File Edit View Terminal Tabs Help

```
No programs found to install
No ogma to cleanup
No crona to clean up
[jm@rat13
[jm@rat13
[jm@rat13
[jm@rat13
[jm@rat13 make
sr1 index.in > index.in3
mkinheader -h > header
sr2html index.in3 > www/index.html
sr1 meta.in > meta.in3
sr2html meta.in3 > www/meta.html
grep -vh "^\.header" menu.in 1_Goal.in 2_The_network.in 3_Building.in 4_Static_NAT.in 5_NAT_overl
sr1 total.in > total.in3
sr2html total.in3 > www/total.html
sr2html 1_Goal.in3 > www/1_Goal.html
sr2html 2_The_network.in3 > www/2_The_network.html
sr2html 3_Building.in3 > www/3_Building.html
sr2html 4_Static_NAT.in3 > www/4_Static_NAT.html
sr2html 5_NAT_overload.in3 > www/5_NAT_overload.html
sr2html 6_Dynamic_NAT.in3 > www/6_Dynamic_NAT.html
touch tag/in3.www
sr2html -n --partonly index.in3 > html/index.html
sr2html -n --partonly meta.in3 > html/meta.html
sr2html -n --partonly total.in3 > html/total.html
sr2html -n --partonly 1_Goal.in3 > html/1_Goal.html
sr2html -n --partonly 2_The_network.in3 > html/2_The_network.html
sr2html -n --partonly 3_Building.in3 > html/3_Building.html
sr2html -n --partonly 4_Static_NAT.in3 > html/4_Static_NAT.html
sr2html -n --partonly 5_NAT_overload.in3 > html/5_NAT_overload.html
sr2html -n --partonly 6_Dynamic_NAT.in3 > html/6_Dynamic_NAT.html
mkinheader > html/header.htm
touch tag/in3.html
make: *** No rule to make target `book.pdf', needed by `tag/in3.tag'.  Stop.
[jm@rat13 romeo
Last login: Fri Jan  8 21:30:22 2010 from 192.168.170.113
Linux 4.1 RR

It is not true that life is one damn thing after
another -- it's one damn thing over and over.
                -- Edna St. Vincent Millay

[jm@romeo:~]$ zmeabart
[jm@romeo:~]$ 
```

