

Advanced Topics in Networking

Lab 2

NAT

Patryk Figiel

December 31, 2024

Contents

1. Goal of the lab	3
2. Preparations	3
3. The main task	3
3.1. Topology of the network	3
3.2. Allocated IP addresses	4
3.3. Configuring the TinyCoreVM server	4
3.4. Dynamic NAT Overload Configuration on R1	5
3.5. SSH Connection to the Server	6
3.6. Displaying the Hosting Webpage on the TinyCore Host	6
4. Summary	7

1. Goal of the lab

The goal of this lab is to configure Dynamic NAT with Overload (PAT) on routers, allowing multiple internal hosts (including a TinyCore host) to communicate with a TinyCore server. The lab first involves understanding Static and Dynamic NAT configurations as examples, then focusing on Dynamic PAT to enable multiple devices to share a pool of public IP addresses while ensuring connectivity for ping, web access, and SSH.

2. Preparations

As part of the preparations for this lab, the network topologies for all NAT types were configured, including Static NAT, Dynamic NAT, Static PAT, and Dynamic PAT, following the examples provided. Each configuration was successfully implemented, ensuring that the routers and devices were properly set up for testing connectivity, NAT translations, and the functionality of the TinyCore server.

3. The main task

3.1. Topology of the network

The network topology consists of:

- Four hosts in an internal private network (Inside Local), one of which is a TinyCore VM
- Router R1 configured with Dynamic NAT Overload to translate private IPs into public IPs
- Router R2
- A TinyCore server (176.16.0.2) connected to the public network, hosting a webpage and accessible via SSH

It is presented as follows:

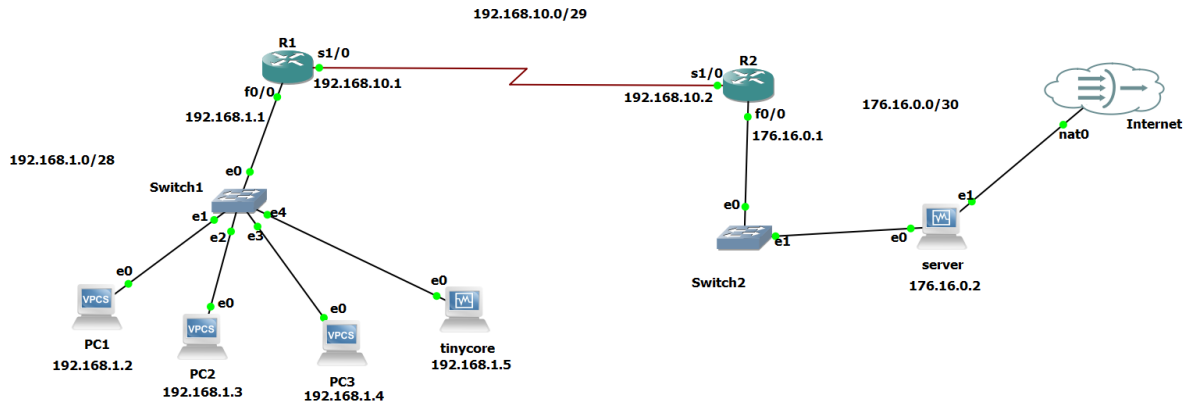


Figure 1. Network's Topology

3.2. Allocated IP addresses

For the purpose of this lab, the following IP addresses were allocated:

- **Inside Local Network (Private):** 192.168.1.0/28
 - PC1: 192.168.1.2
 - PC2: 192.168.1.3
 - PC3: 192.168.1.4
 - TinyCore Host: 192.168.1.5
 - R1 (interface to local network): 192.168.1.1
- **Inter-router Network:** 192.168.10.0/29
 - R1 (interface to R2): 192.168.10.1
 - R2 (interface to R1): 192.168.10.2
- **Outside network:** 176.16.0.0/30
 - R2 (interface to the second network): 176.16.0.1
 - TinyCore Server: 176.16.0.2

