Juniper Networking Technologies

# DAY ONE: DEPLOYING CGNAT ON THE MX SERIES

Imagine you had your own personal JTAC engineer for a day so you could learn how to configure the MX Series to truly fit your operational NATing needs. Whether it's using inline NAT or a MS-DPC Card, it's all here for you to test in your lab.

By Joseph Naughton

# DAY ONE: DEPLOYING CGNAT 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.
- How to monitor and manage the MX Series when it is configured in a CGNAT solution.
- How to 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 978-9367798577

9 789367 798577 52000

JUNIPER NETWORKS

# Day One: Deploying CGNAT on the MX Series

By Joseph Naughton



Imagine you had your own personal JTAC engineer for a day so you could learn how to configure the MX Series to truly fit your operational NATing needs. Whether it's using inline NAT or a MS-DPC Card, it's all here for you to test in your lab.

JUNIPER NETWORKS

© 2014 by Juniper Networks, Inc. All rights reserved. Juniper Networks, Junos, Steel-Belted Radius, NetScreen, and ScreenOS are registered trademarks of Juniper Networks, Inc. in the United States and other countries. The Juniper Networks Logo, the Junos logo, and JunosE 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: Dhvani Vora, Prasad Yadati, Simon Zhong
Editor in Chief: Patrick Ames
Copyeditor and Proofer: Nancy Koerbel
Illustrations: Karen Joice
J-Net Community Manager: Julie Wider

ISBN: 978-1-936779-85-7 (print)
Printed in the USA by Vervante Corporation.

ISBN: 978-1-936779-86-4 (ebook)

Version History: v1, June 2014
2 3 4 5 6 7 8 9 10

**About the Author**
Joseph Naughton has seventeen years of 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.

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

## Welcome to Day One

This book is part of a growing library of *Day One* books, 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 larger and longer suite of *This Week* books, whose concepts and test bed examples are more similar to a weeklong seminar.

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

## Audience

This book is intended for network administrators and common network deployment scenarios, as well as to provide brief background information needed to understand and deploy these solutions in your own environment.

## What You Need to Know Before Reading This Book

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

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

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

## What You Will Learn by Reading This Book

- Understand the hardware needed for your setup
- Understand the different NAT configurations of the MX and how they can fit into your network's needs
- How to monitor and manage the MX when it is configured in a CGNAT solution
- How to build a working model in your lab

## Information Experience

This *Day One* book is singularly focused on one aspect of networking technology that you might be able to do in one day. There are other sources at Juniper Networks, from white papers to webinars to online forums such as J-Net (forums.juniper.net). Look for the following sidebars for access directly to other superb resources for information.

## Before You Begin

This book is not meant to be a network design book or to serve as training material on what NAT is, or how NAT can be applied in one's network – instead use this book to educate yourself on how the MX Series can fit your operational NATing needs.

This book utilizes an imaginary regional service provider called Massachusetts Telcom or MassT for short. You will follow along as MassT goes on its adventure to set up several different types of NAT scenarios using its MX to fit its needs. By using MassT as our model you will see how the MX can be used as a very powerful and flexible NATing solution.

Almost every reader of this book should understand most Network Address Translation (NAT) terms and acronyms, since many of the terms Juniper uses are generic. With that said, best practice recommends reviewing the acronyms and terms this book uses and how these acronyms and terms apply when using the Juniper MX Series.

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
- PCP: Port Control Protocol

## CGNAT

So what is Carrier Grade NAT aka CGNAT as compared to plain NAT?

This book doesn't attempt to officially define *Carrier Grade NAT*; CGNAT, which is also known as Large Scale NAT, is just a buzzword for a highly scalable NAT device that sits between the CPE stuff 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 already 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:

- **NAT44** is IPv4 only. NAT44 is truly traditional NAT, taking the original source IP Address of a packet and changing it to the NAT'd IP Address. Typically used to hide the client IP Address or used with PAT(Port Address Translation) when limited pubic IPv4 addresses are available.
- **NAT66** is just NAT44 but for IPv6. It is mainly used for hiding the client IP address but can be used when limited public IPv6 addresses are available to the ISP.
- **NAT64** 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.
- **Dual-Stack Lite** is when a IPv6 address is used at the customer's premises. The customer's equipment then encapsulates their private IPv4 address target in IPv6. The CGNAT device decapsulates this, takes the private IPv4 address, and maps it to one in its NAT pool.
- **DNAT**, aka **Destination NAT** , allows the MX to ensure that packets destined to a specific virtual IP address are always NAT'd to the same private destination IP. Can be used to hide servers' actual IP addressees behind the NAT device for example, yet still allow the outside world to send them traffic.

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

- **Stactic 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 typically does not need to define an equal sized pool in regard to the number of potential 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-DPC service cards. Let's explain this right now before you move on, so it is clear in your mind.

## Inline NAT Versus 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 Junos. 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-DPC the MX can achieve line rate, low latency NAT translations (up to 120 Gbps/slot) with inline NAT. So performance wise, inline NAT is fantastic. But without the MS-DPC service card's advanced processing, the MX cannot support features like PAT or the ALG DPI packet rewrites. Service providers will look to use inline NAT with such NAT types as basic NAT44, basic NAT66, and DNAT (destination NAT). The other NAT technologies will require a service card.

As you 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 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 Types 1, 2, and 3 line cards. Older cards do not support Inline NAT. As for any newer cards that Juniper releases during the lifespan of this book, you should first check the documentation for inline NAT support and that the total inline NAT bandwidth per slot has been increased with the new card.

## Different Juniper MX Service Cards

Before you move on to the rest of this book, there is one more minor, yet important, topic to review–the different service cards for the MX. The MS-DPC, MS-MPC, and MS-MIC cards are the three options you have for the MX platform as of this book's writing.

The MS-DPC and MS-MPC are the full line card options for your MX-240, MX-480, and MX-960. The MS-MPC is the newer of the two cards with more processing power and memory; it has 4 NPUs versus 2 NPUs on the MS-DPC. It also has 32GB of memory versus the 8GB in the MS-DPC.

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 chassis bringing advanced services to the MX-80 platform.

This book will focus on the MS-DPC service card and its configuration. When using the MS-MPC and MS-MIC service cards please check the Juniper documentation for differences between using the MS-MPC and MS-MIC.

Let's get started!

# Chapter 1

## Configuration

There is more to setting up NAT on the MX then one might imagine. This is because Junos has a very, very flexible range of options so that it fits most operator's needs. This first chapter provides more than just the knowledge of some different CLI options – it attempts to show you what each option can do, and how it can be applied in your network. It should also provide some insight into how you can manage the solution, what to look for when analyzing flows 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 should learn how each section plugs into another, since they are all 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 basics are in the *Service Interfaces* section, which represents 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 also ties the NAT rules to the given interface. That NAT rule determines which traffic will be NAT translated or not. 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, our port configuration for PAT, and potentially our more advanced options.

Right now this material may seem confusing, but once you are done with this first chapter, it will all make much more sense and you will be able to understand what each configuration section truly 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-DPC cards, or the PFEs on the MPC cards, for inline NAT.

Services interfaces can be either a logical `si` interface or a logical `sp` interface. The `Si` interfaces are used for inline NAT, whereas the service PICs themselves are not, and the servicing is handled on the MPC line card itself. The `sp` interfaces are used with the MS-DPC service card, because of the card's support for advanced NAT features.

NOTE Remember, the service interface is not 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 physical interface and manipulating it before it is sent off to its destination through one of the actual physical egress interfaces.

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. This should be a PFE on the MPC line card that is in egress point for reserving the data from the client end so that you do not have to jump the fabric.

NOTE It *has* to be a MPC card. The older DPC line cards cannot anchor services inline.

So for our example, there is a 16 10G port card in FPC slot 3. For each PFE the MPC card has, you need to set a pic, which will map to an `si` interface:

```
[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 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 slot 3. Make sure you set `family inet` if you want to NAT IPv4 and inet6 for IPv6:

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

That's it for setting up the service interface for inline NAT! 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 setup for these types of scenarios using the MS-DPC service card. First you need to configure the correct package for the MS-DPC you are going to use, in this case, the `service-package layer-3`.

Here is what the configuration will look like for each MS-DPC and PIC that will be used in your non-inline NAT setup. The example uses the MS-DPC in FPC slot 2 and both of its PICs will be used in the NAT setup so each one will have the `service-package layer-3` added:

```
[edit chassis]
fpc 2 {
   pic 0 {
      adaptive-services {
         service-package layer-3;
      }
   }
   pic 1 {
      adaptive-services {
         service-package layer-3;
      }
   }
}
```

The Layer 3 services package is required to perform all NAT functionality on the MS-DPC cards. It would also be needed for other advanced services such as IPsec and IDP functionality, though those are not used in this book.

Now that the PICs on the MS-DPC have the package loaded, let's add the logical SP interface for FPC 2 PIC 0:

```
[edit interfaces]
 sp-2/0/0 {
       description "CGN Interface";
       services-options {
            cgn-pic;
       }
```

Note that the `services-options cgn-pic` is 100% optional, and it is only used when you want the PIC on the service card to be used *only* for the CGNAT feature. It must be configured to achieve your best scaling numbers.

This is a great place to show you that the MX also allows you to use redundancy when using the SP interfaces. The RSP interface redundancy feature has one SPIC take over for another SPIC when it goes down. Note that the RSP interface feature does not share session state, which means all of your flows need to be reestablished and NAT'd IP addresses need to be reassigned.

The RSP configuration is set by applying the logical RSP interface. Take a look at this:

```
[edit interfaces rspnumber]
redundancy-options {
   primary sp-fpc/pic/port;
   secondary sp-fpc/pic/port;
}
```

So using the previous example, where it had an MS-DPC in slot 2, both PICs were being used for CGNAT, but if you are going to configure redundancy you need to separate this across multiple cards to avoid outages due to any physical issues on the whole card. For the sake of our example, let's have a second MS-DPC card in FPC slot 8 and plan on using one of its PICs for CGNAT redundancy. Make sure the adaptive-services service-package is configured under the chassis hierarchy:

```
[edit chassis]
fpc 8 {
   pic 0 {
      adaptive-services {
         service-package layer-3;
      }
   }
```

Now add the logical interface:

```
[edit interfaces]
 sp-8/0/0 {
       description "CGN Interface";
       services-options {
            cgn-pic;
       }
```

Now you can add interfaces sp-2/0/0 and sp-8/0/0 under an RSP interface, like so:

```
[edit interfaces]
rsp0 {
redundancy-options {
  primary sp-2/0/0;
  secondary sp-8/0/0;
}
```

NOTE A single **sp** interface can be a secondary for multiple **sp** interfaces. This setup can effectively create N:1, up to 7:1.

The advantage of using the RSP interface is that it causes the least subscriber impact in the event of a service PIC failure. The use of Redundant service PIC (RSP) interfaces allows the active services PIC to perform an immediate switchover to the secondary services PIC in case of a major issue that requires a services PIC reboot.

NOTE One potential pitfall you should be careful about in your setup is to consider the potential event where a full MS-DPC blade fails, so assigning the primary and secondary SP interfaces from the same card under the same RSP is not the best option.

Failover to the secondary PIC occurs under the following conditions:

- The primary PIC, FPC, or Packet Forwarding Engine goes down, resets, or is physically removed from the router.
- The PIC or FPC is taken offline using the `request chassis pic fpc-slot slot-number pic-slot slot-number offline` or `request chassis fpc slot slot-number offline` command.
- The `request interface switchover` command is issued:
  - `Request interface switchover rspX`

The RSP interface redundancy can only tolerate a single failure per RSP grouping. It should also be noted that it can be used with both interface and next hop style service sets. Also, it can be used with load distribution. The recovery times for a failure range from immediate to five seconds or higher, depending on the state of the system.

You can check the status of your RSP interface running this show command:

```
lab@JTAC -re0# run show interfaces redundancy detail
Redundancy interfaces detail
Interface       : rsp1
  State         : On primary
  Last change    : 00:00:55
  Primary       : sp-11/1/0
  Secondary     : sp-10/0/0
  Mode          : warm-standby
  Current status : both up
  Replication state : Disconnected
```

Okay, the services interface is done. Next is 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 Junos MX, so stay with it, or go through it couple of times, and the rest will be downhill. Promise.

## Services NAT

There is going to be plenty to do under the services hierarchy in this configuration. First let's jump under the services / NAT hierarchy and start 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 for 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 here and familiarize yourself with your choices. This book will try to explain these options giving you a full understanding of what you might need to set up to make the pool work for your setup needs. Let's start in the services / NAT hierarchy:

```
[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 is used for a simple static NAT setup where you will be NATing IPv4 to IPv4 using inline NAT. 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;
}
```

Now you'll set up an address prefix for the `pool nat44`, so this pool is used to assign NAT'd traffic an address from the 156.0.0.0/14 range:

```
[edit services nat]
lab@JTAC_setup-re0# show
pool nat44 {
   address 156.0.0.0/14;
}
```

NOTE Addresses that are not allowed include: Martian, multicast, and loopback addresses. Specific addresses are assumed to be in inet.0

One setting needs clarification since it is set under the pool, and that is the `mapping-timeout` setting that can be enabled under the pool and is used to determine how long the Address Pooling Paired, Endpoint Independent Mapping, and Endpoint Independent Filtering mappings remain active (but it will not be important until a later portion of this chapter). These three features are discussed later so you can park this option and revisit it later in the book. For now, let's just remember that the mapping-timeout in seconds can be set here:

```
[edit services nat]
lab@JTAC_setup-re0# show
pool nat44 {
   address 156.0.0.0/14;
}
mapping-timeout 300
```

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

```
lab@JTAC_setup-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 2          REV 11   750-036585   CAAH9396          MS-DPC <-- This is what you want
  CPU          REV 03   710-036586   CAAG9990          DPC PMB 2GB
  PIC 0                BUILTIN      BUILTIN           MS-DPC PIC
  PIC 1                BUILTIN      BUILTIN           MS-DPC PIC
```

Okay, look at the port section 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 ( auto | 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 */
   }
}
```

To understand what is set under the port's section for the pool, you need to understand the settings that come with it. Under the port hierarchy, you have an option called *automatic,* which really means "*I will not define any specific port ranges for PAT and I will have the MX automatically use ports 1024 through 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: `auto` and `random-allocation`.

