, you need to understand that the MX also has different categories of NAT setup, essentially Inline NAT versus using the MS-MPC or MS-MIC service cards. Let's review this right now before you move on, so it is clear in your mind.

Inline NAT Versus NAT on the Service Cards

Inline NAT on the MX is applied when packets are being serviced for NAT in the forwarding plane, much like what is done with standard firewall and policer setups in the Junos OS. With Inline NAT, packets do not need to be steered to a service-PIC hosted on a MX service card for advanced processing. Since the MX does not need to steer packets to the MS-MPC or MS-MIC service cards, the MX can achieve line rate, low latency NAT translations with Inline NAT. So, performance wise, Inline NAT is fantastic. But without advance processing by MS-MPC or MS-MIC service cards, the MX cannot support dynamic NAT or advanced NATing features like PAT or the ALG features such as DPI packet rewrites. Service providers will look to use Inline NAT with such NAT types as basic nat-44, basic nat-66, twice basic nat-44 and dNAT (destination-NAT). Other NAT technologies will require a service card.

As we dig into what each of the NAT types are on the MX and how to configure them, this book will also try to point out which setup requires a service card for processing the NAT type and which setup requires only MPC line cards for an Inline NAT setup. It's important to understand these differences, since doing so will allow you to determine what type of hardware setup you require to fit your need.

NOTE Inline NAT works on the MPC type of line cards. Older cards do not support Inline NAT. As for any newer cards that Juniper releases, you do not need to check the data sheets or documentation for Inline NAT support.

Different MX Series Service Cards

There is one more minor, yet important, topic to review – the different service cards for the MX Series. As of the writing of this book in early 2017, MS-DPC, MS-MPC, and MS-MIC cards are the three options you have for the MX platform.

The MS-DPC and MS-MPC are the full line card options for your MX-240, MX-480, and MX-960. These cards take up a whole FPC slot. The MS-MPC is the newer of the two cards with more processing power and memory; it has four NPUs versus two NPUs on the MS-DPC. It also has 32GB of memory per NPU versus the 8GB per NPU on the MS-DPC. The MS-DPC is the legacy card and some of the configuration settings for it differ from configuration settings for the MS-MPC and MS-MIC.

NOTE It should be noted this book *does not* focus on the MS-DPC card.

The MS-MIC, on the other hand, is a service MIC with 16GB that can fit the MPC-Type1 and MPC-Type2 line cards on the MX-240, MX-480, and MX-960. In addition, the MS-MIC can even fit into the MX-80 and MX-104 chassis bringing advanced services to these platforms and it can be placed into the MX-2010 and MX-2020.

As stated previously, this book will focus on the MS-MPC and MS-MIC service cards and their configurations. When using the older MS-DPC service cards please check the Juniper documentation for differences between using them and the MS-MPC and MS-MIC cards: https://www.juniper.net/documentation/en_US/junos14.1/topics/topic-map/nat-junos-cgn-implementations.html.

Let's get started!

Chapter 1

Configuration

There is more to setting up NAT on the MX Series than one might imagine. This is because the Junos OS has a very, very flexible range of options in order to fit most operators' needs. This first chapter provides more than basic knowledge of different CLI options – it attempts to show you what each option can do, and how that option can be applied in your network. It also provides some insight into how you can manage the solution, what to look for when analyzing NAT'd sessions on the box, how to understand what is actually occurring once the box is in production, and handling paying customers' traffic.

In order to explain the various configurations in a manner that can be understood, this chapter is organized to make understanding the building blocks of the CGNAT setup easier, since, depending on your needs, the configuration can be quite advanced. So in this chapter you will learn how each section plugs into another, since all sections are truly needed before your setup works in even its most basic form. The sections in this chapter are:

- *Service Interfaces*
- *Services NAT*
  - *Pools*
  - *Rules*
- *Service Sets*

The Service Interfaces detail the PICs on the service cards, or the PFEs on the MPC cards, for Inline NAT. All traffic that needs to be processed for NATing capabilities passes through these logical interfaces. These interfaces are then tied to service sets – the component that ties the NAT-Rules to a defined service interface. That NAT-Rule determines which traffic will be NAT Translated. If the traffic will be NAT'd the NAT-Rule calls the NAT-Pool to be used. NAT-Pools are where we are going to configure our NAT IP address pools, and our port configuration for PAT.

This material may seem confusing right now, but once you're done with this first chapter, it will make much more sense and you will be able to understand what each configuration section really means to the functionality of your particular setup.

Service Interfaces

Okay, let's dig in to the configuration and start with the service interfaces, which represent the PICs on the MS-MPC/MS-MIC cards, or the PFEs on the MPC cards, for Inline NAT.

Services interfaces can be either a logical `si` interface or a logical `ms` interface. The `sis` interfaces are used for inline NAT where the NAT servicing is handled on the MPC line card itself within the packet forward engine. The `ms` interfaces, on the other hand, are used with the MS-MPC/MS-MIC service cards. These service cards are being employed because of the card's support for advanced NAT features.

NOTE Remember, the service interface is not tied to a physical interface like an ATM or Ethernet interface is – it is truly a logical interface used to process traffic traversing the MX from the ingress or egress physical interface and manipulating it before it is sent off to its destination.

Let's look at the Inline NAT setup first. Under the chassis configuration hierarchy you need to set the `inline-services` option with a bandwidth value for the amount of NAT services traffic you want this PFE to handle. Whenever possible this should be the PFE on the MPC line card hosting the port that is the egress or ingress point for handling the data from the private network so that you do not have to jump the fabric to get to the PFE that will handle the NATing of the packet. When using interfaces like an AE or IRB this may not always be possible.

NOTE To use the Inline NAT feature you have to use a MPC card. The older DPC line cards cannot anchor services inline on their PFEs.

So, for our example, there is a 16-port, 10G card in FPC slot 3. For each PFE the MPC card has, you need to set a PIC for the PFE you want to use for Inline NAT. This PIC will map to an `si` interface. In our example, the physical interface being used is hosted on PFE 0 set to PIC 0:

```
[edit chassis]
fpc 3 {
    pic 0 {
        inline-services {
            bandwidth 10g;
        }
    }
}
```

When using the `bandwidth` option you can define the amount of bandwidth in gigabits per second to reserve the bandwidth for tunnel services. On MX Series routers, the bandwidth values can be `1g`, `10g, 20g, or 40g`.

Now, let's set up a logical `si` interface to map to PIC 0 hosted in FPC slot 3. Make sure you set `family inet` if you want to NAT IPv4 traffic and `inet6` for IPv6:

```
si-3/0/0 {
    unit 0 {
        family inet;
        }
}
```

If you were also going to use the second, third, and fourth PFE on that line card for INLINE NAT you would then use the same steps for creating si-3/1/0, si-3/2/0, and si-3/3/0.

NOTE Check the Juniper documentation for your line cards to check the number of PFEs they may have, since this varies among line cards.

That's it for setting up the service interface for Inline NAT! Very simple. Now it's time to look at the service interface for when a service card is being used.

Now, as stated before, and as will be stated again throughout this book, there are many NAT setups that require the advanced processing that the service cards bring to the table – think Port Address Translation (PAT), or ALGs/DPI. If you are thinking along these lines you are going to be using at least one service card. So let's see how you get a `service-interface` set up for these types of scenarios using the MS-MPC service card.

Let's add a MS interface for each service PIC to be used for our CGNAT solution. Each MS interface can be used for a single service-set or can be

used against many different service-sets based on your overall configuration. Below we are adding a MS interface that will represent the first service PIC on a MS-MPC card in FPC slot 1. It is set up in a very generic manner to fit both of our different service-set types (and will make more sense to you very shortly):

```
[edit interfaces ms-1/0/0]
unit 0 {
    family inet;
}
unit 1 {
    family inet;
    service-domain inside;
}
unit 2 {
    family inet;
    service-domain outside;
}
```

Now the MS-MPC has four service PICs in total. If you want to use the other three service PICs, do the same for ms-1/1/0, ms-1/2/0, and ms-1/3/0.

This is a great place to demonstrate how the MX also allows you to use load balancing and redundancy when using the MS interfaces with the MS-MPC card. The AMS interface has a feature that allows one service PIC to take over for another service PIC when it goes down. Also, the AMS interface can be used to load balance packets across multiple service PICs from the same service set. The AMS interface is detailed in Chapter 4, but for now, let's show you the basics.

Our example showcases a MS-MPC card in FPC slots 1, 2, and 3. Each card will only use one of its four service PICS in this AMS bundle, with ms-1/0/0 and ms-2/0/0 being used to load balance the traffic received. MS-3/0/0 is the backup service PIC. Let's configure redundancy in this example across multiple MS-MPC cards to separate the AMS bundle across multiple cards to avoid outages due to any physical issues on one whole card or FPC slot:

```
[edit interfaces ams0]
load-balancing-options {
    member-interface mams-1/0/0;
    member-interface mams-2/0/0;
    member-interface mams-3/0/0;
    member-failure-options {
        redistribute-all-traffic {
            enable-rejoin;
        }
    }
    high-availability-options {
        many-to-one {
            preferred-backup mams-3/0/0;
        }
```

```
    }
}
unit 1 {
    family inet;
}
unit 2 {
    family inet;
    service-domain outside;
}
```

NOTE A single `ms` interface can be a secondary for multiple `ms` interfaces. This setup can effectively create N:1 up to 12:1 in version 14.2R5.

The advantage of using the AMS interface is to provide the least subscriber impact in the event of a service PIC failure. The AMS bundle also allows for a simple load balancing setup when you have enough potential NAT traffic that one service PIC alone cannot handle.

NOTE One potential pitfall you should be careful about in your setup is to the possibility of a full MS-MPC blade or FPC slot failing. Assigning the backup MS interfaces from the same card as all of your active interfaces under the same AMS interface is not the best option if other MS-MPC or MS-MIC service cards are installed in the system.

All right, after configuring the services interface the easiest part of the configuration is done. Next, let's dive into the services hierarchy, which is one of the more detailed and difficult sections to explain. Why? Well, there is quite a bit of information to cover. But truth be told the services hierarchy is the real core of the CGNAT setup on the MX Series, so stay with it, or even read through it a couple of times, and the rest will be all downhill. Promise.

Services NAT

There is going to be plenty to do under the services hierarchy in this configuration, so let's jump in under the services / NAT hierarchy and go from there. You are going to configure your NAT IP address pools, your port configuration for when PAT is needed, and look at some other more unique options you can leverage with your NAT pools.

Pools

The following is an outline of some of the options you can set as you define your pool. Take a quick look at what's here and familiarize yourself with the possibilities. This book tries to explain these options by giving you a full understanding of what you might need to set up to make the pool work for your setup's needs.

```
[edit services nat]
pool nat-pool-name {
    address prefix;
    address-range low value high value;
    mapping-timeout 300; /* Min 120, Max 86400, default 300 */
    port (automatic | range low minimum-value high maximum-value | random-allocation) {
        secured-port-block-allocation {
            block-size 256; /* Min 64, Max 64512, default 128 */
            max-blocks-per-user 8; /* Min 1, Max 512, default 8 */
            active-block-timeout 300; /* 0(default), Min 120secs, Max 86400 */
    }
    address-allocation round-robin
}
```

This book's example starts by creating a pool called `nat44` that will be used for a simple static one-to-one NAT setup where you will be NATing IPv4 to IPv4 using inline NAT. For the NAT pool you can specify a single specific address, a prefix, or an address range to be used for this pool's NAT'd IP addresses:

```
[edit services nat]
pool nat-pool-name {
    address prefix;
    address-range low : high value;
    mapping-timeout 300; /* Min 120, Max 86400, default 300 */
    port (automatic | range low minimum-value high maximum-value | random-allocation) {
        secured-port-block-allocation {
            block-size 256; /* Min 64, Max 64512, default 128 */
            max-blocks-per-user 8; /* Min 1, Max 512, default 8 */
            active-block-timeout 300; /* 0(default), Min 120secs, Max 86400 */
    }
    address-allocation round-robin;
}
```

So now let's set up an address prefix for the `pool nat44`, a pool that will be used to assign traffic that is to be NAT'd with an address from the 156.100.100.0/24 range:

```
[edit services nat]
pool nat44 {
    address 156.100.100.0/24;
}
```

NOTE Addresses that are not allowed in to be used in the NAT pool include: Martian, multicast, and loopback addresses.

Now, in simple static NAT setups such as this one for pool nat44, the port configuration section is skipped since it is only needed when PAT is utilized. At this point, though, let's create a second pool called `natpat44` so you can get a PAT setup going. Remember, you need a service card for this NAT type. Run the `show chassis hardware` command to verify you have a MS-MPC or MS-MIC card in one of the slots:

```
user@re0> show chassis hardware
Hardware inventory:
Item             Version  Part number  Serial number     Description
Chassis                                JN109689EAFA      MX960
Routing Engine 0 REV 07   740-013063   1000743729        RE-S-2000
CB 0             REV 08   710-021523   ABBG0290          MX SCB
FPC 0            REV 13   750-038768   CACP8307          MS-MPC
  CPU                     BUILTIN      BUILTIN           MS-MPC-PMB
  PIC 0                   BUILTIN      BUILTIN           MS-MPC-PIC
  PIC 1                   BUILTIN      BUILTIN           MS-MPC-PIC
  PIC 2                   BUILTIN      BUILTIN           MS-MPC-PIC
  PIC 3                   BUILTIN      BUILTIN           MS-MPC-PIC
```

Okay, look at the port section under the pool here:

```
[edit services nat]
pool nat-pool-name {
    address prefix;
    address-range low value high value;
    mapping-timeout 300; /* Min 120, Max 86400, default 300 */
    port {
        automatic ( sequential | random-allocation)
        range
        secured-port-block-allocation {
             block-size 256; /* Min 64, Max 64512, default 128 */
             max-blocks-per-user 8; /* Min 1, Max 512, default 8 */
             active-block-timeout 300; /* 0(default), Min 120secs, Max 86400 */
    }
}
```

You can see that under the port hierarchy you have an `automatic` option which really means "*I will not define any specific port ranges for PAT and I will have the MX automatically use ports 1024 thru 65535*." The ports that are used by the `automatic` option for PAT translation are the 'ephemeral' or non-default ports (1024 – 65535), so operators will use the `automatic` option when they are not concerned about which ports the MX uses when performing PAT. Under the `automatic` option there are two more options you can choose from. These are `sequential` and `random-allocation`.

Using the `sequential` option means that when the MX is assigning ports 1024 thru 65535, it starts assigning them from port 1024 and counts up for each sequential flow. When you set the `automatic` option to `random-allocation` it tells the MX that the starting point for a free port search is random, and the port for each sequential session is chosen randomly. Some service providers may apply `random-allocation` instead of `sequential` to make port prediction tougher (the ability for someone on the public side to predict which port is assigned to a NAT'd session).

As stated for our example of PAT, let's create the pool called `natpat44`, in this example setting a prefix for the IP addresses to be used on the NAT sessions, and let's also have the MX assign any free port from 1024 through 65535 in a sequential manner for each new session by using the `automatic` setting under `port` along with `sequential`:

```
[edit services nat pool natpat44]
address 100.100.100.0/24;
port {
    automatic {
        sequential;
    }
}
```

One thing to point out while looking at the PAT section of the pools is that when you are within the port hierarchy, and if you do not use the `automatic` setting to select the port ranges, you have to choose the range option and define the range of ports you want to be used for PAT:

```
[edit services nat pool natpat44]
address 100.100.100.0/24;
port {
    range low 8888 high 9999;
}
```

By default, the ports under the `range` option are assigned to each session in order, starting from the lowest value and then counting up sequentially for each new session. But to fight port predictability for defined ranges, you can change this to `random-allocation` as shown here (if that is your desired behavior):

```
port {
   range low 8888 high 9999 random-allocation;
}
```

Let's change our example pool `natpat44`, and instead of automatically assigning ports from 1024 through 65535 in sequential order, let's define a port range from 8888 through 9999, and assign these ports randomly to each new flow as follows:

BEST PRACTICES

When using any of the dynamic NAT technologies offered by the MX Series, keep in mind that you need to be aware of the NAT pool size and if you are using more of those potentially valuable public IP addresses for your NAT pool than you truly need! When you are not using the PBA or Deterministic NAT feature, the max pool size should be a /24 if you are using a single service PIC for your NAT pool. More IP addresses in the NAT pool than a /24 would not be used on the service PIC, since it would hit a memory limit. A MS-MPC can manage fifteen million max sessions per service PIC, and the MS-MIC

can manage seven million sessions. That means the total number of sessions used in a dynamic NAT setup can fit nicely into a /24. (The math is: 254 x 65535-1024k = 16 million ports.)

Let's take a step back and take a look at a few operational `show` commands. First, we'll let's go back and set the pool to assign the ports to `automatic` with the `sequential` order:

```
[edit services nat pool natpat44]
address 100.100.100.0/24;
port {
    automatic {
        sequential;
    }
}
```

Using one of our operational CLI commands, let's create an example showing the NAT mappings when using the `natpat44` NAT pool with the ports set to be automatically assigned in a sequential order. In the example, you'll notice there are four sessions originating from the 14.0.0.0/16 network being NAT'd with PAT against each of the four different private subscriber source IPs. Using the example pool above, the MX will assign 100.100.100.1:1024 first, and then 100.100.100.2:1024, and then 100.100.100.3:1024, and so on, until it gets to 100.100.100.255:1024 and then it will wrap around and assign 100.100.100.1:1025. This is how the sequential feature for the port assignment works:

```
labroot@MX960# run show services sessions
Service Set: nat44, Session: 100735100, ALG: none, Flags: 0x0600, IP Action: no, Offload: no,
Asymmetric: no
UDP            14.0.0.8:22391  ->     197.100.1.8:4000   Forward  I           26772
UDP         197.100.1.8:4000   ->   100.100.100.1:1024   Forward  O               0

Service Set: nat44, Session: 134243880, ALG: none, Flags: 0x0600, IP Action: no, Offload: no,
Asymmetric: no
UDP           14.0.110.45:40021 ->     197.100.1.8:4000   Forward  I           26139
UDP         197.100.1.8:4000    ->   100.100.100.2:1024   Forward  O               0

Service Set: nat44, Session: 100737200, ALG: none, Flags: 0x0600, IP Action: no, Offload: no,
Asymmetric: no
UDP            14.0.0.9:41777  ->     197.100.1.8:4000   Forward  I           27559
UDP         197.100.1.8:4000   ->   100.100.100.3:1024   Forward  O               0

Service Set: nat44, Session: 100736499, ALG: none, Flags: 0x0600, IP Action: no, Offload: no,
Asymmetric: no
UDP            14.0.20.4:56456 ->     197.100.1.8:4000   Forward  I           27630
UDP         197.100.1.8:4000   ->   100.100.100.4:1024   Forward  O               0
```

To understand the output you have just seen with the `show services sessions` command you should understand that 14.0.0.0/16 is the internal private network, 100.100.100.0/24 is our NAT pool used to

NAT the traffic, and 197.100.1.8 is a target on the public side towards which everyone is sending UDP traffic. The outputs displayed by the different `show` commands will be detailed in later, in the troubleshooting section of this book, so for now let's use them to show you how things work.

To help visualize this better, Figure 1.1 shows a simple example showing the 14.0.0.0/16 network sitting on one side of the MX and the public server 197.100.1.8 sitting on the other side. You can see when the 14.0.0.0/16 network sends traffic towards the 197.100.1.8 target through the MX that the 14.0.0.0/16 source addresses are NAT'd to an address from the 100.100.100.0/24 NAT pool range and the source port is also changed.

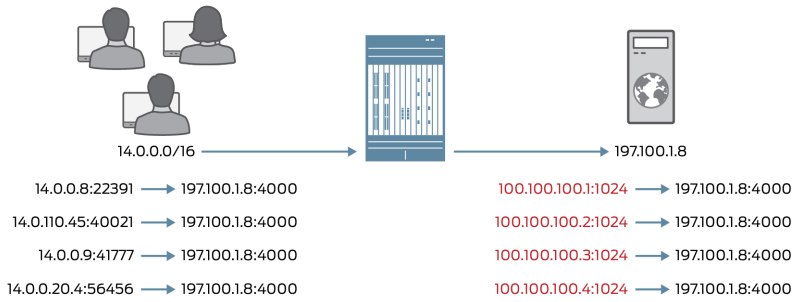

Figure 1.1 The 14.0.0.0/16 Network and the Public Server 197.100.1.8

When 197.100.1.8 needs to send traffic back it will send it to the 100.100.100.0/24 NAT'd range as the destination, and when this hits the MX it will get de-NAT'd back to the original private IP address and port as shown in Figure 1.2.

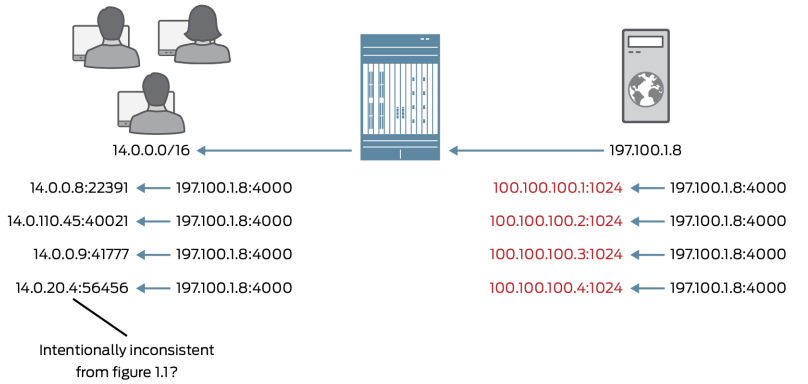

Figure 1.2 When 197.100.1.8 Needs to Send Traffic Back

NOTE The order in which the IP addresses are assigned will be discussed in just a few more paragraphs.

Next let's discuss an example showing the NAT/PAT mappings when using a NAT pool set with the `random-allocation` option under the port setting. You will see that the MX assigns 100.100.100.1 with one random free port to the first session, and then 100.100.100.2 with one random free port to the next session, and so on, until it gets to 100.100.100.255 and then it will wrap around and assign 100.100.100.1 with another free random port:

```
[edit services nat pool natpat44]
address 100.100.100.0/24;
port {
   automatic {
      random-allocation;
   }
}

labroot@MX960# run show services sessions

Service Set: nat44, Session: 67108867, ALG: none, Flags: 0x0000, IP Action: no, Offload: no,
Asymmetric: no
UDP            14.0.0.14:10098 ->     197.100.1.8:4000   Forward  I           18629
UDP         197.100.1.8:4000   ->  100.100.100.1:20857   Forward  O               0

Service Set: nat44, Session: 67108893, ALG: none, Flags: 0x0000, IP Action: no, Offload: no,
Asymmetric: no
UDP            14.0.10.3:60134 ->     197.100.1.8:4000   Forward  I           19729
UDP         197.100.1.8:4000   ->  100.100.100.2:25932   Forward  O               0

Service Set: nat44, Session: 67108901, ALG: none, Flags: 0x0000, IP Action: no, Offload: no,
Asymmetric: no
UDP            14.0.33.25:5532 ->     197.100.1.8:4000   Forward  I           18457
UDP         197.100.1.8:4000   ->  100.100.100.3:37017   Forward  O               0

Service Set: nat44, Session: 67108911, ALG: none, Flags: 0x0000, IP Action: no, Offload: no,
Asymmetric: no
UDP           14.0.1.12:56565  ->     197.100.1.8:4000   Forward  I           17219
UDP         197.100.1.8:4000   ->  100.100.100.4:16083   Forward  O               0
```

Get used to running the `show services sessions` command when setting up and testing your NAT setup on the MX, since it is the one command that will give you a ton of insight into what is happening on the MX with regard to your NAT settings. It is a great place to start analyzing the solution, or even troubleshooting. Also, note that the troubleshooting section of this book goes over quite a few show commands that will help you manage the MX Series NAT setup.

Address Allocation

So, that's how the ports are assigned when PAT is used, but now let's swing back and look at how the IP addresses are assigned. The `address-allocation` feature can be used for your pool to control the order in which IP addresses from the pool get assigned to translated sessions. Let's set a few address ranges under our NAT pool to show an example that uses more than just a single address prefix that was shown in the previous example NAT pools created for the pools called `nat44` and `natpat44`. This new pool will be called *CGN-1*, it uses three address-ranges, and it will use the `port automatic random-allocation` feature to randomly allocate ports between 1024° and 65535 to new sessions. Let's also add our new `address-allocation` option. Look for lots of stuff happening under this pool:

```
[edit services nat]
pool CGN-1 {
    address-range low 10.12.1.100 high 10.12.1.200;
    address-range low 10.12.1.206 high 10.12.1.210;
    address-range low 10.12.1.212 high 10.12.1.216;
    port {
    automatic {
        random-allocation;
        }
    }
    address-allocation round-robin
}
```

When the `address-allocation` feature is set to `round-robin`, the MX starts assigning the first, or the lowest, address in the NAT pool to the first session it processes, and based on the `port/automatic/random-allocation` setting it randomly assigns a source port from the pool.

In this example using the NAT pool CGN-1, when a session needs to be NAT'd the MX starts by assigning the address NAT IP 10.12.1.100 with one of the free ports for that under 10.121.1.100 being selected randomly. The next session is then assigned the address 10.12.1.101, with one of the free ports for that IP address selected randomly. This logic continues through 10.12.1.216 for each new session received by the MX that it needs to NAT. At this point you should understand that even if the first session has been released from the system, and the NAT IP address 10.12.1.100 and the port used by that first session are now available to be used as a NAT'd resource again, the MX will not even try to assign it to a new session until the MX has wrapped through all of the IP addresses of the NAT pool and assigned each IP address with one free port to a session. So, only when the MX has assigned all the IPs to a single NAT session once, would the MX then wrap back around to use 10.12.1.100 and another free port for that IP.

