RCOM Project 2

Gonçalo Miranda (up202108773)
Sophie Large (up202303141)

Table of Contents

Introduction

The goal of this project was to set up a small network and prove that it is correctly setup using a download application, which was also developed for this project.

Part 1 - Development of the Download Application (app.c and app.h)

This part of the project consists in the development of a small app that downloads files using the FTP protocol. The app works when provided with an URL such as `ftp://[<username>[:<password>]@]<host>[:<port>]/<path`.

Example: `ftp://ftp.up.pt/pub/hello.txt`

During the development of the app part of the project, we had the chance to understand Unix utilities to handle URLs and to communicate with sockets, the FTP protocol and the concept of DNS.

Architecture of the Application

1. Parsing the URL string given as the argument to the program to extract its fragments, such as the username, password, host, port, and path.
2. Connecting to the host on the specified port (being 21 the default port, the standard FTP control port).
3. Entering passive mode and receiving the host and port of the data connection.
4. Connecting to the data connection host and port.
5. Notifying the server that it should start transferring the desired file.
6. Reading the incoming data from the server and writing it locally to the download. file, being the extension of the last part of the path (last string after the last "/").
7. Closing the connections and terminating the app.

TCP sockets mediate the network communication. By making use of the URL parser we designed, via a state machine and regular expressions, and getaddrinfo(), the FTP server's IP and control port are obtained (in case there's no port specified, 21 is the value assumed). A TCP connection is established and the pro- gram starts sending commands through the control connection, which will be responded to by the server. Those responses carry response codes that are properly extracted and checked to understand if the program is working as intended. If any of the response codes sent by the server doesn't match the expected code, the program is terminated with exit code -1. In case there's no username specified by the user, "anonymous" is assumed as the username, since it's the default value we decided to provide. After that, passive mode is entered and, if its successfully entered, the server responds with code 227 and the host's IP and port. After establishing a data connection , the retr command is sent and, after that, the buffers of data from the data connection begin getting read into a file named "download.`<extension>`", where the extension is the last string after the last "/" in the file path provided by the user. Finally, the connection is closed and the program terminates.

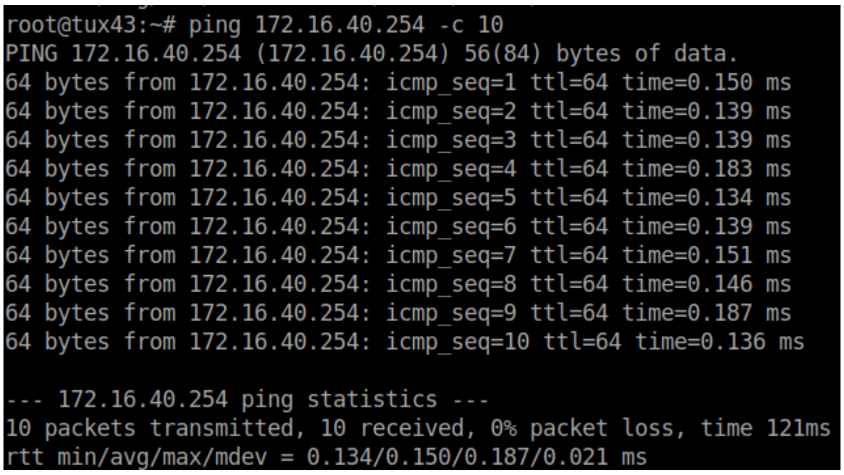
Part 2 - Experiences

Experience 1 - Configure an IP Network

Step By Step

Note that, in this experience, the workbench used was workbench 4.