NAT

ljm

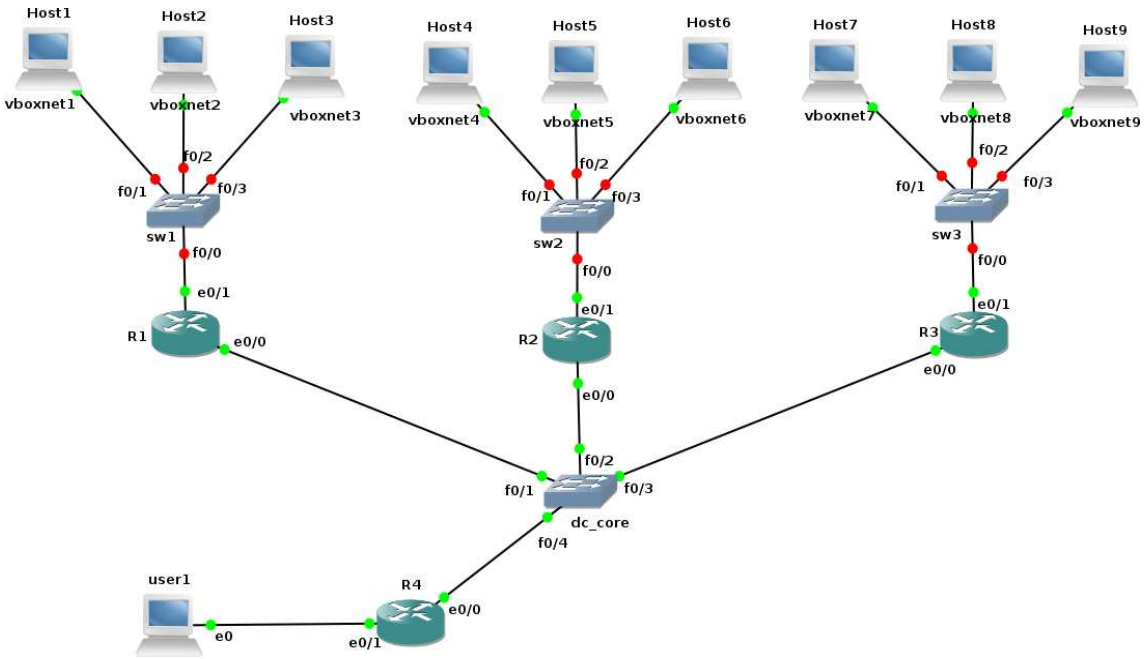
1. Goal

The goal of this network is to explore NAT and PAT.

We'll set-up a network in three zones, each having the same IP-subnet, and a connection zone, which we will call the core-switch. The idea is to have a separate subnet for the connection zone. All other zones will be behind one or two IP adresses from the connection zone, independend from the quantity of devices that may be in the zone.

2. The network

2.1 Connecting the network



All devices are 3640 routers. The switches are the NM-16-ESW switch cards in a slot; we'll be using them as switch. For the hosts, we've just connected them to the different vboxnet interfaces.

For the VirtualBox instances, the following Vagrantfile is used:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
# Vagrantfile API/syntax version. Don't touch unless you know what you're doing!
VAGRANTFILE_API_VERSION = "2"
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
        config.vm.define :precise1 do |t|
                t.vm.box = "hashicorp/precise32"
                t.vm.provider "virtualbox" do |prov|
                        prov.customize ["modifyvm", :id, "--nic2", "hostonly", "--hos-
tonlyadapter2", "vboxnet1" ]
                end
                t.vm.provision "shell", path: "./setup.precise1.sh"
        end
        config.vm.define :precise2 do |t|
                t.vm.box = "hashicorp/precise32"
                t.vm.provider "virtualbox" do |prov|
                        prov.customize ["modifyvm", :id, "--nic2", "hostonly", "--hos-
tonlyadapter2", "vboxnet2" ]
                end
                t.vm.provision "shell", path: "./setup.precise2.sh"
        end
        config.vm.define :precise3 do |t|
                t.vm.box = "hashicorp/precise32"
                t.vm.provider "virtualbox" do |prov|
                        prov.customize ["modifyvm", :id, "--nic2", "hostonly", "--hos-
tonlyadapter2", "vboxnet3" ]
                end
                t.vm.provision "shell", path: "./setup.precise3.sh"
        end
        config.vm.define :precise4 do |t|
                t.vm.box = "hashicorp/precise32"
                t.vm.provider "virtualbox" do |prov|
                        prov.customize ["modifyvm", :id, "--nic2", "hostonly", "--hos-
tonlyadapter2", "vboxnet4" ]
                end
                t.vm.provision "shell", path: "./setup.precise4.sh"
        end
        config.vm.define :precise5 do |t|
                t.vm.box = "hashicorp/precise32"
                t.vm.provider "virtualbox" do |prov|
                        prov.customize ["modifyvm", :id, "--nic2", "hostonly", "--hos-
tonlyadapter2", "vboxnet5" ]
                end
                t.vm.provision "shell", path: "./setup.precise5.sh"
        end
        config.vm.define :precise6 do |t|
                t.vm.box = "hashicorp/precise32"
                t.vm.provider "virtualbox" do |prov|
                        prov.customize ["modifyvm", :id, "--nic2", "hostonly", "--hos-
tonlyadapter2", "vboxnet6" ]
                end
                t.vm.provision "shell", path: "./setup.precise6.sh"
        end
```