Take a look at the output below from the `show services sessions extensive` command; you can see that each sequential session has the next IP address from the NAT pool assigned to it with a random port between 1024 and 65535:

```
user@re0# run show services sessions extensive
ms-1/0/0
Service Set: ss1, Session: 167774734, ALG: none, Flags: 0x2000, IP Action: no, Offload: no,
Asymmetric: no
NAT PLugin Data:
  NAT Action:   Translation Type - NAPT-44
    NAT source       31.0.0.2:4012    ->    10.12.1.100:1830
UDP         31.0.0.2:4012  ->      33.0.0.2:4000  Forward  I         12669
  Byte count: 15445632
  Flow role: Initiator, Timeout: 32
UDP         33.0.0.2:4000  ->    10.12.1.100:1830  Forward  O           110
  Byte count: 0
  Flow role: Responder, Timeout: 32
Service Set: ss1, Session: 167772931, ALG: none, Flags: 0x2000, IP Action: no, Offload: no,
Asymmetric: no

NAT PLugin Data:
  NAT Action:   Translation Type - NAPT-44
    NAT source       31.0.0.2:50222    ->    10.12.1.101:29571
UDP         31.0.0.2:50222  ->      33.0.0.2:4000  Forward  I        820553
  Byte count: 15445632
  Flow role: Initiator, Timeout: 32
UDP         33.0.0.2:4000  ->    10.12.1.101:29571 Forward  O            0
  Byte count: 0
  Flow role: Responder, Timeout: 32
Service Set: ss1, Session: 167773848, ALG: none, Flags: 0x2000, IP Action: no, Offload: no,
Asymmetric: no

NAT PLugin Data:
  NAT Action:   Translation Type - NAPT-44
    NAT source       31.0.0.2:33866    ->    10.12.1.102:15971
UDP         31.0.0.2:33866  ->      33.0.0.2:4000  Forward  I           12
  Byte count: 15445760
  Flow role: Initiator, Timeout: 32
UDP         33.0.0.2:4000  ->    10.12.1.102:15971 Forward  O          892
  Byte count: 0
  Flow role: Responder, Timeout: 32
Service Set: ss1, Session: 167774799, ALG: none, Flags: 0x2000, IP Action: no, Offload: no,
Asymmetric: no

NAT PLugin Data:
  NAT Action:   Translation Type - NAPT-44
    NAT source       31.0.0.2:5008    ->    10.12.1.103:15971
UDP         31.0.0.2:5008  ->      33.0.0.2:4000  Forward  I        542111
  Byte count: 15445760
  Flow role: Initiator, Timeout: 32
UDP         33.0.0.2:4000  ->    10.12.1.103:15971 Forward  O          223
  Byte count: 0
  Flow role: Responder, Timeout: 32
```

```
Service Set: ss1, Session: 167772358, ALG: none, Flags: 0x2000, IP Action: no, Offload: no,
Asymmetric: no

NAT PLugin Data:
  NAT Action:   Translation Type - NAPT-44
    NAT source       31.0.0.2:22224    ->    10.12.1.104:32538
UDP         31.0.0.2:22224  ->      33.0.0.2:4000  Forward  I        420671
  Byte count: 15445760
  Flow role: Initiator, Timeout: 32
UDP         33.0.0.2:4000  ->    10.12.1.104:32538 Forward  O            2
  Byte count: 0
  Flow role: Responder, Timeout: 32
```

You should be aware that without setting the `address-allocation round-robin` option, with some of the older service cards used in the MX and M series product lines, the default behavior was to assign the first IP addresses in the NAT pool and keep using that IP for all NAT mappings until that IP had no more unassigned ports to use. So, in our example the 10.12.1.100 address would see lots of use if `address-allocation round-robin` was *not* set in our example pool above when using these older service cards. Potentially some subscriber's bad behavior would cause issues that could block an IP address from using a certain server for a period of time, and using the same IP address frequently could then cause other subscribers pain. (Possible bad behavior could range from someone cheating on a video gaming server to someone trying to hack a given node on the Internet.) With the MS-MPC and MS-MIC cards, you only assign IP addresses from the NAT pool using `round-robin` so these concerns are not needed any longer.

One other setting under the NAT pool that really needs clarification is the `mapping-timeout` setting. This setting can be enabled under the `pool` to determine how long the address pooling paired (APP) and Endpoint Independent Mapping (EIM) remain active (but this setting will not be important or really make much sense until a later portion of this chapter). These features are discussed later on in this book, so park this option and revisit it later. For now, let's just remember that the `mapping-timeout` can be set here in seconds:

```
[edit services nat]
pool nat44 {
    address 156.100.100.0/24;
}
mapping-timeout 300
```

A Few More Features

These are the basic settings needed to get a NAT pool working. There are a few more items to talk about that you may, or may not, need but you should at least be armed with the knowledge of what such features do. So next up are the `preserve parity` and `preserve range` options you

can enable under the port section. These options allow the operator to preserve the range or parity of the session's source port when the MX is using PAT to allocate a source port for an outbound connection. Let's examine:

```
[edit services nat]
pool nat-pool-name {
    address prefix;
    address-range low value high value;
    mapping-timeout 300; /* Min 120, Max 86400, default 300 */
    port (automatic | range low minimum-value high maximum-value | preserve-
parity | preserve-range) ;
            }
   }
```

Now, let's create an example. Say you need to configure the preserve-parity option because you are implementing some multiple media solutions that use Real-Time Transport Protocol (RTP) or because you expect your private subscribers to use RTP in some manner. The RTP protocol, which handles data transport, uses even ports, and its peer protocol, Real-Time Control Protocol (RTCP), which handles the control packets, uses odd ports. RTP is used with different streaming media types such as IP telephony, television services, and video teleconferencing. When preserve-parity is enabled the MX will follow the logic of finding an odd or an even port to assign to the NAT'd session so no unexpected issues occur with solutions that use protocols like RTP and RTCP. But let's say the MX does not have any available odd source ports in the NAT pool for the odd source range you need to NAT/PAT on, since all these ports are currently in use. What the MX will do is drop the packets/traffic you are trying to NAT, even if you still have available even ports in the NAT pool. This is how the MX behaves with the preserve-parity feature enabled for the NAT Pool in question.

The preserve-range feature is used when an operator needs to make sure their NAT'd sessions are mapped to either the range of defined/privileged ports 0-1023 or the ephemeral non-default ports (1024 – 65535), based on the source port received from the client side. If preserve-range is used and the MX does not have any available source ports for the source range you need, it will drop the traffic. Therefore, if preserve-range is enabled and the source port you received from the client side is 1011, but ports 0 through 1023 for all NAT IP address in the NAT pool are currently assigned to active NAT translated sessions, the MX will drop the new packet/traffic even if there are open ports in the 1024-66535 range that can still be used by the NAT Pool.

When using preserve-range you can expect source ports to be used by both the 0-1023 and 1024 – 65535 ranges. To deal with this situation

you do need to make sure your NAT pool configuration is correct or else the result will be sessions getting dropped unexpectedly. So you need to either have a defined range that overlaps both ranges, or else you need to use the `automatic` option. In this case the `automatic` option allows 0-65535 to be assigned. If you remember a few pages back, the `automatic` option normally uses 1024-65535, but when the `preserve-range` option is present the functionality changes just a bit.

Either of these two setups would work when the `preserve-range` is set since they both offer ports from each port range: the defined/privileged range and the ephemeral/non-default range:

```
pool natpat44 {
    address 100.100.100.0/24;
    port {
        range low 500 high 3500;
        preserve-parity;
        preserve-range;
    }
}

pool natpat44 {
    address 100.100.100.0/24;
    port {
        automatic;
        preserve-parity;
        preserve-range;
    }
}
```

Note that you cannot define two ranges under your port setting, one for the defined/privileged ports 0-1023 range and one for the ephemeral/non-default ports (1024 – 65535). You can only define an overlapping range if you need to define which ports can be used for translation, or else, as stated previously, you can just use the `automatic` option if *all* ports should be made available.

Let's use the `show services session` command to see what the MX is doing. In this example the MX is using NAPT44 with a pool and port range setup to be NAT'd like so (the pool is a /30 so you can see the IPs wrap around in the example output):

```
pool natpat44 {
    address 100.100.100.0/30;
    port {
        range low 5000 high 8000;
    }
```

In the following output you can see the MX has mapped a NAT'd IP and port to each session from the created pool. Starting from port

5000, it sequentially uses the next available IP with port 5000 for the next session. It does not matter what the subscriber source port is:

```
user@re0# run show services sessions

Service Set: nat44, Session: 167772172, ALG: none, Flags: 0x0000, IP Action: no, Offload: no,
Asymmetric: no
UDP           14.0.0.10:56565  ->     197.100.1.8:4000   Forward  I           10764
UDP         197.100.1.8:4000   ->   100.100.100.2:5001   Forward  O               0

Service Set: nat44, Session: 134217760, ALG: none, Flags: 0x0000, IP Action: no, Offload: no,
Asymmetric: no
UDP            14.0.0.4:56565  ->     197.100.1.8:4000   Forward  I           11580
UDP         197.100.1.8:4000   ->   100.100.100.1:5001   Forward  O               0

Service Set: nat44, Session: 134217754, ALG: none, Flags: 0x0000, IP Action: no, Offload: no,
Asymmetric: no
UDP            14.0.0.8:56565  ->     197.100.1.8:4000   Forward  I           11060
UDP         197.100.1.8:4000   ->   100.100.100.2:5000   Forward  O               0

Service Set: nat44, Session: 134217733, ALG: none, Flags: 0x0000, IP Action: no, Offload: no,
Asymmetric: no
UDP            14.0.0.5:56565  ->     197.100.1.8:4000   Forward  I           12010
UDP         197.100.1.8:4000   ->   100.100.100.1:5000   Forward  O               0
```

Once the `preserver-parity` and `preserve-range` options are set, the behavior is a bit different. You can see the MX is mapping odd ports to odd ports and even ports to even ports. You should also know that with this specific NAT Pool configuration the MX will drop any traffic from ports 0-1023 that needs to be NAT'd through this pool since you have not set any source ports from this range to NAT/PAT, so you have *no* available ports to map now that you are enforcing `preserve-range`:

```
pool natpat44 {
        address 100.100.100.0/30;
        port {
            range low 5000 high 8000;
            preserve-parity;
            preserve-range;
        }

user@re0# run show services sessions
ms-5/0/0
Service Set: nat44, Session: 201326601, ALG: none, Flags: 0x0000, IP Action: no, Offload: no,
Asymmetric: no
UDP            14.0.0.7:56565  ->     197.100.1.8:4000   Forward  I           14680
UDP         197.100.1.8:4000   ->   100.100.100.2:5021   Forward  O               0

Service Set: nat44, Session: 100663333, ALG: none, Flags: 0x0000, IP Action: no, Offload: no,
Asymmetric: no
UDP            14.0.0.3:56565  ->     197.100.1.8:4000   Forward  I           14452
UDP         197.100.1.8:4000   ->   100.100.100.1:5021   Forward  O               0
Service Set: nat44, Session: 201326596, ALG: none, Flags: 0x0000, IP Action: no, Offload: no,
```

```
Asymmetric: no
UDP            14.0.0.8:50000  ->     197.100.1.8:4000   Forward  I           25782
UDP         197.100.1.8:4000   ->   100.100.100.1:5022   Forward  O               0

Service Set: nat44, Session: 134217765, ALG: none, Flags: 0x0000, IP Action: no, Offload: no,
Asymmetric: no
UDP            14.0.0.9:50000  ->     197.100.1.8:4000   Forward  I           27760
UDP         197.100.1.8:4000   ->   100.100.100.2:5022   Forward  O               0
```

PBA

You are almost done with the pools configuration hierarchy. As a final task, let's take a look at the port block allocation (PBA) feature. This feature is configured under the `secured-port-block-allocation` hierarchy of your NAT pool. Under this section of the `nat pool` hierarchy you can have the MX set blocks of ports that will be assigned to a subscriber's NAT'd IP address. That is the crux of the PBA feature. Assigning a port block to a subscriber IP address is beneficial because it cuts back on the number of syslog messages the MX needs to generate when mapping or freeing a private source address to a NAT'd address. Nothing has been said about syslog messages and their use to historically track which private subscriber was using what NAT'd IP and port at a given time, but to some operators this is a key part of their business practices.

NOTE Logging benefits may mean more to the reader after Chapter 4's syslog section. Then you will really see how PBA can help.

Aside from the benefits of logging, some operators also like PBA with their PAT setup because it limits the number of ports they assign to each private IP address. There's a simple setting called `max-sessions-per-subscriber` that will be discussed in a bit that limits the number of sessions each subscriber can use. So do not enable PBA if your only need is to limit how many sessions a subscriber can use, since we have a more efficient option with the setting `max-sessions-per-subscriber`.

Looking at the PBA settings:

```
[edit services nat]
pool nat-pool-name {
    address prefix;
    address-range low value high value;
    mapping-timeout 300; /* Min 120, Max 86400, default 300 */
    port (automatic | range low minimum-value high maximum-value | random-allocation) {
        secured-port-block-allocation {
             block-size 256; /* Min 64, Max 64512, default 128 */
             max-blocks-per-user 8; /* Min 1, Max 512, default 8 */
             active-block-timeout 300; /* 0(default), Min 120secs, Max 86400 */
    }
}
```

- `block-size` determines the number of ports per block that will be assigned to a subscriber's NAT'd address.
- `max-blocks-per-user` dictates the total potential blocks a given subscriber can be assigned.
- `active-block-timeout` is useful when `max-blocks-per-user` is set to a value of 2 or higher or if `max-blocks-per-user` is not set all. Note that `active-block-timeout` determines how long the current block will be used to allocate sessions. This is yet another setting that can be used to fight port prediction. Since only one active port block at a time will be used by a given subscriber's private IP address, service providers may want a way to kick over to a new port-block so the same NAT'd source ports are not constantly being used.

Let's create a new NAT pool that utilizes the PBA feature for yet another example of how all this works:

```
[edit services nat pool pba_pool]
address-range low 217.200.200.80 high 217.200.200.94;
port {
    automatic;
    secured-port-block-allocation block-size 100 max-blocks-per-address 4;
}
address-allocation;
```

Port blocks are assigned to NAT'd sessions in order, starting from the lowest value and counting up. Within the MX Series, the process of searching for an available port is to first scan the currently active port-block assigned via the NAT pool to the private IP address and assign free ports in sequential order until the end of the port-block. In the event that a free port is not found in the current port-block, and the `max-blocks-per-user` value has not been exceeded, then the next available port-block is made active, and the search continues. So using the setting `secured-port-block-allocation block-size` with a value of 100, the first subscriber to send traffic through the service PIC to get NAT'd will get the first possible block assigned to it.

Using our NAT pool example above called `pba_pool`, that block would contain ports 1024 through 1123. The next subscriber that needs a new block to be assigned to NAT their traffic would get 1124 through 1223, then 1224 through 1323, and 1324 through 1423, and so on. These blocks could all belong to the same private address for the same subscriber if that subscriber needed the resources before anyone else sent traffic through the MX to get NAT'd. Remember, based on the configuration example using the NAT pool called `pba_pool` each subscriber's private IP address can have four blocks assigned to it before the MX rejects any new packets or traffic for the private IP address in question.

NOTE The port / automatic / random-allocation setting has no effect when PBA is configured. Here is an example so you can remember where the `automatic random-allocation` setting is added:

```
[edit services nat pool natpat44 port]
root@JTAC_setup-re0# show
automatic {
    random-allocation;
}
```

One thing about the PBA feature: be very, very, very careful of how large you configure your port blocks. Make sure you have compared the peak number of unique private IP subscribers you expect to send traffic to be NAT'd through the MX at peak hours against the configured port block size compared to the number of IP addresses available in the pool. For example, if it is expected that during peak hours 10,000 subscribers will access the public network concurrently through the MX, and that these 10,000 subscribers will have their traffic NAT'd by the same NAT pool. Now your NAT pool has 10 IP addresses using a port range of 1024-65535. With these facts at hand do not set your `secured-port-block-allocation block-size` to be 100 or you will run out of port blocks and see lots of sessions getting dropped at the service PIC.

The finding would be: port 1024 through 65535 means there is 64511 ports available per NAT'd IP and 10 NAT'd IPs in the NAT pool means you have 645,110 unique ports total to use for PAT. So 10,000 subscribers with a port block size of 100 would require, at a minimum, 1,000,000 ports available for the NAT pool. If you had to go live with this limitation of having just 10 IP Addresses available for use in your NAT pool against potentially having 10,000 private subscribers needing to be NAT'd concurrently during peak hours, set the `secured-port-block-allocation block-size` to be at most 64 and set `max-blocks-per-address` to 1. In addition, think about making sure your `inactive-session-timeouts` and `mapping-timeout` are low. You will most likely need to free pool resources very quickly here and not leave any mappings around longer than absolutely required. If you need more than 64 active NAT sessions for your subscribers or you need mappings to hang around longer for features like address-pooling paired and endpoint independent mappings requirements, you will plain old need more IP addresses available to be NAT'd.

It should be noted that the `show services nat pool detail` command is amazingly useful to discover if you are having port issues with your PBA setup. If the `port block allocation errors` counters continue to climb, you have an issue with the number of active private subscribers requiring to be NAT'd versus the number of port-blocks available in the NAT pool. The `mapping timeouts`, `inactive-session-timeouts`, the

`port block size` and `max-blocks-per-address` settings all need to be looked at:

```
user@re0> show services nat pool detail
Interface: ms-5/0/0, Service set: nat44
  NAT pool: pba_pool, Translation type: NAPT-44
    Address range: 217.200.200.80-217.200.200.94
    Available addresses: 15
    Configured port range: 1024-65535
    Port range: 1024-65535, Ports in use: 9675, Out of port errors: 0
    Parity port errors: 0, Preserve Range errors: 0
    Max ports used: 9675
    AP-P port allocation errors: 0, AP-P port limit allocation errors: 0
    Memory allocation errors: 0
    Max number of port blocks used: 9675, Current number of port blocks in use: 9675, Port
block allocation errors: 4917
    Port blocks limit exceeded errors: 0
    Unique pool users: 9675
  EIF Inbound session count: 0
  EIF Inbound session Limit exceeded drops: 0
```

With the PBA feature you should also be aware that you are assigning what is most likely going to be a lower ceiling in regard to the number of ports a subscriber can use. Without NAT, a client CPE can have a total of 65,535 TCP ports and another 65,535 UDP ports. With Static NAT, or even Dynamic NAT with PAT, but without the PBA feature, a NAT'd client CPE can use 65,535 ports for both UDP and TCP with most configurations. But once you enable PBA you are limiting the overall number of ports a subscriber can use based on the `block-size` and `max-blocks-per-user` settings. So, be aware of your customers' requirements before you enable this feature since you may not want to accidently starve your end users of resources.

NOTE When an operator has to make a change to the NAT pools PBA configuration, or tries to remove the PBA configuration or even add PBA settings to a NAT pool, the sessions on the service PIC will be removed and re-added, which can cause an issue to any long lived session traffic like a SIP call or someone accessing HTTP servers like banking applications. Make these changes during a maintenance window whenever possible.

Now a few final notes on the NAT pool configuration before moving on to the NAT rules configuration. There are a few best practices you should be aware of to make sure you are not wasting valuable routable IPv4 addresses in your NAT pool, and that you are not setting up the service-PIC to handle more traffic than it can correctly manage.

BEST PRACTICES

Think about pool size and what you really need to make it work. When setting a NAT pool up for a dynamic NAT type like napt-44 that uses port address translation, the number of addresses you can set under the pool is limited to a maximum of 65,536 addresses, for a maximum potential of 4,259,775,000 sessions based on your configuration. (The math would be 65,000 addresses multiplied by 65,535 ports.) A dynamic NAT pool with no address port translation can support up to 16,777,216 addresses. There is no limit on the pool size for static source NAT outside of the number of possible IPv4 or IPv6 addresses. But the question you should ask when setting up the pools should not be how large can the pool be but *are you as the operator wasting IP addresses*? For example, though a napt-44 pool can have 65,536 address the MX service PIC cannot come anywhere near handling the number of active NAT'd sessions this would allow. Try to find the balance of having enough NAT'd IPs in your pool to handle your peak traffic needs, potentially with some extra to spare for unexpected growth. Also, monitor the solution checking out the pools and max sessions from time to time to see if you are outgrowing the current setup. As an operator's business grows, additional IP addresses can be added to a NAT pool as needed until the traffic volume through the service PIC hits the peak that the CPU can process. At the same if you set your pools up to be larger than you think you may need them to be at first, so you can make sure the solution works as required, you can always remove IP addresses from the pools as needed. By the end of this book you will be armed with the knowledge of how to look at the solution so you can make the call if your pool is working for your setup or not. Cool!

Do note that the MX Series system will warn you when you try to make the pools too big for the configuration being applied:

```
[edit services nat rule rule_napt-44]
  'term t1'
    Number of addresses with port translation are limited to at most 65536

[edit services nat rule rule_dyn_nat44]
  'term t1'
    Cannot configure more than 16777216 addresses in pool dyn_nat44 with translation type
dynamic-nat44
error: configuration check-out failed
```

BEST PRACTICES

Always remember the MS-MPC card can support up to 30 million sessions per service PIC and the MS-MIC card can support 15 million. Do not waste IP addresses in your NAT pool by making the NAT Pool not only bigger than you what need, but making it bigger than the service PIC could ever use 100%.

NAT Rules

This next section of Chapter 1 jumps into configuring the NAT rules, another major building block for the MX NAT setup. NAT rules will tie into the NAT pools just configured, and, as you will see, NAT rules can determine what traffic actually gets translated, as well as what NAT translation types the MX uses. Consider the following:

```
[edit services nat]
rule rule-name {
    match-direction (input | output
    term term-name {
        from {
            applications [application-names];
            application-sets [set-names];
            destination-address (address | prefix);
            destination-address-range low value high value
            destination-prefix-list prefix-list-name
            source-address (address | prefix);
            source-address-range low value high value
            source-prefix-list prefix-list-name
        }
        then {

            no-translation
            translated {
                destination-pool nat-pool-name;
                source-pool nat-pool-name;
                translation-type {
                    napt-44; }
            }
        }
    }
}
```

Let's create our first example NAT rule we will call `rule1`. NAT rules operate under the same conditions as the MX firewall-filter matching condition `match-direction`, which needs to match the direction the traffic is flowing across the service PIC. Typically, when using something called an *interface-style service-sets*, most NAT setups are configured to NAT sessions from traffic received on the client-side facing interface, so traffic ingressing the MX. In this case service-set you would set `match-direction input` since that is the direction from which the subscriber's traffic is received. If you are setting the interface-style service-set on the interface facing the public network, where the traffic egresses the MX, you would set `match-direction output`. When using something called a *next-hop style-service set* you always set `match-direction input`.

Here is where you set the `match-direction`:

```
[edit services nat]
rule rule1 {
   match-direction input
```

NOTE Service sets are something you will learn about in a few more pages, so don't' worry if you are not sure what they are yet.

Next you set up a term under your NAT rule. In this example you will simply call the term `term1`. Under this term you can optionally set a `from` statement that can make decisions on whether the traffic being received will get NAT'd by this term or not. The logic you can set is based on source-addresses, destination-addresses, destination-port, and applications. Applications are something this book details in Chapter 3, but in the context of how they are used under the term for matching it means you can use the applications setting to match on protocol, source port, and destination port. For our example `term1` let's set some IP prefixes using the stanza `source-address` and a range of IP addresses set using the stanza `source-address-range`. The service PIC will translate traffic from private CPE nodes that arrive from these source networks:

```
[edit services nat rule rule1 term term1]
from {
    source-address {
        10.0.0.0/24;
        11.0.0.0/24;
    }
    source-address-range {
        low 19.0.0.1 high 22.0.0.204;
    }
}
```

The translation that happens will be based off of this next setting, which is the `then` statement. It is used if the criteria in the `from` statement is met, meaning traffic from CPEs that are at the defined networks.

NOTE The `from` statement is optional for most dynamic translation types like napt-44 and dynamic-nat44 but is required for static methods like basic-nat44 and dnat-44.

If the `from` statement is not set, everything is matched and will use the `then` statement's translation logic. If you have multiple terms under your NAT rule you will want to set `then` statements to determine which traffic needs to use the first term for translation, what needs to the second term and so on.

As for this term, let's attach a NAT pool to it so it can actually assign NAT resources and translate traffic and let's use our NAT pool CGN1 created earlier. After the NAT pool is attached to the term you can then

define the NAT translation type you want to use, in this case, napt-44. Remember napt-44 is NAT with PAT translating IPv4 private addresses to IPv4 public addresses:

```
edit services nat rule rule1 term term1]
then {
    translated {
        source-pool CGN-1;
        translation-type {
            napt-44;
        }
    }
}
```

Let's stop with the configuration for a moment and talk about the `translation-type` options. It is with this portion of the `then` section of the NAT rules that things can get tricky as there are quite a number of `translation-type` options to choose from. Once you understand how each `translation-type` functions, it's much simpler to set up the `then` section of the NAT rules. Let's look at some of the translation types:

- `basic-nat-pt`: Nat-pt is static source address (IPv6 to IPv4) and prefix removal for destination address (IPv6 to IPv4) translation.
- `basic-nat44`: Static source address (IPv4 to IPv4) translation.
- `basic-nat66`: Static source address (IPv6 to IPv6) translation. The same as basic-nat44 but for the IPv6 address family.
- `deterministic-napt44`: Deterministic source napt for IPv4.
- `dnat-44`: Static Destination address (IPv4 to IPv4) translation.
- `dynamic-nat44`: Dynamic source address only (IPv4 to IPv4) translation.
- `napt-44`: Source address (IPv4 to IPv4) and port translation.
- `napt-66`: Source address (IPv6 to IPv6) and port translation [same as napt-44 but for IPv6 address family].
- `stateful-nat64`: Dynamic source address (IPv6 to IPv4) and prefix removal for destination address (IPv6 to IPv4) translation.
- `twice-basic-nat-44`: Source static and destination static translation for IPv4 address family.
- `twice-dynamic-nat-44`: Source dynamic and destination static translation for IPv4 address family.
- `twice-napt-44`: Source napt and destination static translation for IPv4 address family.

That is an exhaustive list! Let's go a bit deeper and look under the covers. First thing when setting up a NAT rule is to be aware that the settings in the NAT pool you are using must match the NAT `translation-type` you want to use. Thought must be put into the `[services nat rule]` and `[service nat pool]` sections of the configuration hierarchy or the two pieces may *not* fit together.

Take a look at this example of a commit gone wrong and you'll get the point:

```
root@JTAC_setup-re0# commit
[edit services nat rule nat44]
  'term other'
   With translation-type basic_nat_44, pool must contain equal or more addresses than the
from clause
error: configuration check-out failed
```

View this `[services nat]` configuration that has a NAT pool and a NAT rule to see why that example commit fails:

```
nat {
        pool nat44 {
            address-range low 186.0.0.128 high 186.0.0.129;
            port {
                automatic;
            }
        }
        rule nat44 {
            match-direction input;
            term t1 {
                from {
                    source-address {
                        10.100.30.0/24;
                    }
                }
                then {
                    translated {
                        source-pool nat44;
                        translation-type {
                            basic-nat44;
                        }
                    }
                }
            }
        }
```

This is a good time to focus in on some of these different translation types as you need to understand which type you will need for your particular use case and you need to see how each one may change the needed configuration that you set on the MX for its use. Let's start with the translation types of *basic-nat44* and *basic-nat66*.

Basic-nat44 and basic-nat66 are really just old-fashioned NAT, or what NAT was in the late 1990s to quite a few of us, typically used in the IPv4 and IPv6 world to hide the host machine's actual IP address from the public network. The relationship is a static NAT one-to-one setup, meaning one public NAT'd IP address will be used for one private IP address. Basic-nat44 is used only with IPv4 translation and basic-nat66 is used only for IPv6 translation. It should be stated here that these translation types can be used as inline NAT types. This means they do not require a service card for the NAT translation. Just set up the `si` logical interface and the MPC line card does the work. Remember it has to be a MPC line card to perform the inline NAT functionality.

NOTE Users looking to set up inline NAT ask this type of question often: *"But what if I have a private subnet of /14 but only have a /28 available for the public pool? What do I do?"* The honest answer is you need to move to using a service card in your MX and then move to a dynamic NAT translation-type since inline NAT supports just one-to-one translation mappings.

