Faculdade de Engenharia da Universidade do Porto

LAB 2 - NETWORK

Students and Authors:

Afonso Domingues up202207313@fe.up.pt

Duarte Assunção up202208319@fe.up.pt

Index

Summary

In this laboratory work, we explored several aspects of network communication, focusing on configuring network interfaces, routing, bridging, DNS resolution, NAT, and FTP operations. The experiments demonstrated essential concepts such as ARP resolution, ICMP communication, routing across subnets, and TCP reliability mechanisms.

Introduction

This report presents the results and analysis of configuring a computer network, conducted as part of the coursework computer networks for the Faculdade de Engenharia da Universidade do Porto. The main objective of this laboratory is to explore and implement key concepts of networking, including device communication, IP configuration, routing, bridging, DNS services, and FTP file transfers.

Through a series of six experiments, we configure network architectures using switches, bridges, and routers, enabling communication across different subnets and domains.

Part 1

Architecture of the Download Application

The program receives an FTP URL in the format:

ftp://[<user>:<password>@]<host>/<url-path>.

The program first parses the URL using the parseURL function, which extracts key parameters such as the user, password, host, IP address, path, and filename. While the user, password, and host are stored as strings, the filename's position within the path is stored instead of the filename itself.

Once the URL is parsed, the program opens a socket to the FTP server using the *openSocket* function, based on the provided IP address and port. This socket is used to send commands and receive replies. After successfully establishing the connection, the program reads the server's initial response with the *readAnswer* function. The response is stored in an array and checked for the last line, indicated by a space following the answer code. This code is returned and checked against the expected one.

The program then sends the **USER** and **PASS** commands to the FTP server for authentication. If these are successful, it sends the **PASV** command to switch to passive mode. A new socket is opened for the data transfer, based on the server's reply, and the **RETR** command is issued to request the file download. The file is then saved locally with the same name.

After a successful transfer, the data socket is closed, and the connection is gracefully terminated with the QUIT command, closing the main socket.

Below is the Wireshark log of a successful file transfer:

No.	Time	Source	Destination	Protocol	Lengtl	Info
18	26.415068911	172.16.1.10	172.16.60.1	FTP	116	Response: 220 ProFTPD Server (Debian) [::ffff:172.16.1.10]
20	26.415202517	172.16.60.1	172.16.1.10	FTP	77	Request: user rcom
22	26.416227855	172.16.1.10	172.16.60.1	FTP	98	Response: 331 Password required for rcom
23	26.416295740	172.16.60.1	172.16.1.10	FTP	77	Request: pass rcom
25	26.590890537	172.16.1.10	172.16.60.1	FTP	112	Response: 230-Welcome, archive user rcom@172.16.1.61 !
26	26.590903527	172.16.1.10	172.16.60.1	FTP	115	Response:
27	26.591050612	172.16.1.10	172.16.60.1	FTP	233	Response:
29	26.591305044	172.16.60.1	172.16.1.10	FTP	72	Request: pasv
31	26.592186857	172.16.1.10	172.16.60.1	FTP	115	Response: 227 Entering Passive Mode (172,16,1,10,163,41).
35	26.592632164	172.16.60.1	172.16.1.10	FTP	87	Request: retr files/crab.mp4
36	26.593568314	172.16.1.10	172.16.60.1	FTP	142	Response: 150 Opening ASCII mode data connection for files/crab.mp4 (29803194 bytes)
31498	29.149098496	172.16.1.10	172.16.60.1	FTP	89	Response: 226 Transfer complete
31500	29.149171200	172.16.60.1	172.16.1.10	FTP	72	Request: quit
31501	29.149668678	172.16.1.10	172.16.60.1	FTP	80	Response: 221 Goodbye.

Fig.1 - FTP protocol for a successful transfer

Part 2

Experience 1

Network Architecture

Two devices, TUXY3 and TUXY4, were connected via a switch on the 172.16.Y0.0/24 subnet. TUXY3 used IP 172.16.Y0.1, and TUXY4 used IP 172.16.Y0.254. This is illustrated in the following figure:

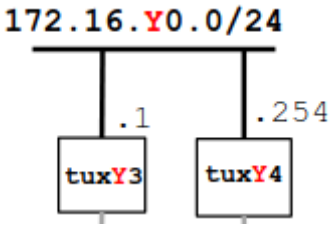


Fig. 2 - Exp1 Network Architecture

Experiment Objectives

Connect two devices, TUXY3 and TUXY4, through a switch and test their connectivity by configuring their network interfaces, verifying communication using the ping command, and analyzing the network behavior.

Main Configuration Commands

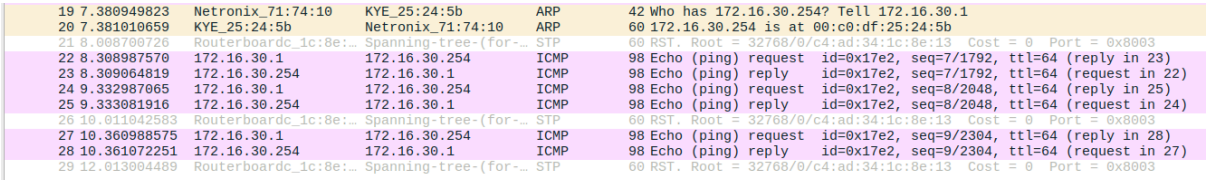
To configure eth1 interface of TUXY3 and eth1 interface of TUXY4 using ifconfig. We used the following commands in TUXY3:

```
Unset
ifconfig eth1 up
ifconfig eth1 172.16.Y0.1
```

And the following commands in TUXY4:

```
Unset
ifconfig eth1 up;
ifconfig eth1 172.16.Y0.254
```