3.3. Configuring the TinyCoreVM server

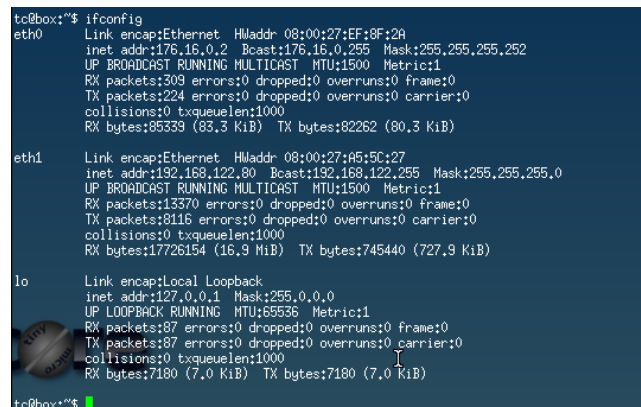
The TinyCore server in this lab setup is configured with two network adapters. The first adapter is used to communicate with the rest of the topology and is assigned the IP address 176.16.0.2, with a default gateway set to 176.16.0.1. This ensures that the server is reachable from devices within the configured network. The second adapter, connected to the internet, is assigned an IP address via DHCP (192.168.122.80), enabling the server to download the required software.

To prepare the server, the following commands were executed using the TinyCore package manager:

```
tce-load -wi openssh      # Enables SSH server functionality
tce-load -wi lighttpd     # Installs a lightweight web server
tce-load -wi dillo        # Installs a graphical web browser
```

These installations allow the server to host a webpage, enable SSH access, and use the dillo browser for visualization.

Below is a screenshot showing the IP addresses assigned to both adapters:



```
tc@box:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:EF:8F:2A
          inet addr:176.16.0.2  Bcast:176.16.0.255  Mask:255.255.255.252
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:309 errors:0 dropped:0 overruns:0 frame:0
          TX packets:224 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:85339 (83.3 KiB)  TX bytes:82262 (80.3 KiB)

eth1      Link encap:Ethernet  HWaddr 08:00:27:A5:5C:27
          inet addr:192.168.122.80  Bcast:192.168.122.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:13370 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8116 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:17726154 (16.9 MiB)  TX bytes:745440 (727.9 KiB)

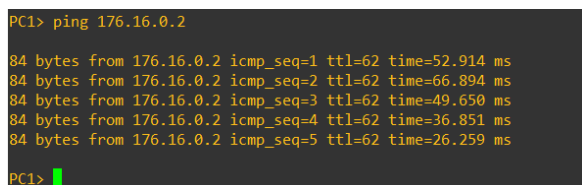
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:87 errors:0 dropped:0 overruns:0 frame:0
          TX packets:87 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:7180 (7.0 KiB)  TX bytes:7180 (7.0 KiB)

tc@box:~$
```

Figure 2. Assigned IP addresses for TinyCore server network adapters

Ping test results from all hosts to the server

Below are the results of the ping tests from all hosts to the server:

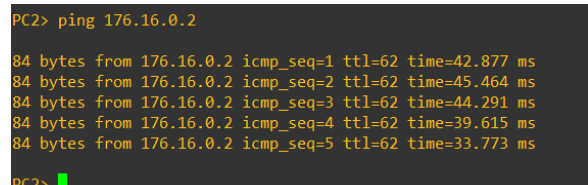


```
PC1> ping 176.16.0.2

84 bytes from 176.16.0.2 icmp_seq=1 ttl=62 time=52.914 ms
84 bytes from 176.16.0.2 icmp_seq=2 ttl=62 time=66.894 ms
84 bytes from 176.16.0.2 icmp_seq=3 ttl=62 time=49.650 ms
84 bytes from 176.16.0.2 icmp_seq=4 ttl=62 time=36.851 ms
84 bytes from 176.16.0.2 icmp_seq=5 ttl=62 time=26.259 ms

PC1>
```

Figure 3. Pinging server from PC1



```
PC2> ping 176.16.0.2

84 bytes from 176.16.0.2 icmp_seq=1 ttl=62 time=42.877 ms
84 bytes from 176.16.0.2 icmp_seq=2 ttl=62 time=45.464 ms
84 bytes from 176.16.0.2 icmp_seq=3 ttl=62 time=44.291 ms
84 bytes from 176.16.0.2 icmp_seq=4 ttl=62 time=39.615 ms
84 bytes from 176.16.0.2 icmp_seq=5 ttl=62 time=33.773 ms

PC2>
```