An example of a NAT rule and NAT pool setup for basic-nat44 translating a /14 network would be as follows:

```
[edit services nat]
pool nat44_basic {
    address 156.100.0.0/14;
}
rule rule_basic_NAT {
    match-direction input;
    term t1 {
        from {
            source-address {
                31.0.0.0/14;
            }
        }
        then {
            translated {
                source-pool nat44_basic;
                translation-type {
                    basic-nat44;
                }
            }
        }
    }
}
```

A good thing to be aware of is that basic-nat44 and basic-nat66 do not translate the source port. Only the IP address is NAT'd and, as stated many times now, it is a static one-to-one mapping. So, let's say you have a CPE on the private network at 31.0.0.5. Using our preceding example pool, this CPE will always have 156.0.05 assigned to it as its NAT'd

address. When using inline NAT the translation is done in the PFE so there is no session table to see these translations being done on each session. But you can set up basic-NAT on the service card that will allow you to use the `show services sessions` command. The next output would be seen when basic-nat44 was set up and used with the service cards (still using the NAT rule and pool example from before). You can see that traffic from 31.0.0.5 is always translated to 150.0.0.5, but the port is not translated since this is a static NAT translation type:

```
user@re0# run show services sessions extensive
ms-1/0/0
Service Set: nat44, Session: 1140850916, ALG: none, Flags: 0x0000, IP Action: no, Offload:
no, Asymmetric: no
NAT PLugin Data:
  NAT Action:   Translation Type - BASIC NAT44
    NAT source             31.0.0.5         ->       150.0.0.5
UDP            31.0.0.5:5003   ->     197.100.1.3:6532   Forward  I               4
  Byte count: 512
  Flow role: Initiator, Timeout: 30
UDP         197.100.1.3:6532   ->       150.0.0.5:5003   Forward  O               0
  Byte count: 0
  Flow role: Responder, Timeout: 30

Service Set: nat44, Session: 1040188013, ALG: none, Flags: 0x0000, IP Action: no, Offload:
no, Asymmetric: no
NAT PLugin Data:
  NAT Action:   Translation Type - BASIC NAT44
    NAT source             31.0.0.5         ->       150.0.0.5
UDP            31.0.0.5:5000   ->     197.100.1.3:6532   Forward  I               4
  Byte count: 512
  Flow role: Initiator, Timeout: 23
UDP         197.100.1.3:6532   ->       150.0.0.5:5000   Forward  O               0
  Byte count: 0
  Flow role: Responder, Timeout: 23

Service Set: nat44, Session: 1140851046, ALG: none, Flags: 0x0000, IP Action: no, Offload:
no, Asymmetric: no
NAT PLugin Data:
  NAT Action:   Translation Type - BASIC NAT44
    NAT source             31.0.0.3         ->       150.0.0.3
UDP            31.0.0.3:5000   ->     197.100.1.3:6532   Forward  I               8
  Byte count: 1024
  Flow role: Initiator, Timeout: 26
UDP         197.100.1.3:6532   ->       150.0.0.3:5000   Forward  O               0
  Byte count: 0
  Flow role: Responder, Timeout: 26
```

After spending so much time looking at the basic-nat44 translation type, it's the perfect time to look at *dynamic-nat44* next. How does the translation type dynamic-nat44 differ from basic-nat44? Dynmaic-nat44 is a one-to-one NAT translation type but it employs the use of the service card to dynamically allocate IP addresses from the NAT

pool. The MX does not have to enforce the fact that your NAT pool may be smaller than the potential range or ranges of private IP addresses that you set in the NAT rule that calls the NAT pool. Using our previous example for basic-nat44, you can change the nat pool from 156.0.0.0/14 to a /30. The following configuration works when using dynamic-nat44 since it is a dynamic type:

```
[edit services]
nat {
    pool nat44_basic {
        address 156.0.0.0/30;
    }
    rule rule1 {
        match-direction input;
        term t1 {
            from {
                source-address {
                    33.0.0.0/14;
                }
            }
            then {
                translated {
                    source-pool nat44;
                    translation-type {
                        dynamic-nat44;
                    }
                }
            }
        }
    }
}
```

WARNING! Be careful with this type of setup. Dynamic-nat44 may allow you to actually configure a smaller NAT pool compared to the private source address ranges, but this does not mean it would work for you in the real world. For example, consider the following: you have a /30 available for your NAT'd IP addresses to service a network that is a /14. What does this mean? You can have two private hosts at a time that have an actual active NAT mapping on the service PIC. That means that if you expect three or more private hosts to try and send data through the MX to be serviced by this NAT pool at the same time, one of those hosts will see all of their packets dropped since you have no free IP addresses to map for their NAT'd IP address. When setting up the dynamic-nat44 translation-type, make sure the NAT pool has enough IP addresses to assign, based on the number of private hosts you feel will need to pass data simultaneously through the MX at peak time. If the number of hosts expected to need NAT servicing simultaneously is 30,000, then you need a /17 for your NAT pool. If you do not have a /17 public range available for your NAT pool you need to look at using one of the translation types that employ Port Address Translation, such as napt-44.

It's a fantastic time to talk about static NAT types versus dynamic NAT types again. Static types like basic-nat44 and basic-nat66 are used in setups where an operator may require every source address listed under the rule's `from` statements to have a readily available NAT IP address in the MX. Once again, static NAT is typically used for security reasons, meaning to hide the private IP address. D-nat44 would be used when you have a range of potential source addresses but there is little to no risk that all of them would ever require a NAT'd address at the same time, or, when you have no choice but to take the risk due to a limited number of public IP addresses. Dynamic types also require a service card, a fact that is admittedly drilled home throughout this book. Dynamic-nat44 can be used to hide the private IP address and it can also be employed by operators who have a limited number of public addresses, so they are fighting IPv4 exhaustion but want a full NAT'd IP address assigned one-to-one to their private IPs versus using a port address translation type which can come with its own complexities.

Now let's look at the translation types *napt-44* and *napt-66*. What is *napt*? Port Address Translation (PAT), NAT Overload Network Address Port Translation (NAPT). Call it what you want, this type of NAT technology translates the source IP and the source port. This means you can have many private IPs using the same public NAT'd IP since multiple sessions can be created using the same public NAT'd IP based on uniquely assigned ports via port address translation. This translation type is very useful for fighting IPv4 exhaustion and is *dynamic* NAT at its essence. These NAT translation types of course do require a service cardcard.

Let's put together a quick configuration for napt-44 that should speak for itself. The following NAT rule will NAT traffic from any source that it receives traffic from, using PAT with the 100.12.1.0/24 range:

```
[edit services]
nat {
    pool CGN-1 {
        address 100.12.1.0/24;
        port {
            automatic {
                 random-allocation;
             }
       }
}
rule CGN-1 {
        match-direction input;
        term other {
            then {
                translated {
                    source-pool CGN-1;
                    translation-type {
                        napt-44;
                    }
```

One other NAT type discussed in this book but not yet shown in any detail, is *deterministic napt44*. Deterministic napt44 is great for when you need to be able to historically map a subscriber's private IP address and port to a NAT'd address and port but do not want to have to use syslogging to look up the historical mappings of a NAT'd public IP back to the private IP for a given period of time. With deterministic-napt44, the MX has an algorithm it uses to assign the NAT'd IP address and blocks of ports in the NAT pool to the source IP ranges set under the NAT rules. This makes this NAT translation type very desirable to some operators since it allows the operator to tell who used what NAT'd IP address and port without needing a syslog server and a storage device for the logs themselves, since a private subscriber will always use the same NAT'd address and port block.

Configuring deterministic NAT is quite simple. Like the PBA settings looked at earlier under the NAT pool section, deterministic-napt44 is a port block type, so under your NAT pool you define the pool as being a deterministic port block allocation type and then you set the `block-size`. Note if you do not define a `block-size`, 512 is the default. Under the NAT rule make sure you add a `source-address` range. With determinstic-napt44 you *must* define the private range or ranges this block will be mapped to:

```
nat {
    pool nat44 {
        address 156.100.100.0/23;
        port {
            automatic;
            deterministic-port-block-allocation block-size 512;
        }
    }
    rule rule1 {
        match-direction input;
        term t1 {
            from {
                source-address {
                    33.33.0.0/20;
                }
            }
            then {
                translated {
                    source-pool nat44;
                    translation-type {
                        deterministic-napt44;
                    }
                }
            }
        }
    }
}
```

So, using our current example, traffic would be seen from the 33.33.0.0/20 private range mapped to an IP address from 156.100.100.0/23 with a port block of 512 ports.

Time to run a `show` command. In this case the `show services nat deterministic-nat nat-port-block` command and look at a couple of subscribers' NAT'd IP addresses. This way you can see that there is a NAT'd address assigned and a block of 512 source ports that the MX will assign to sessions for this subscriber:

```
user@re0# run show services nat deterministic-nat nat-port-block 33.33.1.1
Interface: ms-1/0/0
NAT pool: detnat
Internal Host: 33.33.1.1, NAT IP Address: 156.100.100.3, NAT Port Start: 3072, NAT Port End:
3583

{master}[edit services nat rule detnat]
user@re0# run show services nat deterministic-nat nat-port-block 33.33.1.2
Interface: ms-1/0/0
NAT pool: detnat
Internal Host: 33.33.1.2, NAT IP Address: 156.100.100.3, NAT Port Start: 3584, NAT Port End:
4095
```

If you look closely at the two subscribers' addresses in the output, you will see that incremental private IP addresses and port blocks are NAT'd incrementally, one after the other:

You can also run the `show services nat pool detail` command for a view of the pool in general:

```
user@re0# run show services nat pool detail
Interface: ms-1/0/0, Service set: detnat
  NAT pool: detnat, Translation type: DETERMINISTIC NAPT44
    Address range: 156.100.100.1-156.100.101.254
    Available addresses: 510
    Configured port range: 1024-65535
    Port range: 1024-65535, Ports in use: 0, Out of port errors: 0, Max ports used: 0
    AP-P port allocation errors: 0
    Memory allocation errors: 0
    Max number of port blocks used: 0, Current number of port blocks in use: 0, Port block
allocation errors: 0
    DetNAT subscriber exceeded port limits: 0
    Unique pool users: 0
```

And, you can also change the port block size that is assigned by the deterministic port block. As noted before, 512 is the default block size but you can assign anything from 1 to 65535 ports. It should be noted, however, that not every port in the range can, or will, be used since deterministic NAT will never use privileged ports that are ports 0 through 1023. If you look back at the output from the `show services nat pool detail` command you will see `Configured port range: 1024-65535`.

Let's go back under the `[services nat pool nat44 port]` configuration hierarchy so we can change the block-size to 10:

```
nat {
    pool nat44 {
        address 156.0.0.0/14;
        port {
            automatic;
            deterministic-port-block-allocation block-size 10;
        }
    }
```

As with the PBA feature, when making changes to the deterministic port block size you should restart the service PIC for it to take effect.

You can run the `show services nat deterministic-nat nat-port-block` command again to see that the port block now only contains ten ports:

```
user@re0# run show services nat deterministic-nat nat-port-block 33.33.1.2
Interface: ms-1/0/0
NAT pool: detnat
Internal Host: 33.33.1.2, NAT IP Address: 156.100.100.1, NAT Port Start: 3594, NAT Port End:
3603
```

Also, when running the `show service nat pool` command you can see the block size for your determinsitic-napt44 pool. This is a good example of when the detailed version of a command does not show you a counter that the standard run of a command does:

```
user@re0# run show services nat pool
Interface: ms-1/0/0, Service set: detnat
NAT pool          Type    Address                          Port        Ports used
detnat            DETERMINISTIC NAPT44 156.100.100.1-156.100.101.254 1024-65535 0
Port block type: Deterministic port block, Port block size: 10
```

So why doesn't everyone just use deterministic NAT for their translation type so they have a way to always map the traffic sent by private clients to the NAT'd traffic seen on the public network? The answer is that you may have too large of a source range of private IP addresses that need to be serviced by the NAT pool. Unlike napt44, deterministic napt44 has boundaries around the number of potential source addresses versus the NAT pool address range and port ranges. Simply stated, if the pool and port ranges do not cover the whole of the private source range your commit will fail. This boundary, which is necessary for the logic of deterministic NAT to work, can cause some scenarios to require napt44 as the translation type instead.

Here is what would happen if you tried to commit a configuration that had deterministic NAT configured with a NAT pool that cannot cover the potential range of private IP addresses even when setting an unrealistically small block size of 5 ports for each subscriber:

```
user@re0# commit
[edit services nat rule rule1]
  'term t1'
    Number of addresses and port blocks combination in the NAT pool is less than number of
addresses in 'from' clause
error: configuration check-out failed

[edit services nat pool nat44]
address-range low 156.100.5.1 high 156.100.5.10;
port {
    range low 1025 high 65535;
    deterministic-port-block-allocation block-size 5;
}
[edit services nat rule rule1 term t1]
user@re0# show
from {
    source-address {
        192.168.0.0/13;
    }
}
then {
    translated {
        source-pool nat44;
        translation-type {
            deterministic-napt44;
        }
    }
}
```

To make this configuration work when you are only trying to assign five ports to each private IP address, you have to look at the fact you have a /13 range of private IP addresses, or 524,286 private hosts total. There are ten IP addresses available in the NAT pool and you have 64511 ports per IP. That is a total of 645,110 ports divided by the five ports you are using for the `deterministic-port-block-allocation block-size`. That would mean this setup could handle a boundary of at most 129,022 private source addresses, which is far below the 524,286 you need to handle for this configuration, where each private subscriber would only have five ports. For five total sessions to work you would need to increase the number of IP addresses available in your NAT pool to at least 41 addresses. And even if you had 41 public addresses you could assign to the NAT pool, you still have a block size of five ports which is not going to fit many real world use cases. So in reality you would need to find even more available public IP addresses and you would want to increase the block size considerably. Make sense? Deterministic-NAT like any NAT setup needs to be thought through the design to make sure you do not starve your end users from being able to have their traffic NAT'd.

Okay, the next NAT translation type up is *dnat-44*, otherwise known as *destination NAT*. Destination NAT is when traffic originated from the public network is sent towards a NAT'd address and this is then translated to the private address. A big use case for destination NAT is when nodes on the public side need to reach certain defined targets or networks on the private side but the public nodes are using a virtual IP, or a NAT'd IP, to reach the targets. This keeps the private IP addresses hidden from the public world. The MX in this case, using destination NAT, will translate the destination address to the correct internal private address. Dnat-44 is a static NAT type that does not require a service card.

Let's show an example destination NAT setup where traffic received by the MX Series, destined to 120.100.0.100 through 120.100.0.200, using destination port 8888, would then be de-NAT'd to 140.100.0.100 through 140.100.0.200. So, if traffic was sent to 120.100.0.120 on port 8888 it would be translated to 140.100.0.120:

```
[edit services nat]

pool internal-servers {
    address-range low 140.100.0.100 high 140.100.0.200;
}

rule dnat_rule {
    match-direction input
    term t1 {
        from {
            destination-port {
                range low 8888 high 8888;
            }
            destination-address-range {
                low 120.100.0.100 high 120.100.0.200;
            }
        }
        then {
            translated {
                destination-pool internal-servers;
                translation-type {
                    dnat-44;
                }
            }
        }
    }
}
```

NOTE If you only need to use the destination NAT rule with traffic destined to TCP port 8888, versus both TCP and UDP as the above logic would use, you would use the `applications` setting under the from statement versus using the `destination-port`. (The applications settings are discussed in detail in Chapter 3.)

You can also use the `port-forwarding` feature when using the dnat-44 translation type, allowing you to also hide your actual destination port from the public network. Set your port mappings under the services / nat hierarchy. In this example a map called *map1* is set with two port translations:

```
[edit services nat]
port-forwarding map1 {
   destined-port 8888 translated-port 2000;
}
```

Then add this port forward mapping to the dnat-44 rule you are using. Now, any traffic destined to the addresses 120.100.0.100 through 120.100.0.200 and to destination port 8888 will also have the destination port translated, in this case to 2000. If you do not use the port forwarding feature the MX will NAT the destination IP address but forward the original destination port sent by the source. Look below to see where `port-forward-mapping` is added to the NAT rule:

```
[edit services nat]
rule dnat_rule {
    match-direction input
    term t1 {
        from {
            destination-port {
                range low 8888 high 8888;
            }
            destination-address-range {
                low 120.100.0.100 high 120.100.0.200;
            }
        }
        then {
            port-forwarding-mappings map1;
            translated {
                destination-pool natpat44;
                translation-type {
                    dnat-44;
                }
            }
        }
    }
}
```

It is now time to talk about the translation type *twice NAT,* which is the practice of NATing both the source and the destination address. This translation type is used when you need to NAT the source traffic but at the same time the source NAT'd traffic is destined to a target that requires destination NAT to occur at the MX Series. Junos allows you to set up static mapping with `twice-basic-nat-44`, dynamic mapping with `twice-dynamic-nat-44` and dynamic PAT mappings with `twice-napt-44`.

However, do note that `twice-basic-nat-44` can be used as an inline NAT type, whereas `twice-dynamic-nat-44` and `twice-napt-44` require a service card.

Here is an example using `twice-napt-44`. Like any dynamic NAT type it is only optional to configure the source address field, but for the destination you must define the destination address or addresses with the `destination-address` option and the destination pool must have an equal number of NAT'd IP addresses to map to the addresses added to the `destination-address` option:

```
[edit services nat]

pool dest_pool {
    address 77.76.75.73/32;
}
pool src_pool {
    address 100.100.0.0/25;
    port {
        automatic;
    }
}

rule twice-nat {
            match-direction input;
            term my-term1 {
                from {
                    destination-address {
                        75.76.75.73/32;
                    }
                }
              then {
                  translated {
                      source-pool src_pool;
                      destination-pool dst_pool;
                      translation-type {
                          twice-napt-44;
                      }
                  }
              }
           }
}
```

Stateful nat64 is the last translation type option to explore. Stateful nat64 is an IPv6 transition method, translating incoming IPv6 packets from the private network into IPv4 and then performing the opposite task when IPv4 traffic is destined to the IPv6 private address. When stateful nat64 is used in conjunction with a DNS64 server, no changes are usually required on the IPv6 client side or the IPv4 target when the CPE tries to access resources via name.

NOTE DNS64 is just a DNS server that can take requests for AAAA records (IPv6) from the A records (IPv4). The first part of the synthesized IPv6 address points to an IPv6/IPv4 translator (the MX Series) and the second part embeds the IPv4 address from the A record:

```
rule rule-NAT64 {
    match-direction input;
    term term1 {
        from {
            destination-address {
                64:ff9b::/96;
            }
        }
        then {
            translated {
                source-pool client_nat;
                destination-prefix 64:ff9b::/96;
                translation-type {
                    stateful-nat64;
                }
            }
```

In this example you can see for the `destination-address` and `destination-prefix 64:ff9b::/96` was used. The well-known prefix *64:ff9b::/96* is used in an algorithmic mapping between IPv6 to IPv4 addresses. It is defined out of the 0000::/8 address block, per RFC6052.

NOTE Starting with Junos 12.1, only IPv6 /96 prefixes for this translation pool are supported. So you can use the well-known prefix (64:FF9B::/96) or any arbitrary /96 prefixes. And really, this is based on how the DNS64 server is set up.

If name resolution is not used, the client software, or the end user, will manually have to enter the IPv6 address to be translated. So in our example, to reach IPv4 node 10.12.1.10 you would have to use this IP on the client side as the destination - 64:ff9b::10.12.1.10:

```
user@re0> show services nat pool
Interface: ms-1/0/0, Service set: sset-nat64
NAT pool          Type    Address                          Port        Ports used
client_nat        dynamic 50.1.1.1-50.1.1.254              512-65535   1
Port block type: Unknown port block, Port block size: 0
NAT pool          Type    Address                          Port        Ports used
_jpool_rule-NAT64_term1_ static 64:ff9b::-64:ff9a:ffff:ffff:ffff:ffff:ffff:ffff
Port block type: Unknown port block, Port block size: 0


user@re0> show services sessions
Interface: ms-1/0/0, Service set: sset-nat64
Flow                                              State    Dir       Frm count
```

```
ICMP        10.12.1.10       ->       50.1.1.1       Watch    0               0
    NAT source      10.12.1.10         -> 64:ff9b::a0c:10a
    NAT dest          50.1.1.1         ->         2001::2
ICMPV6         2001::2       ->64:ff9b::a0c:10a      Watch    I              20
    NAT source         2001::2         ->        50.1.1.1
    NAT dest   64:ff9b::a0c:10a        ->      10.12.1.10
```

NOTE One thing to note is that the MX Series does not support using the port block allocation feature with nat64.

Service Sets

By now you should understand the NAT pools and NAT rules configuration and how to set them up. Now it's time to look at service sets, the place where the NAT pool and NAT rule configurations are tied together to the point that you can apply your NAT configuration on the MX Series and actually do address translation to the traffic flowing through it. Without the service set nothing will work when it comes to NAT.

Once again, like everything in the amazingly configurable MX Series, there are options here, and plenty of them. The MX really does offer maximum flexibility. As you start to look at the service set settings understand that there are two different methods, or styles, you can use to apply your service sets based on your needs and preferences: namely *interface-style* and *next-hop style*. In a nutshell, what they really do is determine what data packets will be steered towards your service card's service PIC for CGNAT processing.

Let's look at the difference between these two service set methods and some sample configurations, so you can determine what method you need.

Interface Style

Interface style service sets are generally faster to configure and deploy than next hop style. They are directly applied to the media interfaces and appear as a "bump-in-the-wire" between the media interface and the PFE.

All traffic entering the interface and exiting the interface may traverse the service PIC due to the `interface-style service-set` applied to the interface, as shown in Figure 1.3.

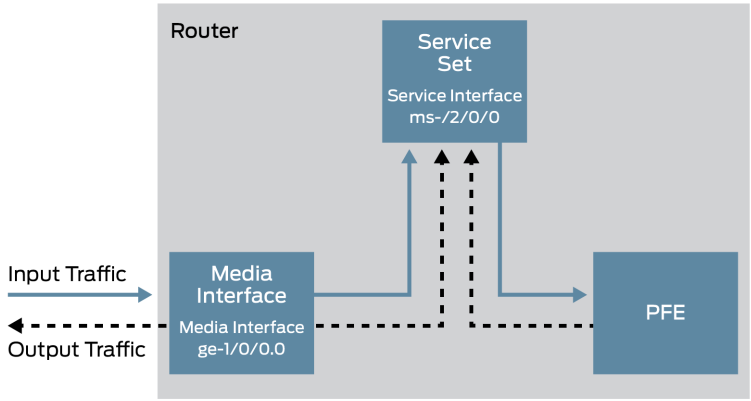

Figure 1.3 Illustration Showing Interface Style Service Set

Next Hop Style

With next hop style service sets you use the routing table to steer traffic destined to specific prefixes to the service PIC. So only traffic that is destined for a specific next hop is serviced by the next hop style service set, as shown in Figure 1.4. In general, the next hop style can provide more flexibility than interface style.

NOTE Only one next hop style service set per MS interface sub unit is allowed.

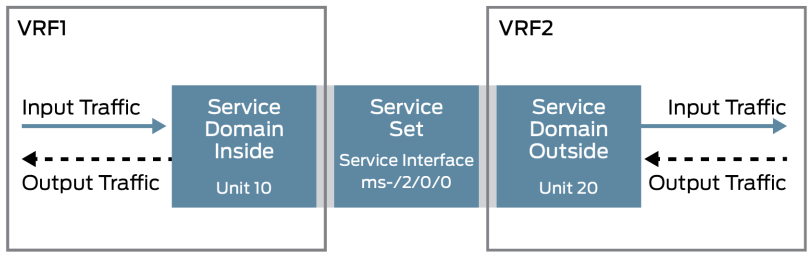

Figure 1.4 Illustration Showing Next Hop Style Service Set

Now let's look at the interface style service-set configuration and get in there and compare it to the next hop style. For a typical source NAT setup using interface style you actually edit the actual physical interfaces that handle the traffic sourced from the private client network or the physical interfaces that this traffic egresses to get to the public network. It is under these interface(s) you will connect to the service-set.

Within the [services] hierarchy you simply add a service-set. For a basic configuration this service-set will call one of our NAT rules, then you will set the interface-service configuration setting to use a service interface you have on your box. This interface-service setting simply points to your si, ms, or ams interfaces that you want to use based on whether you are using inline static NAT, dynamic NAT, or dynamic NAT with load balancing or redundant PICs. Let's start with this example using the ms interface of ms-2/0/0 which you now know means you have a service card in FPC slot 2:

```
[edit services]
service-set nat44 {
    nat-rules rule2;
    interface-service {
        service-interface ms-2/0/0;
    }
}
```

Then, under the physical interfaces unit, you add the options service / input and service / output to the family inet/inet6 that is handling the data traffic and you tie your service-set called nat44 to here. With this configuration this simply means that *all* traffic processed by this interface will be passed to the service-interface ms-2/0/0 attached to the service-set nat44 for NAT servicing based on the NAT configuration set under the NAT rule rule2:

```
[edit interfaces xe-3/3/0]
description "direct to subscriber CPE";
unit 0 {
    family inet {
        service {
            input {
                service-set nat44;
            }
            output {
                service-set nat44;
            }
        }
        address 139.97.68.42/30;
    }
```

As you can see this is a really simple configuration, but the simplicity also means that every packet this physical interface receives gets serviced by the service PIC, even if the NAT rule is configured to only NAT packets received based on source IP, destination IP, application, etc. So, based on the NAT rules you have attached to the service-set, even if a packet gets steered to the service PIC it may not get NAT'd and then just gets passed right back to the interface to proceed along to its destination target based on the MX's routing table. Chapter 4 discusses the service-filters feature and will show you how using these filters

can determine what traffic you do not want to have steered to the service PIC. This feature will be useful for saving CPU processing on the service PIC by not sending traffic to the service PIC that does not need to be NAT'd.

It can be argued that the next hop style can be slightly more complicated to set up but it can offer some control over what packets gets processed by the service PIC since you route traffic to the service PIC to have it serviced. Let's create the next hop style service set and go from there. As you will see the service set is configured very similarly to the interface style– just remove the `interface-service` parameter and add the `next-hop-service` option. Under the `next-hop-service` option there is an `inside-service-interface` and an `outside-service-interface` option, each pointing to a different logical unit set up under the `si`, `ms`, or `ams` interface that you want to steer traffic to for NAT processing:

```
[edit services]
service-set nat44 {
    }
    nat-rules rule2;
    next-hop-service {
        inside-service-interface ms-1/0/0.1;
        outside-service-interface ms-1/0/0.2;
    }
```

Then, under the service interface hierarchy, you create your two logical units – the inside interface is the one that handles ingress traffic from the private network that needs to get NAT'd, which then egresses post-NAT to the public network through the outside interface. Traffic from the public network destined to the NAT'd addresses ingresses through the outside interface, gets de-NAT'd and then egresses towards the private network through the inside interface:

```
[edit interfaces ms-2/0/0]
unit 1 {
    family inet;
    service-domain inside;
}
unit 2 {
    family inet;
    service-domain outside;
}
```

Now you can add static routes that are used to steer the desired destination traffic to the inside service interface for CGNAT processing:

```
[edit]
routing-options {
    static {
        route 197.100.1.0/24 next-hop ms-2/0/0.1;
        route 189.1.1.0/24 next-hop ms-2/0/0.1;
    }
}
```

Let's look at a simple example showing the inside service interface and outside interface sitting in different VRs. Both the physical interface that is used to reach the private network, and the service interface ms-2/0/0.1 that has been configured to handle the inside service-domain, is tied into our first virtual router:

```
[edit routing-instances VR1]
instance-type virtual-router;
interface xe-3/3/0.0;
interface ms-2/0/0.1;
routing-options {
    static {
        route 197.100.1.0/24 next-hop ms-2/0/0.1;
        route 189.1.1.0/24 next-hop ms-2/0/0.1;
    }
}
```

Then you set up the virtual router that handles the traffic destined to the public network. This is done by adding the interface that faces the public network, along with adding the service interface unit that you tied to the outside service domain, that interface being ms-2/0/0.2:

```
[edit routing-instances VR2]
root@JTAC_setup-re0# show
instance-type virtual-router;
interface xe-3/0/2.0;
interface ms-2/0/0.2;
```

At this point, after you have your service set configured and committed, if you look at the routes on the box you should see the desired traffic destinations to be NAT'd will get sent to the inside service interface:

```
VR1.inet.0: 18 destinations, 19 routes (17 active, 0 holddown, 1 hidden)
+ = Active Route, - = Last Active, * = Both
197.100.1.0/24     *[Static51] 04:03:20
                       > via ms-2/0/0.1
189.1.1.0/24     *[Static51] 04:03:20
                     > via ms-2/0/0.1
```

The NAT setup automatically injects dynamic routes into the routing table to steer traffic destined to the NAT'd address towards the service set for interface style setups and towards the outside service interface for next hop style setups.