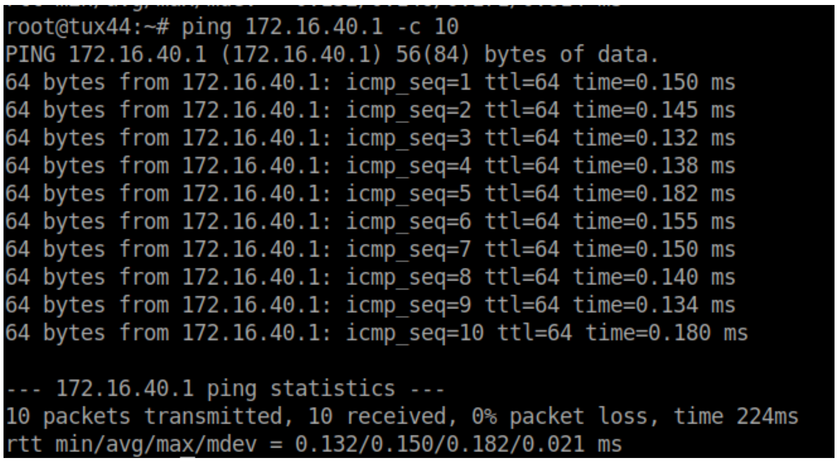
1. Connect E0 on Tux**Y**3 and Tux**Y**4 on any of the switch ports.
2. Configure Tux**Y**3 and Tux**Y**4, by performing the following commands:
    - Tux**Y**3
      ```
      $ ifconfig eth0 up
      $ ifconfig eth0 172.16.Y0.1/24
      ```
    - Tux**Y**4
      ```
      $ ifconfig eth0 up
      $ ifconfig eth0 172.16.Y0.254/24
      ```
3. Now we can check both the IP and MAC addresses of both computers:
    - TuxY3:
        - IP Address: 172.16.**Y**0.1/24
        - MAC Address: 00:21:5a:61:2f:d4
    - TuxY4:
        - IP Address: 172.16.**Y**0.254/24
        - MAC Address: 00:21:5a:5a:7b:ea
4. We now can test the connectivity between the computers by pinging each other (TuxY3 pings TuxY4 and vice-versa).
    - TuxY3
      ```
      $ ping 172.16.Y0.254 -c 10
      ```



```
root@tux43:~# ping 172.16.40.254 -c 10
PING 172.16.40.254 (172.16.40.254) 56(84) bytes of data.
64 bytes from 172.16.40.254: icmp_seq=1 ttl=64 time=0.150 ms
64 bytes from 172.16.40.254: icmp_seq=2 ttl=64 time=0.139 ms
64 bytes from 172.16.40.254: icmp_seq=3 ttl=64 time=0.139 ms
64 bytes from 172.16.40.254: icmp_seq=4 ttl=64 time=0.183 ms
64 bytes from 172.16.40.254: icmp_seq=5 ttl=64 time=0.134 ms
64 bytes from 172.16.40.254: icmp_seq=6 ttl=64 time=0.139 ms
64 bytes from 172.16.40.254: icmp_seq=7 ttl=64 time=0.151 ms
64 bytes from 172.16.40.254: icmp_seq=8 ttl=64 time=0.146 ms
64 bytes from 172.16.40.254: icmp_seq=9 ttl=64 time=0.187 ms
64 bytes from 172.16.40.254: icmp_seq=10 ttl=64 time=0.136 ms

--- 172.16.40.254 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 121ms
rtt min/avg/max/mdev = 0.134/0.150/0.187/0.021 ms
```

- TuxY4:

  ```
  $ ping 172.16.Y0.1 -c 10
  ```

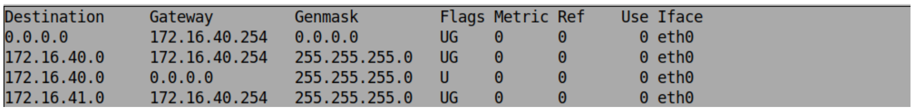
  ```
  root@tux44:~# ping 172.16.40.1 -c 10
  PING 172.16.40.1 (172.16.40.1) 56(84) bytes of data.
  64 bytes from 172.16.40.1: icmp_seq=1 ttl=64 time=0.150 ms
  64 bytes from 172.16.40.1: icmp_seq=2 ttl=64 time=0.145 ms
  64 bytes from 172.16.40.1: icmp_seq=3 ttl=64 time=0.132 ms
  64 bytes from 172.16.40.1: icmp_seq=4 ttl=64 time=0.138 ms
  64 bytes from 172.16.40.1: icmp_seq=5 ttl=64 time=0.182 ms
  64 bytes from 172.16.40.1: icmp_seq=6 ttl=64 time=0.155 ms
  64 bytes from 172.16.40.1: icmp_seq=7 ttl=64 time=0.150 ms
  64 bytes from 172.16.40.1: icmp_seq=8 ttl=64 time=0.140 ms
  64 bytes from 172.16.40.1: icmp_seq=9 ttl=64 time=0.134 ms
  64 bytes from 172.16.40.1: icmp_seq=10 ttl=64 time=0.180 ms

  --- 172.16.40.1 ping statistics ---
  10 packets transmitted, 10 received, 0% packet loss, time 224ms
  rtt min/avg/max/mdev = 0.132/0.150/0.182/0.021 ms
  ```

  The connection is working in case all the packets are correctly received.