Figure 4. Pinging server from PC2

```
PC3> ping 176.16.0.2

84 bytes from 176.16.0.2 icmp_seq=1 ttl=62 time=47.252 ms
84 bytes from 176.16.0.2 icmp_seq=2 ttl=62 time=28.393 ms
84 bytes from 176.16.0.2 icmp_seq=3 ttl=62 time=26.475 ms
84 bytes from 176.16.0.2 icmp_seq=4 ttl=62 time=43.764 ms
84 bytes from 176.16.0.2 icmp_seq=5 ttl=62 time=43.426 ms

PC3>
```

Figure 5. Pinging server from PC3

```
tc@box:~$ ping 176.16.0.2
PING 176.16.0.2 (176.16.0.2): 56 data bytes
64 bytes from 176.16.0.2: seq=1 ttl=62 time=46.306 ms
64 bytes from 176.16.0.2: seq=3 ttl=62 time=39.424 ms
64 bytes from 176.16.0.2: seq=5 ttl=62 time=77.159 ms
64 bytes from 176.16.0.2: seq=7 ttl=62 time=86.410 ms
64 bytes from 176.16.0.2: seq=8 ttl=62 time=95.568 ms
64 bytes from 176.16.0.2: seq=9 ttl=62 time=47.774 ms
^C
--- 176.16.0.2 ping statistics ---
10 packets transmitted, 6 packets received, 40% packet loss
round-trip min/avg/max = 39.424/65.440/95.568 ms
tc@box:~$
```

Figure 6. Pinging server from TinyCore host

3.4. Dynamic NAT Overload Configuration on R1

The configuration of Dynamic NAT with Overload (PAT) on R1 involves several steps to ensure that multiple internal hosts can share a single public IP address for external communication.

Creating an Access Control List (ACL): An ACL was created to identify traffic from the private network 192.168.1.0/28. This allows R1 to selectively translate traffic originating from the internal network. The following command was used: `access-list 1 permit 192.168.1.0 0.0.0.15`

Defining a NAT Pool: A NAT pool was defined using the usable public IP addresses from the range 192.168.10.3 to 192.168.10.6, which fall within the subnet 192.168.10.0/29. These IP addresses are shared by multiple hosts using PAT.

Configuring Interfaces and Applying NAT: The inside and outside interfaces were defined to indicate where NAT translation occurs. Interface f0/0 (connected to the private network) was set as the "inside" interface, and interface s1/0 (connected to R2) was set as the "outside" interface. Finally, PAT was enabled by linking the ACL and NAT pool.

Verifying the Configuration: The commands `show ip nat translations` and `show ip nat statistics` were used to verify that NAT translations were functioning correctly. These commands confirmed that traffic from the private network was being translated to the public IP addresses.

As shown below, all private IPs are correctly translated to public ones via dynamic PAT:

```
*Dec 31 02:48:33.207: NAT*: s=192.168.1.2->192.168.10.3, d=176.16.0.2 [23923]
R1#
*Dec 31 02:48:35.203: NAT*: s=192.168.1.2->192.168.10.3, d=176.16.0.2 [23924]
*Dec 31 02:48:36.075: NAT*: s=192.168.1.3->192.168.10.3, d=176.16.0.2 [23928]
R1#
*Dec 31 02:48:37.215: NAT*: s=192.168.1.2->192.168.10.3, d=176.16.0.2 [23925]
*Dec 31 02:48:38.071: NAT*: s=192.168.1.3->192.168.10.3, d=176.16.0.2 [23929]
R1#
*Dec 31 02:48:40.087: NAT*: s=192.168.1.3->192.168.10.3, d=176.16.0.2 [23930]
R1#
*Dec 31 02:48:42.079: NAT*: s=192.168.1.3->192.168.10.3, d=176.16.0.2 [23931]
R1#
*Dec 31 02:48:43.267: NAT*: s=192.168.1.4->192.168.10.3, d=176.16.0.2 [23935]
*Dec 31 02:48:44.083: NAT*: s=192.168.1.3->192.168.10.3, d=176.16.0.2 [23932]
R1#
*Dec 31 02:48:45.255: NAT*: s=192.168.1.4->192.168.10.3, d=176.16.0.2 [23936]
R1#
*Dec 31 02:48:47.259: NAT*: s=192.168.1.4->192.168.10.3, d=176.16.0.2 [23937]
R1#
*Dec 31 02:48:49.267: NAT*: s=192.168.1.4->192.168.10.3, d=176.16.0.2 [23938]
R1#
*Dec 31 02:48:51.267: NAT*: s=192.168.1.4->192.168.10.3, d=176.16.0.2 [23939]
*Dec 31 02:48:51.287: NAT*: s=176.16.0.2, d=192.168.10.3->192.168.1.4 [13117]
R1#
*Dec 31 02:48:53.751: NAT*: s=192.168.1.5->192.168.10.3, d=176.16.0.2 [9444]
*Dec 31 02:48:54.739: NAT*: s=192.168.1.5->192.168.10.3, d=176.16.0.2 [9640]
R1#
*Dec 31 02:48:54.759: NAT*: s=176.16.0.2, d=192.168.10.3->192.168.1.5 [13643]
*Dec 31 02:48:55.727: NAT*: s=192.168.1.5->192.168.10.3, d=176.16.0.2 [9821]
R1#
*Dec 31 02:48:56.735: NAT*: s=192.168.1.5->192.168.10.3, d=176.16.0.2 [10082]
*Dec 31 02:48:56.767: NAT*: s=176.16.0.2, d=192.168.10.3->192.168.1.5 [13896]
*Dec 31 02:48:57.727: NAT*: s=192.168.1.5->192.168.10.3, d=176.16.0.2 [10263]
R1#
*Dec 31 02:48:58.731: NAT*: s=192.168.1.5->192.168.10.3, d=176.16.0.2 [10509]
*Dec 31 02:48:58.771: NAT*: s=176.16.0.2, d=192.168.10.3->192.168.1.5 [14249]
```