Using this example, let's say that the NAT pool being used in the next hop style setup is employing the NAT prefix 100.100.100.0/25. Then the return route for the client traffic is injected into the VR2 routing instance where the outside service interface exists allowing the return traffic to the private clients to get steered back to the service PIC to get de-NAT'd:

```
user@re0# run show route 100.100.100.0

VR2.inet.0: 68 destinations, 68 routes (68 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

100.100.100.0/25     *[Static/1] 00:15:14
                    > via ms-1/0/0.2
```

With solutions like dnat and twice NAT where destination NAT is used, you can see a dynamically injected route added to the routing instance that hosts the inside interface for a next hop style setup to have any traffic sent to the destination address get steered into the inside service interface to get de-NAT'd:

```
user@re0# run show route 77.76.75.73

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

77.76.75.73/32     *[Static/1] 00:14:53
                    > via ms-1/0/0.1
```

Now, let's look quickly at a couple of additional features you can set under the `service-set` configuration.

First, if you are seeing lots of simultaneous drop flows when viewing the `show services sessions output` command, the drops being due to port scans and the like, you can set `max-flows-drop` under the `service-set` to limit the number of drop flows held in memory. You can set this for both egress and/or ingress directions, as shown here:

```
[edit services]
service-set nat44 {
    max-drop-flows ingress 100 egress 500;
    nat-rules rule2;
    next-hop-service {
             inside-service-interface ms-2/0/0.1;
             outside-service-interface ms-2/0/0.2;
    }
```

Next, when using the napt-44 translation type, if the PBA feature is not used, there is nothing to stop any one internal private IP address from taking up a whole boatload of ports unless you set the `max-sessions-per-subscriber` option. This feature is great for when a handful of

subscribers starve the rest from being able to get any ports to NAT. It does this by limiting the number of sessions any one private IP address can use at a given moment in time. You can set the value for `max-sessions-per-subscriber` from 1 up until 32000. Note that a session is both an ingress and egress flow through the session table that maintains the NAT'd session:

```
[edit services]
service-set nat44 {
    max-drop-flows ingress 100 egress 500;
    nat-rules rule2;
    nat-options {
          max-sessions-per-subscriber 10000;
     }
    next-hop-service {
              inside-service-interface ms-2/0/0.1;
              outside-service-interface ms-2/0/0.2;
    }
```

You can also limit the maximum number of sessions a service set will allow at any given time by employing the `max-flows` configuration option. It's very useful if you have multiple service sets being used by the same service PIC and you want to make sure one service set does not use up all of the memory on the service PIC. Here's an example:

```
[edit services]
service-set nat44 {
   max-flows 1000000;
   max-drop-flows ingress 10 egress 5;
    nat-rules rule2;
    nat-options {
          max-sessions-per-subscriber 10000;
     }
    next-hop-service {
              inside-service-interface ms-2/0/0.1;
              outside-service-interface ms-2/0/0.2;
    }
```

Summary

At this point in the book, you have gone over the basics of the CGNAT setup on the MX Series. Armed with the configuration knowledge you now have, you can get a good NAT setup going that will fit many scenarios. But you are not done yet, not even close. Chapter 2 covers advanced options you can apply that may fit your individual needs.

Let's go over what you have done so far, and if need be, you should re-read any section before moving on:

- Configured a service interface service, either `si` for inline, or a `ms` or `ams` for use with a service card.
- Configured a services NAT pool to assign translated IP addresses, and optionally, ports to the sessions you want to NAT.
- Configured a service NAT rule where you assigned the NAT translation type and optionally defined the specific private IP address ranges that should be NAT'd. From here you also tied in the services NAT pool to use for the sessions that hit this NAT rule.
- Configured the services service set which ties in the service NAT rule and the service interface that the rule will use to process the NAT logic.
- Tied the services service set to your physical interface using the interface-style setup, or, or you tied the services service set to your service interface, which then gets used based on static routes when using a next hop style setup.

Chapter 2

Additional Features

Wow, after all that was covered in Chapter One, there's more? Yes, yes, there are still more features and options that can be applied to your setup. Remember the MX Series can fit all sorts of different NATing environments because of its very deep and rich feature set. Let's get started.

Address Pooling Paired

Address Pooling Paired (APP) is a feature that allows the MX Series to assign the same NAT'd external address for all sessions originating from the same internal private host. The mapping is triggered when the first packet is received from the internal private host and this mapping will stay for a period of time as explained below.

Why use this feature? Because it solves problems that can occur when client-to-server communication opens multiple connections. Some examples being:

- There are many HTTP based applications, like banking software, that require the client to open multiple sources to destination sessions and each one of these sessions needs to be seen from the same IP address.
- Any P2P protocol assuming address stability benefits from address pooling paired being used, so the NAT'd IP address is the same for all sessions from a private host.

Remember, if you are a private host sending dozens of different UDP and TCP sessions through the MX and you are using a dynamic NAT translation type without this feature enabled, you may be seen on the public facing side by many different NAT'd IP addresses. Based on the application needs of your end user, this may not be a good thing.

NOTE It should be noted that the `address-pooling paired` feature is only needed, and only works with, NAT translation types that use service cards.

Enable the feature under the [`services nat rule`], as shown here:

```
[services nat]
rule rule1 {
    match-direction input;
    term t1 {
        from {
            source-address {
                33.0.0.0/14;
            }
        }
        then {
            translated {
                source-pool natpat44;
                translation-type {
                    napt-44;
                }
                address-pooling paired;
            }
        }
    }
}
```

When configuring the `address-pooling paired` feature it is very strongly recommended that you configure your NAT pool to use the `address-allocation` feature set as `round-robin`. If not, the MX Series will allocate ports from the NAT pool in the default mode, which, based on the traffic profile through the MX, could, and most likely will, result in many internal private IPs being mapped to the same, single, public-facing NAT'd IP. This will eventually cause source port exhaustion of this NAT'd IP address, when there are still other available IPs in the NAT pool that still do not have any ports tied up.

Port exhaustion for a single NAT'd IP address in a pool with many available NAT'd IP addresses when address pooling paired is being used is bad, because the MX will drop traffic for the private IP addresses mapped to the NAT'd IP address in question once the ports are exhausted. When address pooling paired is not used, port exhaustion for a single NAT'd IP address in a pool of many available NAT'd IP addresses is not a big deal since any additional subscriber sessions that

come in would just start to use the next available IP address and any related ports in the NAT pool that are available. But to drive this point home, if the `address pooling paired` feature maps the private IP to the same public IP for all its sessions, the MX will not move onto the next available IP. Instead, it will drop any new sessions until the private IP address it is mapped to has a free port, which could take some time, based on the session timeout settings you may have configured and on the length of activity for the actual traffic from the client to its target and back.

Let's look at this process step-by-step.

Remember `address-allocation` with `round-robin` chooses the first available IP address from the NAT pool for a unique private IP address that sends traffic through the MX to get NAT'd. With the `address-pooling paired` feature and `address-allocation` with `round-robin`, the next unique private IP address that sends traffic will be mapped to the next IP address in the NAT pool. Based on the number of private IP addresses and the size of the NAT Pool, multiple private IP addresses can, and may, end up being mapped to the same public NAT IP address.

The example shown here uses `napt-44` as the translation type, so let's set the pool size to a /25 to show how the resources are assigned:

```
[edit services nat]
pool natpat44 {
    address 100.100.0.0/25;
    port {
        automatic {
            random-allocation;
        }
    }
    address-allocation round-robin;
    mapping-timeout 120;
}
```

To start, look at the effects of `address-allocation with round-robin` without the `address-pooling paired` feature being enabled. In the `run show services sessions extensive` output below it can be seen where each session will have a different NAT translated IP and port assigned to it, even though the traffic all originates from the same private source address. In this example the private client at 31.0.0.2 is sending TCP and UDP traffic to a public server at 197.100.1.2 destination port 6532. For each unique session, you can see that a different NAT'd IP address is used since the NAT'd IP addresses are assigned sequentially due to the configuration used:

```
user@re0# run show services sessions extensive
ms-1/0/0
```

```
Service Set: nat44, Session: 1275068568, ALG: none, Flags: 0x0000, IP Action: no, Offload:
no, Asymmetric: no
NAT PLugin Data:
  NAT Action:   Translation Type - NAPT-44
    NAT source             31.0.0.2:17488   ->   100.100.0.1:34059
UDP            31.0.0.2:17488  ->     197.100.1.2:6532   Forward  I               1
  Byte count: 128
  Flow role: Initiator, Timeout: 30
UDP         197.100.1.2:6532   ->   100.100.0.1:34059  Forward  O               0
  Byte count: 0
  Flow role: Responder, Timeout: 30

Service Set: nat44, Session: 1241514401, ALG: none, Flags: 0x0000, IP Action: no, Offload:
no, Asymmetric: no
NAT PLugin Data:
  NAT Action:   Translation Type - NAPT-44
    NAT source             31.0.0.2:27989   ->   100.100.0.2:6122
TCP            31.0.0.2:27989  ->     197.100.1.2:6532   Forward  I               1
  Byte count: 128
  Flow role: Initiator, Timeout: 30
TCP         197.100.1.2:6532   ->   100.100.0.2:6122   Forward  O               0
  Byte count: 0
  Flow role: Responder, Timeout: 30

Service Set: nat44, Session: 1308623554, ALG: none, Flags: 0x0000, IP Action: no, Offload:
no, Asymmetric: no
NAT PLugin Data:
  NAT Action:   Translation Type - NAPT-44
    NAT source             31.0.0.2:30776   ->   100.100.0.3:31043
UDP            31.0.0.2:30776  ->     197.100.1.2:6532   Forward  I               1
  Byte count: 128
  Flow role: Initiator, Timeout: 30
UDP         197.100.1.2:6532   ->   100.100.0.3:31043  Forward  O               0
  Byte count: 0
  Flow role: Responder, Timeout: 30

Service Set: nat44, Session: 1275069075, ALG: none, Flags: 0x0000, IP Action: no, Offload:
no, Asymmetric: no
NAT PLugin Data:
  NAT Action:   Translation Type - NAPT-44
    NAT source             31.0.0.2:11729   ->   100.100.0.4:15314
UDP            31.0.0.2:11729  ->     197.100.1.2:6532   Forward  I               1
  Byte count: 128
  Flow role: Initiator, Timeout: 30
UDP         197.100.1.2:6532   ->   100.100.0.4:15314  Forward  O               0
  Byte count: 0
  Flow role: Responder, Timeout: 30

Service Set: nat44, Session: 1275068755, ALG: none, Flags: 0x0000, IP Action: no, Offload:
no, Asymmetric: no
NAT PLugin Data:
  NAT Action:   Translation Type - NAPT-44
    NAT source             31.0.0.2:1093    ->   100.100.0.5:29645
UDP            31.0.0.2:1093   ->     197.100.1.2:6532   Forward  I               1
```

```
  Byte count: 128
  Flow role: Initiator, Timeout: 30
UDP         197.100.1.2:6532   ->   100.100.0.5:29645  Forward  0              0
  Byte count: 0
  Flow role: Responder, Timeout: 30
```

Now, let's add the `address-pooling paired` feature, which, as stated just a moment earlier, is set under the [`service nat rule`] section:

```
pool natpat44 {
    address 100.100.0.0/25;
    port {
        automatic {
            random-allocation;
        }
    }
    address-allocation round-robin;
}
rule rule1 {
    match-direction input;
    term t1 {
        from {
            source-address {
                33.0.0.0/14;
            }
        }
        then {
            translated {
                source-pool natpat44;
                translation-type {
                    napt-44;
                }
                address-pooling paired;
            }
        }
    }
}
```

With this feature you can see that the same NAT'd IP address is being mapped to all of the sessions originated from the private IP address 31.0.0.2. It is now a dynamic one-to-one mapping, unlike before, when `address-pooling paired` was not used and any free NAT'd IP could be mapped to any session:

```
user@re0# run show services sessions extensive
ms-1/0/0
Service Set: nat44, Session: 1577058893, ALG: none, Flags: 0x0000, IP Action: no, Offload:
no, Asymmetric: no
NAT PLugin Data:
  NAT Action:   Translation Type - NAPT-44
    NAT source             31.0.0.2:30332   ->     100.100.0.1:43590
UDP           31.0.0.2:30332  ->     197.100.1.2:6532   Forward  I              1
  Byte count: 128
  Flow role: Initiator, Timeout: 30
UDP         197.100.1.2:6532   ->     100.100.0.1:43590  Forward  O              0
```

```
  Byte count: 0
  Flow role: Responder, Timeout: 30

Service Set: nat44, Session: 1543504245, ALG: none, Flags: 0x0000, IP Action: no, Offload:
no, Asymmetric: no
NAT PLugin Data:
  NAT Action:   Translation Type - NAPT-44
    NAT source             31.0.0.2:7288    ->     100.100.0.1:50119
UDP            31.0.0.2:7288   ->     197.100.1.2:6532   Forward  I               1
  Byte count: 128
  Flow role: Initiator, Timeout: 30
UDP         197.100.1.2:6532   ->     100.100.0.1:50119  Forward  O               0
  Byte count: 0
  Flow role: Responder, Timeout: 30

Service Set: nat44, Session: 1577058858, ALG: none, Flags: 0x0000, IP Action: no, Offload:
no, Asymmetric: no
NAT PLugin Data:
  NAT Action:   Translation Type - NAPT-44
    NAT source             31.0.0.2:63941   ->     100.100.0.1:35276
UDP            31.0.0.2:63941  ->     197.100.1.2:6532   Forward  I               1
  Byte count: 128
  Flow role: Initiator, Timeout: 30
UDP         197.100.1.2:6532   ->     100.100.0.1:35276  Forward  O               0
  Byte count: 0
  Flow role: Responder, Timeout: 30

Service Set: nat44, Session: 1375732136, ALG: none, Flags: 0x0000, IP Action: no, Offload:
no, Asymmetric: no
NAT PLugin Data:
  NAT Action:   Translation Type - NAPT-44
    NAT source             31.0.0.2:29816   ->     100.100.0.1:25017
UDP            31.0.0.2:29816  ->     197.100.1.2:6532   Forward  I               1
  Byte count: 128
  Flow role: Initiator, Timeout: 30
UDP         197.100.1.2:6532   ->     100.100.0.1:25017  Forward  O               0
  Byte count: 0
  Flow role: Responder, Timeout: 30
```

Figure 2.1 illustrates this show command example.

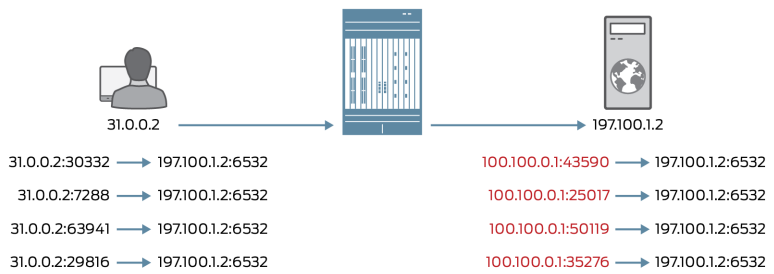

Figure 2.1 Dynamic One-to-One Mapping

The paragraph above stated, "*It is now a one-to-one mapping*" and, yes, that is true; any session originating from 31.0.0.2 will have 100.100.0.1 assigned to it for the lifetime of the mapping. But this is not a true one-to-one mapping in the sense that 100.100.0.1 will be the only private IP ever mapped to 31.0.0.2. The example is using a NAT pool with a /25 for the NAT'd IP address range for this pool. If you just brought the MX Series online and you now have 127 private hosts start to send traffic to be NAT'd at the same time, you will find two private IP addresses now using free ports under the 100.100.0.1 to NAT their traffic IP. If you ran the same test but had 253 private hosts start to send traffic to be NAT'd at the same time, you would have three private addresses now using any free port under 100.100.0.1.

One thing that is important to understand when talking about the `address-pooling paired` feature is the mapping of a private IP address to a NAT'd IP address and how long that mapping stays in memory. Remember that when the private subscriber stops sending or receiving traffic for a single session, the session will timeout after the inactivity timeout has been met, which by default is 30 seconds. *But what about the address pooling paired mapping?*

When all of the sessions for a given subscriber timeout the MX will keep the address pooling paired mapping in memory for five minutes by default. If the subscriber again sends any data from the same private IP address it has used before, the MX will once again map the private IP address to the exact same NAT address. Using the example where any traffic from 31.0.0.2 will see 100.100.0.1 used as the NAT'd IP address, if all of the NAT'd sessions from 31.0.0.2 are inactive for 30 seconds, no traffic is sent from the client or received from the public side. If five minutes have passed the address pooling paired mapping will be released from memory. When they start to send traffic again the MX Series will use the standard logic-based settings like `address-allocation round-robin` on what NAT'd IP address from the NAT pool to map to `address pooling pair` mapping. Your end users may very well have different NAT'd address many times a day if they sporadically send data.

You can change how long the MX Series stores the `address-pooling paired mapping-timeout` value by changing the `mapping-timeout` value under the NAT Pool configuration. The value can be set from anywhere between 120 to 86400 seconds based on your needs:

```
[edit services nat]
pool natpat44 {
    address 100.100.0.0/25;
    port {
        automatic {
            random-allocation;
```

```
        }
    }
    address-allocation round-robin;
    mapping-timeout 1200;
}
```

Since the inactivity timeout for sessions has been mentioned several times, let's quickly look at how you can change the inactivity timeout from its default of 30 seconds to a different value, which can be between 4 and 129600 seconds. This value can be set under the service interface, which affects all the sessions anchored on this interface. You configure one setting `inactivity-tcp-timeout` for TCP, and the other `inactivity-non-tcp-timeout` for every other protocol, thus UDP, ICMP, ESP, etc.:

```
[edit interfaces ms-1/0/0 services-options]
inactivity-tcp-timeout 60;
inactivity-non-tcp-timeout 60;
```

With any NAT feature that affects how NAT resources are assigned, best practice deems that thought needs to go into this. For example, do not apply `address-pooling paired` to a NAT term that handles proxy traffic like DNS from a server that sits on the private network. Proxy servers, especially ones that forward DNS traffic, but use the source IP of the proxy server itself, can send hundreds of thousands of requests a second. When NAT is needed and the `address-pooling paired` feature is enabled for the NAT term used to handle this type of traffic, you need to be aware that you are now limited to only 65535 possible sessions at one time since the proxy servers private IP addresses are mapped to a single NAT'd IP address. That can affect the service to your end users, and you should consider not using `address-pooling paired` for this type of traffic.

Endpoint Independent Mapping (EIM)

Another useful feature on the MX Series for NAT solutions is Endpoint Independent Mapping (EIM), which is used to assign the same external address and port for a specific session from a given private host. If other sessions from the same private host come in from a different source port, the MX assigns a different external address and port for these NAT'd sessions. With EIM a host can send traffic to ten different targets on the public side, and if these ten sessions all use the same source private IP address and source port, the ten targets on the public facing side will all see the same NAT'd IP address and port. If that same host sends traffic to those same ten targets but uses a different internal source port for each session, the MX will choose *any* free IP address and port combo to assign to the ten NAT'd sessions that will each get added to a new EIM mapping.

NOTE It should be noted that EIM is only needed and only works with NAT translation types that use a service card.

The EIM feature is enabled by setting `mapping-type` with the `endpoint-independent` option under the NAT rule, like so:

```
[edit services nat rule rule2]
match-direction input;
term t1 {
    from {
        source-address {
            100.100.0.0/16;
        }
    }
    then {
        translated {
            source-pool natpat44;
            translation-type {
                napt-44;
            }
            mapping-type endpoint-independent;
        }
    }
}
```

As a point of comparison, address-pooling paired is required for assigning the same external NAT IP address for all sessions from an internal private IP for a period of time, while EIM means you have the same external NAT IP address and port mapped to an internal IP address and a port for a period of time.

EIM, like address-pooling paired, also has a period of time where the mapping is held even after all sessions have timed out. Once the session is dropped on the MX, when data is not sent or received for the duration of the inactivity timeout, which by now you should know is a default 30 seconds, the MX will hold the End Point Mapping in memory for five minutes, the default EIM mapping timeout period. Within this five-minute period, if the session is re-initiated by the private subscriber (meaning the subscriber will still be using the same internal IP address and port), the MX will still use the same NAT'd external IP address and port assigned to this session. And if five minutes have passed before the session is re-initiated by the private subscriber, the MX will then assign a new NAT'd IP and port to the session.

Also, like the `address-pooling paired` feature, the mapping timeout default of five minutes can be changed by editing the same `mapping-timeout` value, set in seconds, under the NAT pool settings. Note that this is the exact same setting changed for the address-pooling paired mapping timeout, meaning that the address-pooling paired and EIM

mapping timeouts are tied to the same value when they use the same pool, but you can use the `ei-mapping-timeout` setting if you need to define the EIM mapping to use a separate value than the address-pooling paired value:

```
[edit services nat]
pool natpat44 {
    address 100.100.0.0/25;
    port {
        automatic {
            random-allocation;
        }
   }
    address-allocation round-robin;
    mapping-timeout 1700;
    ei-mapping-timeout 500;
}
```

If your NAT rule is set up to use both the address-pooling paired feature and the endpoint independent mapping feature (as in the example that follows) there are a few things to be aware of about how the `mapping-timeout` works when both are used. First, due to the address-pooling paired option, the MX Series will always use the same NAT'd address for the subscriber's source IP address. Due to the EIM option, the MX will use the same unique NAT'd port for each unique source port – a useful combination.

When the individual traffic sessions expire due to inactivity, the MX will timeout the EIM first. Once all the sessions for the subscribers' private source IP addresses are removed and all of the EIM are expired, then, and only then, does it start the countdown to expire the address-pooling paired mapping. As long as one session is active or one EIM has not timed-out, the MX will not start to count down the address-pooling paired mapping timeout:

```
[edit services nat rule rule2]
match-direction input;
term t1 {
    from {
        source-address {
            31.0.0.0/14;
        }
    }
    then {
        translated {
            source-pool natpat44;
            translation-type {
                napt-44;
            }
            mapping-type endpoint-independent;
```

```
            filtering-type {
                endpoint-independent {
                    prefix-list {
                        0.0.0.0/0;
                    }
                }
            }
            address-pooling paired;
        }
    }
}
```

Endpoint Independent Filtering (EIF)

Configuring only the EIM feature will solve any outbound stability requirements a service provider may have where a target host on the Internet always needs to see the same NAT'd IP address and port for a duration of time, but it will not allow inbound stability to be met.

So let's introduce our next feature, Endpoint Independent Filtering (EIF). In order to allow inbound connections from *any* public node or a defined range of public nodes after the outbound connection has been established, EIF needs to be configured in conjunction with EIM. What this means is when the subscriber's private IP and port are used to create a NAT'd session to a target, the target, and even other public nodes, can later send traffic inbound to the NAT'd address as long as they are sending the traffic destined to the NAT'd IP and source port the subscriber session was created with. This technology is typically called *full cone* when using vendor agnostic terms.

EIM with EIF with APP is required for gaming solutions like X-Box and PlayStation, certain VoIP solutions, and IM solutions. The need arises because of how private end users register with a server on the public network, then through this registration other public clients know how to reach the private client based on the NAT'd IP and port the private client registered with. When EIF is used, these different public clients can now initiate connections with the private client. If EIF is not used when these public clients try to initiate a connection to the private clients through the NAT'd IP address and port, the MX will drop the traffic since the pinhole is not open.

To set up an EIF example, under our NAT rule we add a `filtering-type` of `endpoint-independent` and optionally, a `prefix-list`. The prefix list defines which public nodes are allowed to establish inbound connections back through the session table using the EIM mapping. This actually allows you to control what public sources can use the EIM/EIF pinholes for inbound connections versus just allowing all of them, which could be unwanted based on the operator's needs.

NOTE If you do not add a prefix list any public node can reach back in through the pinhole, so it would act like a prefix list of 0.0.0.0/0.

Here's the EIF example:

```
[edit services nat]
    rule rule2 {
        match-direction input;
        term t1 {
            from {
                source-address {
                    31.0.0.0/16;
                }
            }
            then {
                translated {
                    source-pool natpat44;
                    translation-type {
                        napt-44;
                    }
                    mapping-type endpoint-independent;
                    filtering-type {
                        endpoint-independent {
                            prefix-list {
                                access_list_A;
                            }
                        }
                    }
                    address-pooling paired;
                }
            }
        }
    }
}
```

The prefix list itself is defined under the policy options hierarchy by adding a prefix list for the individual public hosts or public subnets you want the MX to allow traffic to be received through the EIF/EIM mapping and forwarded on to the private clients:

```
[edit policy-options]
  prefix-list access_list_A {
    197.100.1.0/24;
    198.100.100.100/32
}
```

BEST PRACTICE The EIM and address-pooling paired features use a small portion of memory for each mapping. So only use EIM and/or address-pooling paired for applications that actually reuse the source ports and/or require the NATing device to maintain the address or port mappings for the needed traffic, such as applications that use UNSAF processes. Read RFC 3424 for more information.

WARNING! Be careful and thoughtful when setting up EIF. You are allowing public nodes to reach back in through the MX. This setting could and does potentially allow public nodes to be able to create a large number of sessions, possibly exhausting all the memory in the service PIC. An external node can send a nearly unlimited number of sessions through the same EIF pinhole by changing its source IP address/port. This could be a DoS attack on the private IP address or a DoS attack against the service PIC, since it consumes memory through each flow.

To help protect the MX Series, and the subscribers that sit on the private network, the MX has a setting called `eif-flow- limit`. To limit the number of inbound connections on an EIM/EIF mapping, include the `eif-flow-limit` number of flows statement at the `[edit services nat / rule rule-name term term-name, then translated secure-nat-mapping]` hierarchy level, like so:

```
[edit services nat]
    rule rule2 {
        match-direction input;
        term t1 {
            from {
                source-address {
                    31.0.0.0/16;
                }
            }
            then {
                translated {
                    source-pool natpat44;
                    translation-type {
                        napt-44;
                    }
                            secure-nat-mapping {
                           eif-flow-limit 1000;
                }
                    mapping-type endpoint-independent;
                    filtering-type {
                        endpoint-independent {
                            prefix-list {
                                access_list_A;
                            }
                        }
                    }
                    address-pooling paired;
                }
            }
        }
    }
}
```

In the next example you can see you the output of the `show services nat mappings endpoint-independent` command to look at our current EIM mappings. In this case the private address at 31.0.0.33 with a

source port of 44444 has an EIM mapping of 156.100.0.11 with the port 61824:

```
user@re0> show services nat mappings endpoint-independent
Interface: ms-1/0/0, Service set: SS1

NAT pool: nat44
Mapping          : 31.0.0.33     :44444  --> 156.100.0.11    :61824
Session Count    :     6
Mapping State    : Active
```

Using this NAT'd IP and a source port of 156.100.0.11, port 61824 the private client at 31.0.0.33 is sending UDP traffic to a public target at 90.0.0.11 port 389. As stated, when using EIM/EIF, other nodes on the public side can now reach back into 31.0.0.33 by sending traffic to that NAT IP and port, which in this example is 156.100.0.11:61824. As mentioned earlier, with EIF you can configure it to only allow certain ranges/targets to be able to reach back in, but in this example it was left wide open.