Relevant Logs



19	7.380949823	Netronix_71:74:10	KYE_25:24:5b	ARP	42	Who has 172.16.30.254? Tell 172.16.30.1
20	7.381010659	KYE_25:24:5b	Netronix_71:74:10	ARP	60	172.16.30.254 is at 00:c0:df:25:24:5b
21	8.008700726	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:13 Cost = 0 Port = 0x8003
22	8.308987570	172.16.30.1	172.16.30.254	ICMP	98	Echo (ping) request id=0x17e2, seq=7/1792, ttl=64 (reply in 23)
23	8.309064819	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x17e2, seq=7/1792, ttl=64 (request in 22)
24	9.332987065	172.16.30.1	172.16.30.254	ICMP	98	Echo (ping) request id=0x17e2, seq=8/2048, ttl=64 (reply in 25)
25	9.333081916	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x17e2, seq=8/2048, ttl=64 (request in 24)
26	10.011042583	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:13 Cost = 0 Port = 0x8003
27	10.360988575	172.16.30.1	172.16.30.254	ICMP	98	Echo (ping) request id=0x17e2, seq=9/2304, ttl=64 (reply in 28)
28	10.361072251	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x17e2, seq=9/2304, ttl=64 (request in 27)
29	12.013004489	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:13 Cost = 0 Port = 0x8003

Fig.3 - Log from TUXY3

When TUXY3 pings TUXY4 using "**ping 172.16.30.254**" it first resolves TUXY4's MAC address via the ARP protocol. TUXY3 sends an ARP request, and TUXY4 responds with its MAC address (00:c0:df:25:24:5b). Once resolved, ICMP Echo Requests and Replies are exchanged, confirming successful network connectivity.

Experience 2

Network Architecture

Three devices (TUXY2, TUXY3, and TUXY4) were connected to a switch with two bridges: **bridgeY0** (for TUXY3 and TUXY4) and **bridgeY1** (for TUXY2). This setup isolated traffic between the bridges while maintaining connectivity within each. This is illustrated in the following figures.

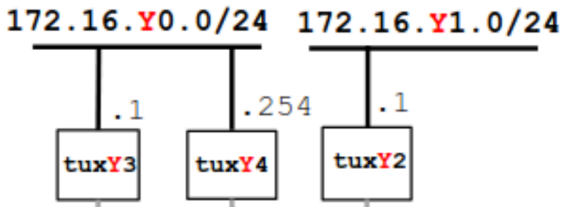


Fig. 4 - Connections

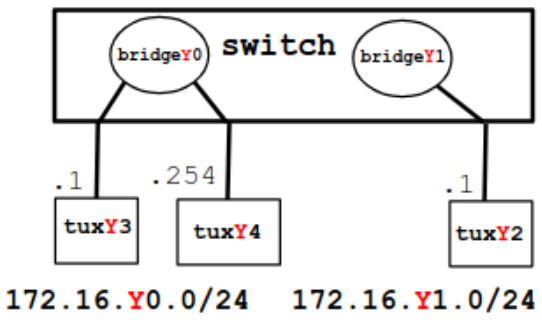


Fig. 5 - Bridges

Experiment Objectives

Configure bridges on a switch and analyze broadcast traffic's impact on devices in different bridges. This includes setting up TUXY2, verifying its IP and MAC addresses, testing traffic between TUXY3, TUXY4, and TUXY2, and analyzing captured packets.

Main Configuration Commands

Firstly, we started by configuring the eth1 of TUXY2 using the commands:

```
Unset

ifconfig eth1 up
ifconfig eth1 172.16.Y1.1/24
```

Then using the GKTerm to access the switch configuration we used the following commands to create the quested bridges:

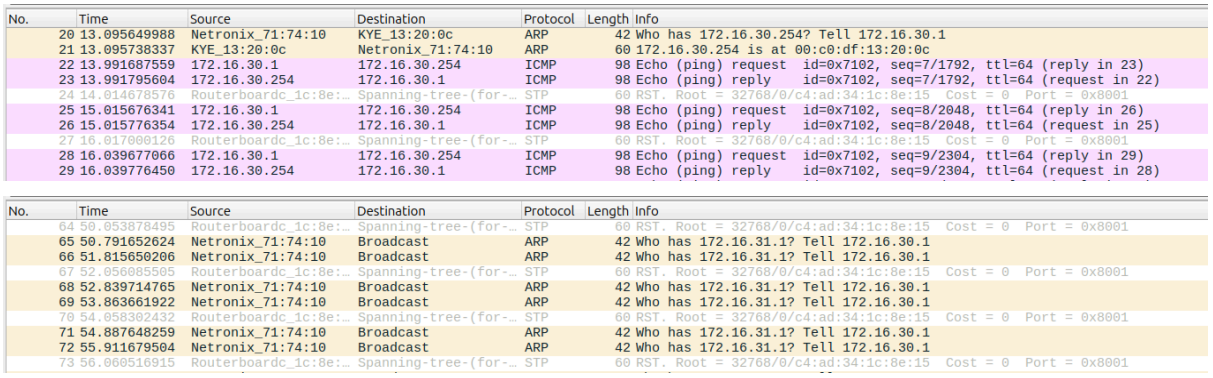
```
Unset

/interface bridge add name=bridge30
/interface bridge add name=bridge31
```

Finally in the last step of the confirmation we used the below commands to remove the ports where tuxY3, tuxY4 and tuxY2 are connected to the default bridge, in our case (ether3, ether4 and ether2) respectively and consequently add this ports to their respective bridge, in this case ether3 and ether4 to bridge 30 and ether2 to bridge31:

```
Unset
/interface bridge port remove [find interface=ether3]
/interface bridge port remove [find interface=ether4]
/interface bridge port remove [find interface=ether2]
/interface bridge port add bridge=bridge30 interface=ether3
/interface bridge port add bridge=bridge30 interface=ether4
/interface bridge port add bridge=bridge31 interface=ether2
```

