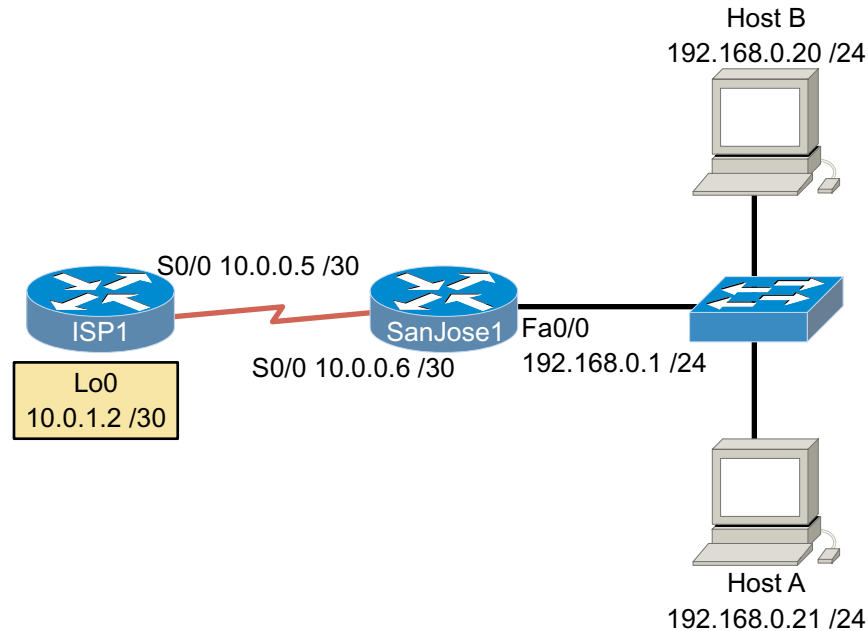


### 11.5.3: Configuring NAT Overload



#### Objective

Configure dynamic NAT with overload.

#### Scenario

When the International Travel Agency (ITA) expanded and updated their network, they chose to use the 192.168.0.0 /24 private addresses and NAT to handle connectivity with the outside world. In securing the outside IP addresses from their ISP, ITA is having to pay a monthly fee per IP address. ITA has asked you to set up a series of prototypes that would demonstrate NAT's capabilities to meet ITA's requirements. The company hopes to be able to get by with 14 real IP addresses (42.0.0.48 /28). For a variety of reasons including security concerns, the company wishes to hide the internal network from the outside.

It appears that the basic dynamic NAT translations will be too limiting and cumbersome to meet ITA's needs, so you have been asked to modify the prototype to use the overload feature.

#### Step 1

Build and configure the network according to the above diagram. This configuration requires the use of subnet zero, so you may need to enter the `ip subnet-zero` command, depending on the IOS version you are using. Both Host A and Host B represent users on the ITA network.

Configure SanJose1 to use a default route to ISP1:

```
SanJose1(config)#ip route 0.0.0.0 0.0.0.0 10.0.0.5
```

On ISP1, configure a static route to the global addresses used by SanJose1 for NAT:

```
ISP1(config)#ip route 42.0.0.48 255.255.255.240 10.0.0.6
```

Define a pool of global addresses to be allocated by the dynamic NAT process. Issue the following command on SanJose1:

```
SanJose1(config)#ip nat pool MYNATPOOL 42.0.0.55 42.0.0.62 netmask
255.255.255.240
```

Configure a standard access list to define which internal source addresses can be translated. Because you are translating all users on the ITA network, use the following command:

```
SanJose1(config)#access-list 2 permit 192.168.0.0 0.0.0.255
```

Specify an interface on SanJose1 to be used by inside network hosts requiring address translation:

```
SanJose1(config)#interface fastethernet0/0
SanJose1(config-if)#ip nat inside
```

You must also specify an interface to be used as the outside NAT interface:

```
SanJose1(config)#interface serial0/0
SanJose1(config-if)#ip nat outside
```

## Step 2

In the last exercise, you saw that a pool of “real” global IP addresses can be used to provide internally addressed hosts with access to the Internet and other outside resources. However, in your previous implementation, each global address could be allocated to only one host at a time.

The most powerful feature of NAT is address overloading, or port address translation (PAT). Overloading allows multiple inside addresses to map to a single global address. With PAT, literally hundreds of privately addressed nodes can access the Internet using only one global address. The NAT router keeps track of the different conversations by mapping TCP and UDP port numbers.

