bib/main.bib

Computer Networks

Redes de Computadores

Bruno Carvalho
up201606517

João Malheiro
up201605926

Carlos Daniel Gomes
up201603404

December 22, 2018

Contents

1 Summary

This project was developed for the curricular course RCOM (Computer Networks). It was finished successfully by implementing all the required functionalities. We built a fully working network application and established the configuration of a network.

2 Introduction

This project goal was to developed a fully working network and a download application. The network consists of two VLAN's inside a switch. To complement that, our group developed a socket based application to download, using a FTP client.

In regards to this report, it's objective is to show and explain everything about this second project and it is divided in these sections:

Part 1 : Download Application: description of its architecture while also analyzing and discussing results.

Part 2: Network Configuration: description of its architecture, every experience objectives and analyzes with the answers to all the questions presented and logs saved during the development.

Conclusion: last analysis of the project and final reflections.

3 Part 1: Download application

3.1 Files

debug.c and debug.h: This files help the user understand the state of the download.

pipeline.c and pipeline.h: Contains the functions responsible for make every step defined in main.c.

main.c: File that organize the functions calls. The order found here mirrors the download sequence:

1. Parse input FTP URL
2. Resolve hostname to server's IPv4 IP address and open protocol socket
3. Login to the server (user + password)
4. Enter passive mode and open passive socket
5. Retrieve command for file
6. Download file
7. Close connection to server

3.2 Download Report

In the report was used the URL ftp://ftp.up.pt/pub/debian/README. However the parser accepts URL's in format ftp://[<user>:<password>@]<host>/<url-path>.

```
1. FTP URL: ftp://ftp.up.pt/pub/debian/project/trace/mirrors.up.pt
 Defaulting username to anonymous
 Defaulting password to upstudent-rcom
 url.protocol: ftp
 url.username: anonymous
 url.password: upstudent-rcom
 url.hostname: ftp.up.pt
 url.pathname: pub/debian/project/trace/
 url.filename: mirrors.up.pt
 url.port: 21
2. Resolve hostname ftp.up.pt and setup control socket
 2.1. Resolved hostname ftp.up.pt successfully
  host->h_name: mirrors.up.pt
  host->h_addrtype: 2 [IPv4=2]
 2.2. Protocol IP Address: 193.137.29.15
      Establishing control connection with FTP server
 2.3. Opened control socket for FTP's protocol connection
 2.4. Connected control socket to 193.137.29.15:21
    [REPLY] 220-Welcome to the University of Porto's mirror archive (mirrors.up.pt)
    [REPLY] 220----------------------------------------------------------------
    [REPLY] 220-
    [REPLY] 220-All connections and transfers are logged. The max number of connections is 200.
    [REPLY] 220-
    [REPLY] 220-For more information please visit our website: http://mirrors.up.pt/
    [REPLY] 220-Questions and comments can be sent to mirrors@uporto.pt
    [REPLY] 220-
    [REPLY] 220-
    [REPLY] 220
 2.5. Successfully established control socket connection
3. Send USER and PASS login commands to FTP server
    [COMMD] USER anonymous
    [REPLY] 331 Please specify the password.
 3.1. Server confirmed user anonymous
    [COMMD] PASS upstudent-rcom
    [REPLY] 230 Login successful.
 3.2. Server acknowledges login, proceeding
4. Establish passive connection with FTP server
    [COMMD] PASV
    [REPLY] 227 Entering Passive Mode (193,137,29,15,229,4).
 4.1. Entered Passive Mode: (193,137,29,15,229,4)
 4.2. Passive IP Address: 193.137.29.15:58628
 4.3. Opened passive socket for FTP's passive connection
 4.4. Connected passive socket to 193.137.29.15:58628
 4.5. Successfully established passive socket connection
5. Retrieve file from Server (retrieve command)
    [COMMD] RETR pub/debian/project/trace/mirrors.up.pt
    [REPLY] 150 Opening BINARY mode data connection for pub/debian/project/trace/mirrors.up.pt (802 bytes).
 5.1. Confirmed, server retrieved pub/debian/project/trace/mirrors.up.pt
6. Download file mirrors.up.pt into current directory
 6.1. Opened output file successfully
      Reading...
 6.2. Done.
```

4 Part 2: Network

4.1 Exp 1. Configure an IP Network

Cable configuration:
`tux31@eth0—tux34@eth0`

Commands:
tux31:
```
ifconfig eth0 up
ifconfig eth0 172.16.30.1/24
route add default gw 172.16.30.254
```
tux34:
```
ifconfig eth0 up
ifconfig eth0 172.16.30.254/24
```

We start by creating a private network 172.16.30.0/24 for computers *tux31* and *tux34*, connecting directly their interfaces `tux31@eth0` and `tux34@eth0`. Interface `tux31@eth0` is given IP address 172.16.30.1/24, and `tux34@eth0` is given 172.16.30.254/24. We test the connectivity of *tux31* and *tux34* with a simple `ping` command.