Figure 7. R1's debug info

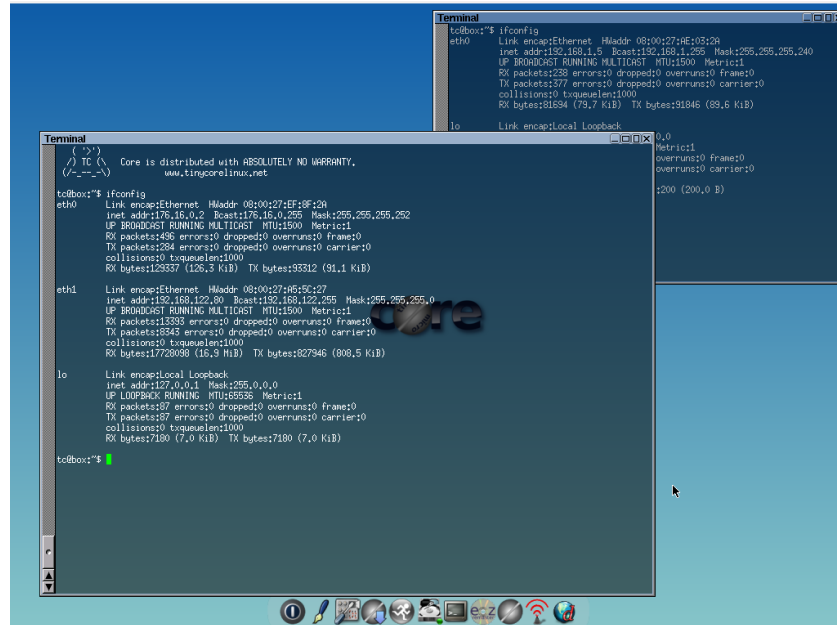
```
Pro Inside global      Inside local      Outside local      Outside global
icmp 192.168.10.3:29021 192.168.1.2:29021 176.16.0.2:29021 176.16.0.2:29021
icmp 192.168.10.3:29533 192.168.1.2:29533 176.16.0.2:29533 176.16.0.2:29533
icmp 192.168.10.3:30045 192.168.1.2:30045 176.16.0.2:30045 176.16.0.2:30045
icmp 192.168.10.3:30557 192.168.1.2:30557 176.16.0.2:30557 176.16.0.2:30557
icmp 192.168.10.3:31069 192.168.1.2:31069 176.16.0.2:31069 176.16.0.2:31069
icmp 192.168.10.3:30813 192.168.1.3:30813 176.16.0.2:30813 176.16.0.2:30813
icmp 192.168.10.3:31325 192.168.1.3:31325 176.16.0.2:31325 176.16.0.2:31325
icmp 192.168.10.3:31837 192.168.1.3:31837 176.16.0.2:31837 176.16.0.2:31837
icmp 192.168.10.3:32349 192.168.1.3:32349 176.16.0.2:32349 176.16.0.2:32349
icmp 192.168.10.3:32861 192.168.1.3:32861 176.16.0.2:32861 176.16.0.2:32861
icmp 192.168.10.3:32605 192.168.1.4:32605 176.16.0.2:32605 176.16.0.2:32605
icmp 192.168.10.3:33117 192.168.1.4:33117 176.16.0.2:33117 176.16.0.2:33117
icmp 192.168.10.3:33629 192.168.1.4:33629 176.16.0.2:33629 176.16.0.2:33629
icmp 192.168.10.3:34141 192.168.1.4:34141 176.16.0.2:34141 176.16.0.2:34141
icmp 192.168.10.3:34653 192.168.1.4:34653 176.16.0.2:34653 176.16.0.2:34653
icmp 192.168.10.3:5454 192.168.1.5:5454 176.16.0.2:5454 176.16.0.2:5454
tcp 192.168.10.3:46420 192.168.1.5:46420 176.16.0.2:22 176.16.0.2:22
R1#show ip nat stat
Total active translations: 17 (0 static, 17 dynamic; 17 extended)
Peak translations: 17, occurred 00:00:31 ago
Outside interfaces:
  Serial1/0
Inside interfaces:
  FastEthernet0/0
Hits: 400 Misses: 0
CEF Translated packets: 400, CEF Punted packets: 0
Expired translations: 25
Dynamic mappings:
-- Inside Source
[Id: 1] access-list 1 pool nat-pool refcount 17
  pool nat-pool: netmask 255.255.255.248
    start 192.168.10.3 end 192.168.10.6
    type generic, total addresses 4, allocated 1 (25%), misses 0
Appl doors: 0
Normal doors: 0
```