5. Now, we can check the forwarding and the ARP tables. We're doing it on TuxY3.

   In:

   ```
   $ route -n
   ```

   out: 

   ```
   Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
   0.0.0.0         172.16.40.254   0.0.0.0         UG    0      0        0 eth0
   172.16.40.0     172.16.40.254   255.255.255.0   UG    0      0        0 eth0
   172.16.40.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
   172.16.41.0     172.16.40.254   255.255.255.0   UG    0      0        0 eth0
   ```

   In:

   ```
   $ arp -a
   ```

   out:

   ```
   ? (172.16.Y0.254) at 00:21:5a:5a:7b:ea [ether] on eht0
   ```

6. We are now deleting the arp table entries in Tux**Y**3. In:

   ```
   $ arp -d 172.16.Y0.254
   ```

   To test if the deletion occurred correctly, we can perform

   In:

   ```
   $ arp -a
   ```

   out:

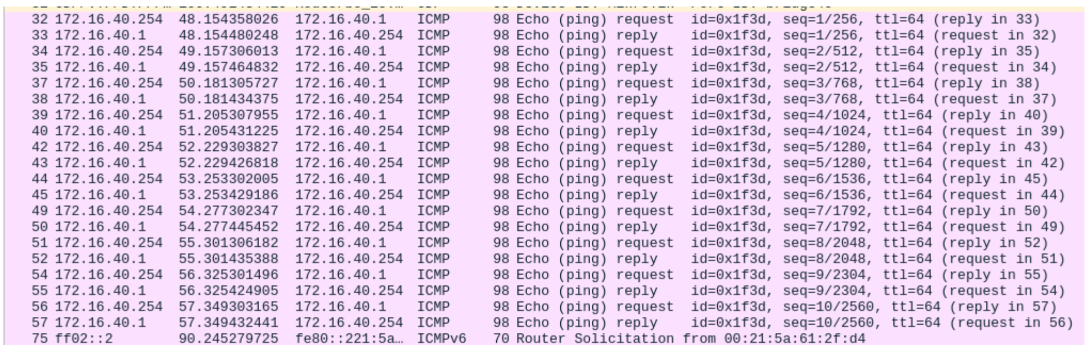
   No entries on the table

7. Now, we can start Wireshark in Tux**Y**3.eth0 and start capturing packets.
8. After that, we can ping Tux**Y**4 from Tux**Y**3 10 times.

   ```
   $ ping 172.16.40.254 -c 10
   ```

9. Finally, we can stop capturing the packets.

10. When saving the logs and analyzing them, we've got the following:



```
32 172.16.40.254  48.154358026  172.16.40.1    ICMP     98 Echo (ping) request  id=0x1f3d, seq=1/256, ttl=64 (reply in 33)
33 172.16.40.1    48.154480248  172.16.40.254  ICMP     98 Echo (ping) reply    id=0x1f3d, seq=1/256, ttl=64 (request in 32)
34 172.16.40.254  49.157306013  172.16.40.1    ICMP     98 Echo (ping) request  id=0x1f3d, seq=2/512, ttl=64 (reply in 35)
35 172.16.40.1    49.157464832  172.16.40.254  ICMP     98 Echo (ping) reply    id=0x1f3d, seq=2/512, ttl=64 (request in 34)
37 172.16.40.254  50.181305727  172.16.40.1    ICMP     98 Echo (ping) request  id=0x1f3d, seq=3/768, ttl=64 (reply in 38)
38 172.16.40.1    50.181434375  172.16.40.254  ICMP     98 Echo (ping) reply    id=0x1f3d, seq=3/768, ttl=64 (request in 37)
39 172.16.40.254  51.205307955  172.16.40.1    ICMP     98 Echo (ping) request  id=0x1f3d, seq=4/1024, ttl=64 (reply in 40)
40 172.16.40.1    51.205431225  172.16.40.254  ICMP     98 Echo (ping) reply    id=0x1f3d, seq=4/1024, ttl=64 (request in 39)
42 172.16.40.254  52.229303827  172.16.40.1    ICMP     98 Echo (ping) request  id=0x1f3d, seq=5/1280, ttl=64 (reply in 43)
43 172.16.40.1    52.229426818  172.16.40.254  ICMP     98 Echo (ping) reply    id=0x1f3d, seq=5/1280, ttl=64 (request in 42)
44 172.16.40.254  53.253302005  172.16.40.1    ICMP     98 Echo (ping) request  id=0x1f3d, seq=6/1536, ttl=64 (reply in 45)
45 172.16.40.1    53.253429186  172.16.40.254  ICMP     98 Echo (ping) reply    id=0x1f3d, seq=6/1536, ttl=64 (request in 44)
49 172.16.40.254  54.277302347  172.16.40.1    ICMP     98 Echo (ping) request  id=0x1f3d, seq=7/1792, ttl=64 (reply in 50)
50 172.16.40.1    54.277445452  172.16.40.254  ICMP     98 Echo (ping) reply    id=0x1f3d, seq=7/1792, ttl=64 (request in 49)
51 172.16.40.254  55.301306182  172.16.40.1    ICMP     98 Echo (ping) request  id=0x1f3d, seq=8/2048, ttl=64 (reply in 52)
52 172.16.40.1    55.301435388  172.16.40.254  ICMP     98 Echo (ping) reply    id=0x1f3d, seq=8/2048, ttl=64 (request in 51)
54 172.16.40.254  56.325301496  172.16.40.1    ICMP     98 Echo (ping) request  id=0x1f3d, seq=9/2304, ttl=64 (reply in 55)
55 172.16.40.1    56.325424905  172.16.40.254  ICMP     98 Echo (ping) reply    id=0x1f3d, seq=9/2304, ttl=64 (request in 54)
56 172.16.40.254  57.349303165  172.16.40.1    ICMP     98 Echo (ping) request  id=0x1f3d, seq=10/2560, ttl=64 (reply in 57)
57 172.16.40.1    57.349432441  172.16.40.254  ICMP     98 Echo (ping) reply    id=0x1f3d, seq=10/2560, ttl=64 (request in 56)
75 ff02::2        90.245279725  fe80::221:5a…  ICMPv6   70 Router Solicitation from 00:21:5a:61:2f:d4
```