4.1.1 Questions

What are the ARP packets and what are they used for? ARP request packets are broadcast in a network by a host to retrieve the MAC address of the machine with a certain IPv4 address. The protocol specifies the packet is sent to every host machine in the network, and only the one with the requested IPv4 address responds with its MAC address. In other words, the ARP protocol serves to convert 32-bit logical IP addresses to 48-bit physical MAC addresses.

What are the MAC and IP addresses of ARP packets and why? The ARP request packet includes the IP and MAC addresses of its source host and the IP address of its target host, whose MAC address it wants to retrieve. The ARP reply includes all four addresses. For example, *tux31* sent an ARP request for 172.16.30.254 on network 172.16.30.0/24, which reached `tux34@eth0` (the only other host on the network). Its ARP reply was MAC address `00:21:5a:5a:7d:7a`.

What packets does the ping command generate? The `ping` commands generates ICMP `ECHO_REQUEST` packets, and expects ICMP `ECHO_REPLY` responses.

What are the MAC and IP addresses of the ping packets? The source IP is 172.16.30.1 (`tux31@eth0`) and the destination IP is 172.16.30.254 (`tux34@eth0`) for the `ECHO_REQUEST` packets, and the reverse of `ECHO_REPLY` packets. Since the packets are directly transmitted, the source and destination MAC addresses match the IP addresses.

How to determine if a receiving Ethernet frame is ARP, IP, ICMP? For Ethernet frames, check the EtherType header field on the MAC frame: `0x0800` for IPv4 and `0x0806` for ARP. Inside the IP packet, the protocol header field indicates the upper layer protocol: 1 for ICMP, 6 for TCP, 17 for UDP, etc.

00:0f:fe:8b:e4…	ff:ff:ff:ff:ff:ff	ARP	Who has 172.16.30.254? Tell 172.16.30.1
00:21:5a:5a:7d…	00:0f:fe:8b:e4:4d	ARP	172.16.30.254 is at 00:21:5a:5a:7d:74
172.16.30.1	172.16.30.254	ICMP	Echo (ping) request id=0x39cc, seq=1/256, ttl=64
172.16.30.254	172.16.30.1	ICMP	Echo (ping) reply id=0x39cc, seq=1/256, ttl=64
172.16.30.1	172.16.30.254	ICMP	Echo (ping) request id=0x39cc, seq=2/512, ttl=64
172.16.30.254	172.16.30.1	ICMP	Echo (ping) reply id=0x39cc, seq=2/512, ttl=64
172.16.30.1	172.16.30.254	ICMP	Echo (ping) request id=0x39cc, seq=3/768, ttl=64
172.16.30.254	172.16.30.1	ICMP	Echo (ping) reply id=0x39cc, seq=3/768, ttl=64
172.16.30.1	172.16.30.254	ICMP	Echo (ping) request id=0x39cc, seq=4/1024, ttl=64
172.16.30.254	172.16.30.1	ICMP	Echo (ping) reply id=0x39cc, seq=4/1024, ttl=64
172.16.30.1	172.16.30.254	ICMP	Echo (ping) request id=0x39cc, seq=5/1280, ttl=64
172.16.30.254	172.16.30.1	ICMP	Echo (ping) reply id=0x39cc, seq=5/1280, ttl=64

Figure 1: *tux31* pings *tux34*

How to determine the length of a receiving frame? For Ethernet frames there is no header field with the frame length — the whole frame must be read and the measured. For IPv4 packets, the header has two length fields: one for header length (4bits, offset 4 bits) and another for packet length (2 bytes, offset 16 bits).

What is the loopback interface and why is it important? 172.0.0.0/8 is a range of 2^{24} IPv4 addresses representing the host machine. Useful for testing or running client-server services in the host. It is generally not represented in the routing table. The address 172.0.0.1 is assigned the local loopback interface `lo`, but the whole range is private and may be used.