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

For our example let's create our pool, called `natpat44`, set a prefix for the IP addresses to NAT, and make the MX assign any free port from 1024 through 65535 in a sequential manner for each new flow by using the `automatic` setting under `port` along with `auto`:

```
[edit services nat pool natpat44]
lab@JTAC_setup-re0# show
address 100.100.0.0/16;
port {
    automatic {
        auto;
    }
}
```

When you are within the port hierarchy, if you do not use `automatic` 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. By default, the ports under the range option are assigned to each flow in order, starting from the lowest value and then counting up sequentially for each new flow, but to fight port-predictability for defined ranges, you can change this to `random-allocation` as seen below (if that is your desired behavior):

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

So, 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:

```
[edit services nat pool natpat44]
lab@JTAC_setup-re0# show
address 100.100.0.0/16;
port {
    range low 8888 high 9999 random-allocation;
}
```

BEST PRACTICES

One thing to keep in mind when using dynamic NAT is that you need to be careful of the pool size! When you are not using the PBA feature and round-robin address allocation, the max pool size should be a /26 for the MS-DPC service card. More IP addresses in the NAT pool than a /26 would not be used on your MS-DPC, since there is a memory limit. A MS-DPC can manage 8.4M max flows per PIC based on the configuration used. That means the total number of flows used in a dynamic PAT setup when EIM and address-pooling are not used can fit nicely into a /26. (The math being 64x64K = 4M sessions which is 8M flows). Combine the EIM and address pooling features and you need additional memory for storing the mappings. This statement is being made a bit prematurely, but it will make sense when you have finished the book and understand the complete features.

Let's take a step back, forget that we are not 100% complete with our configuration, and take a look at a few show commands. Here is an example showing the NAT mappings when using the NAT pool `natpat44`, with the ports set to be automatically assigned in a sequen-

tial order. You'll see there are three flows NAT'd and PAT'd against two different subscriber source IPs:

```
lab@JTAC_setup-re0> show services nat mappings detail
Interface: sp-2/1/0, Service set: CGN-1
NAT pool: CGN-1
Mapping         : 10.0.14.3     :24669  --> 100.100.1.101    : 1026
Session Count   :    2
Mapping State   : Active
Mapping         : 10.0.14.2     :24638  --> 100.100.1.100    : 1025
Session Count   :    2
Mapping State   : Active
Mapping         : 10.0.14.2     :24582  --> 100.100.1.100    : 1024
Session Count   :    2
Mapping State   : Active
```

And here is an example showing the NAT/PAT mappings when using a NAT pool set with random allocations:

```
lab@JTAC_setup-re0> show services nat mappings detail
Interface: sp-2/1/0, Service set: CGN-1
NAT pool: CGN-1
Mapping         : 10.0.14.3     :24669  --> 100.100.1.101    : 11950
Session Count   :    2
Mapping State   : Active
Mapping         : 10.0.14.2     :24638  --> 100.100.1.100    : 44890
Session Count   :    2
Mapping State   : Active
Mapping         : 10.0.14.2     :24582  --> 100.100.1.100    : 25123
Session Count   :    2
Mapping State   : Active
```

Get used to running the `show service nat mapping` command since it is one command that will give you insight into what is happening on the MX in regard to your NAT settings. It is a great place to start analyzing the box, 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 NAT setup.

### Address Allocation

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 flows. Let's set a few address ranges under our pool to show an example that uses more than just a single address prefix that's shown in the previous example pools, `nat44` and `natpat44`. This new pool is called CGN-1, it uses three address ranges, and it randomly allocates ports between 1024 and 65535 to new flows. And let's add a new option, `address-allocation`. 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, aka lowest, address in the NAT pool to the first flow it processes. So, in the above example of the pool CGN-1, the MX starts assigning address 10.12.1.100 and continues through 10.12.1.216. You should understand that even if the first flow has been released from the system, and 10.12.1.100 is available to be used as a NAT'd IP address again, the MX will not use it until the MX has wrapped through the IP addresses of all the Pools. Only when the MX has assigned all the IPs in this example (this would be through 10.12.1.216 to a NAT'd flow) would the MX then wrap back around to 10.12.1.100.

You should also know that without setting the `address-allocation round-robin` option, the MX's default behavior is always to use the first available address in the NAT pool, which means the lowest-valued free IP address to NAT to a new flow. So, unless a long-lived traffic session was using the NAT'd address 10.12.1.100, this address would see lots of use if `address-allocation round-robin` is *not* set. Potentially some subscriber's bad behavior could 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. (Bad behavior could be along the lines of someone cheating on a video gaming server to someone trying to hack a given node on the Internet.) In this case, the operator may want to use address-allocation with round-robin set to lessen the chance that subscribers have issues on the Internet due to an IP-address being blocked because of someone else's malicious intentions.

### A Few More Features

At this point, let's also look at 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 flows source port when the MX is using PAT to allocate a source port for an outbound connection. These options are more corner cases but can be

useful for certain setups, so you should be aware that these features exist. In some cases, they are actually needed more than you may realize. 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
| random-allocation) ;
  preserve-parity;
  preserve-range {
{
   }
```

You may need to configure the `preserve-parity` option if you are implementing some multiple media solutions that use RTP (real-time transport protocol) or if you expect your subscribers to use RTP in some manner. The RTP protocol, which handles data transport, uses even ports, and its peer 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. So when `preserve-parity` is enabled, and let's say the MX does not have any available odd source ports for the odd source range you need to NAT/PAT to, you will drop the traffic even if you still have available even ports. The same behavior holds true for even ports.

The `preserve-range` feature is used when an operator needs to make sure their NAT'd flows 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, you will drop the traffic. So if `preserve-range` is enabled and the source port you received from the client side is 1011, yet ports 0 through 1023 are currently assigned to active translated flows, you will drop the new flow even if there are open ports in the 1024-66535 range that can still be used for the NAT translation.

Now when using preserve-range you can expect ports to be used in 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 flows getting dropped. 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 `automatic` normally uses 1024-65535 but when the preserve-range option is present the functionally changes just a bit.

So, 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 100.100.0.0/16;
        port {
            range low 500 high 3500;
            preserve-parity;
            preserve-range;
        }
    }

pool natpat44 {
    address 100.100.0.0/16;
    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 and the ephemeral/non-default ports (1024 – 65535). You can only define an overlapping range if you need to define what ports can be used for translation or else you can just use the `automatic` option if ALL ports should be made available.

Let's use the `show services stateful-firewall flows` command to see what the MX is doing. In this example the MX is using NAPT with a pool and port range setup to be NAT'd like so:

```
    pool natpat44 {
        address 100.100.0.0/16;
        port {
            range low 5000 high 8000;
        }
```

In the following output you can see the MX has mapped a port to each flow from the pool created. So, starting from port 5000, it sequentially uses the next available port for the next flow. It does not matter what the subscriber source port is. Also in this output, 14.0.0.0/8 is the internal private network, 100.100.0.0/16 as seen above is the IP range we will use to NAT to out flows, and 197.100.1.8 is a target on the public side the 14.0.0.0/8 network is reaching through the MX NAT solution:

```
root@JTAC_setup-re0# run show services stateful-firewall flows terse
Interface: rsp1, Service set: nat44
Flow                                    State   Dir     Frm count
TCP          14.1.0.0:63258  ->    197.100.1.8:2001   Forward  I           3
   NAT source       14.1.0.0:63258   ->    100.100.0.1:5011
```

```
UDP          14.1.0.0:66    ->    197.100.1.8:2004   Forward  I           43
   NAT source       14.1.0.0:66     ->    100.100.0.1:5012
UDP          14.0.20.5:2002   ->    197.100.1.8:2003   Forward  I          480
   NAT source      14.0.20.5:2002    ->    100.100.0.1:5007
UDP          14.0.20.2:2002   ->    197.100.1.8:2003   Forward  I          10
   NAT source      14.0.20.2:2002    ->    100.100.0.1:5004
UDP          14.0.20.1:2002   ->    197.100.1.8:2003   Forward  I          40
   NAT source      14.0.20.1:2002    ->    100.100.0.1:5002
UDP          14.0.20.6:2002   ->    197.100.1.8:2003   Forward  I          440
   NAT source      14.0.20.6:2002    ->    100.100.0.1:5006
UDP          14.0.20.8:2002   ->    197.100.1.8:2003   Forward  I          20
   NAT source      14.0.20.8:2002    ->    100.100.0.1:5000
UDP          14.0.20.9:2002   ->    197.100.1.8:2003   Forward  I          3
   NAT source      14.0.20.9:2002    ->    100.100.0.1:5003
UDP          14.0.20.7:2002   ->    197.100.1.8:2003   Forward  I          480
   NAT source      14.0.20.7:2002    ->    100.100.0.1:5008
UDP          14.0.20.0:2002   ->    197.100.1.8:2003   Forward  I          434
   NAT source      14.0.20.0:2002    ->    100.100.0.1:5009
UDP          14.0.20.3:2002   ->    197.100.1.8:2003   Forward  I          30
   NAT source      14.0.20.3:2002    ->    100.100.0.1:5001
UDP          14.0.20.4:2002   ->    197.100.1.8:2003   Forward  I          81
   NAT source      14.0.20.4:2002    ->    100.100.0.1:5005
```

Once the `preserver-parity` and `preserve-range` options are set, things are different. You can see the MX is mapping odd ports to odd ports and even to even. You can also see the MX is dropping any traffic from ports 0-1023 since you have not mapped 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 100.100.0.0/16;
       port {
          range low 5000 high 8000;
          preserve-parity;
          preserve-range;
       }
```

```
[edit services nat pool natpat44 port]
root@JTAC_setup-re0# run show services stateful-firewall flows terse
Interface: rsp1, Service set: nat44
Flow                                        State    Dir      Frm count
UDP          14.0.8.0:2002   ->    197.100.1.8:2003   Forward  I           33
   NAT source       14.0.8.0:2002    ->    100.100.0.1:5000
UDP          14.1.8.3:66     ->    197.100.1.8:2004   Drop     I           2
UDP          14.1.8.3:2002   ->    197.100.1.8:2003   Forward  I           4
   NAT source       14.1.8.3:2002    ->    100.100.0.1:5012
UDP          14.1.8.4:154     ->    197.100.1.8:2004   Drop     I          29
UDP          14.1.8.4:2002   ->    197.100.1.8:2003   Forward  I           3
   NAT source       14.1.8.4:2002    ->    100.100.0.1:5016
TCP          14.1.8.5:55106  ->    197.100.1.8:2001   Forward  I           3
   NAT source       14.1.8.5:55106   ->    100.100.0.1:5022
TCP          14.1.8.4:53578  ->    197.100.1.8:2001   Forward  I           3
```

```
   NAT source       14.1.8.4:53578   ->    100.100.0.1:5018
TCP          14.0.8.0:60098  ->    197.100.1.8:2001   Forward  I            6
   NAT source       14.0.8.0:60098   ->    100.100.0.1:5002
TCP          14.1.8.1:57801  ->    197.100.1.8:2001   Forward  I            3
   NAT source       14.1.8.1:57801   ->    100.100.0.1:5009
UDP          14.1.8.7:2002   ->    197.100.1.8:2003   Forward  I            1
   NAT source       14.1.8.7:2002    ->    100.100.0.1:5028
```

## PBA

You are almost done understanding the pool's configuration hierarchy. As a final task, let's take a look at the `secured-port-block-allocation` (PBA) options. 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. Assigning a port block to a subscriber IP address is beneficial in that it cuts back on the number of logs the MX needs to generate when mapping a source address to a NAT'd address. (Logging benefits will mean more to the reader in a later section and then you will really see how PBA can help.) Aside from the benefits of logging, some operators use PBA with their PAT setup when they need to limit the number of ports they want to assign to each private IP address. When operators do not have enough public IP addresses to match the number of private addresses that may be NAT'd at any one time, PAT, with a port limit per private IP address, is a solution. Before examining this more closely, let's first look at the configuration:

```
[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 */
   }
}
```

Note the following in this configuration:

- `block-size` determines the number of ports per block that will be assigned a given subscriber's NAT'd address.
- `max-blocks-per-user` dictates the total potential blocks a given subscriber will be assigned.
- `active-block-timeout` is useful when `max-blocks-per-user` is set to a value of 2 or higher. `active-block-timeout` determines how long the current block will be used to allocate sessions from. This is yet just another setting that can be used to fight port predic-

tion. 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 source ports are not constantly being used.

Let's create a new pool that utilizes the PBA feature, for another example:

```
[edit services nat pool NAT-pool2]
lab@JTAC_setup# show
address-range low 217.200.202.80-high 217.200.202.94 ;
port {
    }
    secured-port-block-allocation block-size 100 max-blocks-per-address 4;
}
address-allocation;
```

Port blocks are assigned to NAT'd flows in order starting from the lowest value and counting up. Within the MX 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 a flow through the box will get a block assigned to them. Using our example above, that block would be 1024-1123.The next subscriber that needs a new block to be their active flow would get 1124-1223, then 1224-1323, and 1324-1423, 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 you allow each subscriber's private IP address four blocks before you reject any new flows for them.

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;
}
```

## Syslog Messages

It was stated earlier that the PBA feature is an even more efficient method for tracking subscribers to their NAT and PAT assignments when using syslog messages . You will actually see that the MX has an even better method to track subscribers to their NAT'd addresses instead of using syslog messaging at all, when the book gets to the deterministic NAT translation type in the next section. So, with that said, let's take a break from looking at Junos NAT configurations and look into the syslog messages.

Syslog messages can be written to the RE or to an external server. Those details are discussed later, but right now let's focus on how you can map, via a single syslog message, the subscriber's private IP address and its mapping to the public IP and assigned port block. You'll understand why PBA can be useful when it comes to efficient logging.

So here is the message you would see for a private IP address when a flow hits the MX and needs to be NAT'd when the PBA feature is used. This is the only message you will see until that subscriber has so many active flows open that they require another port block to get assigned or unless you are using the `active-block-timeout` setting and it has expired causing the MX to assign a new port block:

```
Aug 21 20:43:05  JTAC_setup-re0 (FPC Slot 11, PIC Slot 1) 2013-08-21 17:43:04
NAT44[FWNAT]:ASP_NAT_PORT_BLOCK_ALLOC: 14.1.36.0 -> 156.0.0.125:1024-1087
```

Now, without using the PBA feature on the pool, you would have to track each individual NAT'd flow – so every flow sent from that private IP address would cause a message to be written if syslog was enabled. You can already see why PBA is desirable in regard to tracking your NAT'd flows via syslog messages:

```
Aug 21 20:43:05  JTAC_setup-re0 (FPC Slot 11, PIC Slot 1) 2013-08-21 17:43:04
NAT44{nat44}[FWNAT]: ASP_SFW_CREATE_ACCEPT_FLOW  14.1.36.0:2002 [156.0.0.125:1024] ->
197.100.1.1:2003  (UDP)
Aug 21 20:43:05  JTAC_setup-re0 (FPC Slot 11, PIC Slot 1) 2013-08-21 17:43:04
NAT44{nat44}[FWNAT]: ASP_SFW_CREATE_ACCEPT_FLOW  14.1.36.0:2003 [156.0.0.125:1025] ->
197.100.1.2:2004  (UDP)
Aug 21 20:43:05  JTAC_setup-re0 (FPC Slot 11, PIC Slot 1) 2013-08-21 17:43:04
NAT44{nat44}[FWNAT]: ASP_SFW_CREATE_ACCEPT_FLOW  14.1.36.0:2003 [156.0.0.125:1026] ->
197.100.1.1:2004  (UDP)
Aug 21 20:43:05  JTAC_setup-re0 (FPC Slot 11, PIC Slot 1) 2013-08-21 17:43:04
NAT44{nat44}[FWNAT]: ASP_SFW_CREATE_ACCEPT_FLOW  14.1.36.0:43981 [156.0.0.125:1027] ->
197.100.1.1:2048  (ICMP ECHO REQUEST)
```

Earlier, this chapter mentioned malicious users potentially doing things that could get them blocked. Such activity is just one reason you may need to have a way to map a NAT and PAT assignment to the actual private IP address that has been assigned to the subscriber's device. If

you do not capture the syslog messages for NAT flows, there is no other historical data from the MX that can help.

So unless you are using static NAT types, such as basic-nat44, dnat-44, and basic-nat66, or if you are using deterministic NAT (something discussed in the next section) you have to have a way to see which NAT/PAT address/port combo belonged to which subscriber's private IP address during a period of time. You can enable syslogging to get this data. When using syslogging, PBA can be used to make tracking easier since it cuts down on the number of messages written. Typically, PBA is used with the dynamic NAT types such as NAT44 and NAT66.

Syslog messaging for NAT flows is discussed in more detail later on in this book. This PBA section is just a great place to mention the syslog messages so you can start to have an understanding of why they would be helpful.

## More on Port Block Allocation (PBA)

Before moving on let's talk about PBA for just a few more minutes. This feature can be great but it has some *gotchas* where operators really need to think about the consequences of enabling the feature.

One thing about the PBA feature: be very, very, very careful concerning how large you configure your port blocks. Make sure you have compared the number of unique private IP subscribers you expect to access the MX at peak hours against the configured port block size. For example, during peak hours it is expected that 10,000 subscribers will access the public network concurrently through the MX and these 10,000 subscribers will have their traffic NAT'd. Now you have a NAT pool of ten IP addresses with a port range of 1024-65535. Do not set your secured-port-block-allocation block-size to be 100 or you will run out of ports and see lots of flows getting dropped.

The finding would be: Port 1024 through 65535 means there are 64,511 ports available per NAT'd IP. Ten NAT'd IPs in the pool means you have 645,110 unique ports total to use for PAT. Ten thousand subscribers with a port block size of 100 would require, at a minimum, 1,000,000 ports. If you have to go live with this limitation of ten IP addresses you employ for NAT'd address, set the `secured-port-block-allocation block-size` to be 64 and set `max-blocks-per-address` to 1. Also make sure your inactive flows and mapping timeout are low. You need to free pool resources very quickly here. If you need more than 64 active flows, or need mappings to hang around longer for address-pooling paired and endpoint independent mappings requirements, you will just 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 `Out of port error` counter continues to climb, you have an issue and the number of flows versuss the `port block size` and `max-blocks-per-address` needs to be looked at:

```
juniper@JTAC_setup-re0> show services nat pool detail
Interface: rsp1, Service set: SSET1
  NAT pool: POOL, Translation type: dynamic
    Address range: 217.200.202.80-217.200.202.94
    Port range: 1024-65535, Ports in use: 958003, Out of port errors: 65251634, Max
ports used: 958022
    AP-P out of port errors: 891636
    Current EIF Inbound flows count: 0
    EIF flow limit exceeded drops: 0
```

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 client CPE can use 65,535 ports. But once you enable PBA you are limiting the overall number of ports a subscriber can use unless your `block-size` and `max-blocks-per-user` equals 65,535. As stated earlier, some operators can use PBA not just for its efficient syslog messaging but for the plain fact that they have a limited number of public IP addresses so they employ PAT. The operator in question may want to limit the number of ports each private IP address can use, since they do not want a few power users who are using tons of different applications with lots of traffic flows to take up all the NAT port resources. Though later on we will look at the `max-sessions-per-subscriber` feature, which was created purely to limit the number of sessions each subscriber can use at anyone time.

NOTE With respect to the port block allocation, it should be noted that this value should never be changed on a running system when the NAT pool is being used. 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, it is always recommended that the service PIC be restarted after the changes to ensure there are no issues – so make these changes during a maintenance window when a restart is possible.

Now a few final notes on the NAT pool configuration before moving on to the NAT rules configuration. These are a few best practices you

should be aware of to make sure you are not wasting valuable routable IPv4 addresses, and you are not setting up the service PIC to handle more flows than it can correctly manage.

BEST PRACTICES Think about pool sizes before you specify a port for a dynamic NAT type like NAPT44. Address ranges are limited to a maximum of 65,000 addresses, for a total number of 4,259,775,000 flows. (The math would be 65,000 addresses x 65,535 ports.) A dynamic NAT pool with no address port translation supports 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. The MX system will warn you when you try to make the pools too big:

```
[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 This statement is being repeated but it is important. Based on the configuration used, the maximum possible number of flows supported by each service PIC on a MS-DPC card would be 8.4 million, which would be 4.2 million sessions if you consider one reverse flow for each forward flow. This max flow scenario requires that four millions ports be available from the pools for this Service PIC. This would mean a total /26 range of IP addresses would be the correct number of IPs to be used in the pools across this service PIC. (The math being 64 IP addresses with 1024-65535 ports range (64K ports each address.) Of course you could increase the number of IP addresses beyond the /26 range, but if this is done you should decrease the number of ports in the range. The idea behind the design choice you make as the provider is do not get into a scenario where the MX may have over 8 million active flows or mappings in memory at the same time. Why add the potential to fill all physical memory available – you gain nothing but a potential failure in service to your end users and a potential waste of valuable routable IPv4 addresses. Note this Best Practice example is for a setup that does not use the APP, EIM, or EIF features. If any of these options are enabled the total number of flows per Service PIC is lower than 8 million due to the additional mappings needed to store the APP, EIM, or EIF entries. So more thought needs to be put into pool and port size.

## NAT Rules

Hopefully all this is not too overwhelming as there's quite a bit further to go. This section of Chapter 1 jumps into configuring the NAT rules, another major building block for the MX setup. NAT rules will tie into the NAT pools just configured. And as you will see, NAT rules determine what traffic actually gets translated, as well as what NAT translation types the MX uses. Consider the following (with notes in boldface):

```
[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 <-- Used to explicitly skip NATing traffic if desired
         translated {
            destination-pool nat-pool-name;
            source-pool nat-pool-name;
            translation-type {
               napt-44; <-- many more options available              }
         }
      }
   }
}
```

Let's create our first example rule that called rule1. NAT rules operate under the same conditions as the MX firewall filter matching conditions `match-direction`, which needs to match the direction the traffic is flowing across the PIC. Typically, most NAT setups are configured to NAT flows created by the client side, so you typically use `match-direction input` since that is the direction from which the subscriber's traffic is received by the service PIC from the PFE receiving the data from the ingress side:

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

Next you set up a term, simply named term1. Under this term you set a few `from` statements that cover some IP prefixes set using the stanza

source-address and a range of private IP addresses set using the stanza source-address-range. The service PIC will translate traffic from private CPE nodes that arrive from these 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 next thing you set is a then statement that is used if the criteria in the from statement is met, meaning traffic from CPEs that are at the defined networks. In our example let's use our pool CGN1 created earlier. Then you define the NAT type you want to use, in this case, NAPT44. NAPT44 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;
      }
   }
}
```

It is with 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. So 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

- `dnat-44`
  Static destination address (IPv4 to IPv4) translation
- `dynamic-nat44`
  Dynamic source address only (IPv4 to IPv4) translation
- `napt-44`
  Dynamic 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]
- `napt-pt`
  NATPT (source address (IPv6 to IPv4) and source port and prefix removal for destination address (IPv6 to IPv4) translation)
- `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

Wow, that is an exhaustive list! Let's go a bit deeper now and look under the covers. First thing to be aware of when setting up a rule is that the NAT `translation-type` in your pool definitions must match the NAT translation type you want to use. So 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
```

Look at the [`services nat`] configuration that has a pool and a rule

that calls this pool and you can see why this fails:

```
nat {
      pool nat444 {
         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. You need to understand which one you 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. Used in the IPv4 and IPv6 world to hide the host machines IP, the relationship is a static NAT one-to-one setup, meaning one public NATing IP address for each private IP address. Basic-nat44 is used only with IPv4 translation and basic-nat66 is used only for IPv6 translation. These translation types are also inline NAT types. Remember this means they do not use a service card for NAT. Just set up the `si` logical interface and the MPC line card does the work. Also 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 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 different dynamic NAT translation-type.

An example of a NAT rule and NAT pool setup for `basic-nat44` would be:

```
[edit services nat]
pool nat44_basic {
   address 156.100.0.0/14;
}
rule rule1 {
   match-direction input;
   term t1 {
      from {
         source-address {
            14.0.0.0/14;
         }
      }
      then {
         translated {
            source-pool nat44;
            translation-type {
               basic-nat44;
            }
         }
      }
   }
}
```

```
root@JTAC_setup-re0# run show services stateful-firewall flows extensive
Interface: rsp1, Service set: nat44
Flow                                        State   Dir      Frm count
UDP        14.1.16.1:4015  ->   197.100.1.1:55   Forward  I           31
   NAT source      14.1.16.0:4015   ->     156.1.16.1:4015
  Byte count: 16368
  Flow role: Master, Timeout: 22
UDP       197.100.1.1:55   ->    156.1.16.1:4015  Forward  O           31
   NAT dest       156.1.16.1:4015   ->      14.1.16.1:4015
  Byte count: 16368
  Flow role: Responder, Timeout: 23
```

A good thing to remember 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 static one-to-one mapping. So, let's say you have a CPE on the private network at 14.1.16.1. Using our preceding example pool, this CPE will always have 156.1.16.1 assigned to it as its NAT'd address

After looking at the basic-nat44 translation type, it's the perfect time to look at `dynamic-nat44` next. So, how does the translation type dynamic-nat44 differ from basic-nat44? Dynmaic-nat44 employs the use of the service card to dynamically allocate IP addresses from the NAT pool. So 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 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. This configuration works fine 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 {
                    14.0.0.0/14;
                }
            }
            then {
                translated {
                    source-pool nat44;
                    translation-type {
                        dynamic-nat44;
                    }
                }
            }
        }
    }
}
```

WARNING! Be careful with this type of setup, though. Dynamic-nat44 may allow you to actually configure a smaller NAT pool compared to the potential private source address ranges, but this does not mean it would work for you in the real world. Consider the preceding example. You have a /30 for potential 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 have and actual active mappings on the PIC. That means that if you expect three different private hosts trying to send data thru the MX NAT solution 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 to NAT'd flows, based on the number of hosts you feel will need to pass data simultaneously thru the MX at peak time. If the number of hosts is 30,000, then you need a /17 for your NAT pool. If you do not have a /17 range you can use for your NAT pool you need to look at using one of the translation types that employ Port Address Translation, such as NAPT44.

This is a fantastic time to again talk about the static NAT types versus dynamic NAT types, now that we have a real example laid out in front

of us. 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. The dynamic type `dynamic-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, though you would most likely employ napt-44 if this was the case since it uses PAT. Also dynamic types require a service card, a fact that is admittedly drilled home throughout this book.

The next NAT translation type up is `dnat-44`, otherwise known as *Destination NAT*. Destination NAT means if the flow through the MX is trying to reach the destination address, or range defined, and optionally, the destination port or port ranges, then the MX NAT's the destination address. One 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 to reach the targets. 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 require a service card:

```
[edit services nat]
rule rule1 {
   match-direction input
   term t1 {
      from {
         destination-port {
            range low 40 high 8888;
         }
         destination-address-range {
            low 120.100.0.100 high 120.100.0.200;
         }
      }
      then {
         translated {
            destination-pool natpat44;
            translation-type {
               dnat-44;
            }
         }
      }
   }
}
```

And there is a *gotcha*: you can only have one NAT rule per NAT pool if the pool that is being used is against a rule that has a translation type of `dnat-44`. You can also only have one term that calls a pool that is using a specific dnat-44 pool. That is unless you enable the `allow-overlapping-nat-pools` option under the services / nat hierarchy. Look at the example:

```
[edit services nat]
allow-overlapping-nat-pools;
```

You can also use a feature called *port forwarding* when using the `dnat-44` translation type. Under the services / nat hierarchy you can set your port mappings. In our example a map called *map1* is set with two port translations:

```
[edit services nat]
port-forwarding map1 {
    destined-port 53 translated-port 23;
    destined-port 1028 translated-port 20022;
}
```

Then you 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 53 or 1028 will also have their ports translated. 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 at the following configuration to see all this together:

```
[edit services nat]
rule rule1 {
    match-direction input
    term t1 {
        from {
            destination-port {
                range low 40 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;
                }
            }
        }
    }
}
```

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 type translates the source IP and the source Port. This means you can have many private IPs using the same public 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 this is dynamic NAT at its essence. These NAT translation types do require a service card to handle the more advanced processing of the packets.

Let's put together a quick configuration that should hopefully speak for itself:

```
[edit services]
nat {
   pool CGN-1 {
      address-range low 10.12.1.100 high 10.12.1.200;
      port {
         range low 1055 high 10000;
      }
}
rule CGN-1 {
      match-direction input;
      term other {
         from {
            source-address-range {
               low 10.0.14.3 high 10.0.100.13;
            }
         }
         then {
            translated {
               source-pool CGN-1;
               translation-type {
                  napt-44;
               }
```

Remember, the MX will normally tell you when there is an issue with your configuration. For example, if you have not configured the SP interfaces and are trying to use the `si` inline logical interface for a NAT type like NAPT. You will be warned by Junos:

```
[edit services nat rule rule1 term t1 then]
lab@JTAC_setup -re0# commit

[edit services nat rule rule1 term t1 then]
  'translated'
   translation-type has to be 'basic-nat44' or 'dnat-44' where si is the service-
interface
error: configuration check-out failed
```

So another NAT type spoken about, 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 address and port to a NAT'd address and port but do not want to have to use syslogs to map the NAT'd public IP back to the private IP. Syslogs can be intensive and not something you may want to mange. With deterministic-napt44, the MX has an algorithm it uses to assign the NAT'd IP address and blocks

of ports to the source IP ranges set under the NAT rules. This makes this NAT translation type very desirable by many operators since it allows the operator to tell who used what NAT'd IP address and port without needing syslogs:

```
nat {
    pool nat44 {
        address 156.0.0.0/20;
        port {
            automatic;
            deterministic-port-block-allocation;
        }
    }
    rule rule1 {
        match-direction input;
        term t1 {
            from {
                source-address {
                    14.0.0.0/14;
                }
            }
            then {
                translated {
                    source-pool nat44;
                    translation-type {
                        deterministic-napt44;
                    }
                }
            }
        }
    }
}
```

So, using our current example, we would see the 14.0.0.0/14 range mapped with a source port block of 512 ports, which is the default size.