Relevant Logs



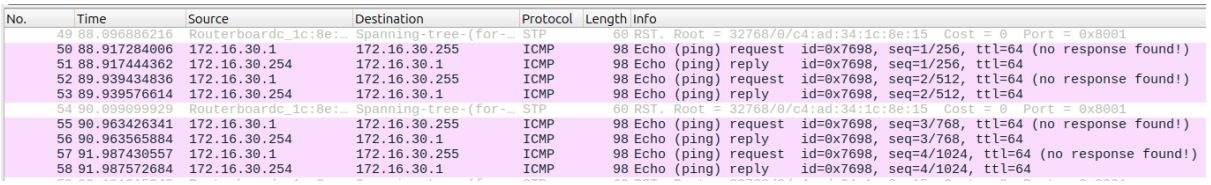
No.	Time	Source	Destination	Protocol	Length	Info
20	13.095649988	Netronix_71:74:10	KYE_13:20:0c	ARP	42	Who has 172.16.30.254? Tell 172.16.30.1
21	13.095738337	KYE_13:20:0c	Netronix_71:74:10	ARP	60	172.16.30.254 is at 00:c0:df:13:20:0c
22	13.991687559	172.16.30.1	172.16.30.254	ICMP	98	Echo (ping) request id=0x7102, seq=7/1792, ttl=64 (reply in 23)
23	13.991795604	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x7102, seq=7/1792, ttl=64 (request in 22)
24	14.014678576	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:15 Cost = 0 Port = 0x8001
25	15.015676341	172.16.30.1	172.16.30.254	ICMP	98	Echo (ping) request id=0x7102, seq=8/2048, ttl=64 (reply in 26)
26	15.015776354	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x7102, seq=8/2048, ttl=64 (request in 25)
27	16.017000126	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:15 Cost = 0 Port = 0x8001
28	16.039677066	172.16.30.1	172.16.30.254	ICMP	98	Echo (ping) request id=0x7102, seq=9/2304, ttl=64 (reply in 29)
29	16.039776450	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x7102, seq=9/2304, ttl=64 (request in 28)

No.	Time	Source	Destination	Protocol	Length	Info
64	50.053878495	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:15 Cost = 0 Port = 0x8001
65	50.791652624	Netronix_71:74:10	Broadcast	ARP	42	Who has 172.16.31.1? Tell 172.16.30.1
66	51.815650206	Netronix_71:74:10	Broadcast	ARP	42	Who has 172.16.31.1? Tell 172.16.30.1
67	52.056085505	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:15 Cost = 0 Port = 0x8001
68	52.839714765	Netronix_71:74:10	Broadcast	ARP	42	Who has 172.16.31.1? Tell 172.16.30.1
69	53.863661922	Netronix_71:74:10	Broadcast	ARP	42	Who has 172.16.31.1? Tell 172.16.30.1
70	54.058302432	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:15 Cost = 0 Port = 0x8001
71	54.887648259	Netronix_71:74:10	Broadcast	ARP	42	Who has 172.16.31.1? Tell 172.16.30.1
72	55.911679504	Netronix_71:74:10	Broadcast	ARP	42	Who has 172.16.31.1? Tell 172.16.30.1
73	56.060516915	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:15 Cost = 0 Port = 0x8001

Fig.6 and Fig.7 - Logs from TUXY3 and TUXY2

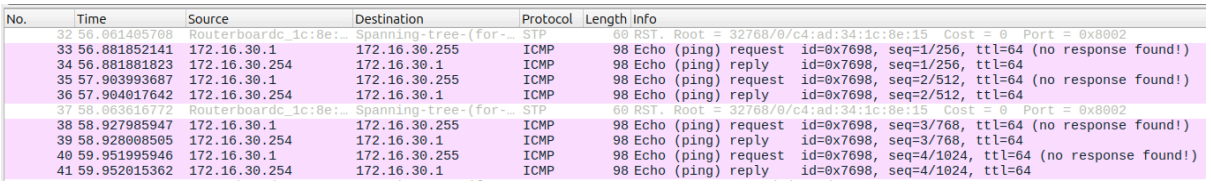
The logs show that pinging TUXY4 from TUXY3 works as expected since both are on the same bridge and subnet (172.16.30.0/24). However, pinging TUXY2 fails because it is on a different subnet (172.16.31.0/24) and bridge, leading TUXY3 to send unresolved ARP requests.

- Fig.8 - TUXY3 log:



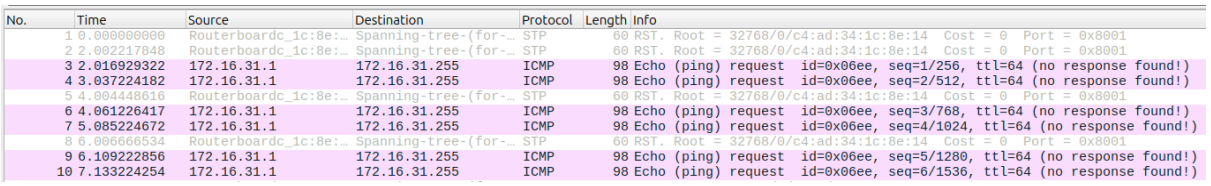
No.	Time	Source	Destination	Protocol	Length	Info
49	88.096886216	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:15 Cost = 0 Port = 0x8001
50	88.917284006	172.16.30.1	172.16.30.255	ICMP	98	Echo (ping) request id=0x7698, seq=1/256, ttl=64 (no response found!)
51	88.917444362	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x7698, seq=1/256, ttl=64
52	89.939434836	172.16.30.1	172.16.30.255	ICMP	98	Echo (ping) request id=0x7698, seq=2/512, ttl=64 (no response found!)
53	89.939576614	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x7698, seq=2/512, ttl=64
54	90.099099929	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:15 Cost = 0 Port = 0x8001
55	90.963426341	172.16.30.1	172.16.30.255	ICMP	98	Echo (ping) request id=0x7698, seq=3/768, ttl=64 (no response found!)
56	90.963565884	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x7698, seq=3/768, ttl=64
57	91.987430557	172.16.30.1	172.16.30.255	ICMP	98	Echo (ping) request id=0x7698, seq=4/1024, ttl=64 (no response found!)
58	91.987572684	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x7698, seq=4/1024, ttl=64

- Fig.9 - TUXY4 log:



No.	Time	Source	Destination	Protocol	Length	Info
32	56.061405708	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:15 Cost = 0 Port = 0x8002
33	56.881852141	172.16.30.1	172.16.30.255	ICMP	98	Echo (ping) request id=0x7698, seq=1/256, ttl=64 (no response found!)
34	56.881881823	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x7698, seq=1/256, ttl=64
35	57.903993687	172.16.30.1	172.16.30.255	ICMP	98	Echo (ping) request id=0x7698, seq=2/512, ttl=64 (no response found!)
36	57.904017642	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x7698, seq=2/512, ttl=64
37	58.063616772	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:15 Cost = 0 Port = 0x8002
38	58.927985947	172.16.30.1	172.16.30.255	ICMP	98	Echo (ping) request id=0x7698, seq=3/768, ttl=64 (no response found!)
39	58.928008505	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x7698, seq=3/768, ttl=64
40	59.951995946	172.16.30.1	172.16.30.255	ICMP	98	Echo (ping) request id=0x7698, seq=4/1024, ttl=64 (no response found!)
41	59.952015362	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x7698, seq=4/1024, ttl=64

The logs show that during the broadcast ping (“ping 172.16.30.255”), TUXY3 sends ICMP Echo Requests, and TUXY4 responds with Echo Replies because both are on the same bridge and subnet, confirming successful communication. TUXY2 does not respond, as it is on a different bridge and subnet, making its log irrelevant.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:14 Cost = 0 Port = 0x8001
2	2.002217848	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:14 Cost = 0 Port = 0x8001
3	2.016929322	172.16.31.1	172.16.31.255	ICMP	98	Echo (ping) request id=0x06ee, seq=1/256, ttl=64 (no response found!)
4	3.037224182	172.16.31.1	172.16.31.255	ICMP	98	Echo (ping) request id=0x06ee, seq=2/512, ttl=64 (no response found!)
5	4.004448616	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:14 Cost = 0 Port = 0x8001
6	4.061226417	172.16.31.1	172.16.31.255	ICMP	98	Echo (ping) request id=0x06ee, seq=3/768, ttl=64 (no response found!)
7	5.085224672	172.16.31.1	172.16.31.255	ICMP	98	Echo (ping) request id=0x06ee, seq=4/1024, ttl=64 (no response found!)
8	6.006666534	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:14 Cost = 0 Port = 0x8001
9	6.109222856	172.16.31.1	172.16.31.255	ICMP	98	Echo (ping) request id=0x06ee, seq=5/1280, ttl=64 (no response found!)
10	7.133224254	172.16.31.1	172.16.31.255	ICMP	98	Echo (ping) request id=0x06ee, seq=6/1536, ttl=64 (no response found!)

Fig.10 - Log from TUXY2

The log for the broadcast “ping 172.16.31.255” from TUXY2 shows multiple ICMP Echo Requests, but no replies are received because no other devices are on the same subnet (172.16.31.0/24). As TUXY2 is on a different bridge than the other TUX devices, their logs are irrelevant.

Experience 3

Network Architecture

TUXY4 will act as a router bridging two subnets. TUXY2 and TUXY3 are connected to separate networks, with TUXY4's eth1 connected to one subnet and eth2 (added to bridgeY1) connected to the other subnet. This is illustrated in the following figures.

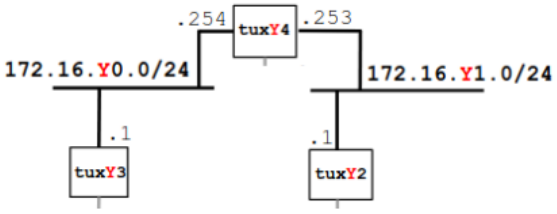


Fig. 11 - Connections

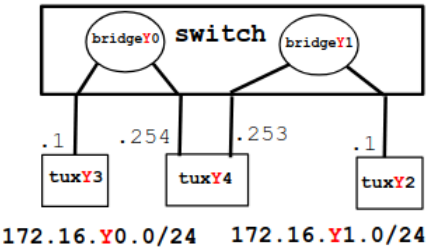


Fig. 12 - Bridges

Experiment Objectives

Configure TUXY4 as a router to enable communication between TUXY3 and TUXY2, which are on different subnets. This involves setting up TUXY4 with two interfaces, enabling IP forwarding, and configuring TUXY3 and TUXY2 to route traffic through TUXY4.

Main Configuration Commands

Firstly, we configured TUXY4's eth2 interface with the following commands, enabled IP forwarding, and disabled ICMP echo-ignore-broadcast:

```
Unset
ifconfig eth2 up
ifconfig eth2 172.16.Y1.253/24
sysctl net.ipv4.ip_forward=1
sysctl net.ipv4.icmp_echo_ignore_broadcasts=0
```

and added it to bridgeY1:

```
Unset
/interface bridge port remove [find interface=ether10]
/interface bridge port add bridge=bridgeY1 interface=ether10
```

Next, we reconfigured TUXY3 and TUXY2 to route traffic through TUXY4, using these commands:

```
Unset
route add -net 172.16.Y1.0/24 gw 172.16.Y0.254 # TUXY3 routing to TUXY2's
subnet via TUXY4
route add -net 172.16.Y0.0/24 gw 172.16.Y1.253 # TUXY2 routing to TUXY3's
subnet via TUXY4
```

Relevant Logs

No.	Time	Source	Destination	Protocol	Length	Info
13	22.764180808	172.16.60.1	172.16.60.254	ICMP	98	Echo (ping) request id=0x356f, seq=1/256, ttl=64 (reply in 14)
14	22.764348986	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x356f, seq=1/256, ttl=64 (request in 13)
15	23.790730230	172.16.60.1	172.16.60.254	ICMP	98	Echo (ping) request id=0x356f, seq=2/512, ttl=64 (reply in 16)
16	23.790900992	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x356f, seq=2/512, ttl=64 (request in 15)