4.2 Exp 2. Implement two virtual LANs in a switch

Now we are going to create a second private network 172.16.31.0/24 for *tux32*, and connect our two networks to two virtual LANs in the switch: vlan 30 for network 172.16.30.0/24 and vlan 31 for network 172.16.31.0/24. Naturally, we connect `tux31@eth0` and `tux34@eth0` to vlan 30 and `tux32@eth0` to vlan 31. Notice there is no connection between the networks yet, so *tux32* cannot communicate with neither *tux31* or *tux34*.

Cable configuration:
`tux31@eth0`—`sw Fa0/1`
`tux34@eth0`—`sw Fa0/4`
`tux32@eth0`—`sw Fa0/2`

Commands:
tux32:
```
ifconfig eth0 up
ifconfig eth0 172.16.31.1/24
```
switch:
```
configure terminal
vlan 30
exit
vlan 31
exit
interface fastethernet 0/1
switchport mode access
switchport access vlan 30
exit
interface fastethernet 0/4
switchport mode access
switchport access vlan 30
exit
interface fastethernet 0/2
switchport mode access
switchport access vlan 31
end
```

4.2.1 Questions

How many broadcast domains are there? How can you conclude it from the logs? The two broadcast domains are the two networks 172.16.30.0/24 and 172.16.31.0/24 because they are not connected.

4.3 Exp 3. Configure a Router in Linux

In this configuration we enable a new interface `tux34@eth2` and connect it to network 172.16.31.0/24 through vlan 31. This way *tux34* connects the two networks, and enabling IP forwarding allows *tux31* and *tux32* to communicate.

4.3.1 Questions

What routes are there in the tuxes? What are their meaning? The routes listed by command `ip route` are as follows:

- *tux31*: `default via 172.16.30.254 dev eth0`
  Default gateway of *tux31*. If no other route matches the destination of an IP packet being sent, it is sent to 172.16.30.254 (through interface `eth0`).
- *tux31*: `172.16.30.0/24 dev eth0`
  Means *tux31* is directly connected to all hosts in the network 172.16.30.0/24 through network interface `eth0`.
- *tux34*: `172.16.30.0/24 dev eth0`
  Means *tux34* is directly connected to network 172.16.30.0/24 through interface `eth0`.

- *`tux34`*: `172.16.31.0/24 dev eth2`
  Means *`tux34`* is directly connected to network 172.16.31.0/24 through interface `eth2`.
- *`tux32`*: `172.16.30.0/24 via 172.16.31.253 dev eth0`
  Means that *`tux32`* is (indirectly) connected to the network 172.16.30.0/24 through gateway 172.16.31.253. So any IP packet whose destination is the network 172.16.30.0/24 will be sent to address 172.16.31.253 (who *`tux32`* expects to forward).
- *`tux32`*: `172.16.31.0/24 dev eth0`
  Means *`tux32`* is directly connected to network 172.16.31.0/24 through interface `eth0`.

What information does an entry of the forwarding table contain? The primary information are the *`Destination`* (host or networks) and the *`Gateway`*. Packets destined to a certain *`Destination`* are routed to the respective *`Gateway`*. For gateway entries, the host is not directly connected to the *`Destination`* but it always directly connected to the *`Gateway`*.

The table displayed by `route -n` shows further information. *`Metric`* scores a given route in terms of cost. It can contain any number of values that help a router or host determine the best route among multiple routes to a destination. A packet will generally be sent through the route with the lowest metric. The most basic metric is typically based on path length, hop count or delay. Some *`Flags`* are U for Up, meaning the route is up; G for Gateway, meaning the route is to a gateway; H for host, meaning the route's destination is a complete host address; D means the route was created by a redirect; and M means the route was modified by a redirect. *`Iface`* is the network interface used for the route, naturally.

What ARP messages, and associated MAC addresses, are observed and why? ARP Request/Reply pairs which are required for IP communication:

- *`tux31`* (172.16.30.1) asks MAC address of 172.16.30.254 (*`tux34`*)
- *`tux34`* (172.16.31.253) asks MAC address of 172.16.31.1 (*`tux32`*)

When the ARP table is clear, the link layer needs to request the MAC address of the destination IP with an ARP request.