Questions

1. What are the ARP packets and what are they used for?
   The ARP (*Address Resolution Protocol*) packets are used to establish a connection between an IP and a MAC address.
2. What are the MAC and IP addresses of ARP packets and why?
   The ARP packets carry 2 IP addresses: the destiny machine's IP address (172.16.**Y**0.254, Tux**Y**4) the the source machine's IP address (172.16.**Y**0.01, Tux**Y**3). Tux**Y**4 then proceeds to send another ARP packet to TuxY3 that contains its MAC address (00:21:5a:5a:7b:ea).
3. What packets does the ping command generate?
   While the destination's MAC address isn't received, *ping* generates ARP packets. After connecting the IP address to the corresponding MAC address,it generates ICMP (*Internet Control Message Protocol*) packets.
4. What are the MAC and IP addresses of the ping packets?
   The IP and MAC addresses used in ICMP are the ones from the machines in use - in this case, Tux**Y**3 and Tux**Y**4.
5. How to determine if a receiving Ethernet frame is ARP, IP, ICMP?
   Each frame's protocol designation can be checked in the "Protocol" column of the Wireshark capture. The protocol is usually indicated in the first 2 bytes of the frame, in its header.
6. How to determine the length of a receiving frame?
   Wireshark has a column that specifies the frame size,in bytes. Usually that is also present in the frame's header.
7. What is the loopback interface and why is it important?
   It's a virtual interface that's always reachable as long as one of the switch's interfaces are on. Because of this, it's possible to periodically check if the network's connections are properly configured.

Experience 2 - Implement two bridges in a switch

Step By Step

Note that, in this experience, the workbench used was workbench 4.

1. Connect E0 on Tux**Y**2 to any of the switch ports. After that,configure Tux**Y**2, by performing the following commands:
   ```
   $ ifconfig eth0 up
   $ ifconfig eth0 172.16.Y1.1/24
   ```
2. Now, we can create two bridges in the switch, using the following commands on GTKTerm.
   - Bridge**Y**0:
     ```
     /interface bridge add name=bridgeY0
     ```
   - Bridge**Y**1:
     ```
     /interface bridge add name=bridgeY1
     ```
3. Afterwards, we can remove the ports where Tux**Y**2, Tux**Y**3, and Tux**Y**4 are connected from the default bridge and add them to the corresponding ports to bridge**Y**0 and bridge**Y**1.
   - Removing the original ports:
     ```
     /interface bridge port remove [find interface=ether1]
     /interface bridge port remove [find interface=ether2]
     /interface bridge port remove [find interface=ether3]
     ```
   - Adding the current ports:
     ```
     /interface bridge port add bridge=bridgeY0 interface=ether1
     /interface bridge port add bridge=bridgeY0 interface=ether2
     /interface bridge port add bridge=bridgeY1 interface=ether3
     ```
4. Now, we should switch to Tux**Y**3 and start monitoring the packets.
5. Ping Tux**Y**4 and Tux**Y**2 from Tux**Y**3.

   In:
   ```
   $ ping 172.16.Y0.254 -c 10
   ```
   Out:
   In Tux**Y**4, all 10 packets have been received successfully!