If you look at the traffic being sent from the Internet towards the NAT'd address and port via the `show services sessions` command, so the traffic is in the `Forward 0` direction, you can see that traffic sourced from different Internet peers is passing back in through the EIF pinhole to get de-NAT'd and then sent to the private IP at 31.0.0.33. Figure 2.2 illustrates what's happening:

```
UDP           90.0.0.11:389  ->    156.100.0.11:61824  Forward  0         2314777
    NAT dest      156.100.0.11:61824   ->     31.0.0.33:44444
  Byte count: 79623716
  Flow role: Master, Timeout: 32
UDP           90.0.0.10:6679  ->    156.100.0.11:61824  Forward  0         2373098
    NAT dest      156.100.0.11:61824   ->     31.0.0.33:44444
  Byte count: 82128564
  Flow role: Master, Timeout: 29
UDP           231.10.110.89:46345  ->    156.100.0.11:61824  Forward  0          665341
    NAT dest      156.100.0.11:61824   ->     31.0.0.33:44444
  Byte count: 22880266
  Flow role: Master, Timeout: 28
UDP           130.0.0.88:32255  ->    156.100.0.11:61824  Forward  0          690824
    NAT dest      156.100.0.11:61824   ->     31.0.0.33:44444
  Byte count: 23855794
  Flow role: Master, Timeout: 32
```

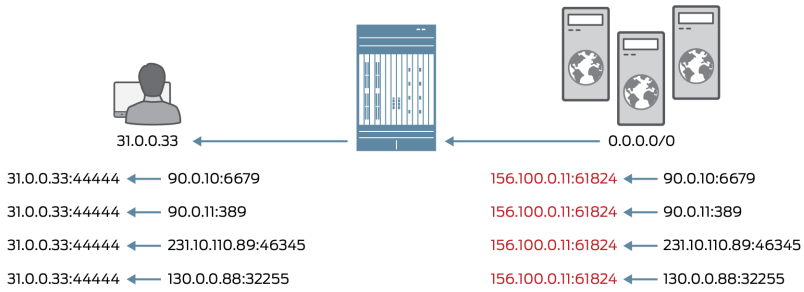

Figure 2.2 Traffic Sourced From Different Internet Peers

If these same public nodes try to reach into this NAT'd address on any other source ports that have not been mapped by the private addresses to an EIM/EIF mapping, the traffic will be dropped and not forwarded on to 31.0.0.33. Figure 2.3 illustrates this traffic path:

```
UDP           90.0.0.11:389     ->    156.100.0.11:9000    Drop      0           97730
  Byte count: 3346280
  Flow role: Initiator, Timeout: 4
UDP           90.0.0.10:6679   ->    156.100.0.11:9000    Drop      0          105284
  Byte count: 3597302
  Flow role: Initiator, Timeout: 4
UDP           231.10.110.89:46345  ->    156.100.0.11:6165    Drop      0           21157
  Byte count: 725900
  Flow role: Initiator, Timeout: 4
UDP           130.0.0.88:32255  ->    156.100.0.11:6165    Drop      0           23302
  Byte count: 796722
  Flow role: Initiator, Timeout: 4
```

Figure 2.3 Traffic Being Dropped

WARNING! Wait, another warning! This EIF feature can be dangerous and you need to make sure you are aware of the pitfalls and configure to avoid them if necessary. With the EIF feature enabled on the MX Series, be aware that nodes on the public side could continue to send traffic to the private address for an unlimited amount of time. Even if the private

client has powered off, traffic can keep coming. This could mean sessions still showing as active on the MX, tying up resources. There is another setting you can use that can be applied under the service NAT rule itself called *mapping-refresh*. If you set this to a value of outbound only, traffic sent from the private side towards the public side is what the MX calculates to see if a session is inactive or not. Once an outbound session from the private network to the public is timed out, the inbound flow from the public network will also be removed, even if the pubic host is still sending data to the private host:

```
[edit services nat]
    rule rule_1 {
        match-direction input;
        term t1 {
            then {
                translated {
                    source-pool natpat44;
                    translation-type {
                        napt-44;
                    }
                 secure-nat-mapping {
eif-flow-limit 1000;
mapping-refresh outbound;
                    }
                    mapping-type endpoint-independent;
                    filtering-type {
                        endpoint-independent {
                            prefix-list {
                                access_list_A;
                            }
                        }
                    }
                    address-pooling paired;
                }
            }
        }
    }
}
```

Best Practices with EIF

At this point a best practices design can be useful to protect the service PICs CPU even when EIF is not used, but in fact, it should be thought about when employing EIF. The truth is – watch out when using the EIF feature with the next hop style of service sets, since you can cause unexpected traffic on the service PIC or even potentially a traffic loop if you are not careful. An example of this occurs when using anything from a basic virtual router with a single interface facing a private network to a VRF hosting an L3 VPN. You need to be aware that traffic ingressing the routing instance will, as expected, use the route that steers the traffic towards the inside service interface to get pro-

cessed by the NAT engine. If the NAT rules process the request, you now have an open pinhole. If EIF is used you now have the potential for any public node to reach back into the open pinhole. A problem occurs when traffic from the public network destined to a NAT'd address hits the routing instance called *client* and does not have a route back out the physical interface facing the private network for the de-NAT'd private IP address. This traffic from the public side reaches back in through the pinhole and gets routed right back through the service interface to get NAT'd again causing issues.

In this best practices example, there is a single physical interface, facing the private network, the 31.0.0.0/16 network:

```
[interfaces xe-3/0/2]
description "to client network";
unit 0 {
    family inet {
        address 31.0.0.1/16;
    }
}
```

This interface has been added to a routing instance called *client* along with the inside service interface for the next hop style service set. There is a single static route added to this routing instance to route all destination traffic into the inside service interface to get NAT'd. This is a typical setup where the private network needs all of its traffic that ingresses the MX to be NAT'd:

```
[edit routing-instances client]
instance-type virtual-router;
interface ms-1/0/0.1;
interface xe-3/0/2.0;
routing-options {
    static {
        route 0.0.0.0/0 next-hop ms-1/0/0.1;
    }
}
```

Now, in the routing instance client route traffic destined to 31.0.0.0/16 back out the xe-3/0/2 interface:

```
user@re0# run show route 31.0.0.0/16

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

0.0.0.0/0          *[Static/5] 02:59:34
                    > via ms-1/0/0.1
31.0.0.0/16        *[Direct/0] 19:50:46
                    > via xe-3/0/2.0
31.0.0.1/32        *[Local/0] 19:50:55
                      Local via xe-3/0/2.0
```

The problem with this setup is if any traffic enters the MX through ge-3/0/2 from a source address other than an expected IP in the 31.0.0.0/16 range, it could possibly get NAT'd based on how the NAT rule is configured. This means when a public node replies to the NAT'd IP address and port and gets de-translated back to a non-31.0.0.0/16 private IP address, the 0.0.0.0/0 will pick this traffic up and send it back through to get NAT'd again creating a new session. Right here, even without EIF, you are sending packets back through the service PIC to get NAT processed, which can eat up CPU cycles on the service PIC. If EIM/EIF was used you could see many public nodes sending traffic to this private client, which also opens up yet another session/pinhole.

For example, if traffic sourced from 44.0.0.2 and destined to 197.100.1.2 hit this routing instance and was NAT'd, and you were using EIF, you are now potentially at risk for many new NAT'd sessions opening up. Here the session sourced from 44.0.0.2 is created:

```
Service Set: nat44, Session: 134274463, ALG: none, Flags: 0x0000, IP Action: no, Offload: no,
Asymmetric: no
NAT PLugin Data:
  NAT Action:   Translation Type - NAPT-44
    NAT source             44.0.0.2:17951   ->     100.100.0.4:6332
UDP            44.0.0.2:17951  ->     197.100.1.2:12229  Forward  I               1
  Byte count: 128
  Flow role: Initiator, Timeout: 420
UDP         197.100.1.2:12229  ->     100.100.0.4:6332   Forward  O               0
  Byte count: 0
  Flow role: Responder, Timeout: 420
```

Here traffic from another node on the public side reaches back into this NAT'd address 100.100.0.4:6332, which will happen since EIM/EIF is used. Remember, it gets steered back through the service PIC because the route in the client routing-instance tells it to:

```
0.0.0.0/0        *[Static/5] 02:59:34
                 > via ms-1/0/0.1

Service Set: nat44, Session: 100724789, ALG: none, Flags: 0x0000, IP Action: no, Offload: no,
Asymmetric: no
NAT PLugin Data:
  NAT Action:   Translation Type - NAPT-44 (EIF)
    NAT source             44.0.0.2:17951   ->     100.100.0.4:6332
UDP        197.100.1.88:6532   ->     100.100.0.4:6332   Forward  O            8170
  Byte count: 1045760
  Flow role: Initiator, Timeout: 420
UDP            44.0.0.2:17951  ->    197.100.1.88:6532   Forward  I               0
  Byte count: 0
  Flow role: Responder, Timeout: 420
```

Let's create a new session for the traffic destined to 44.0.0.2, since the route in the routing instance points back into the service PIC. Now the

Internet host 197.100.1.88 is coming from the inside and taking up a NAT'd resource:

```
user@re0# run show services sessions source-prefix 44.0.0.2 source-port 17951 extensive
ms-1/0/0
Service Set: nat44, Session: 100724847, ALG: none, Flags: 0x0000, IP Action: no, Offload: no,
Asymmetric: no
NAT PLugin Data:
  NAT Action:   Translation Type - NAPT-44
    NAT source         197.100.1.88:6532    ->     100.100.0.5:43165
UDP        197.100.1.88:6532   ->        44.0.0.2:17951  Forward  I            8170
  Byte count: 1045760
  Flow role: Initiator, Timeout: 420
UDP            44.0.0.2:17951  ->     100.100.0.5:43165  Forward  O               0
  Byte count: 0
  Flow role: Responder, Timeout: 420
```

When you can, one thing you can do to protect your setup is to make sure you always define the source address range under the NAT rules so that other unexpected source traffic does not get NAT'd that creates a pinhole, or in the case of EIF, potentially many pinholes. Let's define the source address:

```
[edit services nat]
    rule rule_1 {
        match-direction input;
        term t1 {
    from {
        source-address {
            31.0.0.0/16;
        }
    }
 then {
                translated {
                    source-pool natpat44;
                    translation-type {
                        napt-44;
                    }
            }
                    mapping-type endpoint-independent;
                    filtering-type {
                        endpoint-independent {
                        }
                    }
                    address-pooling paired;
                }
            }
        }
    }
```

You could also apply a filter to your physical interfaces to drop traffic from unexpected networks, or if you are using a VRF with Layer 3 VPN, you can apply the filter to the service interface itself to address this potential design flaw:

```
[edit firewall filter drop]
term 1 {
    from {
        address {
            31.0.0.0/16;
        }
    }
    then accept;
}
term 2 {
    then {
        discard;
    }
}

[edit interfaces ms-1/0/0]
unit 1 {
    family inet {
        filter {
            input drop;
        }
}
    service-domain inside;
}
unit 2 {
    family inet;
    service-domain outside;
}

[edit interfaces ge-3/0/2]
user@re0# show
unit 0 {
    family inet {
        filter {
            input drop;
        }
        address 31.0.0.1/16;
    }
}
```

JTAC has seen incidents where end users on the private network are assigned an IP that is part of the NAT pool by accident, so when traffic sourced from these users comes into the routing instance from the private network, it will get NAT'd. An IP that exists in the NAT pool is now NAT'd. If any traffic from the Internet reaches back through this pinhole it will see the NAT address get de-NAT'd, hit the routing instance and then route back through the NAT engine to get NAT'd again. Since its route is destined to a NAT address whose routes points right back through the outside interface to get de-NAT'd, and then NAT'd all over again, this causes a routing loop and eats up CPU cycles on the service PIC.

Best practice is to set the source address range under the NAT rules.

Chapter 3

Application Layer Gateways and User-Defined Application Controls

Application Layer Gateways (ALGs) get their own chapter, which looks at the five-tuple application matching users can define and use with the NAT rules. This chapter also dives into the ALGs a bit so you can understand what is going on under the hood when it comes to the MX Series helping your NAT applications that may not be NAT friendly after all.

The ALG code allows the MX Series CGNAT features to work with certain applications that need to have some of their characteristics changed to work in a NATing environment. It is one of the strongest functionalities that the service cards bring to the table.

This chapter also includes application definitions that are not ALGs, but are methods an operator can use to decide what to do NAT-wise with traffic matching the definitions of a certain application, which are port and protocol.

Below is a limited list of just a few ALGs supported in the MX Series using the service cards:

- FTP
- TFTP
- RTSP
- PPTP
- MSRPC

- SUNRPC
- TALK
- RSH
- SQL-NET
- SIP
- ICMP
- H232
- IKE/ESP

This chapter could include dozens of pages discussing the functionality of each ALG but since this is a *Day One* book, it briefly covers a few of the ALGs and shows you how to configure them. Use the documentation for details and incidentals.

MORE? ALGs are well covered in the technical documentation for Junos. Search http://www.juniper.net/documentation for *ALG*.

Okay, let's first look at how you generically enable ALGs using the ICMP ALG. This will help you to understand the ALG configuration. *Why have an ICMP ALG in the MX code?* Because without the ICMP ALG there is no way the MX could cleanly handle certain ICMP events when NAT is used. For example, within the ICMP portion of the packet the MX needs the NAT'd IP address to be rewritten for events like the ICMP type `destination unreachable`. This is needed because when the ICMP event occurs, the ICMP packet is sent back to the MX from a public host with the source address in the ICMP/IP payload set to the NAT'd address. The MX then needs to have the service PIC rewrite the packet so that this source address is translated back to the subscriber's private address. This magic of rewriting the application layer and not just the standard IP header is performed by the ICMP ALG!

You add the ALGs to a NAT rule under a defined term. This term will be used only for the applications you apply to this term, as in this example where the ICMP ALG is added:

```
[edit services nat rule rule2 term t1]
from {
    source-address {
        33.33.0.0/16;
    }
    applications junos-icmp-all;
}
```

There are many predefined ALGs and applications you can select from under the applications hierarchy, but in this example the predefined application definition `junos-icmp-all` was used, which is the ICMP ALG.

Here is what your session looks like when you have the ICMP ALG application configured and attached to your rule:

```
user@re0# run show services sessions extensive
ms-1/0/0
Service Set: nat44, Session: 1342177285, ALG: icmp, Flags: 0x2000, IP Action: no, Offload:
no, Asymmetric: no
NAT PLugin Data:
  NAT Action:   Translation Type - NAPT-44
    NAT source   212.27.42.153:51806   ->   150.100.100.4:55916
ICMP     212.27.42.153       ->     188.0.0.10       Forward  I              28
  Byte count: 2352
  Flow role: Initiator, Timeout: 32
ICMP        188.0.0.10       ->  150.100.100.4       Forward  O               28
  Byte count: 0
  Flow role: Responder, Timeout: 32
Service Set: ss1, Session: 33554447, ALG: none, Flags: 0x0000, IP Action: no, Offload: no,
Asymmetric: no
```

And here is what it looks like when you do not:

```
user@re0# run show services sessions extensive
ms-1/0/0
Service Set: nat44, Session: 1342177285, ALG: icmp, Flags: 0x2000, IP Action: no, Offload:
no, Asymmetric: no
NAT PLugin Data:
  NAT Action:   Translation Type - NAPT-44
    NAT source   212.27.42.153:51806   ->   150.100.100.4:55916
ICMP     212.27.42.153       ->     188.0.0.10       Forward  I              28
  Byte count: 2352
  Flow role: Initiator, Timeout: 32
ICMP        188.0.0.10       ->  150.100.100.4       Forward  O               28
  Byte count: 0
  Flow role: Responder, Timeout: 32
Service Set: ss1, Session: 33554447, ALG: none, Flags: 0x0000, IP Action: no, Offload: no,
Asymmetric: no
```

They are identical! It should be stated here that unlike all of the other ALGs the ICMP ALG is in use by default. You actually do not need to define the ICMP ALG under your NAT rule.

NOTE Do not let the ICMP ALGs being used by default confuse you. All other ALGs must be enabled by you as the operator or they will not be used, but this example is good since it informs you, as the operator, that the ICMP ALG is in play by default.

Let's dig a bit deeper into ICMP ALGs since there are options you can configure to give you further control over when to NAT an ICMP packet under your NAT rule. By going to the [applications] hierarchy you can actually define your own ICMP applications.

In the example the ICMP-type option echo-request is added to a user created ICMP application that we will create and call ICMP_USER_ALG. Based on the match direction of the NAT rule, you now have a setup where only ICMP echo requests or pings from the subscriber's private IP addresses, will be NAT'd by the NAT term using this user-defined application. All other ICMP packets will just be sent to the next term under the NAT rule. If this was the only NAT term you defined in your NAT rule, any other ICMP types that were sent through this configuration would be forwarded without being NAT'd.

To make this work, add the application created under the NAT rule to have the logic take effect:

```
[edit applications application ICMP_USER_ALG]
application-protocol icmp;
protocol icmp;
icmp-type echo-request;



edit services nat rule rule1]
match-direction input;
term 2 {
    from {
        applications ICMP_USER_ALG;
    }
    then {
        translated {
            source-pool pool1;
            translation-type {
                napt-44;
            }
        }
    }
}
```

TIP There are many more ICMP codes and ICMP type options you can set under the ICMP ALG. See the Juniper TechLibrary: https://www.juniper.net/techpubs/en_US/junos15.1/topics/concept/alg-support-for-ms-mic-ms-mpc.html.

The purpose of all of this was to show you how the configuration works and how you can use ICMP ALG to control your setup. But the truth is, most operators will never likely need to configure the ICMP ALG. The default behavior is fine in most use cases. With that said, let's' take a look at how we can create a user-defined application to

change the session behavior for just DNS traffic, leaving all other application traffic to use the default settings. Since the MX does not do anything special to DNS traffic you do not have a DNS ALG. This just means with DNS traffic being NAT'd you do not have to analyze or rewrite the layer seven portion of the packet like you do with SIP or certain ICMP packets, nor do you have to open special pin holes as done for PPTP.

But for this example, let's have the MX aggressively timeout DNS sessions. In many networks DNS traffic can be quite heavy at times, but normally they are very short-lived UDP sessions, DNS being just a query and a response. Having all of these DNS NAT sessions sit in memory for 30 seconds, which is, remember, the default inactivity timeout, can actually tie up quite a bit of memory and NAT'd resources since there can be a ton of DNS passing through the service PIC in many mobile and fixed-line operators' setups.

The question you might have is why tie up a session or NAT'd resources for a protocol like DNS for 30 seconds if DNS resolves in under a second on our network? This example will show you that you can set up a user-defined DNS application and then set a lower or even higher inactivity timeout to help efficiently clean up these sessions, or leave them active for longer if need be. Here is an example that defines the user-created DNS application under the `[edit applications]` hierarchy. Note that in the example you want to set a NAT term that calls the DNS application under the NAT rule *before* the generic catch-all rule that was added to handle all other traffic types:

```
[edit applications]
application user_defined-dns-udp {
    protocol udp;
    destination-port 53;
    inactivity-timeout 5;
}


[edit services nat rule rule1]
match-direction input;
term t1 {
    from {
        applications user_defined-dns-udp;
    }
    then {
        translated {
            source-pool nat44;
            translation-type {
                napt-44;
            }
        }
    }
}
```

```
term t2 {
    from {
    }
    then {
        translated {
            source-pool nat44;
            translation-type {
                napt-44;
            }
        }
    }
}
```

By now you should have some basics down: how to configure and define further if needed for an ALG and how to create a user-defined application. It's time to dig down a bit deeper again. This time let's look at the FTP ALG which should give you an even deeper understanding of how ALGs work and how to use them when needed.

When using FTP in active mode, the FTP server actively connects to the FTP client. To set up active mode, the client sends a PORT command to the server, specifying the IP address and port number the client is listening on. When a data connection is required, the server initiates a connection to the client at this address and port. When using NAT, the IP address in the PORT command needs to be rewritten with the NAT'd IP address used for the connection and this is what the FTP ALG does when Active FTP is used.

NOTE When using napt44 the port under the PORT command also needs to be rewritten.

When using FTP in passive mode, the client sends a PASV command to the server. This tells the server to listen for a connection attempt from the client, hence the server is passively waiting. The server replies to PASV with the host and port address that the server is listening on. The client deciphers this reply and when a data connection is required, attempts to initiate the connection to the server at this address.

But what if the FTP server is on the private network, as it is when it is used in a destination NAT scenario? Then the passive IP and port will not be translated by the FTP ALG because the FTP ALG was not designed to re-write this packet during the exchange. So be aware that the FTP ALG works with passive mode when the client is the source address being NAT'd, but not when the private client is actually the FTP server.

In order to flag the FTP ALG that the FTP commands are encrypted, you need to let encrypted packets go through the service PIC without inspecting them. The MX's FTP ALG looks for the AUTH TLS or the legacy AUTH SSL string as a pattern match. It has been seen that there

are some old FTP clients that are still using the AUTH SSL command when they negotiate encryption with the FTP server (explicit SSL).

If the FTP server responds back that it supports the encryption type the MX will consider the FTP packets for these sessions are now encrypted. If the server rejects the encryption or ignores the request and the client does not enforce encryption being used, the MX will treat the FTP traffic as if it is in the clear.

Let's take a look at the PPTP ALG since it is a great example for showing you how the ALG code can be used to open pinholes to allow expected traffic from the public side toward the private side when the public side is initiating a new session.

PPTP used TCP destination port 1723 as a simple control channel whose sole purpose is to bring up a Microsoft-specific secondary GRE IP protocol 47 tunnel. This GRE tunnel contains encapsulated PPP frames and is used for negotiating authentication, encryption, and passing actual data.

The negation is as follows, a TCP three-way handshake followed by PPTP Start Control-Connection-Request and Reply, then outgoing Call Request and Reply, and then, following the last reply from the PPTP public server, the server should send a PPP-LCO GRE encapsulated request to the private client. It is here that the pinhole gate needs to be opened by the PPTP ALG so this first packet for the GRE tunnel initiated from the PPTP server on the public side can actually get de-NAT'd and sent to the private client. Without the PPTP ALG enabled this GRE initiated session from the PPTP server would get dropped by the session table.

As stated at the beginning of this chapter, there are many functionalities of of ALGs. This book is not a comprehensive dive into each one, but here are a few last words on the ALGs. It is not advisable to enable every ALG. Traffic being processed by an ALG is more expensive CPU-wise than standard TCP and UDP traffic, for example. If you use a NAT-friendly technology like STUN for SIP traffic, do not waste CPU cycles by enabling the SIP ALG since it is not needed.

BEST PRACTICE

If application traffic tied to an ALG requires nodes on the public side to reach back in through pinholes, the EIM/EIF features are not required because the ALG code will open the pinholes itself, in the same way as the PPTP ALG works. There is no predefined ALG that requires EIM/EIF.

A recommended NAT configuration for operators that do require EIM/EIF for their solution would be to define the Junos ALGs required (under the `[edit applications]` hierarchy using an `application-set`):

```
[edit applications]
application-set accept-alg {
    application junos-sip;
    application junos-h323;
    application junos-pptp;
    application junos-rtsp;
    application junos-ftp;
}
```

Then you can define a rule that matches this application set that does *not* use EIM/EIF, and a second rule that matches all other application traffic that does use EIM/EIF:

```
[edit services nat]
    rule rule1 {
        match-direction input;
         term ALG {
            from {
                }
                application-sets accept-alg;
            }
            then {
                translated {
                    source-pool nat44;
                    translation-type {
                        napt-44;
                    }
                }
            }
        }
        term non_ALG {
            from {
            }
            then {
                translated {
                    source-pool nat44;
                    translation-type {
                        napt-44;
                    }
                    mapping-type endpoint-independent;
                    filtering-type {
                        endpoint-independent;
                    }
                }
            }
        }
    }
```

Chapter 4

Final Configuration Topics

The topics covered in this final chapter on configuration could have been included anywhere in this book. They have synergy with the content of other chapters, but in trying to keep things streamlined along the way we have moved them to this chapter. Let's dive into Chapter 4, and remember you are almost at the end!

Syslog

Syslogging is a very important activity for many service providers who need to track which subscriber used which public address and port via their private address and port combos, and at what time they used the NAT resource. Without syslogging, the service provider has no means to make this historical match. Syslogging can also be used to log messages concerning issues that may be occurring in regard to the CGNAT resources or the health of the service PICs.

The MX Series can write logs pointing to an external syslog server or logging locally to the Routing Engine (RE) by writing the messages in the /var/log directory. It is recommended that you write to an external syslog server instead of the RE due to the limited space on the RE and the fact that it will now have to process each NAT-based syslog message sent to it from the service PICs, which, based on the setup, could be a high volume. In fact, as you go through this chapter, keep in mind that syslogging puts extra strain on service PIC CPU processing and that you only should enable syslogging when you feel you need it. This is one reason many service providers deploy PBA, or the deterministic NAT translation type, since PBA can cut back on the number

of syslog messages written and deterministic NAT can outright eliminate needing any syslog messages in regard to NAT sessions being created and deleted.

Now, onto our syslog setup. There are three key control points set in the Junos OS hierarchy where the MXs service cards can capture data to write to syslog:

- interfaces / service-interface
- services / service-set
- services / nat / rule /term

The service-interface is the highest point in the Junos hierarchy that you can enable the syslog functionality for CGNAT, but this control point is used for syslogging infrastructure information around the service interface. This setting will not enable logging of information pertaining to the creation and deletion of NAT'd sessions.

It should be noted that the syslog messages created under the interfaces hierarchy can be used to notify you that general issues occurred with your CGNAT setup. For example, when flow/session limits have been exceeded, the following would be a syslog message you receive when syslogging is enabled under the service-interface:

```
Apr  7 18:01:01  JTAC_setup-re0 (FPC Slot 1, PIC Slot 0)  ms10 mspmand[190]: Service set
nat44, flows 750, reached high water mark 75 for maximum flows 1000
Apr  7 18:05:36  JTAC_setup-re0 (FPC Slot 1, PIC Slot 0)  ms10 mspmand[190]: Service set
nat44, flows 0, reached low water mark 1 for maximum flows 1000
```

Remember flow/session limits and SNMP thresholds are set under the `service-set` hierarchy:

```
{master}[edit services service-set nat44]
user@re0# show
max-flows 1k;
snmp-trap-thresholds {
    flow low 1 high 75;
}
```

If you enable the syslog under the service interface, the syslog messages will be enabled across all service sets that use this service interface. The syslog setting under `service-interface` allows you to apply filters on the service to filter out the severity of the events you want to see written to the syslog. Let's set the syslog to write to the local host only for critical issues:

```
[edit interfaces ms-2/0/0]
services-options {
    syslog {
        host local {
            services critical;
```

You could have chosen any one of these levels to decide what gets written:

```
user@re0# set services ?
Possible completions:
  <[Enter]>            Execute this command
  alert                Conditions that should be corrected immediately
  any                   All levels
  critical             Critical conditions
  emergency            Panic conditions
  error                Error conditions
  info                 Informational messages
  none                 No messages
  notice               Conditions that should be handled specially
  warning              Warning messages
```

Also under the syslog settings of the service interface you can add a log-prefix, which prepends the syslog message with the entered string name, in this case PIC0. This can be useful if you have lots of service interfaces or many nodes on the network writing to the syslog server, allowing you to distinguish messages created by certain service interfaces if you need to:

```
[edit interfaces ms-2/0/0]
services-options {
    syslog {
        host local {
            services any;
            log-prefix PIC0;
        }
    }
```

One key setting you can apply to the service interface that will affect syslogging from this service PIC is the `message-rate-limit` feature, which allows you to scale back the number of syslog messages sent per second if the services PIC is hitting performance issues. Many service providers would rather drop syslog messages in order to process NAT sessions:

```
[edit interfaces ms-2/0/0 services-options syslog]
user@re0# set ?
 message-rate-limit  Maximum syslog messages per second allowed from this interface
(messages per second)
```

Next, let's look at the syslogging, which can be enabled at the individual service sets and here is where many operators will look to enable syslogging with the CGNAT solution since they can actually send messages here that pertain to the NAT'd sessions. Here you can apply the log prefix and services filter, but you can also set a class filter which unlocks the CGNAT specific syslog message.