The observed MAC addresses are then:

- `tux31@eth0` (172.16.30.1): `00:0f:fe:8b:e4:4d`
- `tux34@eth0` (172.16.30.254): `00:21:5a:5a:7d:74`
- `tux34@eth2` (172.16.31.253): `00:01:02:21:83:0e`
- `tux32@eth0` (172.16.31.1): `00:21:5a:61:30:63`

```
00:0f:fe:8b:e4:4d ff:ff:ff:ff:ff:ff ARP    Who has 172.16.30.254? Tell 172.16.30.1
00:21:5a:5a:7d:74 00:0f:fe:8b:e4:4d ARP    172.16.30.254 is at 00:21:5a:5a:7d:74
172.16.30.1       172.16.31.1       ICMP   Echo (ping) request  id=0x44cf, seq=1/256, ttl=64
172.16.31.1       172.16.30.1       ICMP   Echo (ping) reply    id=0x44cf, seq=1/256, ttl=63
fc:fb:fb:3a:fa:86 01:80:c2:00:00:00 STP    Conf. Root = 32768/30/fc:fb:fb:3a:fa:80  Cost = 0
172.16.30.1       172.16.31.1       ICMP   Echo (ping) request  id=0x44cf, seq=2/512, ttl=64
172.16.31.1       172.16.30.1       ICMP   Echo (ping) reply    id=0x44cf, seq=2/512, ttl=63
172.16.30.1       172.16.31.1       ICMP   Echo (ping) request  id=0x44cf, seq=3/768, ttl=64
172.16.31.1       172.16.30.1       ICMP   Echo (ping) reply    id=0x44cf, seq=3/768, ttl=63
```

```
00:01:02:21:83:0e ff:ff:ff:ff:ff:ff ARP    Who has 172.16.31.1? Tell 172.16.31.253
00:21:5a:61:30:63 00:01:02:21:83:0e ARP    172.16.31.1 is at 00:21:5a:61:30:63
172.16.30.1       172.16.31.1       ICMP   Echo (ping) request  id=0x44cf, seq=1/256, ttl=63
172.16.31.1       172.16.30.1       ICMP   Echo (ping) reply    id=0x44cf, seq=1/256, ttl=64
fc:fb:fb:3a:fa:87 01:80:c2:00:00:00 STP    Conf. Root = 32768/31/fc:fb:fb:3a:fa:80  Cost = 0
172.16.30.1       172.16.31.1       ICMP   Echo (ping) request  id=0x44cf, seq=2/512, ttl=63
172.16.31.1       172.16.30.1       ICMP   Echo (ping) reply    id=0x44cf, seq=2/512, ttl=64
172.16.30.1       172.16.31.1       ICMP   Echo (ping) request  id=0x44cf, seq=3/768, ttl=63
172.16.31.1       172.16.30.1       ICMP   Echo (ping) reply    id=0x44cf, seq=3/768, ttl=64
```

Figure 2: *tux31* pings *tux32*, seen from `tux34@eth0` and `tux34@eth2`

What ICMP packets are observed and why? When *tux31* pings *tux32*, ICMP `ECHO_REQUEST` packets are sent with source 172.16.30.1 and destination 172.16.31.1. These are routed from `tux31@eth0` through *tux34* to `tux32@eth0`, and back for the `ECHO_REPLY` packets. These packets can be observed in both of *tux34*'s interfaces.

What are the IP and MAC addresses associated to ICMP packets and why? Since ICMP packets are IPv4 packets – they live in the network layer – their IP addresses are fixed. When *tux31* pings *tux32*, the source IP is 172.16.30.1 and destination IP is 172.16.31.1. However, the MAC addresses reflect the hops the packet takes through the network. From *tux31* to *tux34*, the source MAC address is that of `tux31@eth0` and the destination MAC address is that of `tux34@eth0`. From *tux34* to *tux32*, the source MAC address is that of `tux34@eth2` and the destination MAC address is that of `tux32@eth0`. For the `ECHO_REPLY` packets it's the reverse, naturally.

4.4 Exp 4. Configure a Commercial Router and Implement NAT

To allow both networks to communicate with the Internet we connect one of the router's interfaces to network 172.16.31.0/24 and implement NAT, with allowance for *tux31* and *tux32* but not for *tux34*.