   In:
   ```
   $ ping 172.16.Y1.1 -c 10
   ```
   Out:
   In Tux**Y**2 -> connect: Network is unreachable
6. Afterwards, stop the capture.
7. Proceed by starting the capture of eth0 in Tux**Y**2, Tux**Y**3 and Tux**Y**4.
8. From Tux**Y**3, ping 172.16.**Y**0.255.
   ```
   $ ping 172.16.Y0.255 -c 10
   ```
9. Stop the captures.
10. Restart the previous captures.
11. From Tux**Y**2, ping 172.16.**Y**1.255.
12. Enable ICMP.
13. Enable IP forwarding.

Questions

1. How to configure bridgeY0?
   In order to connect Tux**Y**3 and Tux**Y**4, bridge**Y**0 was configured, creating a subnet. To configure the bridge properly, we delete previous configurations and default connections when the machines first connected and configure new ports that connect to each

machine. We can do this by using the commands stated in the previous step-by-step guide.

2. How many broadcast domains are there? How can you conclude it from the logs?
   Since we implemented two bridges (bridge**Y**0 and bridge**Y**0), there are two broadcast domains. We can conclude this because, after analyzing the Wireshark logs, Tux**Y**3 got a response from Tux**Y**4 but not from Tux**Y**2, since they're on different bridges and, therefore, different broadcast domains.

Experience 3 - Configure a router in Linux

Step By Step

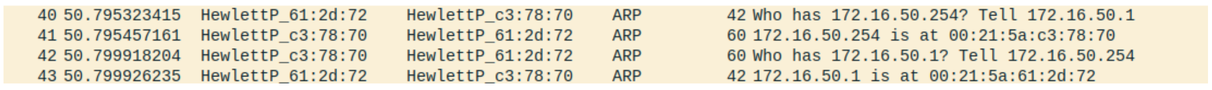
Note that, in this experience, the workbench used was workbench 5.

1. We need to configure Tux**Y**4 as a router, by:
   - Connect Tux**Y**4's eth1 to a switch port.
   - After connecting the cables, config the port itself.
     ```
     $ ifconfig eth1 172.16.Y1.253/24
     ```
   - Connect Tux**Y**2 to the switch console and add it to bridge**Y**1.
     In GTKTerm:
     ```
     /interface bridge port remove [find interface=etherXX]
     /interface bridge port add interface=etherXX bridge=bridgeY1
     ```
     , being XX the switch's port number.
   - Enable the IP forwarding on Tux**Y**4.
     ```
     $ echo 1 > /proc/sys/net/ipv4/ip_forward
     ```
   - Disable ICMP echo-ignore broadcast on Tux**Y**4.
     ```
     $ echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
     ```
2. Now, check the MAC and IP addresses in Tux**Y**4's eth0 and eth1.
   - in:
     ```
     $ ifconfig
     ```
   - out:
     i. eth0:
        MAC Address: 00:21:5a:c3:78:70 IP Address: 172.16.**Y**0.254
     ii. eth1:
        MAC Address: 00:c0:df:08:d5:b0 IP Address: 172.16.**Y**1.253
3. Reconfigure Tux**Y**3 and Tux**Y**2 so that they can reach each other.
   - Tux**Y**2:
     ```
     $ route add -net 172.16.50.0/24 gw 172.16.51.253
     ```
   - Tux**Y**3:
     ```
     $ route add -net 172.16.Y1.0/24 gw 172.16.Y0.254
     ```
4. Now, check the routes available in each machine.
   ```
   $ route -n
   ```
5. Start capturing packets with Wireshark.

6. From Tux**Y**3, ping the other network interfaces and check their connectivity.
   in:
   ```
   $ ping 172.16.Y0.254 -c 10
   $ ping 172.16.Y1.253 -c 10
   $ ping 172.16.Y1.1 -c 10
   ```
   out:
   ```
   All 10 packets for each ping call received correctly in
   every machine.
   ```
7. Stop the log capture and check the logs.
8. Now, start a Wireshark capture on Tux**Y**4's eth0 and eth1.
9. In each machine, clean the ARP tables.
   - Tux**Y**2:
     ```
     $ arp -d 172.16.Y1.253
     ```
   - Tux**Y**3:
     ```
     $ arp -d 172.16.Y0.254
     ```
   - Tux**Y**4:
     ```
     $ arp -d 172.16.Y0.1 $ arp -d 172.16.Y1.1
     ```
10. In Tux**Y**3, ping Tux**Y**2.
    ```
    $ ping 172.16.Y1.1 c 10
    ```
11. Stop both Tux**Y**4 Wireshark captures.