Figure 8. IP NAT translations and statistics

3.5. SSH Connection to the Server

First, OpenSSH was downloaded and installed on the TinyCore host (192.168.1.5) to enable SSH functionality. Once installed, a successful attempt was made to connect to the TinyCore server as the root user using the following command: `ssh root@176.16.0.2`

As we can see below, the `ifconfig` command returns the parameters of our server, confirming that the connection was successfully established.



```
tc@box:~$ ifconfig
eth0: Link encap:Ethernet  HWaddr 08:00:27:AE:03:2A
      inet addr:192.168.1.5  Bcast:192.168.1.255  Mask:255.255.255.240
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:138 errors:0 dropped:0 overruns:0 frame:0
      TX packets:177 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:81634 (79.7 KiB)  TX bytes:91846 (89.6 KiB)

eth1: Link encap:Ethernet  HWaddr 08:00:27:AE:5C:27
      inet addr:192.168.122.80  Bcast:192.168.122.255  Mask:255.255.255.0
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:1333 errors:0 dropped:0 overruns:0 frame:0
      TX packets:1843 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:17728098 (16.9 MiB)  TX bytes:827946 (808.5 KiB)

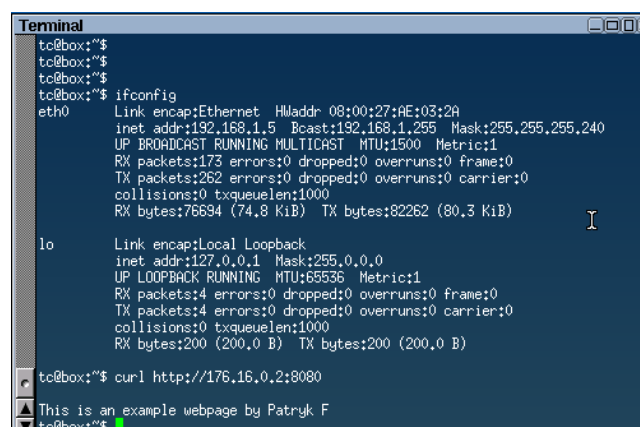
lo: Link encap:Local Loopback
      inet addr:127.0.0.1  Mask:255.0.0.0
      UP LOOPBACK RUNNING  MTU:65536  Metric:1
      RX packets:4 errors:0 dropped:0 overruns:0 frame:0
      TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:200 (200.0 B)  TX bytes:200 (200.0 B)

tc@box:~$
```