No.	Time	Source	Destination	Protocol	Length	Info
39	36.750716714	172.16.60.1	172.16.61.253	ICMP	98	Echo (ping) request id=0x3576, seq=2/512, ttl=64 (reply in 40)
40	36.750886289	172.16.61.253	172.16.60.1	ICMP	98	Echo (ping) reply id=0x3576, seq=2/512, ttl=64 (request in 39)
41	37.774716443	172.16.60.1	172.16.61.253	ICMP	98	Echo (ping) request id=0x3576, seq=3/768, ttl=64 (reply in 42)
42	37.774892931	172.16.61.253	172.16.60.1	ICMP	98	Echo (ping) reply id=0x3576, seq=3/768, ttl=64 (request in 41)

No.	Time	Source	Destination	Protocol	Length	Info
52	50.316114988	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) request id=0x3580, seq=1/256, ttl=64 (reply in 53)
53	50.316435838	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) reply id=0x3580, seq=1/256, ttl=63 (request in 52)
54	51.342699889	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) request id=0x3580, seq=2/512, ttl=64 (reply in 55)
55	51.342964587	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) reply id=0x3580, seq=2/512, ttl=63 (request in 54)

Fig.13/14/15 - Logs from TUXY3

The logs from the ping command executed on TUXY3 show that it successfully sends ICMP requests to the IP addresses of the other TUX devices (172.16.Y0.254, 172.16.Y1.253, 172.16.Y1.1), and receives from each a corresponding ICMP reply. This confirms that TUXY3 can communicate with devices on both the same and different subnets, with TUXY4 acting as a router.

No.	Time	Source	Destination	Protocol	Length	Info
148	263.502649315	KYE_25:40:66	Broadcast	ARP	60	Who has 172.16.60.254? Tell 172.16.60.1
149	263.502682141	3Com_a1:35:69	KYE_25:40:66	ARP	42	172.16.60.254 is at 00:01:02:a1:35:69
150	263.502799198	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) request id=0x36a5, seq=1/256, ttl=64 (reply in 151)
151	263.503102107	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) reply id=0x36a5, seq=1/256, ttl=63 (request in 150)

No.	Time	Source	Destination	Protocol	Length	Info
289	251.010725792	KYE_04:20:8c	Broadcast	ARP	42	Who has 172.16.61.1? Tell 172.16.61.253
290	251.010866176	Netronix_b5:8c:8e	KYE_04:20:8c	ARP	60	172.16.61.1 is at 00:e0:7d:b5:8c:8e
291	251.010879376	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) request id=0x36a5, seq=1/256, ttl=63 (reply in 292)
292	251.011000205	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) reply id=0x36a5, seq=1/256, ttl=64 (request in 291)

Fig.16/17 - Logs from both interfaces (ether1 and ether2) of TUXY4

By analyzing the logs for the eth1 and eth2 interfaces on TUXY4, we see the process when TUXY3 pings TUXY2 (172.16.61.1). As the devices are on different subnets, TUXY4 acts as the router. TUXY3 first sends an ARP request for the gateway IP (172.16.60.254) on its subnet, then another ARP request via eth2 to resolve TUXY2's MAC address. Once TUXY2 responds, the connection is established, and ICMP packets are exchanged.

Experience 4

Network Architecture

RC will act as a router connecting the lab network and bridgeY1. The ether1 interface is connected to the lab network with IP 172.16.1.Y1/24, while ether2 is connected to bridgeY1 with IP 172.16.Y1.254. This setup enables RC to forward traffic between the two subnets.

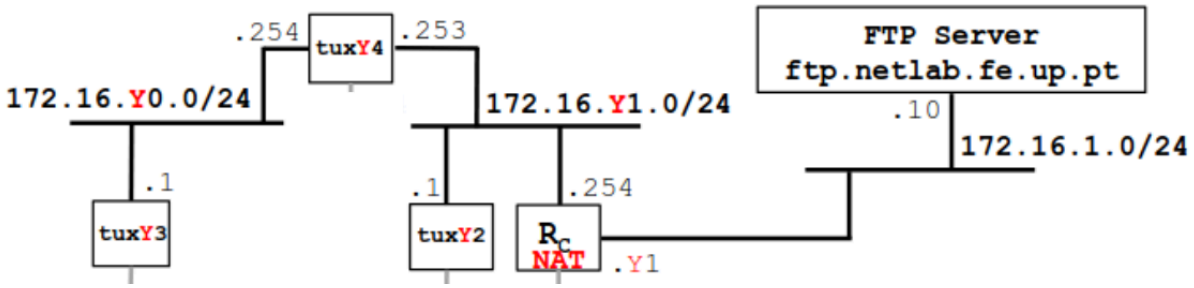


Fig. 18 - Connections

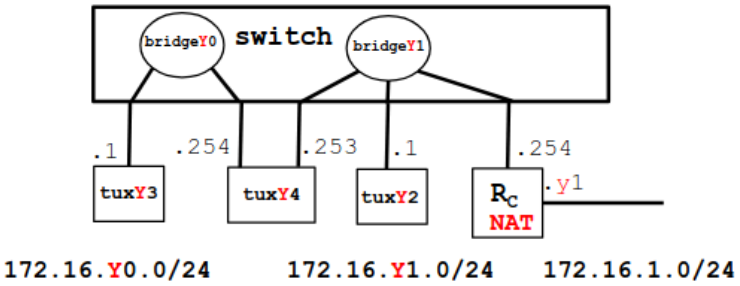


Fig. 19 - Bridges

Experiment Objectives

Set up and configure the RC router to enable communication between subnets. This involves connecting RC's ether1 to the lab network with NAT enabled, and ether2 to bridgeY1, configuring IP addresses and routes. Validate connectivity between devices in different subnets using ping commands and Wireshark, and analyze the impact of ICMP redirects and NAT functionality on traffic flow.