Let's finally run a show command, `show services nat deterministic-nat nat-port-block`, and look at a couple of subscriber NAT'd IP addresses. This way you can see a NAT'd address assigned and a block of 512 source ports that the MX will assign to flows for this subscriber based on the *number of flows they open:*

```
lab@JTAC_setup-re0# run show services nat deterministic-nat nat-port-block 14.1.28.98
Service set: nat44
Interface: rsp1
NAT pool: nat44
Internal Host: 14.1.28.98, NAT IP address: 156.0.2.66, NAT Port Start: 51712, NAT Port
End: 52223

lab@JTAC_setup-re0# run show services nat deterministic-nat nat-port-block 14.1.28.27
Service set: nat44
Interface: rsp1
```

```
NAT pool: nat44
Internal Host: 14.1.28.27, NAT IP address: 156.0.2.66, NAT Port Start: 15360, NAT Port
End: 15871
```

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

```
lab@JTAC_setup-re0# run show services nat deterministic-nat nat-port-block 14.1.128.27
Service set: nat44
Interface: rsp1
NAT pool: nat44
Internal Host: 14.1.128.27, NAT IP address: 156.0.3.13, NAT Port Start: 26624, NAT Port
End: 27135

lab@JTAC_setup-re0# run show services nat deterministic-nat nat-port-block 14.1.128.28
Service set: nat44
Interface: rsp1
NAT pool: nat44
Internal Host: 14.1.128.28, NAT IP address: 156.0.3.13, NAT Port Start: 27136, NAT Port
End: 27647
```

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

```
lab@JTAC_setup-re0# run show services nat pool detail
Interface: rsp1, Service set: nat44
  NAT pool: nat44, Translation type: dynamic
    Address range: 156.0.0.1-156.3.255.254
    Port range: 1024-65535, Ports in use: 0, Out of port errors: 0, Max ports used: 100
    AP-P out of port errors: 0
    Max number of port blocks used: 200, Current number of port blocks in use: 0, Port
block allocation errors: 0, Port block memory allocation errors: 0
    DetNAT subscriber exceeded port limits: 0
    Unique pool users: 0
    Current EIF Inbound flows count: 0
    EIF flow limit exceeded drops: 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. Not every port in the range can, or will, be used. deterministic NAT will never use privileged ports.

Let's look at the following NAT pool definition:

```
lab@JTAC_setup > show configuration services nat pool NAT-pool-
address-range low 80.1.0.0 high 80.1.127.255;
address-range low 80.1.128.0 high 80.1.255.255;
port {
    range low 1 high 65535 random-allocation;
    deterministic-port-block-allocation;
}
mapping-timeout 120;
```

```
snmp-trap-thresholds {
    address-port low 50 high 80;
}
```

As shown in the next output, the privileged ports are not taken into account:

```
lab@JTAC_setup > show services nat pool NAT-pool
Interface: rsp0, Service set: svc-set-TEST-01A
NAT pool        Type    Address          Port       Ports used
NAT-pool dynamic 80.1.0.0-80.1.127.255       1024-65535  0

Port block type: Deterministic port block, Port block size: 512
```

This behavior is correct per the deterministic NAT Draft RFC.

Under the [`services nat pool nat44 port`] configuration hierarchy you can set a block-size when you enable `deterministic-port-block-allocation`:

```
nat {
    pool nat44 {
        address 156.0.0.0/20;
        port {
            automatic;
            deterministic-port-block-allocation block-size 10;
        }
    }
```

As with the PBA feature, when making changes to the deterministic port block size you must restart the service card for this to take effect.

NOTE For the PBA or deterministic NAT configuration changes to take effect on the PIC, you must reboot the service PIC when the below NAT configuration changes are made under the [`edit services nat pool`] hierarchy:

- port rang
- address/address-range
- block-size
- max-blocks-per-address
- active-block-timeout

So, when making these changes, plan around a maintenance window if possible.

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:

```
lab@JTAC_setup-re0# run show services nat deterministic-nat nat-port-block 14.1.128.28
Service set: nat44
Interface: rsp1
NAT pool: nat44
Internal Host: 14.1.128.28, NAT IP address: 156.0.0.16, NAT Port Start: 16684, NAT Port
End: 16693

lab@JTAC_setup-re0# run show services nat pool
Interface: rsp1, Service set: nat44
NAT pool        Type    Address             Port       Ports used
nat44           dynamic 156.0.0.1-156.3.255.254       1024-65535  100
Port block type: Deterministic port block, Port block size: 512
```

So why doesn't everyone just use deterministic NAT for their translation of IPv4 to IPv4 dynamic translation type? 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 source range your commit will fail. This boundary, which is necessary for 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:

```
lab@JTAC_setup-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]
lab@JTAC_setup-re0# show
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]
lab@JTAC_setup-re0# show
from {
    source-address {
        192.168.0.0/13;
    }
}
then {
    translated {
        source-pool nat44;
```

```
        translation-type {
           deterministic-napt44;
        }
    }
}
```

To make this configuration work where you are only trying to assign a 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 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. So for five total flows to work you would need to increase the number of IP addresses available in your NAT pool to at least 41 addresses. Make sense?

It is now time to talk about the translation types *twice NAT*, referring to the practice of NATing both the source and destination address. 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` (note that twice-basic-nat-44 is an inline NAT type, whereas twice-dynamic-nat-44 and twice-napt-44 require a service card):

```
[edit services nat]
rule twice-nat {
            match-direction input;
            term my-term1 {
                from {
                    destination-address {
                        41.41.41.41/32;
                      }
                    source-address-range {
                        low 10.58.254.34 high 10.58.254.35;
                      }
                  }
                then {
                    translated {
                        source-pool src-pool;
                        destination-pool dst_pool;
                        translation-type {
                            twice-basic-nat-44;
                        }
                    }
                }
            }
}
```

`Stateful-nat64` is the next translation type option. 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) and the second part embeds the IPv4 address from 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;
            }
         }
```

For the destination-address and destination-prefix 64:ff9b::/96 was used. The w*ell 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 As of 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 an 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:

```
lab@JTAC_setup-re0> show services nat pool
Interface: sp-2/1/0, 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


lab@JTAC_setup-re0> show services stateful-firewall flows
Interface: sp-2/1/0, Service set: sset-nat64
Flow                                    State    Dir      Frm count
ICMP       10.12.1.10      ->      50.1.1.1      Watch    O             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 does not support using the Port Block Allocation feature with NAT64.

## Service Sets

By now you should understand NAT pools and NAT rules and how to set them up. 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 to the MX and actually do something 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, 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, aka styles, you can use to apply your service sets based on your needs and preference: *interface-style* and *next-hop style*. In a nutshell, what they really do is determine what data packets will be steered towards your service cards' PIC for CGNAT processing.

Let's look at the difference between these two service set methods and sample configurations, so you can determine what 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 traversuss the service PIC via a service filter applied to the interface, as shown in Figure 1.1.

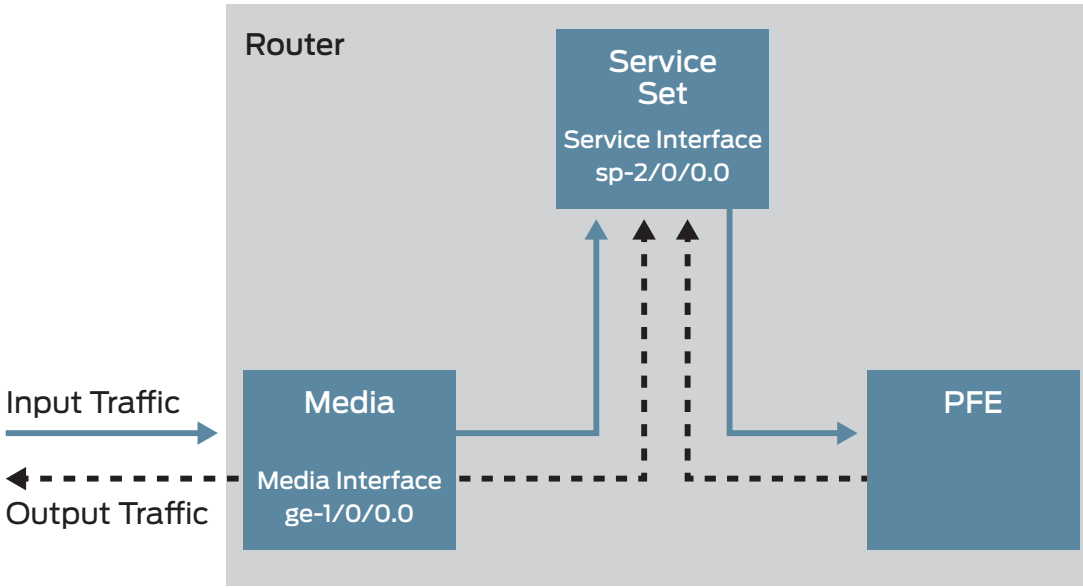


Figure 1.1 Illustration Showing Interface-Style Service Set

## Next-Hop Style

With the next hop style you use the routing table in the virtual router's instance to steer traffic to services. Only traffic that is destined for a specific next hop is serviced by the service set.

In general, next hop style provides more flexibility than interface-style.

NOTE Only one next hop style service set per `sp` interface is allowed.

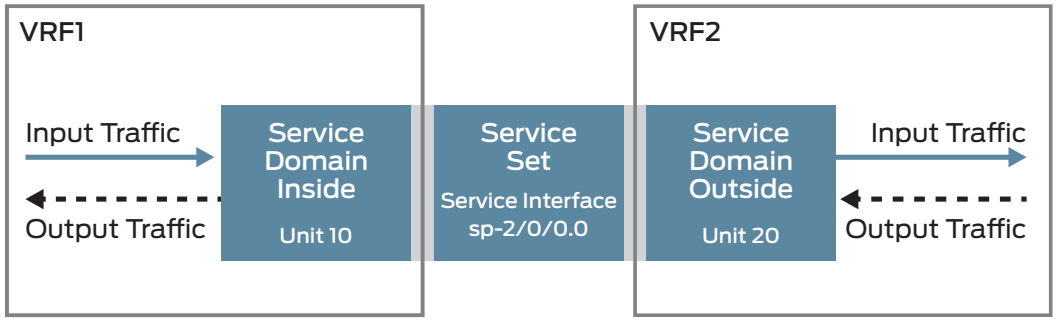


Figure 1.2 Illustration Showing Next Hop Style Service Set

Let's look at the `interface-style service-set` configuration and compare it to the next hop style. For the interface-style you actually edit the actual physical interfaces that handle the client end's egress traffic. This is typically known as the *private network* in a NAT setup. Under these interface(s) you will connect to the service-set.

Under 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. This setting simply points to your `si`, `sp` or `rsp` interfaces that you want to use based on whether you are using inline static NAT, dynamic NAT, or dynamic NAT with a redundant PIC. Let's start here:

```
[edit services]
service-set nat44 {
   nat-rules rule2;
   interface-service {
      service-interface rsp1;
   }
}
```

Then, under the 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 here. This simply means that *all* traffic processed by this interface's PFE will be passed to the service-interface attached to the service-set for NAT servicing via the input setting you just added. Then the service-interface sends the post serviced traffic back to the PFE via the output option, as shown here:

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

```
        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 that this physical interface receives gets serviced by the Service PIC, even if the NAT rules never actually NAT the packet. So, based on the NAT rules you have attached to the service set, a packet will be steered to the service PIC but may not be NAT'd and then passed right back to the PFE of the interface to proceed along to its destination target based on the MX's routing table. So, next hop style may be the style for you, once armed with this knowledge. Let's see.

The next hop style is more complicated but offers more control over what packets get processed by the service PIC. Let's create the next hop style service set and go from there. As you will see the service set is configured very similarly – 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`, `sp`, or `rsp` 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 rsp1.1;
        outside-service-interface rsp1.2;
    }
```

Then, under the service interface hierarchy, you create two logical units. One handles service traffic for ingress and the other handles service traffic for egress. Remember with service interfaces that the inside interface is the one that handles egress traffic and the outside interfaces are the ones that pass the post serviced traffic to the next step in the ingress path:

```
[edit interfaces rsp1]
redundancy-options {
    primary sp-11/1/0;
    secondary sp-10/0/0;
    hot-standby;
```

```
}
services-options {
   cgn-pic;
}
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 service interface for CGNAT processing:

```
[edit]
routing-options {
   static {
      route 197.100.1.0/24 next-hop rsp1.1;
      route 189.1.1.0/24 next-hop rsp1.1;
   }
}
```

Let's look at a simple example showing the inside service interface and outside interface sitting in different VRs. Both the physical interface that receives the egress traffic and the service interface rsp.1, which was 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 rsp1.1;
routing-options {
   static {
      route 197.100.1.0/24 next-hop rsp1.1;
      route 189.1.1.0/24 next-hop rsp1.1;
   }
}
```

Then you set up the virtual router that handles the engress traffic. This is done by adding the interface that faces the egress targets and by adding the service interface unit that you tied to the outside service domain – that interface being rsp1.2:

```
[edit routing-instances VR2]
root@JTAC_setup-re0# show
instance-type virtual-router;
interface xe-3/0/2.0;
interface rsp1.2;
```

At this point, after you have your service set configured, and you look at the routes on the box, you should see any traffic destined to be NAT'd will get sent to the service interface, in this case represented by the service set name `nat44`:

```
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 rsp1.1
```

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

If you are getting lots of drop flows, due to port scans and the like, set the `max-flows-drop` under the service set to limit the amount of drop flows held in memory. You can set this for egress and ingress. Though it should be noted that the creation of drop flows can save the service PIC CPU cycles, so lower this setting with care. Trying to be efficient with memory can hurt the CPU efficiency.