Figure 9. SSH connection to the server

3.6. Displaying the Hosting Webpage on the TinyCore Host

The `lighttpd` web server was configured and running on the server, hosting the webpage. To verify the server's functionality, the first test was performed using the `curl` command from the TinyCore host (192.168.1.5) to request the webpage:



```
tc@box:~$ ifconfig
eth0: Link encap:Ethernet  HWaddr 08:00:27:AE:03:2A
      inet addr:192.168.1.5  Bcast:192.168.1.255  Mask:255.255.255.240
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:173 errors:0 dropped:0 overruns:0 frame:0
      TX packets:262 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:76694 (74.8 KiB)  TX bytes:82262 (80.3 KiB)

lo: Link encap:Local Loopback
      inet addr:127.0.0.1  Mask:255.0.0.0
      UP LOOPBACK RUNNING  MTU:65536  Metric:1
      RX packets:4 errors:0 dropped:0 overruns:0 frame:0
      TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:200 (200.0 B)  TX bytes:200 (200.0 B)

tc@box:~$ curl http://176.16.0.2:8080
This is an example webpage by Patryk F
tc@box:~$
```

Figure 10. Accessing the webpage from TinyCore host

The `curl` command successfully returned the webpage content, confirming that the server was hosting the page correctly. Following this, the Dillo browser was downloaded and installed on the TinyCore host. Using Dillo, the server's IP address (176.16.0.2) along with the port number (8080) was entered in the address bar: `http://176.16.0.2:8080`

The hosting webpage was successfully displayed, confirming that the `lighttpd` web server was properly running on the server and that the page was accessible from the TinyCore host. A screenshot of the webpage is provided below to document the successful connection and page display:

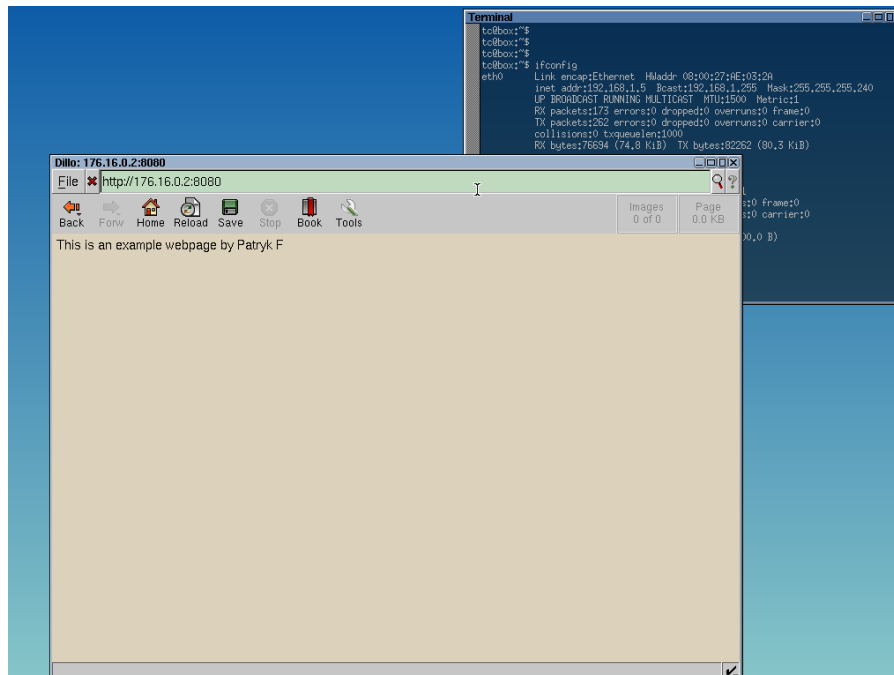


Figure 11. Accessing the webpage from TinyCore host

4. Summary

This lab focused on configuring Dynamic NAT with Overload (PAT) to enable communication between multiple internal hosts and a TinyCore server. The server was set up with two network interfaces and software like OpenSSH, `lighttpd`, and Dillo. The router's NAT configuration allowed internal IPs to be translated to public ones. Ping tests, SSH access, and webpage display via `curl` and Dillo confirmed successful setup, demonstrating the functionality of Dynamic NAT with Overload.