Main Configuration Commands

Firstly, we connected the router ether15 of the switch and added that port to bridge Y1:

```
Unset
/interface bridge port remove [find interface=ether15]
/interface bridge port add bridge=bridgeY1 interface=ether15
```

Secondly, we configured the IP addresses of the router through its console:

```
Unset
/ip address add address=172.16.1.Y1/24 interface=ether1
/ip address add address=172.16.Y1.254/24 interface=ether2
```

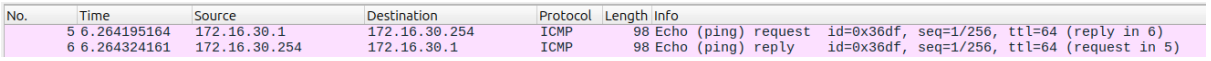
As TUXY2 already had a route to 172.16.Y0.0/24 configured and TUXY3 also already had a route to 172.16.Y1.0/24 in Experience 3, we only needed to configure the routes to 172.16.1.0/24 in the 3 TUX and the route to 172.16.Y0.0/24 in the Rc router:

```
Unset
# TUXY2 & TUXY4:
route add -net 172.16.1.0/24 gw 172.16.Y1.254
# TUXY3:
route add -net 172.16.1.0/24 gw 172.16.Y0.254
#RC
/ip route add dst-address=172.16.Y0.0/24 gateway=172.16.Y1.253
```

Relevant Logs

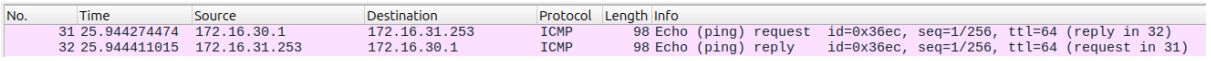
As we can see on the logs below, TUXY3 can reach all the network interfaces of TUXY2, TUXY4 and Rc:

- Fig. 20 - Ping ether1 of TUXY4



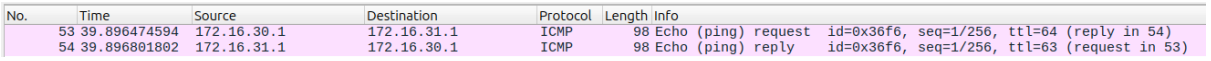
No.	Time	Source	Destination	Protocol	Length	Info
5	6.264195164	172.16.30.1	172.16.30.254	ICMP	98	Echo (ping) request id=0x36df, seq=1/256, ttl=64 (reply in 6)
6	6.264324161	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x36df, seq=1/256, ttl=64 (request in 5)

- Fig. 21 - Ping ether2 of TUXY4



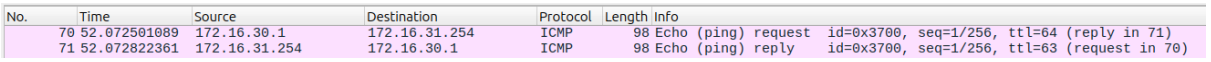
No.	Time	Source	Destination	Protocol	Length	Info
31	25.944274474	172.16.30.1	172.16.31.253	ICMP	98	Echo (ping) request id=0x36ec, seq=1/256, ttl=64 (reply in 32)
32	25.944411015	172.16.31.253	172.16.30.1	ICMP	98	Echo (ping) reply id=0x36ec, seq=1/256, ttl=64 (request in 31)

- Fig. 22 - Ping ether1 of TUXY2



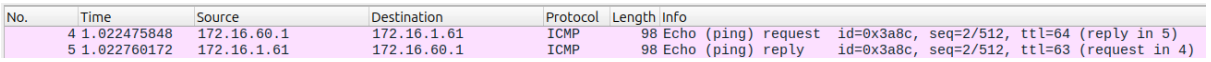
No.	Time	Source	Destination	Protocol	Length	Info
53	39.896474594	172.16.30.1	172.16.31.1	ICMP	98	Echo (ping) request id=0x36f6, seq=1/256, ttl=64 (reply in 54)
54	39.896801802	172.16.31.1	172.16.30.1	ICMP	98	Echo (ping) reply id=0x36f6, seq=1/256, ttl=63 (request in 53)

- Fig. 23 - Ping ether2 of RC



No.	Time	Source	Destination	Protocol	Length	Info
70	52.072501089	172.16.30.1	172.16.31.254	ICMP	98	Echo (ping) request id=0x3700, seq=1/256, ttl=64 (reply in 71)
71	52.072822361	172.16.31.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x3700, seq=1/256, ttl=63 (request in 70)

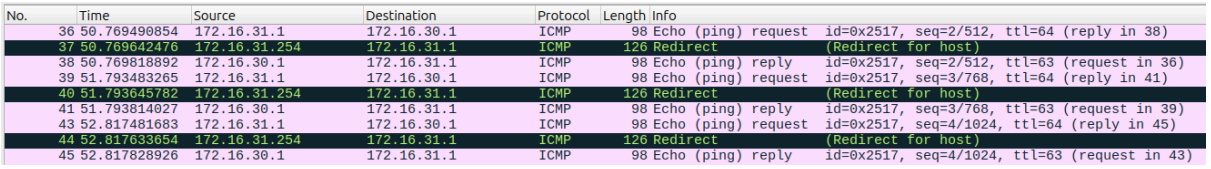
- Fig. 24 - Ping ether1 of RC



No.	Time	Source	Destination	Protocol	Length	Info
4	1.022475848	172.16.60.1	172.16.1.61	ICMP	98	Echo (ping) request id=0x3a8c, seq=2/512, ttl=64 (reply in 5)
5	1.022760172	172.16.1.61	172.16.60.1	ICMP	98	Echo (ping) reply id=0x3a8c, seq=2/512, ttl=63 (request in 4)

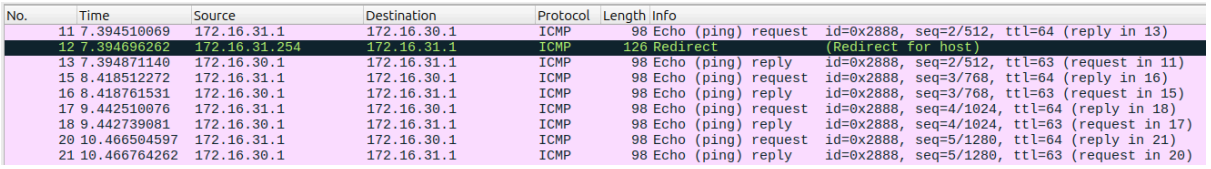
Below, we can see the logs of trying to reach TUXY3 from TUXY2:

- Fig. 25 - ICMP redirect acceptance disabled



No.	Time	Source	Destination	Protocol	Length	Info
36	50.769490854	172.16.31.1	172.16.30.1	ICMP	98	Echo (ping) request id=0x2517, seq=2/512, ttl=64 (reply in 38)
37	50.769642476	172.16.31.254	172.16.31.1	ICMP	126	Redirect (Redirect for host)
38	50.769818892	172.16.30.1	172.16.31.1	ICMP	98	Echo (ping) reply id=0x2517, seq=2/512, ttl=63 (request in 36)
39	51.793483265	172.16.31.1	172.16.30.1	ICMP	98	Echo (ping) request id=0x2517, seq=3/768, ttl=64 (reply in 41)
40	51.793645782	172.16.31.254	172.16.31.1	ICMP	126	Redirect (Redirect for host)
41	51.793814027	172.16.30.1	172.16.31.1	ICMP	98	Echo (ping) reply id=0x2517, seq=3/768, ttl=63 (request in 39)
43	52.817481683	172.16.31.1	172.16.30.1	ICMP	98	Echo (ping) request id=0x2517, seq=4/1024, ttl=64 (reply in 45)
44	52.817633654	172.16.31.254	172.16.31.1	ICMP	126	Redirect (Redirect for host)
45	52.817828926	172.16.30.1	172.16.31.1	ICMP	98	Echo (ping) reply id=0x2517, seq=4/1024, ttl=63 (request in 43)

- Fig. 26 - ICMP redirect acceptance enabled



No.	Time	Source	Destination	Protocol	Length	Info
11	7.394510069	172.16.31.1	172.16.30.1	ICMP	98	Echo (ping) request id=0x2888, seq=2/512, ttl=64 (reply in 13)
12	7.394696262	172.16.31.254	172.16.31.1	ICMP	126	Redirect (Redirect for host)
13	7.394871140	172.16.30.1	172.16.31.1	ICMP	98	Echo (ping) reply id=0x2888, seq=2/512, ttl=63 (request in 11)
15	8.418512272	172.16.31.1	172.16.30.1	ICMP	98	Echo (ping) request id=0x2888, seq=3/768, ttl=64 (reply in 16)
16	8.418761531	172.16.30.1	172.16.31.1	ICMP	98	Echo (ping) reply id=0x2888, seq=3/768, ttl=63 (request in 15)
17	9.442510076	172.16.31.1	172.16.30.1	ICMP	98	Echo (ping) request id=0x2888, seq=4/1024, ttl=64 (reply in 18)
18	9.442739081	172.16.30.1	172.16.31.1	ICMP	98	Echo (ping) reply id=0x2888, seq=4/1024, ttl=63 (request in 17)
20	10.466504597	172.16.31.1	172.16.30.1	ICMP	98	Echo (ping) request id=0x2888, seq=5/1280, ttl=64 (reply in 21)
21	10.466764262	172.16.30.1	172.16.31.1	ICMP	98	Echo (ping) reply id=0x2888, seq=5/1280, ttl=63 (request in 20)

When pinging **TUXY3** from **TUXY2**, packets initially use the default route, which is **172.16.31.254**, the router's IP address. The router then forwards these packets to the target IP. However, when ICMP redirect acceptance is enabled, the router sends an ICMP redirect packet to **TUXY2**, informing it of a more optimal route to reach **TUXY3** directly. Upon receiving this ICMP redirect, **TUXY2** updates its routing table to use the suggested route, bypassing the default gateway for subsequent packets. This behavior optimizes the communication path, reducing unnecessary hops.

In contrast, when ICMP redirect acceptance is disabled, **TUXY2** continues to use the default route for all packets, as the routing table is not updated. This results in a less efficient path for communication.

Experience 5

Network Architecture

The architecture in this experiment remains the same as in the previous one.

Experiment Objectives

The goal is to configure DNS services on TUXY3, TUXY4, and TUXY2 to resolve domain names via the DNS server at 10.227.20.3 (services.netlab.fe.up.pt). The experiment verifies that name resolution works correctly on all three hosts (e.g., using ping hostname or browsing the web).

Main Configuration Commands

No specific configuration commands were used; instead, DNS was set in /etc/resolv.conf with nameserver 10.227.20.3, and name resolution was verified with ping **ftp.netlab.fe.up.pt**.

Relevant Logs

No.	Time	Source	Destination	Protocol	Lengtl	Info
78	7.783739788	10.227.20.53	10.227.20.3	DNS	72	Standard query 0x89ef A santander.pt
79	7.783749706	10.227.20.53	10.227.20.3	DNS	72	Standard query 0x06fa AAAA santander.pt
80	7.784318845	10.227.20.3	10.227.20.53	DNS	104	Standard query response 0x89ef A santander.pt A 2.16.8.96 A 2.16.8.42
81	7.784347969	10.227.20.3	10.227.20.53	DNS	149	Standard query response 0x06fa AAAA santander.pt SOA eur2.akam.net
85	7.808377962	10.227.20.53	10.227.20.3	DNS	82	Standard query 0x9882 PTR 96.8.16.2.in-addr.arpa
86	7.808850720	10.227.20.3	10.227.20.53	DNS	143	Standard query response 0x9882 PTR 96.8.16.2.in-addr.arpa PTR a2-16-8-96.deploy.static.akamaitechno

Fig. 27 - Log from TUXY3

The logs show DNS query and response packets, where TUXY3 (10.227.20.43) queries the DNS server at **10.227.20.3** for ***santander.pt***, receiving the corresponding IPv4 and IPv6 addresses.

Experience 6

Network Architecture

In this experience, the network architecture remains the same as in Experience 4.

Experiment Objectives

The main objective of this experiment is to use our download application to download files from the FTP server located at ftp.netlab.fe.up.pt.