```
[edit services]
service-set nat44 {
   max-drop-flows ingress 10 egress 5;
   nat-rules rule2;
   next-hop-service {
          inside-service-interface rsp1.1;
          outside-service-interface rsp1.2;
   }
```

With `napt-44`, if PBA is not used, there is nothing to stop any single internal private IP address from taking up a whole boatload of ports until 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, 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 firewall, so it's two flows:

```
[edit services]
service-set nat44 {
   max-drop-flows ingress 10 egress 5;
   nat-rules rule2;
   nat-options {
         max-sessions-per-subscriber 10000;
    }
   next-hop-service {
          inside-service-interface rsp1.1;
          outside-service-interface rsp1.2;
   }
```

You can also limit the maximum number of flows a service set will allow at any given time by employing the `max-flows` configuration option like this:

```
[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 rsp1.1;
           outside-service-interface rsp1.2;
   }
```

## Summary

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

Let's go over what you have done so far, and if need be, you should review before moving on:

- Configured a service interface, either `si` for inline, `or sp` or `rsp` for a service PIC.
- Configured a services NAT pool to assign translated IP addresses, and optionally, ports to the flows you want to NAT.
- Configured a service NAT rule where you assigned the NAT translation type and defined the private IP addresses that should be NAT'd. From here you also tied in the services NAT pool to use for the flow that hits 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 flows.
- Tied the services `service-set` to your physical interface using the `interface-style` setup…or….you tied the services `service-set` to your `service-interface` which then gets used based on static routes, when using the next hop style setup.

# Chapter 2

## Additional Features

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

### Address Pooling Paired

Address pooling paired (APP) is a feature that makes the MX 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.

Why use this feature? It solves the problems that can occur when client to application communication opens multiple connections. Some examples are:

- If a SIP client is sending RTP and RTCP, it should be expected that they come from the same IP address, even after they go through a NATing device.
- Any P2P protocol that negotiated ports assuming address stability will benefit from address pooling paired being used so the NAT'd IP address is the same for all flows from a private host.

Remember, if you are a private host sending dozens of different flows 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 APP only works with NAT Translation types that use the service card. So translation types like `basic-nat44` cannot leverage this feature.

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

```
[services nat]
rule rule1 {
   match-direction input;
   term t1 {
      from {
         source-address {
            14.0.0.0/14;
         }
      }
      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 will allocate ports from the NAT Pool in the default mode, which, based on the traffic through the MX, could and most likely will result in many internal private IPs being mapped to the same public facing NAT'd IP. This will eventually cause source port exhaustion, where there are other available IPs to be NAT'd that still do not have any ports tied up.

Port exhaustion for a NAT'd IP address is bad because the MX will drop traffic once no more ports are available. When address pooling paired is not used, it is not a big deal since the additional subscribers' flows would just start to use the next available IP address and any related ports that are available. But when the address pooling paired feature maps the private IP to the same public IP, the MX will not move on to the next available IP. It will drop any new flows until the private IP address it is mapped to has a free port, which could take some time, based on the different timeout settings you may have.

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

Remember `address-allocation` with `round-robin` randomly chooses an available IP address and port for each new flow. The example shown here uses `napt-44` as our translation-type:

```
[edit services nat]
pool natpat44 {
   address 100.100.0.0/16;
   port {
      automatic {
         random-allocation;
      }
   }
   address-allocation round-robin;
   mapping-timeout 120;
}
```

The effects of `address-allocation round-robin` can be seen here, where each flow tends to have a random NAT translated IP and port assigned to it – in this example the private client at 14.0.16.6, sending TCP and UDP traffic to a public server at 197.100.1.1. For each flow, regardless of the source, you can see that a different NAT'd IP address is used:

```
root@JTAC_setup-re0# run show services stateful-firewall flows terse
Interface: rsp1, Service set: nat44
Flow                                      State   Dir      Frm count
UDP         14.0.16.6:2002   ->    197.100.1.1:2003   Forward  I            6
   NAT source      14.0.16.6:2002    ->   100.100.0.13:56332
UDP         14.0.16.6:66     ->    197.100.1.1:2004   Forward  I           60
   NAT source      14.0.16.6:66      ->    100.100.0.7:1004
TCP         14.0.16.6:55951  ->    197.100.1.1:2001   Forward  I            3
   NAT source      14.0.16.6:55951   ->   100.100.0.14:44949
```

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

```
pool natpat44 {
   address 100.100.0.0/16;
   port {
      automatic {
         random-allocation;
      }
   }
   address-allocation round-robin;
}
rule rule1 {
   match-direction input;
   term t1 {
      from {
         source-address {
            14.0.0.0/14;
         }
      }
      then {
         translated {
            source-pool natpat44;
            translation-type {
               napt-44;
```

```
            }
            address-pooling paired;
         }
      }
   }
}
```

You can see that the same unique NAT'd IP address is being mapped for our private IP address flow. It is now a one-to-one mapping, unlike before, when address-pooling paired was not used and any free NAT'd and IP port could be mapped to any flow:

```
root@JTAC_setup-re0# run show services stateful-firewall flows terse
Interface: rsp1, Service set: nat44
Flow                                    State   Dir      Frm count
UDP         14.0.16.6:2002   ->    197.100.1.1:2003   Forward  I           67
   NAT source      14.0.16.6:2002    ->    100.100.0.3:50424
UDP         14.0.16.6:66     ->    197.100.1.1:2004   Forward  I          669
   NAT source      14.0.16.6:66      ->    100.100.0.3:846
TCP         14.0.16.2:55951  ->    197.100.1.1:2001   Forward  I            9
   NAT source      14.0.16.6:55951   ->    100.100.0.3:53303
```

One thing to it's important to understand is that when the a private subscriber stops sending or receiving traffic for a flow, the flow 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 flows for a given subscriber timeout, the MX will keep the address-pooling paired mapping in memory for five minutes by default. So if the subscriber again sends data from the same private IP address, the MX will once again map the private IP address to the same NAT address. If five minutes have passed the MX will use the standard logic based on settings like address-allocation round-robin on what IP address to use as the NAT'd IP. So your end users may very well have a new NAT'd address if they sporadically send data flows.

You can change how long the MX 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 156.100.0.0/16;
   port {
      automatic {
         random-allocation;
      }
   }
   address-allocation round-robin;
   mapping-timeout 1200;
}
```

Since we mentioned inactivity timeout for flows let's quickly look how we can change the inactivity timeout from its default 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 flows anchored on this interface:

```
[edit interfaces rsp1 services-options]
inactivity-timeout 20;
```

## End Point Independent Mapping (EIM)

Another feature of the MX Series is End Point Independent Mapping (EIM), which is used to assign the same external address and port for a flow from a given host. If other flows from the host come from a different source port, the MX assigns a different external address and port for the NAT'd traffic. With EIM a host can send traffic to ten different targets on the public side and if these ten flows all use the same internal IP address and port, the ten targets on the public facing side will 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, the MX is free to choose *any* free IP address and port combo to assign to the NAT'd traffic.

NOTE It should be noted that EIM only works with NAT Translation types that use the service card. So translation types like `basic-nat44` cannot leverage this feature.

The feature is enabled by setting `mapping-typtte` 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 what is required for assigning the same external IP address for all flows from an internal IP for a period of time, while EIM means you have a same external IP address and port mapped to an internal IP address and port for a period of time.

EIM, like address pooling paired, also has a period of time where the mapping is held even after traffic flows have timed out. Once the flow is dropped on the MX, when data is not sent or received for the duration of the inactivity timeout, which by default is 30 seconds, the MX will hold the End Point Mapping in memory for five minutes, the default EIM mapping timeout period. So within this five-minute period, if the flow 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 have the same NAT'd external IP address and port assigned to this flow. And if five minutes have passed before the flow is re-initiated by the private subscriber, the MX will assign a new NAT'd IP and Port to the flow.

Also like address pooling paired, this mapping timeout default of five minutes can be changed by editing the same `mapping-timeout` value (in seconds) under the NAT pool settings that was changed for the address pooling paired mapping. Note this means that the address pooling paired and EIM mapping timeouts are tied to the same value when they use the same pool, though functionally there should be no reason for these features to have different values:

```
[edit services nat]
pool natpat44 {
   address 100.100.0.0/16;
   port {
      automatic {
         random-allocation;
      }
  }
   address-allocation round-robin;
   mapping-timeout 1700;
}
```

Now, if your rule is set up to use both the *address pooling paired* feature and the *endpoint independent mapping* feature (such as in the example shown next) there are a few things to be aware of about how `mapping-timeout` works when both are used. First, due to the address pooling paired option, the MX 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.

When the individual traffic flows expire due to inactivity, the MX will timeout the endpoint independent mapping first. Once all the flows for the subscribers private source IP addresses are removed and all of the endpoint independent mappings are expired, then and only then does it start the count down to expire the address pooling paired mapping. As long as one flow is active or one endpoint independent mapping 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 {
            14.0.0.0/16;
        }
    }
    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 the EIM feature alone 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 after the outbound connection has been established, EIF needs to be configured in conjunction with EIM. This means that when the subscriber private IP is used to create a flow to a target, the target, and even other public nodes, can later send traffic inbound to the NAT'd address as long as it is sending to the source port the subscriber flow was created with, and the mapping-timeout for the EIM mapping has not expired.

So to set up an EIF example, let's add a filtering type of `endpoint-independent` and add a `prefix-list`. The prefix list determines which public nodes are allowed to establish inbound connections back through the stateful firewall using the EIM mapping. This actually allows you to control what targets can use EIM/EIF for inbound connections versus just allowing all, which could be dangerous and unwanted (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):

```
[edit services nat]
   rule rule2 {
      match-direction input;
      term t1 {
         from {
            source-address {
               14.0.0.0/16;
            }
         }
         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;
            }
         }
      }
   }
}
```

Then under the policy-options hierarchy add a prefix list for the individual public hosts or public subnets you want the MX to allow traffic to be received from and forward it on to the private client IP address:

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

BEST PRACTICE

EIM and address pooling paired uses a small mapping of memory for each flow it is attached to. 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 and/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 potentially allow public nodes to be able to create a large number of flows, 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 and/or a DoS attack on the MX, since it consumes memory through each flow.

To help protect the MX, and the subscribers that sit on the private network, the MX has a feature called `eif-flow-limit`. To limit the number of inbound connections on an EIM 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 {
               14.0.0.0/16;
            }
         }
         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 the private address at 14.0.0.33:44444 -> sends traffic to a public target at 90.0.0.11:55, With EIM/EIF enabled our NAT is mapping 156.100.0.11:61824 that public addresses set under our prefix- list can reach:

```
lab@JTAC-re0> show services nat mappings endpoint-independent
Interface: sp-5/0/0, Service set: SS2

NAT pool: nat44
Mapping        : 14.0.0.33    :44444  --> 156.100.0.11    :61824
Session Count    :    6
Mapping State    : Active
```

Now targets on the public side can reach back into 14.0.0.33 by sending traffic to 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 you are leaving it wide open:

```
UDP         90.0.0.11:44444  ->    156.100.0.11:61824  Forward  0       2314777
   NAT dest     156.100.0.11:61824   ->    14.0.0.33:44444
  Byte count: 79623716
  Flow role: Master, Timeout: 59898
UDP         90.0.0.10:44444  ->    156.100.0.11:61824  Forward  0       2373098
   NAT dest     156.100.0.11:61824   ->    14.0.0.33:44444
  Byte count: 82128564
  Flow role: Master, Timeout: 59897
UDP         231.10.110.89:44444  ->    156.100.0.11:61824  Forward  0        665341
   NAT dest     156.100.0.11:61824   ->    14.0.0.33:44444
  Byte count: 22880266
  Flow role: Master, Timeout: 59916
UDP         130.0.0.88:44444  ->    156.100.0.11:61824  Forward  0        690824
   NAT dest     156.100.0.11:61824   ->    14.0.0.33:44444
  Byte count: 23855794
  Flow role: Master, Timeout: 59916
```

If these same targets try to reach into this NAT'd address on any source ports that have not been created and mapped by the private addresses, the traffic will be dropped.

```
UDP         90.0.0.11:44444  ->    156.100.0.11:9000   Drop    0         97730
  Byte count: 3346280
  Flow role: Initiator, Timeout: 4
UDP         90.0.0.10:44444  ->    156.100.0.11:9000   Drop    0        105284
  Byte count: 3597302
  Flow role: Initiator, Timeout: 4
UDP         231.10.110.89:44444  ->    156.100.0.11:6165   Drop    0         21157
  Byte count: 725900
  Flow role: Initiator, Timeout: 4
UDP         130.0.0.88:44444  ->    156.100.0.11:6165   Drop    0         23302
  Byte count: 796722
  Flow role: Initiator, Timeout: 4
```

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. With the EIF feature enabled on the MX, 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 end CPE has powered off UDP, traffic can keep coming. This could mean many flows still showing as active on the MX, tying up resources for no reason. So, 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 flow is inactive or not. So once an outbound flow from the private network 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 rule2 {
      match-direction input;
      term t1 {
         from {
            source-address {
               14.0.0.0/16;
            }
         }
         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;
            }
         }
      }
   }
}
```

## DS-Lite and PCP

The MX supports both DS-Lite and PCP, a technology combination that many service providers are using to rollout IPv6 in what's still very much an IPv4 world.

The Port Control Protocol (PCP) controls the forwarding of incoming packets by public nodes through devices such as the MX when acting as a NAT device to clients on the private network.

The setup on the MX is quite simple – under `service-set` add a `softwire-rule` and a `PCP-rule`, like so:

```
services {
   service-set DSLITE-TEST {
      softwire-rules RULE-AFTR1;
      pcp-rules PCP-RULE1;
      nat-rules NAT-RULE11;
      next-hop-service {
         inside-service-interface sp-1/0/0.101;
         outside-service-interface sp-1/0/0.102;
      }
   }
```

Under the services softwire configuration let's add the rule added to the service set and then add the `softwire-concentrator` configuration that will make the MX an actual softwire concentrator.

In this example, let's give the MX the IPv6 address of 2a01:1:1:0500:c:c:cccd:1234, which will be the IP address DSLITE clients use to reach the MX. Set the IPv6 MTU here and set a flow-limit that will limit the number of active flows for this concentrator if you want to make sure softwire flows do not take up too many resources, thus starving other flows through the stateful firewall.

For the rule itself, there is not much to configure. You state the direction from which you want to match traffic and then you point to the DSLITE concentrator you want to use:

```
services{
   softwire {
      softwire-concentrator {
         ds-lite AFTR1 {
            softwire-address 2a01:1:1:0500:c:c:cccd:1234;
            mtu-v6 1492;
            flow-limit 16384;
         }
      }
      rule RULE-AFTR1 {
         match-direction input;
         term 1 {
            then {
               ds-lite AFTR1;
```

```
            }
         }
      }
   }
```

For the services PCP configuration, you add the PCP rule added to the service-set and then add the PCP server configuration that will make the MX able to anchor PCP requests.

In this example let's give the MX the same IPv6 address of 2a01:1:1:0500:c:c:cccd:123 used for the softwire-concentrator, which will be the IP address PCP clients use to reach the MX. Here you can also state how long you want your PCP mappings to be active.