Configure address overloading on SanJose1 with the following command:

```
SanJose1(config)#ip nat inside source list 2 pool MYNATPOOL
overload
```

After you configure the overload feature, ping both interfaces of ISP1 (10.0.1.2 and 10.0.0.5) from Host A. The pings should be successful. Next, issue the **show ip nat translations** command:

```
SanJose1#show ip nat translation
Pro Inside global      Inside local      Outside local      Outside global
icmp 42.0.0.55:1536      192.168.0.21:1536 10.0.0.5:1536      10.0.0.5:1536
icmp 42.0.0.55:1536      192.168.0.21:1536 10.0.1.2:1536      10.0.1.2:1536
```

1. What port number is the source of the ping?  
\_\_\_\_\_
2. What port number is the destination of the ping?  
\_\_\_\_\_

In addition to tracking the IP addresses translated, the translations table also records the port numbers being used. Also notice that the first column (Pro) shows the protocol used.

Now look at the output of the **show ip nat translation verbose** command:

```
SanJose1#show ip nat translation verbose
Pro Inside global      Inside local      Outside local      Outside global
icmp 42.0.0.55:1536    192.168.0.21:1536  10.0.0.5:1536      10.0.0.5:1536
      create 00:00:09, use 00:00:06, left 00:00:53,
      flags:
extended, use_count: 0
icmp 42.0.0.55:1536    192.168.0.21:1536  10.0.1.2:1536      10.0.1.2:1536
      create 00:00:04, use 00:00:01, left 00:00:58,
      flags:
extended, use_count: 0
```

**Note:** The timeout for these overloaded dynamic translations of ICMP is 60 seconds. Note also that each session has its own timeout timer. New activity only resets one specific session's timer. You may need to **ping** again to see the result on your router.

From the MS-DOS prompt of Host A, quickly issue the following commands and then return to the SanJose1 console to issue the **show ip nat translation** command. You must work fast, because you are racing against the 60-second timeout:

```
HostA:\>ping 10.0.0.5
HostA:\>telnet 10.0.0.5    (Do not login. Return to command window)
HostA:\>ftp: 10.0.0.5      (It will fail. Don't worry about it)
```

**Note:** To quit the Windows FTP program, type **bye** and press Enter.

After you initiate these three sessions, the output of the **show ip nat translation** command should look something like the following:

```
SanJose1#show ip nat translation
Pro Inside global      Inside local      Outside local      Outside global
icmp 42.0.0.55:1536    192.168.0.21:1536  10.0.0.5:1536      10.0.0.5:1536
tcp  42.0.0.55:1095    192.168.0.21:1095  10.0.0.5:21        10.0.0.5:21
tcp  42.0.0.55:1094    192.168.0.21:1094  10.0.0.5:23        10.0.0.5:23
```

Although the NAT router has a pool of eight IP addresses to work with, it chooses to continue to use the 42.0.0.55 for both workstations. The Cisco IOS will continue to overload the first address in the pool until it is maxed out and then move to the second address, and so on.

### Step 3

In this step, you examine the timeout values in more detail. From Host A, initiate FTP and HTTP sessions with ISP1 at 10.0.0.5. Since ISP1 is not configured as an FTP server or Web server, both sessions will fail:

```
HostA:\>ftp: 10.0.0.5
```

To open an HTTP session, type ISP1's IP address in the URL field of a Web browser window.

After you attempt both FTP and HTTP sessions, use the **show ip nat translation verbose** command and examine the time left entries, as shown here:

```
SanJose1# show ip nat translation verbose
Pro Inside global      Inside local      Outside local      Outside global
icmp 42.0.0.55:1536    192.168.0.21:1536  10.0.0.5:1536      10.0.0.5:1536
```

```

        create 00:00:29, use 00:00:26, left 00:00:33,
        flags:
extended, use_count: 0
tcp 42.0.0.55:1114      192.168.0.21:1114  10.0.0.5:21        10.0.0.5:21
        create 00:00:16, use 00:00:15, left 00:00:44,
        flags:
extended, timing-out, use_count: 0
tcp 42.0.0.55:1113      192.168.0.21:1113  10.0.0.5:23        10.0.0.5:23
        create 00:00:22, use 00:00:22, left 23:59:37,
        flags:
extended, use_count: 0
tcp 42.0.0.55:1115      192.168.0.21:1115  10.0.0.5:80        10.0.0.5:80
        create 00:00:12, use 00:00:11, left 23:59:48,
        flags:
extended, use_count: 0

```

Notice that some of the TCP transactions are using a 24-hour timeout timer. To see the other timers that can be set, use the `ip nat translation ?` command while in global configuration mode, as shown here:

```

SanJose1(config)#ip nat translation ?
dns-timeout      Specify timeout for NAT DNS flows
finrst-timeout   Specify timeout for NAT TCP flows after a FIN or
                 RST
icmp-timeout     Specify timeout for NAT ICMP flows
max-entries      Specify maximum number of NAT entries
port-timeout     Specify timeout for NAT TCP/UDP port specific flows
syn-timeout      Specify timeout for NAT TCP flows after a SYN and
                 no further
                 data
tcp-timeout      Specify timeout for NAT TCP flows
timeout          Specify timeout for dynamic NAT translations
udp-timeout      Specify timeout for NAT UDP flows