NOTE You have to have the `services` setting be at least at the info level to get the class messages to write.

Here are the classes you can filter on when using CGANT:

- `nat-logs`: Log Network Address Translation events
- `session-logs`: Log session open and close events

And here is the filter:

```
[edit services service-set nat44]
 syslog {
    host local {
        services info;
        class {
            session-logs;
            nat-logs;
        }
    }
}
```

Note that the other classes that can be set under the service set are not supported with the CGNAT solution on the MS-MIC or MS-MPC:

```
user@re0# commit
re0:
 [edit services service-set nat44 syslog host local]
  'class'
    alg-logs are not supported on ms-interface.
```

Setting the session logs class will write the messages shown next. You can see a UDP session being created and then closed out a few minutes later. Session logs are what you want to employ if you are using napt44, for example, without PBA and need to log the sessions to see which private IP address used which pubic IP address and port at a specific historical time:

```
Apr  7 17:01:27  JTAC_setup-re0 (FPC Slot 2, PIC Slot 0) 2016-04-07 21:01:26: {nat44}
JSERVICES_SESSION_OPEN: application:none, ge-3/0/2.0 31.0.20.197:55568
[100.100.0.122:62282] ->  197.100.1.149:6332 (UDP)

Apr  7 17:03:37  JTAC_setup-re0 (FPC Slot 2, PIC Slot 0) 2016-04-07 21:03:36: {nat44}
JSERVICES_SESSION_CLOSE: application:none, ge-3/0/2.0 31.0.20.197:55568
[100.100.0.122:62282] ->  197.100.1.149:6332 (UDP)
```

You should set the NAT logs when you need to track PBA blocks. Since the PBA feature allocates a block of ports to a private address, the MX does not need to log the creation and deletion of every session. So to cut back dramatically on the possible number of syslog messages created and sent you may remove the class type session logs when the PBA feature is being used, but make sure you set the `class` type `nat-logs`, though, so you can see the port blocks being allocated when

assigned to a private subscriber and then released when all of the sessions and mappings have timed out for the block. Below is an example of what will be written to the syslog when PBA is being used and the NAT-log class is set:

```
Apr  7 17:23:46  JTAC_setup-re0 (FPC Slot 2, PIC Slot 0) 2016-04-07 21:23:45: {nat44}
[jservices-nat]: JSERVICES_NAT_PORT_BLOCK_ALLOC: 31.0.18.96 -> 100.100.0.7:24224-24323
0x5706cfe2

Apr  7 17:28:35  JTAC_setup-re0 (FPC Slot 2, PIC Slot 0) 2016-04-07 21:28:35: {nat44}
[jservices-nat]: JSERVICES_NAT_PORT_BLOCK_RELEASE: 31.0.18.96 -> 100.100.0.7:24224-24323
0x5706cfe2
]:
```

Now there are several other syslog messages that are written when using the NAT class outside of the `JSERVICES_NAT_PORT_BLOCK_ALLOC` and `JSERVICES_NAT_PORT_BLOCK_RELEASE` messages that can be used to historically see what occurred with the NAT solution when viewing the syslog records.

When the PBA feature is not used, but the `nat-logs class` is set under the syslog hierarchy, the MX will create a syslog message when each port is freed:

```
Apr  7 15:47:37  archer-MX960-re0 (FPC Slot 1, PIC Slot 0) 2016-09-01 19:47:37: {ss1}
[jservices-nat]: JSERVICES_NAT_POOL_RELEASE: natpool release 100.100.0.7:44429[1]
```

When using any NAT translation type that employs port address translation, you will log when a pool is out of ports, meaning the pool in question is at 100% capacity when the `nat-logs class` is set under the syslog hierarchy:

```
Apr  7 16:55:21 beans (FPC Slot 3, PIC Slot 2) 2015-01-17 00:55:21: {ss5}[jservices-nat]:
JSERVICES_NAT_OUTOF_PORTS: natpool nat_pool5 is out of ports
```

When using the dynamic nat44 translation type, which, if you remember, does not use port address translation, log when a pool is out of IP addresses, meaning the pool in question is at 100% capacity when the `nat-logsnatl class` is set under the syslog hierarchy:

```
Apr  7 16:00:06 beans (FPC Slot 3, PIC Slot 2) 2015-01-17 00:00:05: {ss5}[jservices-nat]:
JSERVICES_NAT_OUTOF_ADDRESSES: natpool nat_pool5 is out of addresses
```

The last place you can also set syslogging is at the [`services nat rule`] level where you can tell the system to log additional NAT log messages when a given term is hit under a NAT rule. This can be useful for initial setup or even analysis when you need to know which NAT terms are actually being used and how often, but it can also create a large number of syslog messages that may not have much historical relevance. This is why this NAT class syslog message is enabled in this

special way. This special syslog message will only get written if syslogging is enabled under the service set configured to use this NAT rule and if the NAT class is being used for syslog under the service set:

```
{master}[edit services nat rule rule_1]
user@re0# show
match-direction input;
term 1 {
    then {
        translated {
            source-pool pool_1;
            translation-type {
                napt-44;
            }
            mapping-type endpoint-independent;
            filtering-type {
                endpoint-independent;
            }
            address-pooling paired;
        }
        syslog;
    }
```

```
JSERVICES_NAT_RULE_MATCH: proto 17 (UDP) application: any, ge-3/0/2.0:31.0.21.233:43018 ->
197.100.1.103:6332, Match NAT rule-set: (null), rule: rule_1, term: 1
```

To verify that the service PIC is writing syslog messages, run the `show services service-sets statistics syslog` command:

```
[edit services service-set nat44]
user@re0# run show services service-sets statistics syslog
Interface: ms-2/0/0
  Rate limit: 10000
  Sent: 0
  Dropped: 0
  Service-set: nat44
    Sent: 1014530
    Dropped: 0
```

If you do not set the syslog stanza under services / nat / rule and services / service-set hierarchy, and only set it at the service interface level, you will see that it is the functional point where these events are created:

```
[edit services service-set nat44]
user@re0# run show services service-sets statistics syslog
Interface: ms-2/0/0
  Rate limit: 10000
  Sent: 750
  Dropped: 0
```

NOTE When using deterministic NAT you should not need to set the NAT log or session log classes. As stated several times in this book, that is where deterministic NAT shines. You know which private IP address and port map to which NAT'd public IP address and port. It makes syslogging practically unnecessary.

Service Filters and Other Filtering Options

So far we have not looked at any ways to control what traffic gets NAT'd or what really gets steered into the service PICs outside of using next hop style service sets where you can define what destination networks are steered to the service PIC. There are some good options to show you that it can be really helpful, especially for interface style service sets, since so far the configuration used for interface style drives all traffic entering or exiting the interface to get serviced based on the NAT rules direction.

To start, let's use the example of an operator who doesn't want you to NAT ICMP packets. There are three options you can use.

First, if you employ interface style service sets you can create what is called a *service filter* and attach it to the physical interfaces so the ICMP traffic never even goes to the service PIC when ICMP packets from the private network are received. Note this does not drop the packet like a firewall filter, it just tells the packet that its next hop is *not* the service PIC and it continues along to the egress point without ever touching the service PIC:

```
[edit firewall family inet service-filter ICMP]
term t1 {
    from {
        protocol icmp;
    }
    then skip;
}

[edit interfaces xe-0/3/0]
unit 0 {
    family inet {
        filter {
            input counter;
        }
        service {
            input {
                service-set SSET101 service-filter ICMP
```

The `service-filter` configuration has a flexible list of match conditions, from source and destination addresses, to protocol, to ports, to even more granular settings like ICMP options and if a packet is

fragmented. The list is quite extensive, making it a very flexible and powerful feature for choosing what type of traffic you want to get serviced when using the interface style service set.

Now the second option you can employ to control what traffic gets serviced is something similar to a service filter but at the service PIC itself. As you know, you can configure a NAT term under the NAT rule to *not* translate traffic based on the source and destination address and ports. You can also match on the user-define application definitions, such as ICMP or a set of TCP and or UDP ports. The packets still pass through the service PIC when using this method, so when using an interface style service set it is best to employ the service filter at the interface layer to not cause the service PICs CPU to have to process packets you are not going to service. Once again, like the service filter, this method does not drop the packet and it is sent through the MX to the public network:

```
[edit applications application ICMP]
protocol icmp;

[edit services nat rule test_rule term t1]
from {
    applications ICMP;
}
then {
    no-translation;
}
```

These first two types of filtering options may be considered undesirable to certain operator's setups because they can show the public world the private IP address, since NAT has been removed from processing the ICMP packets but the original un-NAT'd packet still passes through the MX. The third option is to use the stateful firewall on the service PIC. By using the stateful firewall you can configure the setup to drop *only* traffic from the private network, and only ICMP or even certain defined ICMP types, if required:

```
[edit applications application ICMP]
protocol icmp;

[edit services stateful-firewall]
rule SFW_Rule {
    match-direction input-output;
    term t1 {
        from {
            source-address {
                10.0.0.0/8;
            }
            applications ICMP;
        }
        then {
```

```
            discard;
        }
    }
    term t2 {
        then {
            accept;
        }
    }
}
```

And you attach the `stateful-firewall-rule`, the one termed `SFW-Rule` right under the `service--set` you are using:

```
[edit services service-set nat44]
max-flows 7500000;
stateful-firewall-rules SFW_Rule;
nat-rules rule2;
interface-service {
    service-interface ms-2/0/0;
}
```

This third option of using the stateful firewall does have the packet land on the service PIC, but since the stateful firewall application runs on the service PIC just like the NAT engine does, it still takes up CPU cycles, although it does allow you to discard the traffic and not forward it.

Setting Up Load Balancing

Load balancing is used when you have the potential for more data or packets per second to be passed through a service PIC than a single service PIC alone can handle. Enabling load balancing allows you to spread the sessions to be NAT'd among multiple service PICs using a hash (#) to steer the data based on the private source IP address being sent from the CPE. This leads to a pretty even distribution among service PICs setup in the load-balanced scenario.

The load balancing setup can also be used as a form of redundancy when you need to make sure your traffic is able to flow to a service PIC if one of the other service PICs in your system goes down for some unexpected reason. It should be noted that load balancing does *not* offer a shared state scenario, meaning if a service PIC does go down the sessions need to be recreated on the other service PICs using a different NAT pool. That's because the NAT'd IP, and potentially also the port traffic, will take a momentary hit as the applications on the client end establish new sessions to the public servers with the new NAT'd IP and port information. Still this scenario should be preferred to having all sessions stop, while going through the MX until the offending service PIC is recovered.

There are two ways to achieve this, starting first with the AMS (aggregate multi-service) interface. This whole book has been about setting up CGNAT service sets to use a single MS interface. *But what if you want to load balance the traffic across many MS interfaces? What if you want to have redundancy so if a service PIC encounters an issue its traffic can be picked up by another healthy service PIC?* These are functions the AMS interface performs.

Let's start by looking at a simple load--balancing scenario. Assume you have a single service set configured on your MX and you have a single MS-MPC card in slot 2. That card has four service PICs so let's employ all four. Add the MS interfaces to the AMS bundle by calling out the MS interfaces as a MAMS (member aggregate multi-service) interface. Your standalone MS interface MS-2/0/0, which is service PIC number 0 in the MS-MPC card that you have been using throughout this book and will be known as `mams-2/0/0` in the AMS bundle. The service PIC in slot 1 of that MS-MPC card will be known as `mams-2/1/0` and so on. So here is what the AMS bundle will look like when configured for `load-balancing` in a next hop style service set:

```
{master}[edit interfaces ams0]
load-balancing-options {
    member-interface mams-2/0/0;
    member-interface mams-2/1/0;
    member-interface mams-2/2/0;
    member-interface mams-2/3/0;
}
unit 1 {
    family inet;
    service-domain inside;
}
unit 2 {
    family inet;
    service-domain outside;
}
```

NOTE When enabling the AMS interface, the AMS MAMS cannot have the same unit configured as their parent MS interface. So if you enable unit 1 and unit 2 under your AMS bundle containing a MAMS, that is also configured as a standalone MS interface, and make sure that the MS interface does not have unit 1 and/or a 2 also enabled, or else you will get a commit error. Here's an error example:

```
{master}[edit]
user@re0# commit
re0:
[edit interfaces ams0]
  'unit'
    ams0 and ms-2/0/0 can't share  unit(1)
error: configuration check-out failed
```

Setting up the AMS on an interface style service set is a bit different. You have to define the hashing used here for load balancing the traffic. When using a next hop style, the ECMP logic used on the PFE will load balance traffic passed into the AMS bundle.

Under the `service-set` for interface style you must set the `load-balancing-option` with a `hash` of `ingress-key source-ip` for source NAT'd traffic and `egress-key destination-ip` for NAT destination traffic. There are other options you can configure and use in regard to the AMS interface and what it should key off of for load balancing. But for the CGNAT use case you want to use only the `ingress-key source-ip` and `egress-key destination-ip` to make sure that all the traffic from a given private IP address is always sent to the same MAMS interface to make sure features like certain ALGS and settings like `address-pooling paired` work as expected by using the same NAT'd IP address for all traffic originated from that private client:

```
[edit services service-set nat44]
nat-rules rule_1;
interface-service {
    service-interface ams0.1;
    load-balancing-options {
        hash-keys {
            ingress-key source-ip;
            egress-key destination-ip;
        }
    }
}
```

NOTE If you do not set these options when using an AMS interface with an interface style service set you will get this commit error:

```
[edit services]
  'service-set nat44'
    service-set policies inconsistent for ams-interface
error: configuration check-out failed
```

NOTE The AMS interface for an interface style service set must use a sub-unit and this cannot be unit 0. So configure any free unit under the AMS and set family inet against the unit. In the preceding configuration example unit 1 was used, as shown with `service-interface ams0.1`.

Along with load balancing, you could also set up redundancy to try and make traffic loss as minimal and as least disruptive as possible if a service PIC goes offline. What will happen on the MX with the next example when traffic is load balanced across mams-2/0/0, mams-2/1/0, and mams-2/2/0? If one of these service PICs has an issue and goes offline mams-2/3/0 will handle all of the traffic that was being steered through the mams interface that just went down. This was done by

calling out mams-2/3/0 as the `preferred-backup` under the `high-availability-options` and `many-to-one` hierarchy:

```
[edit interfaces ams0]
user@re0# show
load-balancing-options {
    member-interface mams-2/0/0;
    member-interface mams-2/1/0;
    member-interface mams-2/2/0;
    member-interface mams-2/3/0;
    high-availability-options {
        many-to-one {
            preferred-backup mams-2/3/0;
        }
    }
}
```

You can verify the current status of the AMS bundle with the following command:

```
user@re0# run show interfaces load-balancing ams0 detail
Load-balancing interfaces detail
Interface        : ams0
  State          : Up
  Last change    : 00:01:48
  Member count   : 4
  HA Model       : Many-to-One
  Members        :
      Interface    Weight   State
      mams-2/0/0   10       Active
      mams-2/1/0   10       Active
      mams-2/2/0   10       Active
      mams-2/3/0   10       Backup
```

Let's' show you how this works. Here is the NAT pool being used by the NAT rule attached to our service set using the AMS bundle:

```
user@re0# run show configuration services nat pool pool_1
address 100.100.0.0/25;
port {
    automatic {
        random-allocation;
    }
}
```

There are three MAMS interfaces active as primary and one as backup. The NAT pool will be split between the three primary MAMS interfaces. You can see below how the pool is broken into individual segments and each segment will be assigned to one MAMS interface:

```
user@re0# run show services nat pool detail
Interface: mams-4/0/0 (ams0), Service set: nat44
  NAT pool: pool_1, Translation type: NAPT-44
    Address range: 100.100.0.1-100.100.0.42
    Configured port range: 0-0
    Port range: 1024-65535, Ports in use: 0, Out of port errors: 0, Max ports used: 0
```

```
    AP-P port allocation errors: 0
    Memory allocation errors: 0

Interface: mams-4/1/0 (ams0), Service set: nat44
  NAT pool: pool_1, Translation type: NAPT-44
    Address range: 100.100.0.43-100.100.0.84
    Configured port range: 0-0
    Port range: 1024-65535, Ports in use: 0, Out of port errors: 0, Max ports used: 0
    AP-P port allocation errors: 0
    Memory allocation errors: 0

Interface: mams-4/2/0 (ams0), Service set: nat44
  NAT pool: pool_1, Translation type: NAPT-44
    Address range: 100.100.0.85-100.100.0.126
    Configured port range: 0-0
    Port range: 1024-65535, Ports in use: 0, Out of port errors: 0, Max ports used: 0
    AP-P port allocation errors: 0
    Memory allocation errors: 0
```

Now if one service PIC goes offline for any unexpected reason, the backup MAMS interface will take over management of that pool. There is no traffic redistribution performed on the existing healthy service PICs – their sessions remained unchanged – although traffic on the service PIC that went down needs to be freshly installed on the backup service PIC that has now taken over. The backup member replaces the failed active member. In terms of traffic takeover, the backup is like a freshly-booted PIC, so all sessions that are now dropped from the previous primary MAMs interface that went down are now steered to the backup MAMS. These sessions have to re-establish, so traffic may be NAT'd' differently, meaning long lived flow and open pinholes will no longer be present and new NAT'd IPs and ports will be assigned to each session. Your SIP and XBOX registration needs to be recreated by the private client end and any long-lived HTTP connections, for example, to banking websites, will be interrupted. Redundancy still has traffic interrupting since the AMS bundle is not stateful redundancy.

But what happens when you are just using the AMS bundle for load-balancing, not redundancy, and a service PIC goes offline? By default, for 120 seconds, all traffic is dropped that is being steered to that PIC just in case it comes back online quickly. Then it will redistribute the traffic to the other PICs that are still up. The IP addresses in the NAT pool being used by the interface that went offline will not be redistributed, just the sessions will. When the service PIC recovers the operator needs to manually tell the system to put that service PIC back into the AMS bundle, like this example that shows re-adding mams-4/1/0 after it went down:

```
user@re0# run request interface load-balancing revert mams-4/1/0
request succeeded
```

If you want the MAMS interface to be able to rejoin right away after it recovers, set the `member-failure-options drop-member-traffic enable-rejoin`, but note this is not the default option, and you must set this as the desired behavior:

```
[edit interfaces ams0]
user@re0# show
load-balancing-options {
    member-interface mams-4/0/0;
    member-interface mams-4/1/0;
    member-interface mams-4/2/0;
    member-interface mams-4/3/0;
    member-failure-options {
        drop-member-traffic {
            enable-rejoin;
        }
    }
}
```

You can also have the sessions be redistributed instantly to the existing MAMS interfaces that are still up once the service PIC in question is seen as being *down* by enabling the `member-failure-options redistribute-all-traffic enable-rejoin`. This changes the default behavior of waiting 120 seconds before redistribution, in case the service PIC recovers quickly, to the traffic being redistributed instantly:

```
{master}[edit interfaces ams0]
user@re0# show
load-balancing-options {
    member-interface mams-4/0/0;
    member-interface mams-4/1/0;
    member-interface mams-4/2/0;
    member-interface mams-4/3/0;
    member-failure-options {
        redistribute-all-traffic {
            enable-rejoin;
        }
    }
}
```

NOTE When using redundancy versus just load balancing, always remember the default behavior is to instantly redistribute the session to the backup MAMS interface. The default behavior acts like this setting:

```
member-failure-options {
redistribute-all-traffic
```

Let's say a second MS-MPC card has been added in FPC slot 4. So let's set up two AMS interfaces and use two service PICs from each MS-MPC card in each AMS bundle. This will allow us to load balance traffic across three of the MAMs interfaces while leaving the fourth as a redundant backup. Each AMS bundle can be added to its own unique service set:

```
[edit interfaces ams0]
user@re0# show
load-balancing-options {
    member-interface mams-2/0/0;
    member-interface mams-2/1/0;
    member-interface mams-4/0/0;
    member-interface mams-4/1/0;
    member-failure-options {
        drop-member-traffic;
    }
    high-availability-options {
        many-to-one {
            preferred-backup ms-4/1/0;
        }
    }
}

[edit interfaces ams1]
user@re0# show
load-balancing-options {
    member-interface mams-2/2/0;
    member-interface mams-2/3/0;
    member-interface mams-4/2/0;
    member-interface mams-4/3/0;
    member-failure-options {
        drop-member-traffic;
    }
    high-availability-options {
        many-to-one {
            preferred-backup ms-4/3/0;
        }
    }
}
```

In general, one final note about using the AMS interfaces is that the number of NAT'd IP addresses must be greater or equal to the number of MAMS interfaces that you have added to the AMS bundle for load balancing. The NAT pools are split among the MAMS members of the AMS bundle but they are split by IP address, meaning each MAMS will have one or more NAT IP addresses from the NAT pool that it is responsible for. The solution does not split a single IP up against multiple MAMS interfaces via a port range or anything else along those lines. You will receive a commit if you try something like setting a NAT pool with only three addresses against four load-balanced MAMS interfaces:

```
user@re0# commit
re1:
error: AMS-NAT constraint check failed for pool pool1 service-set nat44 interface ams0
size of the pool should be >=  AMS configured active member count
[edit services]
  'service-set nat44'
    service-set policies inconsistent for ams-interface
error: configuration check-out failed
```

Beyond the AMS interface and what it brings to your table, you can also load balance traffic to several individual next hop style service sets that are configured to use MS interfaces. First you need to set up the `forwarding-option` hierarchy that allows you to load balance based on the `source-address`:

```
[edit forwarding-options]
enhanced-hash-key {
    services-loadbalancing {
        family inet {
            layer-3-services {
                source-address;
            }
        }
    }
}
```

Then you need to set up the policy that will load balance per session. Do not worry that the key word is `per-packet` this will actually work per session:

```
[edit policy-options policy-statement lb]
term t1 {
    then {
        load-balance per-packet;
    }
}
```

Then in the default routing instance you need to add this just-created policy to the forwarding table:

```
[edit routing-options]
forwarding-table {
    export lb;
```

Using the next hop style (not interface style) for the service sets, let's add our inside service interfaces and the private client-facing ingress interfaces that will receive data to our virtual router called `client`. Here you add a static route for the destination traffic you want to get NAT'd, and the next hop will be the service interfaces you want added to the load balance pool to be chosen from:

```
[edit routing-instances client]
instance-type virtual-router;
interface xe-3/1/0.0;
interface xe-4/2/1.0;
interface ms-2/0/0.1;
interface ms-2/1/0.1;
routing-options {
    static {
        route 70.100.0.0/24 next-hop [ ms-2/0/0.1 ms-2/1/0.1 ];
    }
}
```

Looking at our services setup for this example, you now have two service-sets, each calling a different NAT Rule:

```
[edit services]
service-set nat44_a {
    nat-rules rule1;
    next-hop-service {
        inside-service-interface ms-2/0/0.1;
        outside-service-interface ms-2/0/0/0.1;
    }
}
service-set nat44_b {
    nat-rules rule2;
    next-hop-service {
        inside-service-interface ms-2/0/0.1;
        outside-service-interface ms-2/0/0.1;
    }
}
```

Let's also break out the NAT pool into two pools. Each with half of the public IP addresses used:

```
[edit services]
nat {
    pool nat44_a {
        address-range low 156.100.0.1 high 156.100.0.10;
        port {
            automatic {
                random-allocation;
            }
    }

    pool nat44_b {
        address-range low 156.100.0.11 high 156.100.0.20;
        port {
            automatic {
                random-allocation;
            }
    }
```

Now create our two NAT rules so that each one calls one of our NAT pools:

```
    rule rule1 {
        match-direction input;
        term t1 {
            from {
                source-address {
                    10.0.0.0/8;
                }
            }
            then {
                translated {
                    source-pool nat44_a;
                    translation-type {
                        napt-44;
                    }
```

```
            }
        }
    }
}

rule rule2 {
    match-direction input;
    term t1 {
        from {
            source-address {
                10.0.0.0/8;
            }
        }
        then {
            translated {
                source-pool nat44_b;
                translation-type {
                    napt-44;
                }
            }
        }
    }
}
```

This setup should work for most scenarios, but one of the main weaknesses is whether you have limited public IP addresses that you can add to your NAT pool, and an issue may arise when redundancy has kicked in. Spreading your IP pool out to two pools for load balancing should not be an issue since you are evenly distributing the sessions across the service PICs. But if one service PIC goes down, you are now using only half of all the available IP addresses for your NAT'd sessions until that service PIC recovers, which could mean certain sessions cannot get an available NAT'd IP address if the pool becomes exhausted. Still, this should be preferred to no sessions passing through the box.

When setting up load balancing it will probably be the preferred option to use one service set employing the AMS interface, but you have options here, and the operator can make the right choices for your network needs.

Chapter 5

Example Use Case

Massachusetts Telecom will be our sample use case as they too have read this book and will decide how they are going to configure their MX to fit their NAT needs.

Massachusetts Telecom (MassT) is a mobile and wire line provider that has been very successful. Too successful, in fact, and they are now hitting an IPv4 exhaustion point. They need a NAT solution to help utilize their limited IPv4 resources. But they have other NAT needs. They also host some servers in their network that the owners want to keep secure and hidden from the public. On top of that they also have a DMZ design where traffic routed from a specific network and destined to a cloud data center (managed by MassT) will be NAT'd.

NAT the Data Center

The first NAT setup requirement is for the data center where their partner has requested IP traffic from 100.0.0.0/16 to get NAT'd to 156.0.0.0, due to business requirements for hosting the cloud data center. The IP addresses that will be used in the NAT pool are not part of MassT's routable block: the data center itself sits on a private network. Also, the clients that need to access the data center are from a known defined IP range. So MassT can use a one-to-one NAT setup. This will be a simple case of inline NAT, so a MPC line card will be used, but no service card will be needed. MassT has a MPC card in slot 3 that has four PFEs, so they are good to move forward. The interface xe-3/3/0 is the ingress point where the traffic from 100.0.0.0/16, destined to the data center, will ingress the MX Series.

First enable the inline services for FPC3, PIC 3, and create a `si` interface to use to tie the service-set to this PFE:

```
[edit chassis]
fpc 3 {
    pic 3 {
        inline-services {
            bandwidth 10g;
        }
    }

}

[edit interfaces si-3/3/0]
unit 0 {
    family inet;
}
```

Next, create a `service-set` (called `static_nat`), and attach the si-3/3/0 interface to this service set, creating a NAT rule name called `static_rule`:

```
[edit services]
service-set static_nat {
    }
    nat-rules static_rule;
    interface-service {
        service-interface si-3/3/0.0;
    }
}
```

After the service set is created, define the NAT rule `static_rule`. Note that the plan is to attach this service set to the interface facing the client network 100.0.0./16, whose ingress traffic requires NATing, so set the `match-direction` as `input`.