```
        config.vm.define :precise7 do |t|
                t.vm.box = "hashicorp/precise32"
                t.vm.provider "virtualbox" do |prov|
                        prov.customize   ["modifyvm",   :id,   "--nic2",   "hostonly",
"--hostonlyadapter2", "vboxnet7" ]
                end
                t.vm.provision "shell", path: "./setup.precise7.sh"
        end
        config.vm.define :precise8 do |t|
                t.vm.box = "hashicorp/precise32"
                t.vm.provider "virtualbox" do |prov|
                        prov.customize   ["modifyvm",   :id,   "--nic2",   "hostonly",
"--hostonlyadapter2", "vboxnet8" ]
                end
                t.vm.provision "shell", path: "./setup.precise8.sh"
        end
        config.vm.define :precise9 do |t|
                t.vm.box = "hashicorp/precise32"
                       t.vm.provider "virtualbox" do |prov|
                        prov.customize   ["modifyvm",   :id,   "--nic2",   "hostonly",
"--hostonlyadapter2", "vboxnet9" ]
                end
                t.vm.provision "shell", path: "./setup.precise9.sh"
        end
end
```

I have tried to do it in a loop, but I have some problems understanding the Ruby way of looping.

2.2 Basic IP plan

This is not the complete IP plan, but the information required to get it all working.

network	subnet	netmask
user1	10.128.101.0	255.255.255.0
core	10.128.1.0	255.255.255.0
all natted networks	10.128.2.0	255.255.255.0

3. Building

We'll be building the network step by step.

3.1 User network

First a hint: always work from GNS3. If you start the vpc's separately, they will not connect correctly to the network.

Start user1 and open a console. For the user PC, we'll use 10.128.101.100. In the VPC-window, type

```
ip 10.128.101.100 255.255.255.0 10.128.101.1
```

Next is the user-router r4. In config mode:

```
ip address 10.128.101.1 255.255.255.0
no shut
```

While we're on r4, we'll also configure the core-interface

```
int e0/0
ip address 10.128.1.4 255.255.255.0
no shut
```

Now ping from user1 to r4 should work:

```
user1> ping 10.128.101.1
84 bytes from 10.128.101.1 icmp_seq=1 ttl=255 time=10.001 ms
84 bytes from 10.128.101.1 icmp_seq=2 ttl=255 time=7.588 ms
84 bytes from 10.128.101.1 icmp_seq=3 ttl=255 time=7.053 ms
84 bytes from 10.128.101.1 icmp_seq=4 ttl=255 time=8.238 ms
84 bytes from 10.128.101.1 icmp_seq=5 ttl=255 time=7.610 ms
user1> ping 10.128.1.4
84 bytes from 10.128.1.4 icmp_seq=1 ttl=255 time=9.227 ms
84 bytes from 10.128.1.4 icmp_seq=2 ttl=255 time=8.108 ms
84 bytes from 10.128.1.4 icmp_seq=3 ttl=255 time=7.840 ms
84 bytes from 10.128.1.4 icmp_seq=4 ttl=255 time=7.331 ms
84 bytes from 10.128.1.4 icmp_seq=5 ttl=255 time=7.855 ms
```

3.2 The core

First, we'll configure all the e0/0 interfaces of the routers. This is basicaly the same as for the user-router:

router	interface	ipaddress
r1	e0	10.128.1.1
r2	e0	10.128.1.2
r3	e0	10.128.1.3
r4	e0	10.128.1.4

Because the 16 ESW card acts as a switch, all routers can now ping eachother.

3.3 Enable routing

We'll go for the easy way: we'll use rip. On all the routers:

```
router rip
version  2
no auto-summary
network 10.128.1.0
```

and on r4:

```
network 10.128.101.0
```

Because we're now routing, user1 can ping all the routers:

```
user1> ping  10.128.1.2
84 bytes from 10.128.1.2 icmp_seq=1 ttl=254 time=13.610 ms
84 bytes from 10.128.1.2 icmp_seq=2 ttl=254 time=17.965 ms
84 bytes from 10.128.1.2 icmp_seq=3 ttl=254 time=18.455 ms
84 bytes from 10.128.1.2 icmp_seq=4 ttl=254 time=18.108 ms
84 bytes from 10.128.1.2 icmp_seq=5 ttl=254 time=18.003 ms
```

3.4 The r1-network

In our first attempt to get a working network, we'll use 10.128.2.0/24 for the network behind r1. On r1:

```
int e0/1
ip address 10.128.2.1 255.255.255.0
no shut
router rip
network 10.128.2.0
```

And we'll provision the servers with:

```
#!/bin/bash
apt-get install traceroute
ifconfig eth1 10.128.2.101 netmask 255.255.255.0
route add -net 10.128.0.0 netmask 255.255.0.0 gw  10.128.2.1
echo BOX 1
ifconfig -a
netstat -rn
```

(for precise 2 etc we use the equivalent). When provisioning, this gives:

```
[[ lines removed for brevity ]]
   precise2: eth0      Link encap:Ethernet  HWaddr 08:00:27:12:96:98
   precise2:           inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
   precise2:           inet6 addr: fe80::a00:27ff:fe12:9698/64 Scope:Link
   precise2:           UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
   precise2:           RX packets:427 errors:0 dropped:0 overruns:0 frame:0
   precise2:           TX packets:330 errors:0 dropped:0 overruns:0 carrier:0
   precise2:           collisions:0 txqueuelen:1000
   precise2:           RX bytes:98691 (98.6 KB)  TX bytes:37706 (37.7 KB)
   precise2:
   precise2: eth1      Link encap:Ethernet  HWaddr 08:00:27:7b:9a:67
   precise2:           inet addr:10.128.2.102  Bcast:10.128.2.255  Mask:255.255.255.0
   precise2:           inet6 addr: fe80::a00:27ff:fe7b:9a67/64 Scope:Link
   precise2:           UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
   precise2:           RX packets:3 errors:0 dropped:0 overruns:0 frame:0
   precise2:           TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
   precise2:           collisions:0 txqueuelen:1000
   precise2:           RX bytes:471 (471.0 B)  TX bytes:0 (0.0 B)
   precise2:
   precise2: lo        Link encap:Local Loopback
   precise2:           inet addr:127.0.0.1  Mask:255.0.0.0
   precise2:           inet6 addr: ::1/128 Scope:Host
   precise2:           UP LOOPBACK RUNNING  MTU:16436  Metric:1
   precise2:           RX packets:0 errors:0 dropped:0 overruns:0 frame:0
   precise2:           TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
   precise2:           collisions:0 txqueuelen:0
   precise2:           RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
   precise2: BOX 2
   precise2: Kernel IP routing table
    precise2: Destination     Gateway         Genmask         Flags   MSS Window  irtt
Iface
   precise2: 0.0.0.0         10.0.2.2        0.0.0.0         UG        0 0          0
eth0
   precise2: 0.0.0.0         10.0.2.2        0.0.0.0         UG        0 0          0
eth0
   precise2: 10.0.2.0        0.0.0.0         255.255.255.0   U         0 0          0
eth0
   precise2: 10.128.0.0      10.128.2.1      255.255.0.0     UG        0 0          0
eth1
   precise2: 10.128.2.0      0.0.0.0         255.255.255.0   U         0 0          0
eth1
```

And we'll be able to ping the servers form our user1:

```
user1> ping 10.128.2.101
84 bytes from 10.128.2.101 icmp_seq=1 ttl=62 time=39.953 ms
84 bytes from 10.128.2.101 icmp_seq=2 ttl=62 time=25.692 ms
84 bytes from 10.128.2.101 icmp_seq=3 ttl=62 time=39.988 ms
84 bytes from 10.128.2.101 icmp_seq=4 ttl=62 time=25.149 ms
84 bytes from 10.128.2.101 icmp_seq=5 ttl=62 time=25.152 ms
user1> ping 10.128.2.102
10.128.2.102 icmp_seq=1 timeout
84 bytes from 10.128.2.102 icmp_seq=2 ttl=62 time=23.082 ms
84 bytes from 10.128.2.102 icmp_seq=3 ttl=62 time=29.684 ms
84 bytes from 10.128.2.102 icmp_seq=4 ttl=62 time=21.853 ms
84 bytes from 10.128.2.102 icmp_seq=5 ttl=62 time=29.929 ms
user1> ping 10.128.2.103
10.128.2.103 icmp_seq=1 timeout
84 bytes from 10.128.2.103 icmp_seq=2 ttl=62 time=26.572 ms
84 bytes from 10.128.2.103 icmp_seq=3 ttl=62 time=28.058 ms
84 bytes from 10.128.2.103 icmp_seq=4 ttl=62 time=30.783 ms
84 bytes from 10.128.2.103 icmp_seq=5 ttl=62 time=30.273 ms
user1>
```

This means that our infrastructure is working.

4. Static NAT

The r2-network will be statically natted:

host	precise	ip	nat-IP
host4	precise4	10.128.2.104	10.128.1.104
host5	precise5	10.128.2.105	10.128.1.105
host6	precise6	10.128.2.106	10.128.1.106

The gateway address on the sw2-side of r2 will be 10.128.2.1.

For NAT, we'll need to define which network is 'inside' and which network is 'outside'. In our case, the sw2 network is inside, while the dc_core is outside.

This should be all the information required to set-up NAT.

The core-interface of r2 has already been configured:

```
R2#sh ip int br
Interface                  IP-Address      OK? Method Status                Protocol
Ethernet0/0                10.128.1.2      YES NVRAM  up                    up
Ethernet0/1                unassigned      YES NVRAM  administratively down down
Ethernet0/2                unassigned      YES NVRAM  administratively down down
```

The e0/1 should be 10.128.2.1 on a /24 network:

```
int e0/1
ip address 10.128.2.1 255.255.255.0
no shut
```

Now, host4 should be available:

```
R2#ping 10.128.2.104
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.128.2.104, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/13/32 ms
R2#
```

To configure NAT, we'll issue the following commands:

```
ip nat inside source static 10.128.2.104 10.128.1.104
ip nat inside source static 10.128.2.105 10.128.1.105
ip nat inside source static 10.128.2.106 10.128.1.106
int e0/1
ip nat inside
int e0/0
ip nat outside
```

Now, we cannot ping host4 on its real IP address, but we should be able to ping it on its NATted address. From user1:

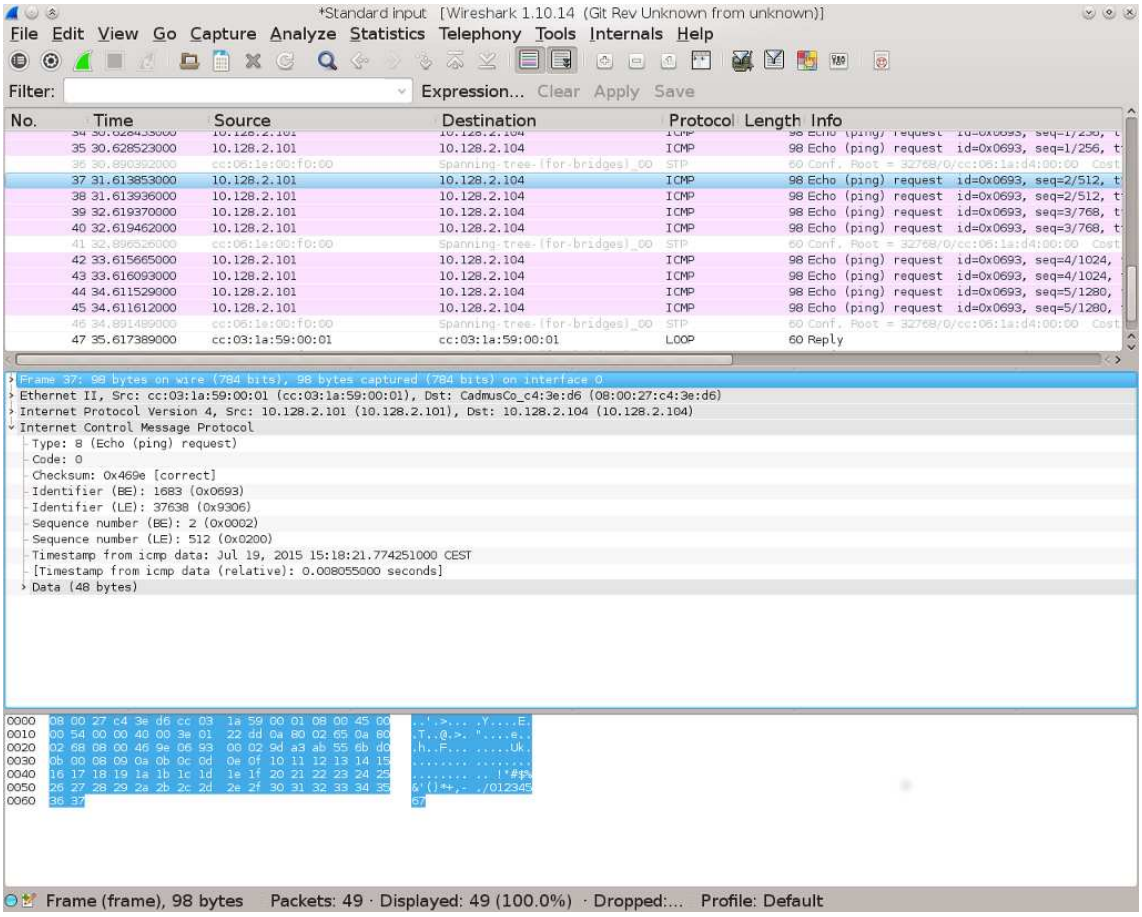
```
user1> ping 10.128.2.104
10.128.2.104 icmp_seq=1 timeout
10.128.2.104 icmp_seq=2 timeout
10.128.2.104 icmp_seq=3 timeout
10.128.2.104 icmp_seq=4 timeout
10.128.2.104 icmp_seq=5 timeout
user1> ping 10.128.1.104
10.128.1.104 icmp_seq=1 timeout
84 bytes from 10.128.1.104 icmp_seq=2 ttl=62 time=24.148 ms
84 bytes from 10.128.1.104 icmp_seq=3 ttl=62 time=39.642 ms
84 bytes from 10.128.1.104 icmp_seq=4 ttl=62 time=36.118 ms
84 bytes from 10.128.1.104 icmp_seq=5 ttl=62 time=23.619 ms
user1>
```

This means that NAT works.

From host1, we're not able to ping host5:

```
vagrant ssh precise1 -c 'ping -c5 10.128.1.104'
PING 10.128.1.104 (10.128.1.104) 56(84) bytes of data.
--- 10.128.1.104 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 3999ms
Connection to 127.0.0.1 closed.
```

So, what goes wrong? Starting a trace on e0/1 of r2 shows the following:



All the requests come in, but no replies come through. The problem is that host4 thinks that 10.128.2.0/24 is directly connected and, therefore, will try to find the reply address via arp. And of course, that does not work.

5. NAT overload

5.1 Configuring

For the sw3 network, we'll also configure NAT, but with NAT overload.

Once again, we'll need to define the inside and outside of the network. The outside is ofcourse dc_core again. And we'll configure e0/1 as 10.128.2.1:

```
int e0/1
ip address 10.128.2.1 255.255.255.0
no shut
```

We'll define the NAT interfaces:

```
int e0/0
ip nat outside
int e0/1
ip nat inside
```

For static NAT, we defined the translations next. NAT overload uses an ACL as source:

```
ip access-list standard NAT
permit 10.128.2.1 0.0.0.255
```

For the ACL, we used a named ACL. Also, for ACLs, wildcard masks are used not netmasks.

And lastly, configure the NAT:

```
ip nat inside source list NAT interface e0/0 overload
```

And we're able to ping host4 from host8:

```
vagrant ssh precise8 -c 'ping -c1 10.128.1.4'
PING 10.128.1.4 (10.128.1.4) 56(84) bytes of data.
64 bytes from 10.128.1.4: icmp_req=1 ttl=254 time=25.9 ms
--- 10.128.1.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 25.949/25.949/25.949/0.000 ms
Connection to 127.0.0.1 closed.
```

And the result for the NAT will be:

```
R3#sh ip nat translations
Pro Inside global      Inside local       Outside local      Outside global
icmp 10.128.1.3:1363   10.128.2.107:1363  10.128.1.4:1363    10.128.1.4:1363
icmp 10.128.1.3:1381   10.128.2.108:1381  10.128.1.4:1381    10.128.1.4:1381
icmp 10.128.1.3:1339   10.128.2.109:1339  10.128.1.4:1339    10.128.1.4:1339
R3#
```

Or, if you start an ssh from host8 to host5:

```
R3#sh ip nat translations
Pro Inside global      Inside local       Outside local      Outside global
tcp 10.128.1.3:51148   10.128.2.108:51148 10.128.1.105:22    10.128.1.105:22
R3#
```

6. Dynamic NAT

The last form of NAT is dynamic NAT. This allows the use of multiple IP addresses on the outside.

The source wil be an ACL and the outside addresses will be in a pool.

```
ip access-list standard NAT
permit 10.128.2.1 0.0.0.255
```

This is the same as for NAT overload.

```
ip nat pool NATPOOL 10.128.1.65 10.128.1.66 netmask 255.255.255.248
```

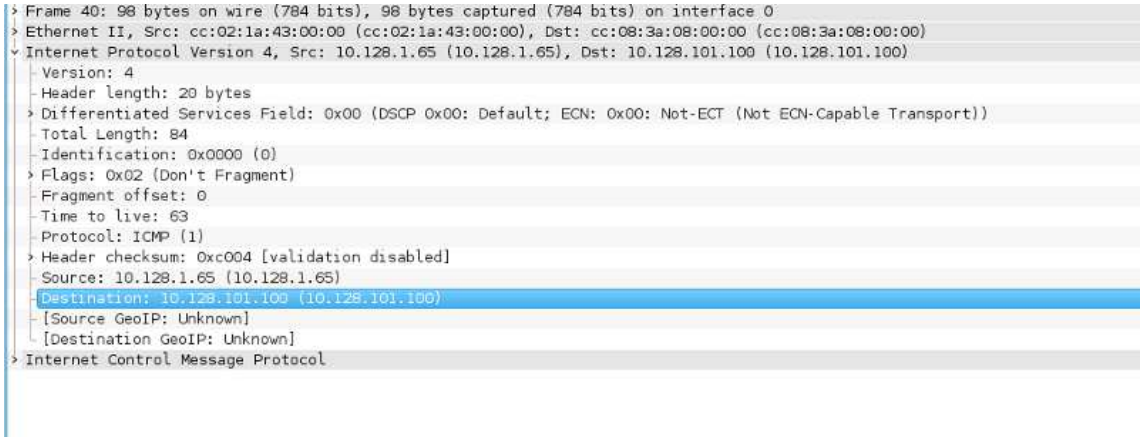
The netmask is a piece of redundant information that needs to be supplied.

```
int e0/0
ip nat outside
int e0/1
ip nat inside
exit
ip nat inside source list NAT pool NATPOOL
```

So what do we see if we ping from host1 to the user1?