```

The actual timeout options vary with versions of the IOS. The defaults for some of the more common times are

- **dns-timeout**      DNS session (60 seconds)
- **finrst-timeout**    TCP session after a FIN or RST / end of session (60 seconds)
- **icmp-timeout**     ICMP session (60 seconds)
- **tcp-timeout**       TCP port session (86,400 seconds – 24 hours)
- **timeout**            Dynamic NAT translations (86,400 seconds – 24 hours)
- **udp-timeout**       UDP port session (300 seconds – 5 minutes)

The **finrst-timeout** timer makes sure that TCP sessions close the related port 60 seconds after the TCP termination sequence.

Dynamic NAT sessions can only be initiated by an internal host. It is not possible to initiate a NAT translation from outside the network. To some extent, this adds a level of security to the internal network. It may also help to explain why the dynamic timeout timer for overload sessions is so short. The window of opportunity stays open just long enough to make sure that legitimate replies like Web pages, FTP and TFTP sessions, and ICMP messages can get in.

You saw in Lab 11.5.1 that outside hosts can ping our static NAT translations at any time, provided the inside host is up. This is so you can share Web, FTP, TFTP, DNS, and other types of servers with the outside world.

With dynamic NAT not configured for overload, the translation stays up for 24 hours and could allow an outside host to try to access the translation and therefore the host. But with the overload option, the outside host has to be able to re-create the NAT IP address plus the port number, thereby reducing the likelihood of an unwanted host gaining access to your system.

#### Step 4

To see the actual translation process and troubleshoot NAT problems, you can use the `debug ip nat` command and its related options.

Remember as with all `debug` commands, this can seriously impair the performance of your production router and should be used judiciously. The `undebug all` command turns off all debugging.

On SanJose1, use the `debug ip nat` command to turn on the debug feature.

From A, ping ISP1's serial interface (10.0.0.5), and observe the translations as shown here:

```
SanJose1#debug ip nat
IP NAT debugging is on
06:37:40: NAT: s=192.168.0.21->42.0.0.55, d=10.0.0.5 [63]
06:37:40: NAT*: s=10.0.0.5, d=42.0.0.55->192.168.0.21 [63]
06:37:41: NAT*: s=192.168.0.21->42.0.0.55, d=10.0.0.5 [64]
06:37:41: NAT*: s=10.0.0.5, d=42.0.0.55->192.168.0.21 [64]
06:37:42: NAT*: s=192.168.0.21->42.0.0.55, d=10.0.0.5 [65]
06:37:42: NAT*: s=10.0.0.5, d=42.0.0.55->192.168.0.21 [65]
06:37:43: NAT*: s=192.168.0.21->42.0.0.55, d=10.0.0.5 [66]
06:37:43: NAT*: s=10.0.0.5, d=42.0.0.55->192.168.0.21 [66]
06:38:43: NAT: expiring 42.0.0.55 (192.168.0.21) icmp 1536 (1536)
```

Turn off debugging.

```
SanJose1#undebug all
All possible debugging has been turned off
```

Notice that we can see both translations as the pings pass both ways through the NAT router. Note that the number at the end of the row is the same for both translations of each ping. The `s=` indicates the source, `d=` indicates the destination and `->` shows the translation.

The **06:38:43** entry above shows the expiration of the NAT translation.

The `detailed` option can be used with `debug ip nat` to provide the port numbers as well as the IP address translations, as shown here:

```
SanJose1#debug ip nat detailed
IP NAT detailed debugging is on
07:03:50: NAT: i: icmp (192.168.0.21, 1536) -> (10.0.0.5, 1536) [101]
07:03:50: NAT: address not stolen for 192.168.0.21, proto 1 port 1536
07:03:50: NAT: ipnat_allocate_port: wanted 1536 got 1536
07:03:50: NAT*: o: icmp (10.0.0.5, 1536) -> (42.0.0.55, 1536) [101]
07:03:51: NAT*: i: icmp (192.168.0.21, 1536) -> (10.0.0.5, 1536) [102]
07:03:51: NAT*: o: icmp (10.0.0.5, 1536) -> (42.0.0.55, 1536) [102]
07:03:52: NAT*: i: icmp (192.168.0.21, 1536) -> (10.0.0.5, 1536) [103]
07:03:52: NAT*: o: icmp (10.0.0.5, 1536) -> (42.0.0.55, 1536) [103]
07:03:53: NAT*: i: icmp (192.168.0.21, 1536) -> (10.0.0.5, 1536) [104]
07:03:53: NAT*: o: icmp (10.0.0.5, 1536) -> (42.0.0.55, 1536) [104]
```