This rule *must* define the private network under the `source-address` field since it is used for basic-nat44, a one-to-one static NAT type. Then create the NAT pool `static_nat44` and attach it to the NAT rule `static_rule`:

```
nat {
    pool static_nat44 {
        address 156.0.0.0/16;
    }
    rule static_rule {
        match-direction input;
        term t1 {
            from {
                source-address {
                    100.100.0.0/16;
                }
            }
            then {
                translated {
                    source-pool static_nat44;
```

```
                    translation-type {
                        basic-nat44;
                    }
                }
            }
        }
    }
}
```

The final task is to attach the service set to the physical interface facing the private clients:

```
[edit interfaces xe-3/3/0]
description “direct to subscriber CPE”;
unit 0 {
    family inet {
        service {
            input {
                service-set static_nat;
            }
            output {
                service-set static_nat;
            }
        }
        address 139.97.68.42/30;
    }
}
```

Part of this configuration could have gone a different way. The NAT rule could have been configured to be `match-direction output` and attached the interface style `service-set` to the interface facing the cloud data center, where this traffic will egress the MX, instead of using the interface facing the private clients whose ingress traffic needs to access the data center through the MX. For example, it could have been done this way:

```
[edit interfaces xe-3/3/1]
description “direct to data center”;
unit 0 {
    family inet {
        service {
            input {
                service-set static_nat;
            }
            output {
                service-set static_nat;
            }
        }
        address 220.220.220.1/24;
    }
}
nat {
    }
    rule static_rule {
        match-direction output;
```

IPv4 Address Exhaustion

The next item up is to address MassT's IPv4 address exhaustion issue. MassT will implement an interface style service set using a dynamic NAT type with PAT, and set this against their AE0.1 interface, which contains the physical interfaces facing their private network where their paying customers sit and where their traffic ingresses the MX when destined to the Internet. A dynamic NAT with PAT setup requires they use a service card and MassT has purchased two MS-MPC cards for their MX-960. For this setup, MassT will use MS-MPC cards in slot 10 and 11 under a load-balanced AMS interface.

First things first; they set up their AMS bundle and attach the second service PIC from FPC 10 and 11:

```
{master}[edit interfaces ams0]
load-balancing-options {
    member-interface mams-10/1/0;
    member-interface mams-11/1/0;
    member-failure-options {
        redistribute-all-traffic {
            enable-rejoin;
        }
    }
}
unit 1 {
    family inet;
}
```

Next up, they attach a `service-set` called `napt44` under AE0 unit 1, which is where all of their private clients' traffic ingresses the MX:

```
[edit interfaces ae0]
description "direct to subscriber CPEs";
unit 1 {
    family inet {
        service {
            input {
                service-set napt44;
            }
            output {
                service-set napt44;
            }
        }
        address 139.97.68.42/30;
    }
```

Now create the `service-set` called `napt44`, and attach a NAT rule called `internet_rule` and then attach the `ams.1 interface`. Remember, with the AMS bundles, unlike the MS interfaces or the SI interfaces, you have to add the load balancing logic here in the service set. Add the logic to `hash` off of source IP for the `ingress-key` since this is a source NAT type:

```
[edit services]
service-set nat44 {
    nat-rules internet_rule;
     interface-service {
        service-interface ams.1;
        load-balancing-options {
            hash-keys {
                 ingress-key source-ip;
                 egress-key destination-ip;
             }
         }
     }
```

Moving on, let's create that NAT rule called `internet_rule`. Simply add the `translation-type` of `natp44` for dynamic NAT with PAT, and then create a NAT Pool that will attach to this NAT rule. In this case the NAT pool will be called `natpat44`. Assign the NAT'd IP addresses in a round-robin fashion and the ports will be allocated randomly:

```
nat {
    pool natpat44 {
        address 180.100.100.0/25;
        port {
            automatic {
                random-allocation;
            }
        address-allocation round-robin;
    }

    rule internet_rule {
        match-direction input;
        term t1 {
            }
            then {
                translated {
                    source-pool natpat44;
                    translation-type {
                        napt-44;
                    }
                }
            }
        }
    }
```

Creating a Service Filter

Once the interface style service set has been deployed, MassT sees some potential issues with using the interface style for their consumer-facing interfaces. That's because some of their consumers still have public addresses assigned to them, so the private IPv4 scheme of 10.10.0.0/16 being rolled out is not yet out to 100% of its consumers. In addition, there is a service complex within MassT's network that sits

on a private range of 172.100.100.0/24 that the consumers may need to access. Traffic destined to 172.100.100.0/24 does not need to be NAT'd.

So MassT creates a `service-filter` they can apply to the AE0.1 interface. It skips servicing any traffic destined to the 172.100.100.0/24 network first, since MassT sees no reason to take up any cycles on the service PICs with this traffic that does not need to be NAT'd. The next thing the filter does is make sure any traffic coming from 10.10.0.0/16 that has not hit the first term (being destined to 172.100.100.0/24) gets NAT'd. The third and final term then makes sure all other traffic does not get NAT'd, so this should be the traffic generated from the consumers that are still using public addresses:

```
[edit firewall family inet service-filter private_network]
term 1 {
    from {
        destination-address {
            170.100.100.0/24;
        }
    }
    then skip;
}
term 2 {
    from {
        source-address {
            10.10.0.0/16;
        }
    }
    then service;
}
term 3 {
    then skip;
}
```

Then attach this `service-filter` to the AE0.1 interface:

```
[edit interfaces ae0]
description "direct to subscriber CPEs";
unit 1 {
    family inet {
        service {
            input {
                service-set napt44 private_network;
            }
            output {
                service-set napt44
            }
        }
        address 139.97.68.42/30;
    }
```

Changing to Next Hop Style Service Sets

The current setup is working just fine now, but let's say MassT decides to make a drastic change to their configuration and move to the next hop style service sets, allowing them to control which traffic based on destination route actually goes to the service PICs. Though they will lose the ability to use service filters to steer traffic based on the source address (so the Internet routable clients' traffic will get steered to the service PIC even though the MX will not NAT their traffic), MassT makes the trade-off. Why? Because they will add two separate routing instances to help manage and visualize their setup, one routing instance will face the private networks, the other will face the data center, the service complex, and the Internet. First, under their AMS and SI interfaces they create a `service-domain inside` and `service-domain outside` unit:

```
{master}[edit interfaces ams0]
unit 1 {
    family inet;
    service-domain inside;
}
unit 2 {
    family inet;
    service-domain outside;
}

[edit interfaces si-3/0/0]
unit 1 {
    family inet;
    service-domain inside;
}
unit 2 {
    family inet;
    service-domain outside;
}
```

Next they remove the interface style service sets from the interfaces they were attached to:

```
[edit interfaces ae0.1]
unit 0 {
    family inet {
        address 139.97.68.42/30;
    }
}

[edit interfaces xe-3/3/0]
description "direct to subscriber CPE";
unit 0 {
    family inet {
        address 139.97.68.42/30;
    }
}
```

Then they convert the service sets to next hop style service sets by attaching `inside-service interface` and `outside-service interfaceservice`:

```
[edit services]
service-set static_nat {
  nat-rules static_rule;
  next-hop-service {
        inside-service-interface si-3/0/0.1;
        outside-service-interface si-3/0/0.2;
    }
}
service-set nat44 {
    }
    nat-rules internet_rule;
    next-hop-service {
        inside-service-interface ams0.1;
        outside-service-interface ams0.2;
    }
}
```

Finally, they change to the dynamic NAT rule to make sure they don't NAT traffic from anything but the private network of 10.10.0.0/16, adding the `from/source-address` stanza to the NAT rule created earlier called `internet_rule`:

```
    rule internet_rule {
        match-direction input;
        term t1 {
            from {
                source-address {
                    10.10.0.0/16;
                }
            }
            then {
                translated {
                    source-pool natpat44;
                    translation-type {
                        napt-44;
                    }
                }
            }
        }
    }
}
```

To finish off the configuration they create two `routing-instances`, one called `Client`, which will host the client-facing interfaces and the inside service domain service interfaces, and a second, called `Public`, that will host the Internet-facing interfaces and the outside service domain interfaces:

```
[edit routing-instances Client]
instance-type virtual-router;
interface ae0.1
interface xe-3/3/0.0;
interface ams0.1
interface si-3/0/0.1;
routing-options {
    static {
        route 110.100.100.0/24 next-hop si-3/0/0.1;
        route 0.0.0.0/0 next-hop ams0.1;
    }
}

[edit routing-instances Public]
user@re0# show
instance-type virtual-router;
interface xe-3/0/0.0;
interface xe-3/0/2.0;
interface xe-3/1/0.0;
interface ams0.2
interface si-3/0/0.2;
```

Enabling EIM With EIF

Now MassT can finish off their general router configuration and add their routing protocols to VR2 or whatever configuration they feel may be needed. And MassT is in business with their MX960 NATing the desired traffic.

But they are getting complaints from their subscribers that they cannot send data to one another via applications like Apple FaceTime, XBOX live, Skype Torrents, PlayStation peering, and other applications that were previously working fine for MassT's clientele until the move to NAT was done.

MassT realizes that they need to enable EIM with EIF for their NAT rule that is being used to handle traffic destined to the Internet. They need to allow nodes sitting on the outside to reach back into the private side and contact a private CPE based on their NAT'd IP address, which is very important for registration like services and applications as seen in the list of failing solutions. The private client will register itself with a register server using its NAT'd IP address, but the public clients who find out how to reach this private client through its NAT'd address cannot initiate a traffic session to the private client through its NAT'd address since there is not an open pinhole for them to use. By default, the only pinhole open will be the one to the registration server since that is the destination the private client connected to.

So MassT adds Endpoint Independent Mapping and Endpoint Independent Filtering to their rule, like this:

```
rule internet_rule {
  match-direction input;
  term t1 {
      from {
          source-address {
              100.100.0.0/16;
          }
      }
      then {
          translated {
              source-pool natpat44;
              translation-type {
                  napt-44;
              }
             mapping-type endpoint-independent;
             filtering-type {
                  endpoint-independent;
              }
          }
      }
  }
```

Adding ALG

MassT is still not servicing their private end users' needs. They also have consumers who use applications like SIP, RTSP, FTP, and PPTP which are not NAT-friendly. The good folks at MassT have to take these applications into consideration through their MX Series NATing device. The traffic from these protocols requires ALG logic to rewrite the application layer portion of the packet, or to open the needed pinholes.

So MassT sets a rule that NAT's the defined ALG traffic first, and a more generic rule that uses EIM secondly, since EIM is not needed by the ALGs to function because the ALG handles the EIM logic on its own. And MassT will *not* configure EIM for their ALGs.

First they define an `application-set` called `ALG` underneath the `applications` hierarchy:

```
[applications]
        application-set ALG {
            application junos-ftp;
            application junos-pptp;
            application junos-sip;
            application junos-rtsp;

    }
```

Then they create a new NAT term under the NAT rule called `alg` and add the `application-set` under the `from` stanza. This term needs to go before the more generic term called `t1` that's been used, so make sure the term `alg` is first in order:

```
[nat services nat]
     rule internet_rule {
        match-direction input;
                  term alg {
                    from {
               source-address {
                   100.100.0.0/16;
               }
                       application-sets ALG;
                     }
                     then {
                       translated {
                         source-pool napt-44;
translation-type {
          napt-44;
}

          term t1 {
             from {
                 source-address {
                     100.100.0.0/16;
                 }
             }
             then {
                 translated {
                     source-pool natpat44;
                     translation-type {
                         napt-44;
                     }
                    mapping-type endpoint-independent;
                    filtering-type {
                         endpoint-independent;
                     }
                 }
             }
        }
```

Using the Port Block Allocation Feature

MassT now wants to have a way to historically track which private IP addresses and ports were mapped to which public NAT IP addresses and ports from their dynamic pool `natpat44`. But they are concerned about lots of sessions being created and deleted because it could create a lot of overhead on the syslog server. So MassT wants to implement the PBA feature to make it more efficient. They go into the pool `natpat44` and they create port blocks that are 1024 ports in size allowing a max of six blocks to be allocated to each subscriber, like this:

```
pool natpat44 {
        address 180.100.100.0/25;
                  port {
                          range low 1024 high 65535 random-allocation;
                          secured-port-block-allocation block-size 1024 max-blocks-per-
address 6;
                                                                  }
        address-allocation round-robin;
    }
```

But where MassT is thinking that their problem is with their public IP addresses – they just do not have that many of them. So they are going make the pool a bit smaller, like this:

```
pool natpat44 {
        address-range low 180.100.100.1 high 180.100.100.100
        port {
            range low 1024 high 65535 random-allocation;
            secured-port-block-allocation block-size 1024 max-blocks-per-address 6;
        }
```

Was this a wise move? Does MassT have enough IP addresses now and did they have enough to begin with?

Since they are using the Port Block Allocation (PBA) feature you can look at this a bit differently when trying to find the sweet spot on how many IP addresses you should use in the pool. Each service PIC on the MS-MPC card can handle 30 million sessions. Remember the single service-PIC on the MS-MIC card can handle 15 million sessions. So normally, large IP ranges assigned to a NAT pool do not help and end up wasting IP addresses that could be used elsewhere. Think about a /22 address – 1022 host IP addresses that can be used with 64511 non-privileged ports each, so that would be a total of 65,930,242 sessions. Each PIC cannot handle that many sessions so you would, under normal circumstances, be better off using a /23 address. There are times when breaking this rule can be useful, such as when using deterministic NAT or the PBA feature.

Let's look at a PBA example, since MassT just changed over to PBA and they made their pools smaller. *What does this possibly mean for their end users?*

MassT now has 100 IP addresses available with a port block size of 1024 and 64511 ports available per IP. So MassT has 62 port blocks per NAT pool IP address when looking at their block range of 1024 against 64511 ports, so 62 blocks against 100 IP addresses in the pool says that they can have 6200 private subscribers, at most, managed by this pool concurrently.

Overall this setup allows you to have over 16,221,928 sessions, which

is only half of what the service PIC can handle, so there is room to grow, meaning MassT can add more public IP addresses to the pool in the future if they need to. But for now, if they feel their average subscribers typically use less than 1024 active sessions, or if they feel they would have less than 6200 private subscribers coming through the box at peak hours, they are okay.

Now that they are live, before making any further changes to their NAT Pool MassT should, at a minimum, always check to see how many active subscribers they have, who their peak active subscribers are, and how many ports and sessions are actually being used. This will give them an idea of how to proceed. At a minimum, run the `show services sessions count` command or the `show services sessions analysis` command to get that insight.

Even better, MassT should look to monitor the service PIC using SNMP so they have historical data especially around the peak hours.

Using SNMP

Let's look at using SNMP to help manage the MassT setup.

When using SNMP to monitor the CGNAT solution on the service cards start at the hierarchy 1.3.6.1.4.1.2636.3.59, which in the tree is the services MIB root:

```
1.3.6.1.4.1.2636.3.59
{iso(1) identified-organization(3) dod(6) internet(1) private(4) enterprise(1) 2636
jnxMibs(3) jnxSvcsMibRoot(59)}

From there you drop down to 1, which is the NAT services root of the tree. When in doubt on
what OID to grab you can just walk 1.3.6.1.4.1.2636.3.59.1 and down:
1.3.6.1.4.1.2636.3.59.1
{iso(1) identified-organization(3) dod(6) internet(1) private(4) enterprise(1) 2636
jnxMibs(3) jnxSvcsMibRoot(59) jnxNatMIB(1)
```

And under the NAT MIB there are three branches, or *children,* to the NAT hierarchy: Notifications, Objects, and Traps! Notifications are for thresholds. Traps are, well, the TRAPS! Objects are for nearly everything else you as the operator would want to monitor and manage the solution:

```
NAT child OIDs:
jnxNatNotifications (0)
jnxNatObjects (1)
jnxNatTrapsVars (2)

1.3.6.1.4.1.2636.3.59.1.0.1
```

```
{iso(1) identified-organization(3) dod(6) internet(1) private(4) enterprise(1) 2636
jnxMibs(3) jnxSvcsMibRoot(59) jnxNatMIB(1) jnxNatNotifications(0)
jnxNatAddrPoolThresholdStatus(1)}

1.3.6.1.4.1.2636.3.59.1.1
{iso(1) identified-organization(3) dod(6) internet(1) private(4) enterprise(1) 2636
jnxMibs(3) jnxSvcsMibRoot(59) jnxNatMIB(1) jnxNatObjects(1)

Let's see what is under the NAT object child:
NAT Objects child OIDs:
jnxSrcNatStatsTable (1)
jnxNatRuleTable(2)
jnxNatPoolTable (3)

1.3.6.1.4.1.2636.3.59.1.1.1.1
{iso(1) identified-organization(3) dod(6) internet(1) private(4) enterprise(1) 2636
jnxMibs(3) jnxSvcsMibRoot(59) jnxNatMIB(1) jnxNatObjects(1) jnxSrcNatStatsTable(1)
jnxSrcNatStatsEntry(1)}

Src Nat Stats Entry Child OIDs:
jnxNatSrcPoolName(1)
 jnxNatSrcXlatedAddrType(2)
 jnxNatSrcPoolType(3)
 jnxNatSrcNumPortAvail(4)
 jnxNatSrcNumPortInuse(5)
 jnxNatSrcNumAddressAvail(6)
 jnxNatSrcNumAddressInUse(7)
 jnxNatSrcNumSessions(8)

1.3.6.1.4.1.2636.3.59.1.1.2.1
{iso(1) identified-organization(3) dod(6) internet(1) private(4) enterprise(1) 2636
jnxMibs(3) jnxSvcsMibRoot(59) jnxNatMIB(1) jnxNatObjects(1) jnxNatRuleTable(2)
jnxNatRuleEntry(1)}

1.3.6.1.4.1.2636.3.59.1.1.3.1
{iso(1) identified-organization(3) dod(6) internet(1) private(4) enterprise(1) 2636
jnxMibs(3) jnxSvcsMibRoot(59) jnxNatMIB(1) jnxNatObjects(1) jnxNatPoolTable(3)
jnxNatPoolEntry(1)}


1.3.6.1.4.1.2636.3.59.1.2
{iso(1) identified-organization(3) dod(6) internet(1) private(4) enterprise(1) 2636
jnxMibs(3) jnxSvcsMibRoot(59) jnxNatMIB(1) jnxNatTrapVars(2)}

NAT Objects child OIDs:
jnxNatAddrPoolUtil(1)
jnxNatTrapSrcPoolName(2)
```

Okay let's walk the jnxNatMibs to look at some example output:

```
user@re0# run show snmp mib walk ascii 1.3.6.1.4.1.2636.3.59.1
jnxNatSrcXlatedAddrType."pool1" = 1
jnxNatSrcPoolType."pool1" = 2
jnxNatSrcNumPortAvail."pool1" = 7483392
jnxNatSrcNumPortInuse."pool1" = 0
jnxNatSrcNumAddressAvail."pool1" = 116
jnxNatSrcNumAddressInUse."pool1" = 0
jnxNatSrcNumSessions."pool1" = 1203
jnxNatRuleType."rule1:1" = 4
jnxNatRuleTransHits."rule1:1" = 2609398
jnxNatPoolType."pool1" = 4
jnxNatPoolTransHits."pool1" = 2609398
```

Notes:

- This means you have a NAT Pool called pool1 that has 116 IP addresses available to be NAT'd:

```
jnxNatSrcNumAddressAvail."pool1" = 116
```

- This shows you the total number of active sessions using this pool:

```
jnxNatSrcNumSessions."pool1" = 1203
```

- The `jnxNatSrcNumAddressInUse` always remains 0 if natp-44 is used since this counter does not currently support this translation-type.

Chapter 6

Troubleshooting

Useful CLI commands are the key to troubleshooting no matter how the MX Series is configured. This final chapter in this *Day One* book focuses on the commands you need to analyze the session table and CGNAT functionality to verify what is happening on the service PIC. Some of these commands have already been used as you went through this book, but this chapter reviews them all as a quick reference.

NOTE Most of these commands are used for troubleshooting the service PICs for the MS-MPC and MS-MIC service cards and *are not used* for Inline NAT.

shows services nat pool detail

Our first command shows NAT statistics related to IP and port usage for your NAT pools. The output shows you each pool and what service interface it belongs to. You can see what IP and port range the pool is set up to use, and you can also see PBA and address-pooling paired stats, if your pool uses these features:

```
user@re0# run show services nat pool detail
Interface: ms-1/0/0, Service set: ss1
  NAT pool: pool1, Translation type: NAPT-44
    Address range: 188.0.0.0-188.0.0.1
    Configured port range: 0-0
    Port range: 1024-65535, Ports in use: 5675, Out of port errors: 0, Max ports used: 75676
    AP-P port allocation errors: 43226
    Memory allocation errors: 0
```

```
    Max number of port blocks used: 525, Current number of port blocks in use: 444, Port block
allocation errors: 0
    Port blocks limit exceeded errors: 14430521
    Unique pool users: 444
```

You can tell that the NAT pool called `pool1` has not encountered out-of-port error hits due to every port being assigned to an active subscriber. The fact that, `Out of port errors: is 0`, is great news, and shows that at least the pool has always had free ports and has not been 100% starved at any point in time since the service PIC was last rebooted:

```
Port range: 1024-65535, Ports in use: 5675, Out of port errors: 0, Max ports used: 75676
```

Though it should be noted that not all is well with this NAT pool in this example. This pool cannot currently meet all the sessions that it is expected to handle and an investigation is required. Take note that the AP-P out of port errors counter shows 44226:

```
AP-P port allocation errors: 43226
```

This means the address-pooling paired feature is being used and one of the NAT'd IP addresses has been, and still could be, starved of free ports. It would be smart to run this command again several times to see if the AP-P out of ports counter increments to tell you if you currently have a resource issue.

Also, the `Port block limit exceeded error` has quite a few hits, which means the pool is also being used with the Port Block Allocation feature. This counter tells us that at least one subscriber has been assigned their max number port blocks and has used every port in those blocks. Now each new session from one of these subscribers that needs to get NAT'd will get dropped with this error counter incrementing until some of their ports get freed up:

```
Port blocks limit exceeded errors: 14430521
```

You know from the fact that the Port Block Allocation error counter being 0 that the system never actually assigned every port block from this pool, so the issue is localized to only certain subscribers pushing their limit as defined by the PBA settings:

```
Port block allocation errors: 0
```

The NAT pool has not exceeded assigning its total available IP and ports, but at least one subscriber has exceeded the total port block resources allocated to them. This is actually quite typical since there can be some real power users on the network, such as heavy peer-to-peer sharing users. But the thing that should really be of concern with this pool is the AP-P out-of-port errors. This means that at one point

there was enough traffic getting NAT'd that one of the IP addresses in this pool hit its limit and had no further ports or port blocks to allocate. This is a clear indication that something needs to be done. Increasing the number of available IP addresses in the NAT pool would be the first thing to try. If there are no more available IP addresses for you to add to the NAT pool, then you could try lowering the port block size to free up more un-used ports. Many subscribers may not be using all of their available ports in their assigned port blocks, but note that this change could also cause more subscribers to exceed their port block limits.

This bring us to the `show service nat pool brief` output. One thing to like about the brief output is that this command will show you your current port block efficiency usage. If this value is really low when running the command during peak hours, then you may have plenty of room to make your port blocks smaller. If this value is really high, many of your subscribers are being starved of free ports and you may want to find more IP addresses to add to your NAT pool so you can increase your block size to fit your general subscribers' needs:

```
user@re0# run show services nat pool brief
Interface: ms-1/0/0, Service set: ss1
NAT pool         Type                Address                      Port         Ports used
pool1           NAPT-44   188.0.0.0-188.0.0.0           1024-65535    100
Port block type: Secured port block, Port block size: 50, Max port blocks per address: 1,
Active block timeout: 120, Effective port range: 1024-65523, Effective number of port blocks:
1290, Effective number of ports: 64500, Port block efficiency: 0.200000
```

Everything may look good in this output, but if you dig deeper into the data there is an issue: you have a single IP address in this pool and you are assigning very small port blocks, a block size of 50 ports, and one block per private address. You currently have less than one percent of the ports in the total port block in use, and you can see the efficiency is only .2 percent.

But if you run `show services nat pool detail` you will see that only two unique pool users are using the pool right now, and the port blocks limit exceeded errors continue to increment:

```
user@re0# run show services nat pool detail
Interface: ms-1/0/0, Service set: ss1
  NAT pool: pool1, Translation type: NAPT-44
    Address range: 188.0.0.0-188.0.0.0
    Configured port range: 0-0
    Port range: 1024-65535, Ports in use: 100, Out of port errors: 0, Max ports used: 49550
    AP-P port allocation errors: 0
    Memory allocation errors: 0
    Max number of port blocks used: 1000, Current number of port blocks in use: 2, Port block
allocation errors: 0
    Port blocks limit exceeded errors: 14797543
    Unique pool users: 2
```

Based on this (very simple) example, just because the pool looks like it might be okay, does not mean it is. Yes, there are free ports but the two subscribers using the NAT pool at this time have exceeded their limits. This begs the question of whether assigning a single port block to each subscriber whose block size is 50 ports will be enough to satisfy your end users or if you need to increase this block size.

show services sessions

The `show services sessions count` command is a simple CLI command that you can run to see the total number of current sessions being handled by the MX. In this case, there is a standalone MS interface, `ms-1/0/0`, handling service set `ss1` and then there is an AMS interface using all four service PICs from the MS-MPC card in FPC slot 4 handling service set `nat44`. You can use this command to see the load balancing of the traffic through the AMS bundle:

```
user@re0# run show services sessions count
Interface   Service set                         Sessions count
ms-1/0/0    ss1                                 1027
mams-4/0/0  nat44                               30245
mams-4/1/0  nat44                               30889
mams-4/2/0  nat44                               31034
mams-4/3/0  nat44                               30333
```

You can use the same command, removing the `count` parameter, to check the creation of the pre-NAT, post NAT, and non-NAT'd sessions within the service interfaces. This means you get to view the details of each session that is currently in memory on the service PIC:

```
user@re0# run show services sessions
ms-1/0/0

Service Set: ss1, Session: 2028242634, ALG: none, Flags: 0x200000, IP Action: no, Offload:
no, Asymmetric: no
UDP       10.202.0.12:53852  ->   77.76.75.144:23308  Forward  I            87
UDP      77.76.75.144:23308  ->      188.0.0.0:47837  Forward  O           1020

Service Set: ss1, Session: 1994621085, ALG: none, Flags: 0x200000, IP Action: no, Offload:
no, Asymmetric: no
UDP       10.202.0.12:50711  ->   77.76.75.123:9453   Forward  I            1322
UDP      77.76.75.123:9453   ->      188.0.0.0:47833  Forward  O           1332

Service Set: ss1, Session: 2095286559, ALG: none, Flags: 0x200000, IP Action: no, Offload:
no, Asymmetric: no
TCP       10.202.0.12:19019  ->   77.76.75.139:15862  Forward  I            2
TCP      77.76.75.139:15862  ->      188.0.0.0:47836  Forward  O           1
```

The MX automatically creates both input and output direction session entries after the first packet is received and the session is created. This

output will show you the protocol used such as ESP, UDP, TCP, ICMP, etc. It will also show you NAT'd and non-NAT'd traffic. If any traffic flows into the service PIC but does not get NAT'd based on the NAT rules term logic, the session still passes through the session table and gets created.

For source NAT translation types, you will see traffic entering the service PIC from the private source IP address. It will show up as `Forward I`. Then the return traffic will show up in the output of this command as being destined to the NAT'd address. This will show up as `Forward O`.