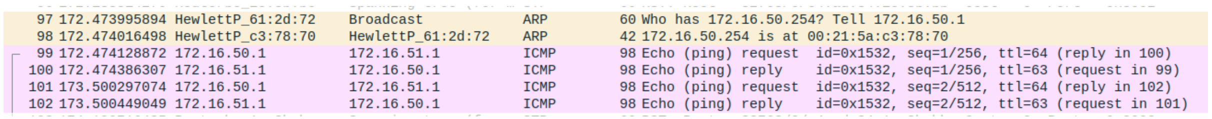
Questions

1. What are the commands required to configure this experience?
   The necessary commands to configure the experience are the ones displayed in the step-by-step guide presented before.
2. What routes are there in the tuxes? What are their meaning?
   There is a route on Tux**Y**2 and another in Tux**Y**3. Both those routes have Tux**Y**4 as the gateway, since Tux**Y**4 is the only one common to the created bridges (it's a router).
3. What information does an entry of the forwarding table contain?
   Each entry of the forwarding table has the destiny address and its gateway.
4. What ARP messages, and associated MAC addresses, are observed and why?
   When pinging Tux**Y**2 from Tux**Y**3, the ARP packages carry the IP and MAC addresses of Tux**Y**3 and Tux**Y4** instead of the Tux**Y**2 as the destiny address. This happens because of the routing we created: here, Tux**Y**2's address is not known by Tux**Y**3. The latter only knows the Tux**Y**4's address (gateway), that will later lead to Tux**Y**2.



```
40 50.795323415  HewlettP_61:2d:72   HewlettP_c3:78:70   ARP   42 Who has 172.16.50.254? Tell 172.16.50.1
41 50.795457161  HewlettP_c3:78:70   HewlettP_61:2d:72   ARP   60 172.16.50.254 is at 00:21:5a:c3:78:70
42 50.799918204  HewlettP_c3:78:70   HewlettP_61:2d:72   ARP   60 Who has 172.16.50.1? Tell 172.16.50.254
43 50.799926235  HewlettP_61:2d:72   HewlettP_c3:78:70   ARP   42 172.16.50.1 is at 00:21:5a:61:2d:72
```

5. What ICMP packets are observed and why?
   The ICMP packets carry the source (Tux**Y**3) and destiny's IP addresses (Tux**Y**2). The fact that those two are the IP's carried means the network is correctly configured.



```
 97 172.473995894 HewlettP_61:2d:72   Broadcast           ARP    60 Who has 172.16.50.254? Tell 172.16.50.1
 98 172.474016498 HewlettP_c3:78:70   HewlettP_61:2d:72   ARP    42 172.16.50.254 is at 00:21:5a:c3:78:70
 99 172.474128872 172.16.50.1         172.16.51.1         ICMP   98 Echo (ping) request  id=0x1532, seq=1/256, ttl=64 (reply in 100)
100 172.474386307 172.16.51.1         172.16.50.1         ICMP   98 Echo (ping) reply    id=0x1532, seq=1/256, ttl=63 (request in 99)
101 173.500297074 172.16.50.1         172.16.51.1         ICMP   98 Echo (ping) request  id=0x1532, seq=2/512, ttl=64 (reply in 102)
102 173.500449049 172.16.51.1         172.16.50.1         ICMP   98 Echo (ping) reply    id=0x1532, seq=2/512, ttl=63 (request in 101)
```

6. What are the IP and MAC addresses associated to ICMP packets and why?
   Each ICMP packet observed after the ping from Tux**Y**3 to Tux**Y**2, carry Tux**Y**3's IP address as the source IP address and the Tux**Y**3's IP address as the destiny address,

however, the ICMP packets carry Tux**Y**4's MAC address, due to the fact that this machine is the one connecting the two bridges.

Experience 4 - Configure a Commercial Router and Implement NAT

Step By Step

Note that, in this experience, the workbench used was workbench 5.