```
vagrant ssh precise1 -c 'ping -c5 10.128.101.100'
```

If we capture a packet at r4 e0/0:

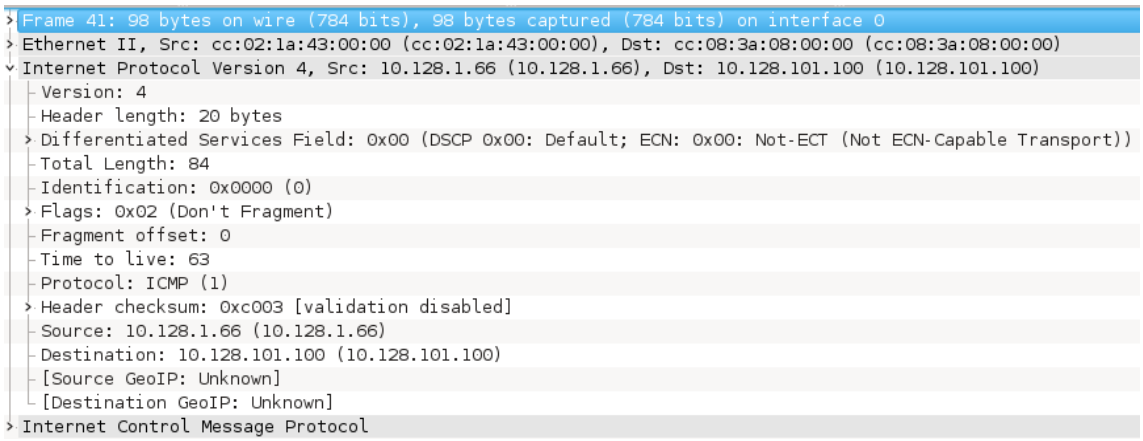


As we expect, the source address for the ping is the first address of the pool (10.128.1.65). So this works nicely.

And pinging from the next host, host2 with

```
vagrant ssh precise2 -c 'ping -c5 10.128.101.100'
```

gives:



So the second address in the dynamic NAT pool. So now both addresses of the NAT pool are in use. What would happen if we try the same from host3?

```
vagrant ssh precise3 -c 'ping -c5 10.128.101.100'
PING 10.128.101.100 (10.128.101.100) 56(84) bytes of data.
From 10.128.2.1 icmp_seq=1 Destination Host Unreachable
From 10.128.2.1 icmp_seq=1 Destination Host Unreachable
From 10.128.2.1 icmp_seq=2 Destination Host Unreachable
From 10.128.2.1 icmp_seq=2 Destination Host Unreachable
From 10.128.2.1 icmp_seq=3 Destination Host Unreachable
--- 10.128.101.100 ping statistics ---
3 packets transmitted, 0 received, +5 errors, 100% packet loss, time 2002ms
```

The pool is exhausted, there is no NAT address available for host3. You can see this on r1:

```
R1#sh ip nat translations
Pro Inside global      Inside local       Outside local      Outside global
--- 10.128.1.65        10.128.2.101       ---                ---
--- 10.128.1.66        10.128.2.102       ---                ---
R1#
```

So, if we reload r1 and start the other way around:

```
ljm@verlaine nat]$ vagrant ssh precise3 -c 'ping -c1 10.128.101.100'
PING 10.128.101.100 (10.128.101.100) 56(84) bytes of data.
64 bytes from 10.128.101.100: icmp_req=1 ttl=62 time=28.1 ms
--- 10.128.101.100 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 28.182/28.182/28.182/0.000 ms
Connection to 127.0.0.1 closed.
[ljm@verlaine nat]$ vagrant ssh precise2 -c 'ping -c1 10.128.101.100'
PING 10.128.101.100 (10.128.101.100) 56(84) bytes of data.
64 bytes from 10.128.101.100: icmp_req=1 ttl=62 time=41.4 ms
--- 10.128.101.100 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 41.455/41.455/41.455/0.000 ms
Connection to 127.0.0.1 closed.
[ljm@verlaine nat]$ vagrant ssh precise1 -c 'ping -c1 10.128.101.100'
PING 10.128.101.100 (10.128.101.100) 56(84) bytes of data.
From 10.128.2.1 icmp_seq=1 Destination Host Unreachable
--- 10.128.101.100 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
Connection to 127.0.0.1 closed.
```

and:

```
R1#sh ip nat translations
Pro Inside global      Inside local       Outside local      Outside global
--- 10.128.1.66        10.128.2.102       ---                ---
--- 10.128.1.65        10.128.2.103       ---                ---
R1#
```

Which is exactly what we'd expect.

CONTENTS

1. Goal 0

2. The network 0
   2.1 Connecting the network 0
   2.2 Basic IP plan 2

3. Building 2
   3.1 User network 2
   3.2 The core 3
   3.3 Enable routing 3
   3.4 The r1-network 3

4. Static NAT 6

5. NAT overload 8
   5.1 Configuring 8

6. Dynamic NAT 9