Forward and Drop sessions are the two states that can be seen when running the `show services sessions` command.

- A Forward state just means the packet is passed through the session table when the packet is NAT'd or even when it is not NAT'd.
- A Drop state just means it drops any packets in that session. A drop can occur when using a nat44 setup, for example, and traffic from the public network is destined to a NAT'd address and port but there isn't a matching session already created for that traffic. Or it could be protocol errors like a TCP session being started by a FIN and not a SYN. It can even occur when you have run out of available NAT IP addresses or ports, or port blocks in one of your NAT pools.

show services sessions extensive

When using the `show services sessions` command with the extensive option you can see some addition data, such as the inactivity timeout values for each session, and some info around the NAT Mapping:

```
Service Set: ss1, Session: 1894010314, ALG: none, Flags: 0x200000, IP Action: no, Offload:
no, Asymmetric: no
NAT PLugin Data:
  NAT Action:   Translation Type - NAPT-44
    NAT source        10.202.0.12:26390   ->      188.0.0.0:47837
UDP       10.202.0.12:26390  ->    77.76.75.87:1997   Forward  I            111
  Byte count: 92
  Flow role: Initiator, Timeout: 12
UDP       77.76.75.87:1997   ->      188.0.0.0:47837  Forward  O            0
  Byte count: 0
  Flow role: Responder, Timeout: 12

Service Set: ss1, Session: 1927517352, ALG: none, Flags: 0x200000, IP Action: no, Offload:
no, Asymmetric: no
NAT PLugin Data:
  NAT Action:   Translation Type - NAPT-44
```

```
   NAT source        10.202.0.12:12730   ->      188.0.0.0:47833
TCP       10.202.0.12:12730  ->   77.76.75.134:13467  Forward  I            1232
  Byte count: 92
  Flow role: Initiator, Timeout: 30
TCP      77.76.75.134:13467  ->      188.0.0.0:47833  Forward  O            1833
  Byte count: 0
  Flow role: Responder, Timeout: 30

Service Set: ss1, Session: 1893964435, ALG: none, Flags: 0x200000, IP Action: no, Offload:
no, Asymmetric: no
NAT PLugin Data:
  NAT Action:   Translation Type - NAPT-44
    NAT source        10.202.0.12:32780   ->      188.0.0.0:47838
TCP       10.202.0.12:32780  ->    77.76.75.94:41776  Forward  I            12
  Byte count: 92
  Flow role: Initiator, Timeout: 27
TCP       77.76.75.94:41776  ->      188.0.0.0:47838  Forward  O            13
  Byte count: 0
  Flow role: Responder, Timeout: 27
```

Show services service-sets < ... >

The `show services service-sets` command focuses on the health of the overall service set from a system point of view. It shows you such things as flow/session drops caused by CPU or memory limits being reached, or if the configured sessions limits are set and have been reached. The command will show the drops per service set and the service interface being used by that service set, including the individual MAMS that are part of any AMS bundles you are using:

```
lab@jtac_setup# run show services service-sets statistics packet-drops
                                 Cpu limit      Memory limit    Flow limit
Interface      Service Set       Drops          Drops           Drops
ms-1/0/0       natp44               0              0              0
ms-1/0/0       twice-nat            0              0              0
mams-4/0/0     nat                  0              0              0
mams-4/1/0     nat                  0              0              0
mams-4/2/0     nat                  0              0              0
mams-4/3/0     nat                  0              0              0
ms-5/0/0       cust_nat             0              0              0
```

Here is an additional series of commands to run against the service sets to gather some very useful data on how the service set is currently behaving:

```
user@re0# run show services service-sets summary
                    Service sets                                        CPU
Interface           configured          Bytes used   Policy bytes used  UtilizationU
ms-1/0/0         3  1219043875      ( 4.16%)      4103480 ( 0.76%)      35.22 %
ms-4/0/0         2  1219016539      ( 4.16%)      3657976 ( 0.68%)      0.94 %
ms-4/1/0         2  1219016667      ( 4.16%)      3618448 ( 0.67%)      0.95 %
ms-4/2/0         2  1219016667      ( 4.16%)      3618448 ( 0.67%)      0.94 %
```

```
ms-4/3/0           2  1219016667      ( 4.16%)      3618448 ( 0.67%)      0.95 %
ms-5/0/0           2  1219044291      ( 4.16%)    3010960 ( 0.56%)      35.21 %


user@re0# run show services service-sets cpu-usage
                                                                          CPU
Interface   Service Set(or system category)                          Utilization
ms-1/0/0    ipsec_ss                                                      0.00 %
ms-1/0/0    ss1                                                           0.00 %
ms-1/0/0    System                                                        0.70 %
ms-1/0/0    Idle                                                         64.77 %
ms-1/0/0    Receive                                                       0.00 %
ms-1/0/0    Transmit                                                      0.00 %
mams-4/0/0  nat44                                                         0.00 %
mams-4/0/0  System                                                        0.95 %
mams-4/0/0  Idle                                                         99.04 %
mams-4/0/0  Receive                                                       0.00 %
mams-4/0/0  Transmit                                                      0.00 %
mams-4/1/0  nat44                                                         0.00 %
mams-4/1/0  System                                                        0.94 %
mams-4/1/0  Idle                                                         99.05 %
```

If the CPU is hitting 90%, the service PIC is too busy. Most likely it is handling too many packets per second and there is just too much traffic to process. You are going to want to monitor the interface to see what the traffic is, in Packets Per Second (PPS), for Input and Output. For example, `monitor interface ms-5/0/0`, and then look to see if you can load balance the traffic among additional service PICs.

Note if you are viewing a MS or AMS interface using the `show interfaces ms-5/1/0 extensive` command and its input or output drops counter has any hits, and is increasing, then you will want to check the CPU usage:

```
user@re0> show interfaces ms-5/1/0 extensive
Physical interface: ms-5/1/0, Enabled, Physical link is Up
  Interface index: 180, SNMP ifIndex: 565, Generation: 183
  Type: Adaptive-Services, Link-level type: Adaptive-Services, MTU: 9192, Clocking:
Unspecified, Speed: 40000mbps
  Device flags   : Present Running
  Interface flags: Point-To-Point SNMP-Traps
  Link type      : Full-Duplex
  Link flags     : None
  Physical info  : Unspecified
  Hold-times     : Up 0 ms, Down 0 ms
  Damping        : half-life: 0 sec, max-
suppress: 0 sec, reuse: 0, suppress: 0, state: unsuppressed
  Current address: Unspecified, Hardware address: Unspecified
  Alternate link address: Unspecified
  Last flapped   : 2016-11-30 03:35:14 PST (04:20:44 ago)
  Statistics last cleared: Never
  Traffic statistics:
   Input  bytes  :           4974889433           1570882800 bps
```

```
   Output bytes  :            3868098872            1221415968 bps
   Input  packets:               7594338                299753 pps
   Output packets:               2531513                 99919 pps
   IPv6 transit statistics:
    Input  bytes  :                    0
    Output bytes  :                    0
    Input  packets:                    0
    Output packets:                    0
  Input errors:
    Errors: 0, Drops: 0, Framing errors: 0, Runts: 0, Giants: 0, Policed discards: 0,
Resource errors: 0
  Output errors:
   Carrier transitions: 2, Errors: 0, Drops: 0, MTU errors: 0, Resource errors: 0
```

You never want to see the memory being used by the service interface being in the orange or the red memory zone when using the `show services service-sets memory-usage zone` command. If it is in either zone the service set will drop packets destined for new sessions until the memory drops back into the yellow warning zone. The green zone is great, and it is what's desired. The yellow zone will not change how the service PIC functions but it does indicate that you may have many sessions piling up. Think of the yellow zone as a warning that memory could soon be a problem. If you are hitting the yellow, orange, or red zones you may want to make sure your inactivity timeouts and mapping-refresh timeouts are not holding expired sessions in memory for too long. If the zone hits `Orange` or `Red` when running this command, you would start to see new sessions tied to your busiest service sets hosted on that service interface start to get into a `DROP` state:

```
user@re0# run show services service-sets memory-usage zone
Interface   Memory zone
ms-1/0/0    Orange
mams-4/0/0  Green
mams-4/1/0  Green
mams-4/2/0  Green
mams-4/3/0  Green
ms-5/0/0    Yellow
```

You never want to be in the orange or red zone. If you are, the box is either configured poorly in regard to settings like long inactivity timeouts based on the volume of sessions you are receiving, or else the service PIC is just handling too many sessions and you need to better distribute the load among other service PICs.

The last `services service-sets statistics` command in this chapter is the `show services service-sets statistics integrity-drops`. This command will tell you if the session table dropped any packets due to a packet error. It is more of a corner case command, required when you are looking to see if the session table is dropping packets because the packets themselves are incorrect somehow:

```
user@re0# run show services service-sets statistics integrity-drops
Interface: ms-1/0/0
  Service set: ss1
    Errors:
      IP: 0, TCP: 0
      UDP: 0, ICMP: 0
    IP errors:
      IP packet length inconsistencies: 0
      Illegal source address: 0
      Illegal destination address: 0
      TTL zero errors: 0, Illegal IP protocol number (0 or 255): 0
      Land attack: 0
      Non-IPv4 packets: 0
      Non-IPv6 packets: 0
      Bad checksum: 0
      Illegal IP fragment length: 0
      IP fragment overlap: 0
      IP fragment reassembly timeout: 0
      IP fragment limit exceeded: 0
      Unknown: 0
    TCP errors:
      TCP header length inconsistencies: 0
      Source or destination port number is zero: 0
      Illegal sequence number and flags combinations: 0
    UDP errors:
      IP data length less than minimum UDP header length (8 bytes): 0
      Source or destination port number is zero: 0
    ICMP errors:
      IP data length less than minimum ICMP header length (8 bytes): 0
      ICMP error length inconsistencies: 0
```

show services sessions analysis

The `show services sessions analysis` command will show you current active session numbers along with historical peak numbers. This is a good command in general, but it is very important to run if you have ever hit any memory zone issues and you want to see how many sessions were actually on the box for a peak value:

```
user@re0# run show services sessions analysis
  Services PIC Name:    ms-1/0/0

Session Analysis Statistics:

  Total sessions Active                  :7482942
  Total TCP Sessions Active              :0
      Tcp sessions from gate             :0
      Tunneled TCP sessions              :0
      Regular TCP sessions               :6482942
      IPv4 active Session                :6482942
      IPv6 active Session                :0
  Total UDP sessions Active              :1000000
```

```
    UDP sessions from gate              :0
    Tunneled UDP sessions               :0
    Regular UDP sessions                :1000000
    IPv4 active Session                 :1000000
    IPv6 active Session                 :0
  Total Other sessions Active           :0
    IPv4 active Session                 :0
    IPv6 active Session                 :0
  Created sessions per Second           :24713
  Deleted sessions per Second           :24736
  Peak Total sessions Active            :9231813
  Peak Total TCP sessions Active        :8231812
  Peak Total UDP sessions Active        :1000000
  Peak Total Other sessions Active      :0
  Peak Created Sessions per Second      :24774
  Peak Deleted Sessions per Second      :44656
  Packets received                      :34008777
  Packets transmitted                   :25007846
  Slow path forward                     :58900118
  Slow path discard                     :9000444
```

shows services nat mappings

The `show services nat mapping summary` command shows you how many private IP to public IP NAT mappings you have per service interface for the address-pooling paired (APP) and EIM features. It also tells you how many of these EIMs have an EIF assigned to them. Note the MAMS interfaces display as just MS interfaces since this output displays the total mapping for an individual service PIC and a MS interface that is part of an AMS bundle can also be used by another service set as a standalone interface:

```
user@re0# run show services nat mappings summary

Service Interface:                                  ms-1/0/0
Total number of address mappings:                   100
Total number of endpoint independent port mappings: 100
Total number of endpoint independent filters:       0

Service Interface:                                  ms-4/0/0
Total number of address mappings:                   0
Total number of endpoint independent port mappings: 0
Total number of endpoint independent filters:       0

Service Interface:                                  ms-4/1/0
Total number of address mappings:                   0
Total number of endpoint independent port mappings: 0
Total number of endpoint independent filters:       0

Service Interface:                                  ms-4/2/0
Total number of address mappings:                   0
Total number of endpoint independent port mappings: 0
```

```
Total number of endpoint independent filters:              0

Service Interface:                                         ms-4/3/0
Total number of address mappings:                          0
Total number of endpoint independent port mappings:        0
Total number of endpoint independent filters:              0

Service Interface:                                         ms-5/0/0
Total number of address mappings:                          98432
Total number of endpoint independent port mappings:        0
Total number of endpoint independent filters:              0
```

You can also run the `show services nat mappings detail` command to see each private to public mapping. It will display the mapping if the session is active, or if the session has timed out it will show you the timeout value left before the EIM or APP mapping is removed. Remember from earlier in the book the EIM mappings start their timeout countdown only after the session has timed out and the and APP starts its timeout only after all sessions mapped to it are timed out:

```
{master}[edit services nat rule rule1]
user@re0# run show services nat mappings detail
Interface: ms-1/0/0, Service set: ss1

NAT pool: pool1
Mapping          : 10.10.10.12     :50000  --> 150.100.100.7   :10926
Session Count    :     1
Mapping State    : Active
Mapping          : 10.10.10.10     :50000  --> 150.100.100.6   :10926
Session Count    :     1
Mapping State    : Active
Mapping          : 212.27.42.153   :52062  --> 150.100.100.5   :46665
Session Count    :     0
Mapping State    : Timeout (223s)
```

show services nat statistics

The `show services nat statistics` command presents the operator with a lot of details on what is occurring with the NAT mappings. This command is most likely going to be more useful when an issue is occurring so it can be shared with JTAC:

```
user@re0# run show services nat statistics interface ms-1/0/0
Interface: ms-1/0/0

Session statistics

Session statistics
    Total Session Interest events                          :123931
    Total Session Create events                            :123931
    Total Session Destroy events                           :123931
    Total Session Pub Req events                           :0
```

```
    Total Session Accepts                                :123929
    Total Session Discards                               :0
    Total Session Ignores                                :2
    Total Session Time events                            :0
    Session interest thru pub event                      :0
    ALG Session interest                                 :0
    ALG Session Create                                   :0
    Packet  Dst in NAT route                             :0
    Packet drop in backup state                          :0
    Session Ext Alloc Failures                           :0
    Session Ext Set Failures                             :0
    Session Created for EIF                              :0
    Session Created for EIM                              :0
    NAT rule lookup failures                             :2
    Pool session count update failed on create           :0
    Pool session count update failed on close            :4

NAT Allocation statistics
    NAT allocation Successes                             :123929
    NAT allocation Failures                              :0
    NAT Free Successes                                   :123925
    NAT Free Failures                                    :4
    NAT EIM mapping reused                               :0
    NAT EIM mapping allocation failures                  :0
    NAT EIM mapping Duplicate entry                      :0
    NAT EIM mapping create failed                        :0
    NAT EIM mapping Created                              :0
    NAT EIM mapping Updated                              :0
    NAT EIF mapping Free                                 :0
    NAT EIM mapping Free                                 :0
    NAT EIM waiting for init                             :0
    NAT EIM waiting for init failed                      :0
    NAT EIM lookup and hold success                      :0
    NAT EIM lookup entry in timeout                      :0
    NAT EIM lookup timer cleared for timeout entry       :0
    NAT EIM lookup timeout entry without timer           :0
    NAT EIM release without entry                        :0
    NAT EIM release entry in timeout                     :0
    NAT EIM release race                                 :0
    NAT EIM release set entry for timeout                :0
    NAT EIM timer entry refreshed                        :0
    NAT EIM timer invalid timer started                  :0
    NAT EIM timer entry freed                            :0
    NAT EIM timer entry updated                          :0
    NAT EIM entry drained                                :0

Packet statistics
    Total Packets Processed                              :87716758
    Total Packets Forwarded                              :87716758
    Total Packets Discarded                              :0
    Total Packets Translated                             :87716756
    Total Packets Restored                               :2

Translation statistics
```

```
    Src  IPv4   Translations                          :87716756
    Src  IPv4   Restorations                         :0
    Dst  IPv4   Translations                         :0
    Dst  IPv4   Restorations                         :2
    Src  IPv6   Translations                         :0
    Src  IPv6   Restorations                         :0
    Dst  IPv6   Translations                         :0
    Dst  IPv6   Restorations                         :0
    Src  Port   Translations                         :87716756
    Src  Port   Restorations                         :0
    Dst  Port   Translations                         :0
    Dst  Port   Restorations                         :2
    ICMP ID     Translations                         :0
    ICMP ID     Restorations                         :0
    ICMP Error  Translations                         :0
    ICMP Drops                                       :0
    ICMP Allocation Failure                          :0
    TCP  Port   Translations                         :0
    TCP  Port   Restorations                         :0
    UDP  Port   Translations                         :87716756
    UDP  Port   Restorations                         :2
    NAT Unexpected Protocol With Port Xlation        :0
    GRE  CallID Translations                         :0
    GRE  CallID Restorations                         :0
    GRE  Wrong protocol value                        :0
    SRC IP restored in ICMP Error                    :0
    DST IP restored in ICMP Error                    :0
    SRC IP translated in ICMP Error                  :0
    DST IP translated in ICMP Error                  :0
    New SRC IP translated in ICMP Error              :0
    Inner SRC IP restored in ICMP Error              :0
    Inner SRC port restored in ICMP Error            :0
    Inner DST port restored in ICMP Error            :0
    Inner DST IP restored in ICMP Error              :0
    Inner SRC IP translated in ICMP Error            :0
    Inner SRC port translated in ICMP Error          :0
    Inner DST port translated in ICMP Error          :0
    Inner DST IP translated in ICMP Error            :0

Misc Errors
    NAT error - no policy                            :0
    NAT error - IP version                           :0
    NAT error - xlate free called with null ext      :0
    NAT error - ext free failed                      :0
    NAT error - policy add failed                    :0
    NAT error - policy delete failed                 :0
    NAT error - prefix filter allocation failed      :0
    NAT error - prefix filter name failed            :0
    NAT error - prefix list create failed            :0
    NAT error - prefix filter tree add failed        :0

Misc Counters
```

```
    NAT prefix filter created                                :0
    NAT prefix filter changed                                :0
    NAT prefix filter control free                           :0
    NAT prefix filter match                                  :0
    NAT prefix filter no match                               :0
    NAT prefix filter mapping add                            :0
    NAT prefix filter mapping remove                         :0
    NAT prefix filter mapping free                           :0
    NAT prefix filter unsupported IP version                 :0
    NAT unsupported layer-4 header for port translation      :0
    NAT unsupported icmp id for port translation             :0

NAT64 Counters
    NAT64 - IP options drop                                  :0
    NAT64 - UDP checksum zero drop                           :0
    NAT64 - Unsupported ICMP type drop                       :0
    NAT64 - Unsupported ICMP code drop                       :0
    NAT64 - Unsupported header drop                          :0
    NAT64 - Unsupported L4 drop                              :0
    NAT64 - MTU exceeded                                     :0
    NAT64 - dfbit set                                        :0
    NAT64 - Unsupported ICMP error                           :0
    NAT64 error - mapping ipv4 source                        :0
    NAT64 error - mapping ipv6 destination                   :0
    NAT64 error - MTU exceed build                           :0
    NAT64 error - TTL exceed build                           :0
    NAT64 error - MTU exceed send                            :0
    NAT64 error - TTL exceed send                            :0

NAT Subscriber extension counters
    NAT subscriber extension allocated                       :3000004
    NAT subscriber extension invalid parameters              :0
    NAT subscriber extension no memory                       :0
    NAT subscriber extension freed                           :0
    NAT subscriber extension is null                         :0
    NAT subscriber extension is invalid                      :0
    NAT subscriber extension link successful                 :2
    NAT subscriber extension link already exists             :123927
    NAT subscriber extension link failed                     :0
    NAT subscriber extension link unknown return value       :0
    NAT subscriber extension unlink successful               :2
    NAT subscriber extension unlink fail                     :0
    NAT subscriber extension unlink on busy                  :0
    NAT subscriber extension resource in use                 :0
    NAT subscriber extension svc set is not active           :0
    NAT subscriber extension svc set is null                 :0
    NAT subscriber extension timer start successful          :4
    NAT subscriber extension timer start failed              :0
    NAT subscriber extension delay timer start successful    :2
    NAT subscriber extension delay timer start failed        :0
    NAT subscriber extension reuse from timer                :1
    NAT subscriber extension timer callback called           :4
    NAT subscriber extension refcount decrement failed       :0
    NAT subscriber extension subscriber reset failed         :0
```

```
    NAT subscriber extension session count update ignored   :0
    NAT subscriber extension incorrect state                :0
    NAT subscriber extension unknown error unlinking        :0
    NAT subscriber extension queue inconsistent             :0
    NAT subscriber extension return to prealloc queue error :0
    NAT subscriber extension dec invalid session count      :0
    NAT subscriber extension dec invalid eim count          :0
    NAT subscriber extension ports in use error             :0
    NAT subscriber extension error while setting state      :0
    NAT subscriber extension nat extension is missing       :0
    NAT subscriber extension unexpected eim refcount        :0

NAT jflow-log counters
    NAT jflow-log error - session extension get fail        :0
    NAT jflow-log error - memory allocation fail            :0
    NAT jflow-log - memory allocation success               :0
    NAT jflow-log - memory free success                     :0
    NAT jflow-log error - memory free fail null record      :0
    NAT jflow-log error - memory free fail null data        :0
    NAT jflow-log error - invalid nat translation type      :0
    NAT jflow-log - memory free success fail queuing        :0
    NAT jflow-log - invalid input arguments                 :0
    NAT jflow-log - invalid allocation error type           :0
    NAT jflow-log - rate limit fail to get pool name        :0
    NAT jflow-log - rate limit fail to get nat pool         :0
    NAT jflow-log - rate limit fail to get pool given id    :0
    NAT jflow-log - rate limit fail to get service set      :0
    NAT jflow-log - rate limit fail invalid current time    :0
```

show interfaces load-balancing

You can make sure your service PICs are okay when using an AMS interface bundle by running thee `show interfaces load-balancing detail` command.

The following output of the command shows you a load-balanced AMS bundle using all four PICs within the MS-MPC in FPC slot 4. All of the PICs are in an active state so they all look good from the AMS point of view and should receive traffic from the MX for processing:

```
user@re0# run show interfaces load-balancing detail
Load-balancing interfaces detail
Interface        : ams0
  State          : Up
  Last change    : 4d 08:49
  Member count   : 4
  HA Model       : None
  Members        :
      Interface    Weight   State
      mams-4/0/0   10       Active
      mams-4/1/0   10       Active
      mams-4/2/0   10       Active
```

```
      mams-4/3/0   10       Active
```

Thee next output of the command shows you a load-balanced AMS bundle using all four PICs within the MS-MPC in FPC slot 4 *and* slot 5 with the service PIC in FPC slot 5, pic slot 4, being designated the backup to the other seven service PICs:

```
user@re0# run show interfaces load-balancing detail
Load-balancing interfaces detail
Interface        : ams0
  State          : Up
  Last change    : 00:00:09
  Member count   : 8
  HA Model       : Many-to-One
  Members        :
      Interface    Weight   State
      mams-4/0/0   10       Active
      mams-4/1/0   10       Active
      mams-4/2/0   10       Active
      mams-4/3/0   10       Active
      mams-5/0/0   10       Active
      mams-5/1/0   10       Active
      mams-5/2/0   10       Active
      mams-5/3/0   10       Backup
```

The other states that a service PIC can be in, aside from Active and Backup, are Discard and Inactive. Discard states occur when an active member fails with no backup available and the `member-failure-option` is set to `drop-member-traffic`, and `rejoin-timeout` is set. In this case, the member moves from Active to Discard until the rejoin time is reached. In the time that it takes to happen, all traffic toward that MAMs interface will get dropped. If the MAMS cannot rejoin, it moves to the Inactive state. Inactive state is a state that means the MAMS interface is removed and no traffic will get steered towards it. When something happens to a MAMS and the setting or system require it to be removed it goes into this state and traffic is distributed to the other MAMS in the AMS bundle.

Inline NAT

To wrap up this final chapter, let's look at the system when Inline NAT is used. Remember with Inline NAT there is no service card being used. You will never have a session table without a service card, so be aware when building your MX system that without the service card you cannot see the NAT'd sessions, because in reality with Inline NAT you are just changing the header of transient traffic based on the PFE's programming.

The following couple of commands can help you tell if NAT is even

occurring, if there are any errors, and what POOLs are being used.

show services inline nat statistics

```
user@re0#run show services inline nat statistics

 Service PIC Name                                      si-2/0/0

 Control Plane Statistics
     Received IPv4 packets                                 0
     ICMPv4 error packets pass through                     0
     ICMPv4 error packets locally generate                 0
     Dropped IPv4 packets                                  0
     Received IPv6 packets                                 0
     ICMPv6 error packets pass through for NPTv6           0
     ICMPv6 error packets locally generated for NPTv6      0
     Dropped IPv6 packets                                  0

 Data Plane Statistics           Packets                  Bytes
     IPv4 NATed packets            17691                20521614
     IPv4 deNATed packets          12311                17764566
     IPv4 error packets            0                        0
     IPv4 skipped packets          0                        0
     IPv6 NATed packets            0                        0
     IPv6 deNATed packets          0                        0
     IPv6 error packets            0                        0
     IPv6 skipped packets          0                        0
```

show services inline nat pool

```
user@re0# run show services inline nat pool
Interface: si-2/0/0, Service set: NAT_SS1
  NAT pool: NAT_POOL1, Translation type: BASIC NAT44
    Address range: 19.200.0.1-19.200.100.1
    NATed packets: 213, deNATed packets: 209, Errors: 0, Skipped packets: 0
```

CGNAT Up and Running on the MX Series Summary

When designing your CGNAT setup, the main things you should think about are how many NAT'd IP addresses and port combinations you have. This is especially true when setting up a dynamic NAT type. You should also consider the number of *mappings* and potential sessions, and then do the math. Do not design a setup where you have two public IP addresses each having 64,511 ports available that need to handle two million concurrent sessions at one time. Some things are just not possible.

Also, do not waste valuable public IP addresses in your NAT pools. Let's say (unrealistically for the sake of this example) you have 327,600 potential NAT'd IP addresses for use and you want to use deterministic NAT as your NAT translation type. Deterministic NAT does not use ports 0-1023, so each NAT'd IP can potentially use 64,511 ports. If you set the `deterministic-port-block-allocation block-size` to 256, or 64,511 ports divided by 256 blocks equals 251 port block assignments per NAT'd IP. So that means 327,600 potential NAT'd IPs with 251unique port blocks, for each one is a total of 82,227,600 private subscriber mappings and 21,133,803,600 potential sessions. Very few operators have 82 million subscribers for their whole network. If these 327,600 potential NAT'd IP addresses are valuable Internet routable IP addresses, you may want to tie up far, far fewer of them with your MX Series CGNAT solution.

Remember to think about the features you are enabling. ALGs may very well be required, but enable just the ones you need since you do not want to spend CPU cycles on the service PICs handling NAT traffic through an ALG that does not require it.

EIM with EIF is a very good feature that is needed for certain applications to work as expected when NAT is added, but always remember to be aware of the risk of traffic from the outside opening up lots of sessions. Monitoring the solution is one of the best things you can do. Looking at the number of active sessions, the memory, the CPU load on the service PIC, and the packet per second through the service interfaces are all good data points to use to make sure events are not causing the solution to be pushed beyond the boundaries.

Just do the math, design accordingly, monitor the solution when in production, and the MX Series can deliver a scalable and powerful CGNAT service for you!