Cable configuration:
`tux31@eth0—sw Fa0/1`
`tux34@eth0—sw Fa0/4`
`tux34@eth2—sw Fa0/7`
`tux32@eth0—sw Fa0/2`
`rt Giga0/1—sw Fa0/9`
`rt Giga0/0—172.16.1.254`

Commands:
tux44:
`route add default gw 172.16.31.254`
tux42:
`route add default gw 172.16.31.254`
switch:
```
configure terminal
interface fastethernet 0/9
switchport mode access
switchport access vlan 31
end
```

4.4.1 Questions

How to configure a static route in a commercial router? The command's basic syntax is

```
ip route DestinationIP Mask GatewayIP
```

What are the paths followed by the packets in the experiments carried out and why?

- *tux31* pings 172.16.30.254 at *tux34*:
  172.16.30.1(*tux31*) → 172.16.30.254(*tux34*)
- *tux31* pings 172.16.31.253 at *tux34*:
  172.16.30.1(*tux31*) → 172.16.30.254(*tux34*)
- *tux31* pings 172.16.31.1 at *tux32*:
  172.16.30.1(*tux31*) → 172.16.30.254(*tux34*) → 172.16.31.1(*tux32*)
- *tux31* pings 172.16.31.254 at *router*:
  172.16.30.1(*tux31*) → 172.16.30.254(*tux34*) → 172.16.31.254(*rt*)
- *tux31* pings 172.16.1.39 at *router*:
  172.16.30.1(*tux31*) → 172.16.30.254(*tux34*) → 172.16.31.254(*rt*)
- *tux31* pings 172.16.1.254 (with NAT implemented in the router):
  172.16.30.1(*tux31*) → 172.16.30.254(*tux34*) → 172.16.31.254(*rt*) → 172.16.1.254(*lab*)
- When *tux32* pings *tux31* with redirects disabled and no gateway route to 172.16.30.0/24:
  172.16.31.1(*tux32*) → 172.16.31.254(*rt*) → 172.16.31.253(*tux34*) → 172.16.30.1(*tux31*)
  The ping reply skips the router — *tux34* forwards directly to *tux32* from *tux31*.
- When *tux32* pings *tux31* with redirects enabled but no gateway route to 172.16.30.0/24, the first ping is routed to 172.16.30.254(*rt*) — like the previous case — but further pings are routed directly to 172.16.30.253(*tux34*) — like the next case — because of the ICMP Redirect sent by the router.
- When *tux32* pings *tux31* with a gateway route to 172.16.30.0/24 through *tux34*: 172.16.31.1(*tux32*) → 172.16.31.253(*tux34*) → 172.16.30.1(*tux31*)

How to configure NAT in a commercial router? Specify which interface is private with `ip nat inside` and which interface is public with `ip nat outside`. Then create an access list of IP addresses which are allowed to pass through the NAT:

```
ip nat pool ovrld 172.16.1.39 172.16.1.39 prefix 24
ip nat inside source list 1 pool ovrld overload
access-list 1 permit 172.16.30.0 0.0.0.7
access-list 1 permit 172.16.31.0 0.0.0.7
```

Because we chose mask 0.0.0.7 for the access list for each network, *tux34* is not allowed past the NAT and its packets addressed to the Internet are rejected.

What does NAT do? NAT (*Network Address Translation*) translates IP addresses from a private network (172.16.30.0/24 and 172.16.31.0/24 in our case) to a public IP address (172.16.1.39), mapping each private IP address and port pair to a port on the public IP address. In other words, NAT implements a *bidirectional function* between pairs (`PrivateIP`, `PrivatePort`), where `PrivateIP` is an allowed IP address present in the access list, to pairs (`PublicIP`, `PublicPort`), where `PublicIP` in our case is always 172.16.1.39. This function's *mapping* is generated on demand and stored in the NAT *mapping table*. Packets arriving in the router from the private network have their IP and port translated according to this *mapping* and then routed to their destination; those arriving from the public network are translated back to their private IP and ports accordingly.

4.5 Exp 5. DNS

With Internet access and a proper DNS server configured, *`tux31`* and *`tux32`* can now resolve and ping domains on the Internet. *`tux34`* can't if we keep the access-list specified in the previous section, as 172.16.31.253 is not allowed past the router's NAT.

4.5.1 Questions

How to configure the DNS service at an host? Edit `/etc/resolv.conf` by hand or with `resolveconf`, providing a nameserver which implements the DNS protocol. In the lab we use domain netlab.fe.up.pt.

What packets are exchanged by DNS and what information is transported? The host sends DNS queries to the nameserver and receives DNS responses. Information transported in DNS messages include, among other things, the question — with the domain name and type of record requested — the answer — resolved IP, resource records of the domain name — and the authority.