Again, for the PCP rule, there is not much to configure. You state the direction from which you want to match traffic and then you point to the PCP server you want to use:

```
services{
   pcp {
      server PCP-SERVER1 {
         ipv6-address 2a01:1:1:0500:c:c:cccd:1234;
         softwire-concentrator AFTR1;
         mapping-lifetime-minimum 120;
         mapping-lifetime-maximum 86400;
         max-mappings-per-client 100;
         pcp-options third-party prefer-failure;
      }
      rule PCP-RULE1 {
         match-direction input;
         term 10 {
            then {
               pcp-server PCP-SERVER1;
            }
         }
      }
   }
}
```

As for the NAT rule, this is used to assign NAT'd IP addresses to the IPv4 private IP address the MX services from the DSLITE client. This rule is the same as any NAT rule that is used. Same with the pool. Once DS-Lite is decapsulated at the MX DS-Lite concentrator's end, nothing changes NAT-wise to the MX. It will just be a regular IPv4 flow that needs to be NAT'd:

```
rule NAT-RULE1 {
         match-direction input;
         term 1 {
            from {
               applications [junos-ftp ];
            }
            then {
               translated {
```

```
            source-pool POOL2;
            translation-type {
               napt-44;
            }
            mapping-type endpoint-independent;
            filtering-type {
               endpoint-independent;
            }
         }
      }
   }
```

# Chapter 3

## Application Layer Gateways

Application Layer Gateways (ALGs), our "Layer 7 DPI lite," get their own chapter. This chapter goes above and beyond the five tuple match rules many people use for filtering and uses our DPI code to look under the hood in order to give an operator better control. It allows the MX CGNAT features to work with certai,n applications that need to have some of their characteristics changed to work in a NATing environment. And it is one of the strong functionalities that the service cards bring to the table.

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

- FTP
- TFTP
- RTSP
- PPTP
- DNS
- MSRPC
- SUNRPC
- TALK
- RSH
- SQL-NET
- SIP
- ICMP

This book could spend dozens of pages talking about the functionality of each ALG the MX supports, but since this is a *Day One* book, this chapter briefly covers a few of the ALGs and then shows you how to configure them.

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

Let's first look at how we generically enable ALGs using the ICMP ALG to help 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 handle ICMP events when NAT is used, since a NATing device requires the source IP address to be rewritten for events like the ICMP type *destination unreachable* with a code of Network Unreachable. When this event occurs, the ICMP packet sent back to the MX from a private host has the source address in the ICMP/IP payload set to the NAT'd address. The MX then needs to have the service card re-write the packet so that this source address under the ICMP/IP payload is translated back to the subscriber's private address. This magic to re-write data in the ICMP layer is performed by the ICMP ALG!

Okay, so let's create an ALG example using ICMP:

```
[edit applications application ICMP_ALG]
application-protocol icmp;
protocol icmp;
```

You can then add to a rule with a given rule to be used only for certain application data, if you wish:

```
[edit services nat rule rule2 term t1]
from {
   source-address {
      100.100.0.0/16;
      139.97.68.0/24;
   }
   applications ICMP_ALG;
}
```

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

```
root@JTAC_setup-re0> show services stateful-firewall conversations
Interface: rsp1, Service set: nat44
Conversation: ALG protocol: icmp
  Number of initiators: 1, Number of responders: 1
Flow                                    State    Dir      Frm count
ICMP      139.97.68.41       ->    197.100.1.1       Watch    I           136
   NAT source   139.97.68.41        ->    100.100.0.1
ICMP       197.100.1.1       ->    100.100.0.1       Watch    O           136
   NAT dest      100.100.0.1        ->    139.97.68.41
```

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

```
root@JTAC_setup-re0> show services stateful-firewall conversations
Interface: rsp1, Service set: nat44
Conversation: ALG protocol: icmp
  Number of initiators: 1, Number of responders: 1
Flow                                    State   Dir      Frm count
ICMP      139.97.68.41       ->    197.100.1.1       Watch   I          136
   NAT source   139.97.68.41        ->    100.100.0.1
ICMP       197.100.1.1       ->    100.100.0.1       Watch   O          136
   NAT dest      100.100.0.1        ->   139.97.68.41
```

They are identical! It should be stated that the ICMP ALG is used by default but there are items you can configure to give you control over what you actually do NAT. In the example below, the `ICMP-type` option and `state echo-request` have been added to the ALG configuration. Based on the direction of the rule, you now have a setup where only ICMP echo requests, a.k.a. *pings* from the subscriber's private IP addresses, will be NAT'd. All other ICMP packets would just be forwarded and the NAT logic would not touch them.

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

TIP There are many more ICMP-codes and ICMP-type options you can set under the ICMP ALG.

This is a good place to talk about the stateful firewall and about the state of these different flows that you have seen in the output. *Forward*, *Watch*, and *Drop* flows have been seen in our different examples.

- Watch flows are nothing but ALG-specific flows such as ICMP, FTP, TFTP, and SIP to name a few. A watch flow state indicates that the control flow is monitored by the ALG for information in the payload. NAT processing is performed on the header and payload as needed. Think of the watch flow as a pinhole in the firewall allowing the required packets from the public side to be able to reach the private client.t
- A Forward flow just forwards the packets without monitoring the payload as NAT is performed on the header as needed.
- A Drop flow just drops any packets that match the flow without having to go to the Service PIC for additional processing.

So, looking back at out ICMP ALG setup, the I/O flows are Watch flows because the ICMP header is an error code such as `TTL exceed` or `destination unreachable`. In these cases the MX knows to convert the header IPs from public to private, but it also knows that it needs to change the payload.

Other protocols like HTTP, SSH, and TELNET may not need to have their payload processed and are just forwarded, so you don't have these *Watch* flows.

Let's look at another ALG and a use case where it may be desired or even required – the DNS ALG. DNS traffic can at times be quite heavy but normally it is short-lived UDP queries that can actually tie up flows.

The inactivity timeout you have set on the MX in our example is seven minutes as seen here:

```
sp-3/0/0 {
       services-options {
          inactivity-timeout 420;
          tcp-tickles 0;
       }
       unit 0 {
          family inet;
       }
   }
```

But why tie up a flow and resources for a protocol like DNS for seven minutes or even the default inactivity timeout if 30 seconds? You can set up the DNS ALG and set a low inactivity timeout to help efficiently clean up these flows. Here is a simple example that defines the DNS ALG under the [`edit applications`] hierarchy. Note in our example you want to set the term that calls the DNS ALG under the NAT rule before the generic catch all rule that will handle all other traffic types:

```
[edit applications]
lab@JTAC_setup-re0# show
application junos-dns-udp {
   protocol udp;
   destination-port 53;
   inactivity-timeout 10;
}


[edit services nat rule rule1]
lab@JTAC_setup-re1# show
match-direction input;
term t1 {
   from {
      source-address {
         0.0.0.0/0;
      }
      applications junos-dns-udp;
   }
   then {
      translated {
         source-pool nat44;
         translation-type {
            napt-44;
```

```
            }
            address-pooling paired;
        }
    }
}
term t2 {
    from {
        source-address {
            0.0.0.0/0;
        }
    }
    then {
        translated {
            source-pool nat44;
            translation-type {
                napt-44;
            }
            address-pooling paired;
        }
    }
}
```

BEST PRACTICE

If 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. So there is no predefined ALG that requires EIM/EIF, and if you use EIM/EIF with traffic matching any of the Junos predefined ALGs, you are actually wasting the Service PICs memory for EIM/EIF mappings that are not required. A recommended configuration for service providers 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;
    application junos-ipsec-esp;
    application junos-ike;
}
```

Then you 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 t0 {
            from {
```

```
            source-address {
                192.168.0.0/16;
            }
            application-sets accept-alg;
        }
        then {
            translated {
                source-pool nat44;
                translation-type {
                    napt-44;
                }
            }
        }
    }
    term t1 {
        from {
            source-address {
                192.168.0.0/16;
            }
        }
        then {
            translated {
                source-pool nat44;
                translation-type {
                    napt-44;
                }
                mapping-type endpoint-independent;
                filtering-type {
                    endpoint-independent;
                }
            }
        }
    }
}
```

# Chapter 4

## Syslog

Syslogging is a very important activity for many Service Providers who need to track which subscriber used which public address via their private address and port combo at what time. Without syslogging the Service Provider has no means by which to make this match.

The MX 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 Routing Engine due to the limited space on the Routing Engine and the fact that the RE will now have to process each syslog. In fact, as you go through this chapter, keep in mind that syslogging puts extra strain on service PIC processing and that you only should enable syslogging if 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 eliminate any messages having to be created for a NAT flow.

Now, on to our syslog setup. There are three key control points in the Junos hierarchy where the MX can capture the NAT data to write to syslog:

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

The service-interface is the highest point in the Junos hierarchy to which you can enable the syslog functionality for CGNAT. If you set it here, syslog will be enabled across all service sets that use this

service interface. The syslog setting under the 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 for only critical issues:

```
[edit interfaces sp-7/0/0]
services-options {
   syslog {
      host local {
         services critical;
```

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

```
lab@JTAC_setup-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 PIC7. 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 if you need to:

```
[edit interfaces sp-7/0/0]
services-options {
   syslog {
      host local {
         services any;
         log-prefix PIC7;
      }
   }
   open-timeout 30;
   close-timeout 10;
   inactivity-timeout 30;
   tcp-tickles 0;
   cgn-pic;
```

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 service PIC is hitting performance issues. Many Service Providers would rather drop syslog messages in order to process flows:

```
[edit interfaces sp-7/0/0 services-options syslog]
lab@JTAC_setup-re0# set ?
 message-rate-limit   Maximum syslog messages per second allowed from this interface
(messages per second)
```

If you do not want to have syslogging enabled at the service interface level, you can set it at the individual service sets. Here you can apply the log-prefix and services filter, but you can also set a class filter. Here are the classes you can filter on:

- `alg-logs` – Log application-level gateway events
- `nat-logs` – Log Network Address Translation events
- `packet-logs` – Log general packet-related events
- `session-logs` – Log session open and close events
- `session-logs open` – Log session open events only
- `session-logs close` – Log session close events

```
[edit services service-set nat44]
 syslog {
    host local {
        services any;
        log-prefix service_set_nat44;
        class {
            session-logs;
            packet-logs;
            stateful-firewall-logs;
            alg-logs;
            nat-logs;
        }
    }
}
```

NOTE If you enable syslogging at the service-set level, it overrides the syslog setting you may have enabled at the service-interface.

The last place you can also set syslogging is at the `services / nat / rule` hierarchy but this cannot be filtered at the class and service level, so it's not recommended to enable syslogging in a production environment.

```
[edit services nat rule rule1 term t1]
from {
    source-address-range {
        low 192.168.0.1 high 192.170.127.254;
    }
}
then {
     translated {
        source-pool natpat44;
```

```
        translation-type {
            napt-44;
        }
    }
    syslog;
}
```

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

```
[edit services service-set nat44]
lab@JTAC_setup-re0# run show services service-sets statistics syslog
Interface: sp-7/0/0
  Rate limit: 10000
  Sent: 0
  Dropped: 0
  Service-set: nat44
    Sent: 1014530
    Dropped: 0
```

If you disable the syslog stanza under the `services / nat / rule` and `services / service-set` hierarchies, you'll see that the service interface starts to be the functional point where these events are controlled:

```
[edit services service-set nat44]
lab@JTAC_setup-re0# run show services service-sets statistics syslog
Interface: sp-7/0/0
  Rate limit: 10000
  Sent: 750
  Dropped: 0
```

Next is an example of some syslog messages. In this example, you can see an ICMP flow being created and then deleted. See how the messages are pre-pended with the log-prefix PIC7. The logging of flows like this can be enabled or disabled under the service-set by setting or removing the class type session-logs:

```
Line 33: Aug 31 00:58:43  JTAC_setup-re0 (FPC Slot 11, PIC Slot 1) 2013-08-30 21:58:43
PIC7{nat44}[FWNAT]: ASP_SFW_CREATE_ACCEPT_FLOW  139.97.68.41:64841 [100.100.0.7:50103]
->  189.1.1.4:2048  (ICMP ECHO REQUEST)

Line 54: Aug 31 00:59:37  JTAC_ setup-re0 (FPC Slot 11, PIC Slot 1) 2013-08-30 21:59:37
PIC7{nat44}[FWNAT]:ASP_NAT_POOL_RELEASE: natpool release 100.100.0.7:50103[1]

Line 55: Aug 31 00:59:37  JTAC_ setup-re0 (FPC Slot 11, PIC Slot 1) 2013-08-30 21:59:37
PIC7{nat44}[FWNAT]: ASP_SFW_DELETE_FLOW  139.97.68.41:64841 [100.100.0.7:50103] ->
189.1.1.4:2048  (ICMP ECHO REQUEST)
```

Let's look at the Port Block Allocation (PBA) feature again, this time with syslogging enabled. Since PBA allocates a block of ports to a private address, the MX does not need to log the creation and deletion of every flow. So it is recommended that you remove the class type session-logs. Make sure you set the class type `nat-logs` though, so you can see the port blocks being allocated. Below is an example of what will be sysloged when PBA is being used and the `nat-log` class is set and the `session-log` class is not:

```
Oct 24 10:19:04.708  JTAC_ setup-re0 (FPC Slot 7, PIC Slot 0) 2013-10-24 17:19:04 PIC7
[FWNAT]:ASP_NAT_PORT_BLOCK_ALLOC: 192.169.0.188 -> 100.100.0.12:54272-55295
Oct 24 10:19:04.733  JTAC_ setup-re0 (FPC Slot 7, PIC Slot 0) 2013-10-24 17:19:04 PIC7
[FWNAT]:ASP_NAT_PORT_BLOCK_ALLOC: 192.169.0.12 -> 100.100.0.12:55296-56319
Oct 24 10:19:04.758  JTAC_ setup-re0 (FPC Slot 7, PIC Slot 0) 2013-10-24 17:19:04 PIC7
[FWNAT]:ASP_NAT_PORT_BLOCK_ALLOC: 192.169.0.193 -> 100.100.0.12:56320-57343
Oct 24 10:19:05.096  JTAC_ setup-re0 (FPC Slot 7, PIC Slot 0) 2013-10-24 17:19:04 PIC7
[FWNAT]:ASP_NAT_PORT_BLOCK_ALLOC: 192.169.0.134 -> 100.100.0.12:57344-58367
Oct 24 10:19:05.526  JTAC_ setup-re0 (FPC Slot 7, PIC Slot 0) 2013-10-24 17:19:04 PIC7
[FWNAT]:ASP_NAT_PORT_BLOCK_ALLOC: 192.169.0.200 -> 100.100.0.12:58368-59391
Oct 24 10:19:05.649  JTAC_ setup-re0 (FPC Slot 7, PIC Slot 0) 2013-10-24 17:19:05 PIC7
[FWNAT]:ASP_NAT_PORT_BLOCK_RELEASE: 192.169.0.3 -> 100.100.0.6:43008-44031 0x52695579
Oct 24 10:19:05.875  JTAC_ setup-re0 (FPC Slot 7, PIC Slot 0) 2013-10-24 17:19:05 PIC7
[FWNAT]:ASP_NAT_PORT_BLOCK_ALLOC: 192.169.0.74 -> 100.100.0.12:59392-60415
Oct 24 10:19:06.351  JTAC_ setup-re0 (FPC Slot 7, PIC Slot 0) 2013-10-24 17:19:05 PIC7
[FWNAT]:ASP_NAT_PORT_BLOCK_ALLOC: 192.169.0.129 -> 100.100.0.12:60416-61439
Oct 24 10:19:06.649  JTAC_ setup-re0 (FPC Slot 7, PIC Slot 0) 2013-10-24 17:19:06 PIC7
[FWNAT]:ASP_NAT_PORT_BLOCK_RELEASE: 192.169.0.81 -> 100.100.0.7:23552-24575
0x526955a2
Oct 24 10:19:07.284  JTAC_ setup-re0 (FPC Slot 7, PIC Slot 0) 2013-10-24 17:19:06 PIC7
[FWNAT]:ASP_NAT_PORT_BLOCK_ALLOC: 192.169.0.89 -> 100.100.0.12:61440-62463
Oct 24 10:19:08.142  JTAC_ setup-re0 rshd[38297]: root@re1 as root: cmd='/sbin/sysctl
net.inet.ip_control_plane'
Oct 24 10:19:08.886  JTAC_ setup-re0 (FPC Slot 7, PIC Slot 0) 2013-10-24 17:19:08 PIC7
[FWNAT]:ASP_NAT_PORT_BLOCK_ALLOC: 192.169.0.143 -> 100.100.0.12:62464-63487
```

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 maps to which NAT'd public IP address and port. It makes syslogging practically unnecessary.

It should also be noted that the syslog messages can be used to troubleshoot general issues with the CGNAT setup. For example, when flow limits have been exceeded the following is an example of a syslog message you'd receive:

```
Oct  9 12:30:52.071  JTAC_ setup-re0 (FPC Slot 7, PIC Slot 0) 2013-10-09 19:30:51
NAT44{nat44}[FWNAT]:ASP_SVC_SET_MAX_FLOWS_EXCEEDED: Number of flows (currently 10000)
exceeded configured limit (10000) 37865 times in previous 60 seconds
```

## Setting Up Filters

Since you have some options, let's review filtering in regard to NAT for a moment. Let's use the example that you do *not* want to NAT ICMP packets, so there's three options you can use.

First, you can create a service filter and attach it to the physical interfaces so you don't have to even go to the service PIC to NAT ICMP packets from the private network. Note this does not drop the packet:

```
[edit firewall family inet service-filter ICMP]
lab@MBG2# show
term t1 {
    from {
        source-address {
            10.0.0.0/0;
        }
        protocol icmp;
    }
    then skip;
}

[edit interfaces xe-0/3/0]
lab@MBG2# set unit 0 family inet service input service-set SSET101 service-filter ICMP

[edit interfaces xe-0/3/0]
lab@MBG2# set unit 0 family inet service output service-set SSET101 service-filter ICMP
```

Secondly, you could do something similar but at the service PIC itself. Once again, this does not drop the packet:

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

[edit services nat rule test_rule term t1]
from {
    source-address {
        10.0.0.0/8;
    }
    applications ICMP;
}
then {
    no-translation;
}
```

These first two types of filters are dangerous because they can show the public world the private IP address since NAT has been removed for ICMP packets. The third option is to use the Stateful Firewall on the SPIC. This can allow you to drop *only* traffic from the private network, and only certain defined ICMP types, if required:

```
[edit applications application ICMP]
protocol icmp;
icmp-code host-unreachable;
```

```
[edit services stateful-firewall]
rule SFW_Rule {
    match-direction input-output;
    term t1 {
        from {
            source-address-range {
                low 10.0.0.0 high 255.255.255.255;
            }
            applications ICMP;
        }
        then {
            discard;
        }
    }
}
```

And you attach the `stateful-firewall-rules SFW_Rule` right under the service set:

```
[edit services service-set nat44]
max-flows 7500000;
stateful-firewall-rules SFW_Rule;
nat-rules rule2;
interface-service {
    service-interface sp-7/0/0;
}
```

## Setting up Load Balancing on the MX for CGNAT

Load balancing is obviously used when you have the potential for more data or packets per second to be passed through a service PIC than the service PIC can handle. So enabling load balancing allows you to spread the flows 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 set up 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 flows need to be recreated on the second service PIC using a different NAT Pool. Since the NAT'd IP, and potentially also the port, will have different features like Address Pooling Paired, EIM, and EIF scenarios will take a momentary hit as the applications on the client end need to 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 flows stop going through the box until the offending service PIC is recovered.

Let's show how to set up load balancing with the MS-DPC card.

First you need to set up the forwarding option 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 flow. Do not worry that the key word is `per-packet` – this will actually work per flow:

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

Then in the default routing instance you need to add this policy just created 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 egress interfaces that will receive data from a virtual router. 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 added to the load balance pool to be chosen from:

```
[edit routing-instances inside]
instance-type virtual-router;
interface xe-3/1/0.0;
interface xe-4/2/1.0;
interface sp-10/0/0.100;
interface sp-11/0/0.100;
routing-options {
   static {
      route 70.100.0.0/24 next-hop [ sp-10/0/0.100 sp-11/0/0.100 ];
   }
}
```

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 sp-10/0/0.100;
        outside-service-interface sp-10/0/0.101;
    }
}
service-set nat44_b {
    nat-rules rule2;
    next-hop-service {
        inside-service-interface sp-11/0/0.100;
        outside-service-interface sp-11/0/0.101;
    }
}
```

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;
            }
    }
```

Then we create our two NAT rules that each call 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;
            }
         }
      }
   }
}
```

Now this setup should work for most scenarios. One of the main weaknesses, though, is whether you have limited public IP addresses that you can add to your NAT pool, and an issue may arise when redundancy kicks in. Spreading your IP Pool out to two pools for load balancing should not be an issue since you are evenly distributing the flows 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 flows until that service PIC recovers, which could mean certain flows cannot get an available NAT'd IP address if the pool becomes exhausted. Still, this should be preferred to no flows passing through the box.

## Use Case for Massachusetts Telecom

Okay, it's time for Massachusetts Telecom (MassT) to sit down with all the information they have digested in this book and decide how they are going to configure their MX to fit their NAT needs.

Massachusetts Telecom is a mobile and wireline provider that has been very successful as of late. Too successful, in fact, and they are now hitting an IP exhaustion point. They need NAT 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 route traffic to a data center where the third party that manages the data center requires a one-to-one NAT so that all traffic from MassT comes from a single IP address.

The first thing MassT should look at configuring on their MX is the NAT setup required for the data center where their partner has

requested IP addresses get NAT'd to subnet 100.0.0.0/16 due to business requirements from the partner hosting the data center. This will be a simple case of inline NAT, so a MPC line card, but no service-card, will be needed. MassT has a MPC with 4 PFEs in slot 3 and they have decided that they will use the first PFE to handle the inline NAT function. Interface XE-3/3/0 is the ingress where the CPEs will send traffic destined to the data center.

```
[edit chassis]
fpc 3 {
    pic 0 {
        inline-services {
            bandwidth 10g;
        }
    }
    power on;
    number-of-ports 12;
}

[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;
    }
}

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

[edit services]
service-set static_nat {
    }
    nat-rules static_rule;
    interface-service {
        service-interface si-3/0/0.0;
    }
}
nat {
    pool 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 nat44;
                    translation-type {
                        basic-nat44;
                    }
                }
            }
        }
    }
}
```

To address their IP address exhaustion issue MassT should implement interface style dynamic NAT with PAT. A dynamic NAT and PAT setup requires that they use a service card. Mass Telecom will use the second PIC on the MS-DPC cards in slot 10 and 11 under a RSP interface. Remember interface style sends *all* the ingress traffic that hits the physical interface to the MS-DPC to get NAT'd:

```
[edit chassis]
fpc 10 {
    pic 1 {
        adaptive-services {
            service-package layer-3;
        }
    }
    power on;
}
fpc 11 {
    pic 1 {
        adaptive-services {
            service-package layer-3;
        }
    }
    power on;
}

[edit interfaces rsp1]
redundancy-options {
    primary sp-11/1/0;
    secondary sp-10/0/0;
    hot-standby;
}
services-options {
    cgn-pic;
}
unit 0 {
```

```
    family inet;
}

[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;
    }

[edit services]
service-set nat44 {
    nat-rules internet_rule;
    interface-service {
        service-interface rsp1;
    }
}
nat {
    pool natpat44 {
        address 180.100.100.0/26;
        port {
            automatic {
                random-allocation;
            }
        address-allocation round-robin;
    }
    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;
                    }
                }
            }
        }
    }

}
```

MassT has an issue using the NAT interface style option in their current setup. They need to NAT their traffic differently based on what its destination is – one destination being the data center network, the other being the Internet. Using the interface style can work if they also add destinations to the `from` portion of their rules. Remember, these options are available:

```
[edit services nat rule rule1 term t1 from]
lab@JTAC_setup-re0# set destination-?
Possible completions:
> destination-address  Match IP destination address
> destination-address-range  Match IP destination address range
> destination-port
> destination-prefix-list  One or more named lists of destination prefixes to match
```

But MassT decides to make a change to their configuration and move to the next hop style, allowing them to control what traffic based on destination route actually goes to the service PIC. They also add two separate routing instances to help manage and visualize their setup:

```
[edit interfaces rsp1]
redundancy-options {
    primary sp-11/1/0;
    secondary sp-10/0/0;
    hot-standby;
}
services-options {
    cgn-pic;
}
unit 1 {
    family inet;
    service-domain inside;
}
unit 2 {
    family inet;
    service-domain outside;
}

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

[edit interfaces xe-3/3/0]
unit 0 {
```

```
    family inet {
        }
        address 139.97.68.42/30;
    }
}

[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 rsp1.1;
        outside-service-interface rsp1.2;
    }
}
nat {
     pool nat44 {
        address 156.0.0.0/16;
    }
     pool natpat44 {
        address 180.100.100.0/24;
        port {
            automatic {
                random-allocation;
            }
        }
        address-allocation round-robin;
    }
    rule static_rule {
        match-direction input;
        term t1 {
            from {
                source-address {
                    100.100.0.0/16;
                }
            }
            then {
                translated {
                    source-pool nat44;
                    translation-type {
                        basic-nat44;
                    }
                }
            }
        }
```

```
    }
    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;
                    }
                }
            }
        }
    }
}

[edit routing-instances VR1]
instance-type virtual-router;
interface xe-3/3/0.0;
interface rsp1.1;
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 rsp1.1;
    }
}

[edit routing-instances VR2]
root@ setup-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 rsp1.2;'
interface si-3/0/0.2;
```

Now MassT can finish off their general router configuration and add their routing protocols to VR2 or whatever other configuration they feel may be needed. And now that they have their setup running, they want to see traffic running through the service PIC:

```
Monitor interface sp-x/x/x
run show services stateful-firewall flows count
```

But MassT is not done, yet. They are getting complaints that their

subscribers cannot send data to one another via applications like Apple Facetime. Also, servers on the outside cannot always send data in to the subscribers CPE, which is affecting some of the P2P solutions – XBOX, torrent, skype, PS3, and others – that previously worked for MassT's clientele. 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 targets setting on the outside to reach back in to the private side and contact a CPE based on their NAT'd IP address. So:

```
rule internet_rule {
  match-direction input;
  term 1 {
     from {
        applications junos-sip;
        }
        then {
          translated {
             source-pool napt-44;
             translation-type {
                napt-44;
             }
             address-pooling paired;

  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;
           }
        }
     }
  }
```

MassT also has SIP users and the good folks at MassT have to take this application into consideration through their MX NAT device. SIP traffic requires an Application Layer Gateway (ALG) to allow SIP servers and clients on the public side of the MX NATing device to communicate with the SIP hosts in the private side. The SIP ALG that MassT will implement takes care of opening the pinholes in the CGNAT router that permit the forwarding of outbound traffic based

on any supported SIP feature and rewriting the packets data when needed.

Since Endpoint Independent Mapping is not needed by SIP to function, or by the SIP ALG to create the flows for forwarding the SIP traffic, they do *not* configure EIM together with SIP-ALG, since it will add extra overhead in the processing with no benefit or usage.

So MassT sets a rule that NAT's SIP traffic first, and a more generic rule that uses EIM second:

```
rule internet_rule {
  match-direction input;
  term t1 {
     from {
        applications junos-sip;
     }
     then {
        translated {
           source-pool napt-44;
           translation-type {
              napt-44;
           }
           address-pooling paired;
              }
      }
  term t2 {
     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;
           }
         }
      }
   }
}
```

Another thing MassT wants to do is use syslogging, so they have a way to historically track what private IP address was mapped to what public IP address in their dynamic pool natpat44. Their concern here is they have the potential to have lots of flows being created and it could be a lot of overhead on the PIC and on the syslog server, so they implement PBA to make this more efficient:

```
pool natpat44 {
       address 180.100.100.0/24;
               port {
                      range low 1024 high 65535 random-allocation;
                      secured-port-block-allocation block-size 1024 max-blocks-per-
address 6;
                                                    }
       address-allocation round-robin;
   }
```

And MassT is having issues with their public IP addresses – they just do not have 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 180.100.100.248
       port {
          range low 1024 high 65535 random-allocation;
          secured-port-block-allocation block-size 1024 max-blocks-per-address 6;
       }
```

But was this a wise move? Do they have enough IP addresses now?

Since they are using the Port Block Allocation 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-DPC can handle between 5.8 and 8.4 million flows based on your configuration, so normally, huge IP ranges assigned to a pool do not help and end up wasting IP addresses.

Let's say you have 248 IP addresses available with a block range of 1024 and 64511 ports available per IP. So you have 62 port blocks per IP address when looking at a block range of 1024 against 64511 ports. 62 blocks against 248 IP addresses says that you can at most have 15376 subscribers managed by this pool.

Overall this setup allows you to have over 16,221,928 flows, which is more than the service PIC can handle, but if you feel your average subscribers typically use less than 1024 active flows you are okay for right now.

So to make sure the MX can handle more subscribers with public IPv4 addresses you can either add more IP addresses to the pool, or you can set lower block ranges because right now the port block size and number of blocks per address may be the weakness. Right now there's an aggressive setting for each subscriber, 1024 being the `block-size` and `max-blocks-per-address` being 6. 6411 ports, which is typically more than enough for most subscribers.

But before making the change it is smart to see how many active subscribers you have and how many ports and flows are actually being used. This will give you an idea of how to proceed. Run these commands on the MX to get that insight:

```
show services stateful-firewall subscriber-analysis
show services stateful-firewall flow-analysis
```

With this setup MassT is ready to move forward.

# Chapter 5

## Troubleshooting

Useful CLI Commands are the key to troubleshooting the MX Series, no matter how the box is configured. This last chapter focuses on the commands you need to analyze the stateful firewall and CGNAT functionality and verify what's going on with the box. Some of these commands you have already used while going through this book, but this chapter reviews them all as a quick reference when you need to recall what to use, and when.

NOTE These commands are used for troubleshooting the service interface and *not* Inline NAT.