1. Connect the router's eth1 port to the port **Y**1 of the server rack in the lab - NAT is, by default, enabled.
2. Connect the router's eth2 port to the switch.
3. Delete the default ports of the switch's etherXX (XX represents the port number on which the cable is connected) and connect it to bridge**Y**1.
    - Using GTKTerm, in the switch's console, run:
      ```
      /interface bridge port remove [find interface=etherXX]
      /interface bridge port add bridge=bridgeY1 interface=etherXX
      ```
4. Switch the cable that was connected to the switch to the router.
5. Reset the router's settings and configure its IP address.
    - Using GTKTerm, switch the baudrate to 115200 and the serial port to /dev/ttyS0. After that, run:
      ```
      /system reset-configuration
      ```
    - Login with username "admin" and no password.
    - Run:
      ```
      /ip address add address=172.16.1.Y9/24 interface=ether1 /ip
      address add address=172.16.Y1.254/24 interface=ether2
      ```
6. Configure all the computer's default routes, as well as the router's.
    - Tux**Y**2:
      ```
      $ route add default gw 172.16.Y1.254
      ```
    - Tux**Y**3:
      ```
      $ route add default gw 172.16.Y0.254
      ```
    - Tux**Y**4:
      ```
      $ route add default gw 172.16.Y1.254
      ```
    - Using GTKTerm on the router's console, run:
      ```
      /ip route add
      dst-address=172.16.Y0.0/24 gateway=172.16.Y1.253
      /ip route add dst-address=0.0.0.0/0 gateway=172.16.1.254
      ```
7. Start a Wireshark capture on Tux**Y**3 to check if it can ping all the network interfaces of Tux**Y**2, Tux**Y**4 and the router.

   In:
   ```
   $ ping 172.16.Y0.254 -c 10
   $ ping 172.16.Y1.1 -c 10
   $ ping 172.16.Y1.254 -c 10
   ```
   Out:

   All 10 packets for each ping call received correctly in every machine.
8. On Tux**Y**2, disable accept_redirects.
   ```
   $ sysctl net.ipv4.conf.eth0.accept_redirects=0
   $ sysctl net.ipv4.conf.all.accept_redirects=0
   ```

9. Remove the route that connects Tux**Y**2 to Tux**Y**4.
```
$ route del -net 172.16.Y0.0 gw 172.16.Y1.253 netmask
255.255.255.0
```
10. On Tux**Y**2, start a Wireshark capture and ping Tux**y**3. In:

    In:
    ```
    $ ping 172.16.Y0.1
    ```
    Out:

    All 10 packets of the ping call received correctly (connection was using the proper router instead of Tux**Y**4 as a router).

11. Now, let's check the connection's route. In:

    In:
    ```
    $ traceroute -n 172.16.Y0.1
    ```
    Out:

    traceroute to 172.16.**Y**0.1 (172.16.**Y**0.1), 30 hops max, 60 byte packets
    1 172.16.**Y**1.254 (172.16.**Y**1.254) 0.200 ms 0.204 ms 0.224 ms
    2 172.16.**Y**1.253 (172.16.**Y**1.253) 0.354 ms 0.345 ms 0.344 ms
    3 tux**Y**1 (172.16.**Y**0.1) 0.596 ms 0.587 ms 0.575 ms

12. Recreate the route that connects Tux**Y**2 to Tux**Y**4.
    ```
    $ route add -net 172.16.Y0.0/24 gw 172.16.Y1.253
    ```
13. Trace the route from Tux**Y2** to Tux**Y**3.

    In:
    ```
    $ traceroute -n 172.16.Y0.1
    ```
    Out:

    traceroute to 172.16.**Y**0.1 (172.16.**Y**0.1), 30 hops max, 60 byte packets
    1 172.16.**Y**1.253 (172.16.**Y**1.253) 0.196 ms 0.180 ms 0.164 ms
    2 tux**Y**1 (172.16.**Y**0.1) 0.414 ms 0.401 ms 0.375 ms

14. Reactivate accept_redirects.
    ```
    $ sysctl net.ipv4.conf.eth0.accept_redirects=1
    $ sysctl net.ipv4.conf.all.accept_redirects=1
    ```
15. From Tux**Y**3, ping the Lab's router.
    ```
    $ ping 172.1.254
    ```
16. Deactivate the router's NAT. On GTKTerm, on the router's console:
    ```
    /ip firewall net disable 0
    ```
    On Tux**Y**3, ping the Lab's router again.

    In:
    ```
    $ ping 172.16.1.254
    ```
    Out:
    ```
    No connection.
    ```

17. Reactivate the router's NAT.
    ```
    /ip firewall nat enable 0
    ```

Questions

1. How to configure a static route in a commercial router?
   We start by resetting its settings, add it to the corresponding bridge and give it its own external and internal IP address.
2. What are the paths followed by the packets in the experiments carried out and why?
   In the first part of the experience, where Tux**Y**2 wasn't connected to Tux**Y**4,the data packets were redirected (ICMP redirect) through the implemented router to the destination IP address.
   In the second part of the experience, there were no redirects, since the shortest connection in the network was available.
3. How to configure NAT in a commercial router? Using the GTK Terminal command `/ip firewall nat enable 0` in the router's terminal.
4. What does NAT do? NAT (Network Address Translation) translates addresses from the local network to a single public address and vice-versa. Because of this, when a packet is sent to an external network, the package is sent with the public address as the source. When the destiny machine sends back a response, it sends that response to the public address, which is then re-translated again to the local address that sent the packet in the first place. This is useful, since we can reduce the number of public addresses used on a network.