4.6 Exp 6. TCP connections

4.6.1 Questions

How many TCP connections are opened by the FTP application? In what connection is transported the FTP control information? The FTP protocol uses two TCP connections: the control connection, where the client sends the server FTP commands and receives FTP replies (server port 21), and the data connection, opened after a `PASV` command, through which the server sends retrieved files to the client.

What are the phases of a TCP connection? The *connection establishment*; *connection established*; and *connection termination* phases. The initiation is a *3-way handshake*, and the termination is a *4-way handshake* with a timeout.

Figure 3: *tux32* pings *tux31*, seen from `tux32@eth0`

172.16.31.1	172.16.30.1	ICMP	Echo (ping) request	id=0x4b3b, seq=1/256, ttl=64
172.16.31.254	172.16.31.1	ICMP	Redirect	(Redirect for host)
172.16.31.1	172.16.30.1	ICMP	Echo (ping) request	id=0x4b3b, seq=1/256, ttl=63
172.16.30.1	172.16.31.1	ICMP	Echo (ping) reply	id=0x4b3b, seq=1/256, ttl=63
fc:fb:fb:3a:fa…	01:80:c2:00:00:00	STP	Conf. TC + Root = 32768/31/fc:fb:fb:3a:fa:80 Cost	
172.16.31.1	172.16.30.1	ICMP	Echo (ping) request	id=0x4b3b, seq=2/512, ttl=64
172.16.31.254	172.16.31.1	ICMP	Redirect	(Redirect for host)
172.16.30.1	172.16.31.1	ICMP	Echo (ping) reply	id=0x4b3b, seq=2/512, ttl=63
172.16.31.1	172.16.30.1	ICMP	Echo (ping) request	id=0x4b3b, seq=3/768, ttl=64
172.16.30.1	172.16.31.1	ICMP	Echo (ping) reply	id=0x4b3b, seq=3/768, ttl=63
fc:fb:fb:3a:fa…	01:00:0c:cc:cc:cc	CDP	Device ID: tux-sw3 Port ID: FastEthernet0/2	
fc:fb:fb:3a:fa…	01:80:c2:00:00:00	STP	Conf. Root = 32768/31/fc:fb:fb:3a:fa:80 Cost = 0	
172.16.31.1	172.16.30.1	ICMP	Echo (ping) request	id=0x4b3b, seq=4/1024, ttl=64
172.16.31.254	172.16.31.1	ICMP	Redirect	(Redirect for host)
172.16.30.1	172.16.31.1	ICMP	Echo (ping) reply	id=0x4b3b, seq=4/1024, ttl=63
172.16.31.1	172.16.30.1	ICMP	Echo (ping) request	id=0x4b3b, seq=5/1280, ttl=64
172.16.31.254	172.16.31.1	ICMP	Redirect	(Redirect for host)
172.16.30.1	172.16.31.1	ICMP	Echo (ping) reply	id=0x4b3b, seq=5/1280, ttl=63

(a) With ICMP redirects disabled and no gateway route

172.16.31.1	172.16.30.1	ICMP	Echo (ping) request	id=0x4bae, seq=1/256, ttl=64
172.16.31.254	172.16.31.1	ICMP	Redirect	(Redirect for host)
172.16.30.1	172.16.31.1	ICMP	Echo (ping) reply	id=0x4bae, seq=1/256, ttl=63
172.16.31.1	172.16.30.1	ICMP	Echo (ping) request	id=0x4bae, seq=2/512, ttl=64
172.16.30.1	172.16.31.1	ICMP	Echo (ping) reply	id=0x4bae, seq=2/512, ttl=63
fc:fb:fb:3a:fa:84	01:80:c2:00:00:00	STP	Conf. Root = 32768/31/fc:fb:fb:3a:fa:80 Cost = 0	
172.16.31.1	172.16.30.1	ICMP	Echo (ping) request	id=0x4bae, seq=3/768, ttl=64
172.16.30.1	172.16.31.1	ICMP	Echo (ping) reply	id=0x4bae, seq=3/768, ttl=63
172.16.31.1	172.16.30.1	ICMP	Echo (ping) request	id=0x4bae, seq=4/1024, ttl=64
172.16.30.1	172.16.31.1	ICMP	Echo (ping) reply	id=0x4bae, seq=4/1024, ttl=63
fc:fb:fb:3a:fa:84	01:80:c2:00:00:00	STP	Conf. Root = 32768/31/fc:fb:fb:3a:fa:80 Cost = 0	
fc:fb:fb:3a:fa:84	fc:fb:fb:3a:fa:84	LOOP	Reply	
172.16.31.1	172.16.30.1	ICMP	Echo (ping) request	id=0x4bae, seq=5/1280, ttl=64
172.16.30.1	172.16.31.1	ICMP	Echo (ping) reply	id=0x4bae, seq=5/1280, ttl=63