### show services nat pool detail

Our first command shows global NAT statistics related to port usage for your 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, Address Pooling Paired, and EIF stats, if your pool uses these features.

```
user@ JTAC_ setup-re0 > show services nat pool detail
Interface: sp-5/0/0, Service set: sset2
NAT pool: pool1, Translation type: dynamic
   Address range: 129.0.0.1-129.0.0.240
   Port range: 512-65535, Ports in use: 35591, Out of port errors: 0, Max ports
used: 40178

Interface: rsp1, Service set: sset_Gi
```

```
  NAT pool: wap-pool1, Translation type: dynamic
    Address range: 129.0.0.241-129.0.0.242
    Port range: 1024-65535, Ports in use: 4608, Out of port errors: 0, Max ports used:
30720
    AP-P out of port errors: 0
    Max number of port blocks used: 120, Current number of port blocks in use: 18, Port
block     allocation errors: 0,
    Port block memory allocation errors: 0
    Port blocks limit exceeded errors: 0
    Unique pool users: 1368
    Current EIF Inbound flows count: 0
    EIF flow limit exceeded drops: 0
  NAT pool: wap-pool2, Translation type: dynamic
    Address range: 129.0.0.243-129.0.0.244
    Port range: 1024-65535, Ports in use: 51200, Out of port errors: 84433, Max ports
used: 129022
    AP-P out of port errors: 645362
    Max number of port blocks used: 503, Current number of port blocks in use: 250, Port
block allocation   errors: 0,
    Port block memory allocation errors: 0
    Port blocks limit exceeded errors: 345
    Unique pool users: 7384
    Current EIF Inbound flows count: 0
    EIF flow limit exceeded drops: 0
```

You can see that everything looks okay for the moment. You can tell that historically NAT pool `wap-pool2` has encountered out-of-port error hits due to every port being assigned to an active subscriber:

```
NAT pool: wap-pool2, Translation type: dynamic
    Address range: 129.0.0.243-129.0.0.244
    Port range: 1024-65535, Ports in use: 51200, Out of port errors: 84433, Max ports
used: 129022
```

This pool cannot currently meet all the flows that it is expected to handle and an investigation is required.

You can see that the `Max ports used` figure is `129022`, which is all available ports. There are two IP addresses in this NAT Pool, shown in `Address range: 129.0.0.243-129.0.0.244`; and, each can use 64511 ports as shown in `Port range: 1024-65535`.

So, based on this data we know that this pool has been starved of ports, though based on the current counter, `Ports in use: 51200`, this pool is not currently starved.

Another thing to take note of is that the AP-P out of port errors counter, which shows 84433. This means the address-pooling paired feature is being used and one of the NAT'd IP addresses could be starved of ports. It would be smart to run this command again to see if the AP-P out of ports counter increases.

## show services stateful-firewall flows count

Here is a simple command you can run to see the total number of flows being handled by the MX at the moment. In this case:

```
user@ JTAC_ setup-re0 > show services stateful-firewall flows count
Interface  Service set Flow count
sp-5/0/0     sset2          35591
rsp1   sset_Gi    56009
```

You can use the same command, removing the `count` parameter, to check the creation of the softwires, pre-NAT, and post-NAT flows within it. Note that if you have millions of flows on the MX this command will only display several thousand, since you do not want to crash the PIC by taking up all the memory to display millions of flows:

```
lab@ JTAC_ setup-re0 > show services stateful-firewall flows
Interface: sp-2/1/0, Service set: CGN-1
Flow                                    State                   Dir     Frm
count
UDP        10.0.14.3:8890  ->    10.12.1.10:8888  Forward  I       62800259
   NAT source      10.0.14.3:8890    ->    10.12.1.100:1061
UDP        10.0.14.3:8891  ->    10.12.1.10:8888  Forward  I       62805075
   NAT source      10.0.14.3:8891    ->    10.12.1.100:1079
UDP        10.0.14.3:8888  ->    10.12.1.10:8888  Forward  I       125608481
   NAT source      10.0.14.3:8888    ->    10.12.1.100:1058
UDP        10.0.14.3:8889  ->    10.12.1.10:8888  Forward  I       62813576
   NAT source      10.0.14.3:8889    ->    10.12.1.100:1067
UDP        10.0.14.3:454   ->    10.12.1.10:8888  Forward  I       62824746
   NAT source      10.0.14.3:454     ->    10.12.1.100:1071
UDP        10.0.14.8:8888  ->    10.12.1.10:8888  Forward  I       62834362
   NAT source      10.0.14.8:8888    ->    10.12.1.100:1057
```

NOTE When looking at the stateful firewall flows and using the `show services stateful-firewall flows` you may see that some TCP sessions are in a `watch` state, and do not move into a `forward` state. These TCP sessions on the stateful firewall will stay in a watch state until the return packets are received. The MX automatically creates both input and output direction flow entries on the first packet received for the flow.

## show services stateful-firewall statistics detail

Here's a good command to see if flows are encountering any issues and why. It will show you your service sets and the service interface they belong to once again:

```
root@ JTAC_ setup-re0> show services stateful-firewall statistics detail
Interface: rsp1
  Service set: nat44
    New flows:
      Rule Accepts: 91, Rule Discards: 0, Rule Rejects: 0
    Existing flow types packet counters:
      Accepts: 61143, Drop:540, Rejects: 0
    Hairpinning counters:
      Slow Path Hairpinned Packets: 0, Fast Path Hairpinned Packets: 0
    Drops:
      IP option: 0, TCP SYN defense: 0
      NAT ports exhausted: 0, Sessions dropped due to subscriber flow limit: 0
    Errors:
      IP: 0, TCP: 6
      UDP: 0, ICMP: 0
      Non-IP packets: 0, ALG: 0
```

You can see `Rule Accepts: 91` which tells you this service-set has handled 91 unique flows through the firewall, and the `Accepts: 61143` value tells you the number of packets that have traversusd successfully. You also have `Drop:540`, which could be normal if you have any rules with an action of *drop*, or this could be indicative of an issue if you run out of ports, the subscriber hit a configured flow limit, or you timed out trying to rebuild a fragmented packet. It could even be an issue with lack of available memory on the service card.

## show services service-sets <... >

Here is another command that focuses on flow drops:

```
root@ JTAC_ setup-re0 > show services service-sets statistics packet-drops
                              Cpu limit    Memory limit    Flow limit
Interface   Service Set    Drops         Drops          Drops
rsp1        nat44            0            0             0
```

Here is a series of commands to run against the service sets to gather some very useful data:

```
root@ JTAC_ setup-re0 > show services service-sets statistics tcp-mss
Interface   Service Set                  SYN Received     SYN Modified
rsp1        nat44                       0              0

root@ JTAC_ setup-re0 > show services service-sets cpu-usage
CPU
Interface   Service Set(or system category)     Utilization
rsp1        nat44                              4.27 %
```

```
root@ JTAC_ setup-re0 > show services service-sets summary
Service sets
CPU
Interface   configured      Bytes used         Policy bytes used      utilization
rsp1              2      177912342 ( 6.17 %) 15120 ( 0.01 %)        4.95 %

root@ JTAC_ setup-re0 > show services service-sets memory-usage
Interface   Service Set     Bytes Used
rsp1       nat44          179925326

root@ JTAC_ setup-re0 > show services service-sets memory-usage zone
Interface   Memory zone
rsp1       Green
```

Note this last command shows you if a service-set is running out of memory. Green zone is great and what is desired. Yellow zone will not change how the MS-DPC functions but it does indicate that you may have too many flows piling up. You may want to make sure your inactivity timeouts and mapping-refresh timeouts are not holding expired flows in memory for too long. Now if the zone hits Orange you would start to see new flows tied to your busiest service sets start to get into a `DROP` state. You never want to be in the Orange zone. The box is either configured poorly in regard to timeouts based on the volume of flows you are receiving, or else the PIC is just handling too many flows and you need to better distribute the load among other PICs. The last zone you can hit is Red and when you get into this zone the card has no more memory and all new flows will be dropped until you get back into a better state, which would be the Orange zone.

## show services stateful-firewall flow-analysis

This command shows you statistics pertaining to your service interface: current active flows, peak flows, created flows per second, flow life times, etc.:

```
root@ JTAC_ setup-re0 > show services stateful-firewall flow-analysis
  Services PIC Name:       rsp1

Flow Analysis Statistics:

  Total Flows Active             :571970
  Total TCP Flows Active          :0
  Total UDP Flows Active          :571970
  Total Other Flows Active        :0
  Total Predicted Flows Active     :0
  Created Flows per Second        :18882
  Deleted Flows per Second        :18938
  Peak Total Flows Active         :583588
  Peak Total TCP Flows Active      :0
  Peak Total UDP Flows Active      :583588
  Peak Total Other Flows Active    :0
  Peak Created Flows per Second    :19002
```

```
  Peak Deleted Flows per Second    :18942
  Average HTTP Flow Lifetime(ms)   :0
  Packets received                :17055304
  Packets transmitted              :17055303
  Slow path forward               :17006582
  Slow path discard               :0

Flow Rate Data:
  Number of Samples: 71327

 Flow Rate Distribution(sec)

 Flow Operation :Creation

    300000+          :0
    250000  - 300000   :0
    200000  - 250000   :0
    160000  - 200000   :0
    150000  - 160000   :0
     50000  - 150000   :0
     40000  -  50000   :0
     30000  -  40000   :0
     20000  -  30000   :0
     10000  -  20000   :1796
      1000  -  10000   :1
         0  -   1000   :69530

 Flow Operation :Deletion

    300000+          :0
    250000  - 300000   :0
    200000  - 250000   :0
    160000  - 200000   :0
    150000  - 160000   :0
     50000  - 150000   :0
     40000  -  50000   :0
     30000  -  40000   :0
     20000  -  30000   :0
     10000  -  20000   :1765
      1000  -  10000   :1
         0  -   1000   :69561

Flow Lifetime Distribution(sec):

              TCP              UDP              HTTP
  240+        :0               48               0
  120 - 240    :0               0
   60 - 120    :0               12
   30 - 60     :0               12
   15 - 30     :0               10012
    5 - 15     :0               0
    1 - 5      :0               0
    0 - 1      :0               33431110
```

This command is great for seeing if the service PIC in question is too busy. These counters show you if the PIC has too many flows currently, or if the peak flow count has ever been too high. Remember each PIC can hold at most 8.4 million (with EIM/EIF this drops to 5.8 Million flows):

```
Total Flows Active              :571970
Peak Total Flows Active         :583588
```

If the command `show services service-sets cpu-usage` shows any PIC running with high CPU usage, these numbers will be useful for JTAC to analyze:

```
Created Flows per Second        :18882
Deleted Flows per Second        :18938
Peak Created Flows per Second    :19002
Peak Deleted Flows per Second    :18942
```

## show services nat mappings summary

This command shows you how many private IP to public IP NAT mappings you have per service interface. It also tells you how many of these are Endpoint Independent Mappings and how many of those EIM mappings have an Endpoint Independent Filter assigned to them:

```
root@ JTAC_ setup-re0 > show services nat mappings summary

Service Interface:                                  rsp1
Total number of address mappings:                     32322
Total number of endpoint independent port mappings:       1020
Total number of endpoint independent filters:             0
```

## services stateful-firewall flows extensive

Let's look at an exercise using the `show services stateful-firewall flows extensive` command to look at our flow timeout values and how these values take effect:

```
lab@ JTAC_ setup-re0 # run show services stateful-firewall flows extensive
Interface: rsp1, Service set: nat44
Flow                                      State    Dir      Frm count
UDP       197.100.1.1:2003   ->    156.0.0.125:1028   Forward  O           24
   NAT dest      156.0.0.125:1028    ->      14.0.32.0:2002
  Byte count: 12672
  Flow role: Responder, Timeout: 23
UDP       197.100.1.2:2004   ->    156.0.0.125:1029   Forward  O          466
   NAT dest      156.0.0.125:1029    ->      14.0.32.0:2003
  Byte count: 479048
  Flow role: Responder, Timeout: 23
UDP         14.0.32.0:2002   ->    197.100.1.1:2003   Forward  I           24
   NAT source      14.0.32.0:2002    ->    156.0.0.125:1028
```

```
  Byte count: 12672
  Flow role: Master, Timeout: 23
UDP         14.0.32.0:2003   ->    197.100.1.2:2004   Forward  I          233
   NAT source      14.0.32.0:2003    ->    156.0.0.125:1029
  Byte count: 239524
  Flow role: Master, Timeout: 23
```

Here you can see the timeout values for each flow. Remember, by default a flow on the MX will be active for 30 seconds waiting for a new packet to be received before the MX takes the flow down. This 30-second period is the inactivity timeout. If a packet does come in through the firewall flow, the counter is renewed to 30 seconds.

Now, remember you can change the inactivity timeout from its default to a different value, between 4 and 129600 seconds. This value can be set under the service interface, which would affect all the flows anchored on this interface:

```
 [edit interfaces rsp1 services-options]
inactivity-timeout 20;
```

Or you can set this value under the individual applications in case you need to control how long a flow stays active based on the given application's needs:

```
[edit applications application UDP_5555]
protocol udp;
source-port 5555;
inactivity-timeout 120;
```

When using RSP interfaces you can make sure your PICs are okay by running this command:

```
root@ setup-re0# run show interfaces redundancy rsp1
Interface  State       Last change  Primary    Secondary    Current status
rsp1      On primary   00:02:21     sp-11/1/0  sp-10/0/0    both up
```

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 stateful firewall flow without a service card, so be aware when building your MX system that without the service card you cannot see the NAT'd flows, because in reality with inline NAT you are just changing the header of transient traffic. Note you can use the stateful firewall with inline NAT types if you do have a service card!

## Last Thoughts

When designing your CGNAT setup, one of the main things you should think about is how many NAT'd IP addresses and port combinations you have when setting up dynamic NAT. You should also consider the number of *mappings* and potential flows, and then do the math. Don't design a setup where you have two public IP addresses and 65000 ports available that need to handle two million concurrent flows at one time. Some things are just not possible.

Let's say you have 327600 potential NAT'd IP addresses for use and you want to use deterministic NAT as the NAT type. Deterministic NAT does not use ports 0-1023 so each NAT'd IP can potentially use 64511 ports. The `deterministic-port-block-allocation block-size` is 256, so:

64511 ports/256 blocks = 251 port block assignments per NAT'd IP.

327,600 potentials NAT'd IPs with 251 unique port blocks for each one is a total of 82,227,600 potential mappings at one moment in time. There isn't a single PIC that can handle that many mappings, or the 21,050,265,600 potential flows that would come with it. So if these 327,600 potential NAT'd IP addresses are valuable Internet routable IP addresses, you may want to tie up fewer on the MX CGNAT solution since you cannot use them all.

Just do the math and design accordingly and the MX can deliver a great service for you!

## More Books on the MX Series

Download these *Day One* books at http://www.juniper.net/dayone.