Experience 5 - DNS

Step by Step

Note that, in this experience, the workbench used was workbench 4 and in room I320.

1. Configure DNS in all machines.
   - Tux**Y**2:
     `$ nano /etc/resolv.conf` and add the line `nameserver 172.16.2.1`
   - Tux**Y**3:
     `$ nano /etc/resolv.conf` and add the line `nameserver 172.16.2.1`
   - Tux**Y**4:
     `$ nano /etc/resolv.conf` and add the line `nameserver 172.16.2.1`
2. In all 3 machines, ping website names to check if they can be used as host.
   In:
   ```
   $ ping google.com -c 10
   ```
   Out:
   All 10 packets correctly transmitted and received from every machine.

Questions

1. How to configure the DNS service at a host?
   DNS is configured by adding the line "nameserver " to the end of the file /etc/resolv.conf of all connected machines.
2. What packets are exchanged by DNS and what information is transported? The first few packets transported are DNS packets, so that the router can identify and translate the destination IP address.

Experience 6 - TCP Connections

Step By Step

Note that, in this experience, the workbench used was workbench 4.

1. Compile the download application in TuxY3.
2. In TuxY3, start a Wireshark capture and execute the application.
3. Check if the file has correctly arrived and stop the Wireshark capture.
4. Check the Wireshark log and analyze the data packets exchanged.
5. Start the Wireshark capture again and restart the download in TuxY3. After it started, switch to TuxY2 and start a download there too.
6. Stop the capture and analyze the logs.

Questions

1. How many TCP connections are opened by your FTP application?
   The application has 2 TCP connections, one to send commands and another to receive the file.
2. In what connection is transported the FTP control information?
   The control information is transported through the connection that has been setup to send commands to the server and fetch its responses.
3. What are the phases of a TCP connection?
   A TCP connection is divided in 3 main phases:
   - Connection Establishment (Three-Way Handshake), where the client sends a TCP segment to the server with the **SYN** flag. That is followed by a response from the server, which is a TCP segment that carries **both** the **SYN** and the **ACK** flags. To end this three-way handshake, the client sends a last TCP segment with the **ACK** flag, which acknowledges the receipt of the previous server response.
   - Data Transfer, where the data is encapsulated into TCP segments and is exchanged between the server and the client.
   - Connection Termination (Four-Way Handshake), where the client starts by sending a TCP segment with the **FIN** flag to the server, who responds with a segment carrying the **ACK** flag and another one that carries another **FIN** flag. Then, the client answers back with the last TCP segment, that carries an ACK flag.
4. How does the ARQ TCP mechanism work? What are the relevant TCP fields? What relevant information can be observed in the logs?
   Automatic Repeat Request (ARQ) is used to retransmit information on a congested network, which is a network where packets have been lost. In order to lose packets, many have to be sent at the same time (Additive Increase). We can use Slow Start when, instead of adding a one to CongestionWindow in each transmission, we double the CongestionWindow in each retransmission. Packet loss can either occur by timeout (there's a Multiplicative Decrease, making Congestion Window 1 and increasing it back until its half the value that's gotten via slow start. After that, Additive Increase starts being incremented by 1) or due to 3 consecutive ACKs (there's a Multiplicative Decrease, reducing Congestion Window to half and the Additive Increase starts being incremented by 1).
5. How does the TCP congestion control mechanism work? What are the relevant fields. How did the throughput of the data connection evolve along the time? Is it according to

the TCP congestion control mechanism?
Each sender determines the channel capacity in order to send one or less packets. Because of that, there's another new parameter in the connection - CongestionWindow. If the network's congestion level increases, the CongestionWindow decreases and vice-versa.

6. Is the throughput of a TCP data connection disturbed by the appearance of a second TCP connection? How?
   Yes - by creating more than once TCP connection, the bandwidth will be divided by all the connections, slowing each one of them down.

Conclusions

By correctly implementing all the steps of the project, from the app to the experiences, we had the chance to better understand how the TCP protocol works, how the configuration of a complex network is done and how the many kinds of packets behave and what information they carry.

Annexes

1. Code

- **Available in folder app.**

2. Wireshark Captures

- **Logs and prints availables in resource folder.**