(b) With ICMP redirects enabled

172.16.31.1	172.16.30.1	ICMP	Echo (ping) request	id=0x4c27, seq=1/256, ttl=64
172.16.30.1	172.16.31.1	ICMP	Echo (ping) reply	id=0x4c27, seq=1/256, ttl=63
172.16.31.1	172.16.30.1	ICMP	Echo (ping) request	id=0x4c27, seq=2/512, ttl=64
172.16.30.1	172.16.31.1	ICMP	Echo (ping) reply	id=0x4c27, seq=2/512, ttl=63
fc:fb:fb:3a:fa…	01:80:c2:00:00:00	STP	Conf. Root = 32768/31/fc:fb:fb:3a:fa:80 Cost = 0	
172.16.31.1	172.16.30.1	ICMP	Echo (ping) request	id=0x4c27, seq=3/768, ttl=64
172.16.30.1	172.16.31.1	ICMP	Echo (ping) reply	id=0x4c27, seq=3/768, ttl=63
172.16.31.1	172.16.30.1	ICMP	Echo (ping) request	id=0x4c27, seq=4/1024, ttl=64
172.16.30.1	172.16.31.1	ICMP	Echo (ping) reply	id=0x4c27, seq=4/1024, ttl=63
fc:fb:fb:3a:fa…	01:80:c2:00:00:00	STP	Conf. Root = 32768/31/fc:fb:fb:3a:fa:80 Cost = 0	
172.16.31.1	172.16.30.1	ICMP	Echo (ping) request	id=0x4c27, seq=5/1280, ttl=64
172.16.30.1	172.16.31.1	ICMP	Echo (ping) reply	id=0x4c27, seq=5/1280, ttl=63

(c) With gateway through *tux34*

172.16.30.1	172.16.1.1	DNS	Standard query 0x5deb A fe.up.pt
172.16.1.1	172.16.30.1	DNS	Standard query response 0x5deb A fe.up.pt A 10.227.240.205

Figure 4: DNS lookup by *tux31*

172.16.30.1	172.16.1.1	DNS	Standard query 0x711c A ftp.up.pt
172.16.1.1	172.16.30.1	DNS	Standard query response 0x711c A ftp.up.pt CNAME mirrors.up.pt A 193.137.29.15
172.16.30.1	193.137.29.15	TCP	53311 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=7335102 TSecr
193.137.29.15	172.16.30.1	TCP	21 → 53311 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1380 SACK_PERM=1 TSval=32
172.16.30.1	193.137.29.15	TCP	53311 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=7335103 TSecr=328556104
193.137.29.15	172.16.30.1	FTP	Response: 220-Welcome to the University of Porto's mirror archive (mirrors.up.p
193.137.29.15	172.16.30.1	FTP	Response: 220--

(a) TCP connection establishment

172.16.30.1	193.137.29.15	FTP	Request: QUIT
172.16.30.1	193.137.29.15	TCP	53311 → 21 [FIN, ACK] Seq=72 Ack=599 Win=29312 Len=0 TSval=7335171 TSecr=328
193.137.29.15	172.16.30.1	TCP	58731 → 42399 [ACK] Seq=1186 Ack=2 Win=29056 Len=0 TSval=328556172 TSecr=733
193.137.29.15	172.16.30.1	FTP	Response: 221 Goodbye.
172.16.30.1	193.137.29.15	TCP	53311 → 21 [RST] Seq=73 Win=0 Len=0
193.137.29.15	172.16.30.1	TCP	21 → 53311 [FIN, ACK] Seq=613 Ack=73 Win=29056 Len=0 TSval=328556173 TSecr=7
172.16.30.1	193.137.29.15	TCP	53311 → 21 [RST] Seq=73 Win=0 Len=0

(b) TCP connection termination

Figure 5: TCP connection phases

5 Conclusion