Main Configuration Commands

No specific configuration commands are required for this task.

Relevant Logs

No.	Time	Source	Destination	Protocol	Lengtl	Info
18	26.415068911	172.16.1.10	172.16.60.1	FTP	116	Response: 220 ProFTPD Server (Debian) [::ffff:172.16.1.10]
20	26.415202517	172.16.60.1	172.16.1.10	FTP	77	Request: user rcom
22	26.416227855	172.16.1.10	172.16.60.1	FTP	98	Response: 331 Password required for rcom
23	26.416295740	172.16.60.1	172.16.1.10	FTP	77	Request: pass rcom
25	26.590890537	172.16.1.10	172.16.60.1	FTP	112	Response: 230-Welcome, archive user rcom@172.16.1.61 !
26	26.590903527	172.16.1.10	172.16.60.1	FTP	115	Response:
27	26.591050612	172.16.1.10	172.16.60.1	FTP	233	Response:
29	26.591305044	172.16.60.1	172.16.1.10	FTP	72	Request: pasv
31	26.592186857	172.16.1.10	172.16.60.1	FTP	115	Response: 227 Entering Passive Mode (172,16,1,10,163,41).
35	26.592632164	172.16.60.1	172.16.1.10	FTP	87	Request: retr files/crab.mp4
36	26.593568314	172.16.1.10	172.16.60.1	FTP	142	Response: 150 Opening ASCII mode data connection for files/crab.mp4 (29803194 bytes)
31498	29.149098496	172.16.1.10	172.16.60.1	FTP	89	Response: 226 Transfer complete
31500	29.149171200	172.16.60.1	172.16.1.10	FTP	72	Request: quit
31501	29.149668678	172.16.1.10	172.16.60.1	FTP	80	Response: 221 Goodbye.

Fig. 28 - FTP protocol: download a file from ftp.netlab.fe.up.pt

The log illustrates an FTP session between **172.16.60.1** (TUXY3) and **172.16.1.10** (ftp.netlab.fe.up.pt server). The client authenticates, enters passive mode, requests the file **CRAB.mp4**, and successfully transfers it. The session ends with a QUIT command and a **221 Goodbye** response, indicating a clean and successful FTP connection and file transfer.

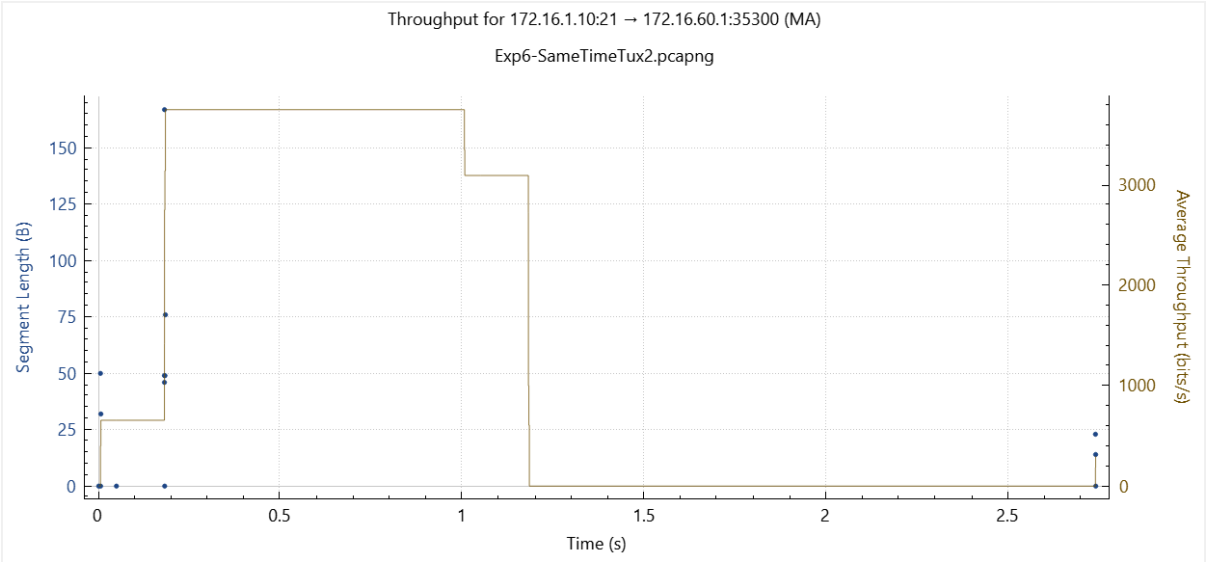


Fig. 29 - Throughput analyses of simultaneously download on TUXY3 and TUXY2

The second log shows a decrease in throughput from approximately 3570 (bits/s) to 3100 (bits/s) due to the simultaneous download of the **crab.mp4** file by **TUXY3** and **TUXY2** from **ftp.netlab.fe.up.pt**. This indicates that shared bandwidth impacts download performance when multiple clients access the same resource concurrently.

Conclusions

We set up and tested different network architectures, using tools like Wireshark to check connections and analyze how data moves through the network. We learned how ARP finds MAC addresses, how ICMP redirects improve routing, and how NAT changes packet transmission.

When testing an FTP application, we saw how TCP works, including starting connections, transferring data, and ending connections.

These activities showed us some common challenges, like managing broadcast domains, enabling IP forwarding, and making sure DNS and NAT work correctly. Overall, these tests were useful for understanding important networking concepts and how they are applied in real life.

References

- Ricardo, M., & Prior, R.. *The Network Layer*. Universidade do Porto.

- Ricardo, M., Teixeira, F. B., & Almeida, E. N. . *Computer Networks: Lab 2 - Computer Networks*. Universidade do Porto.

Annexes

Questions

Experience 1

1. **What are the ARP packets and what are they used for?**

ARP packets are small units of data used to find the physical (MAC) address of a device associated with a given IP address.There are two types: ARP Request (asking for a MAC address) and ARP Reply (responding with the MAC address).

2. **What are the MAC and IP addresses of ARP packets and why?**

The ARP Request package is sent to an IP address in order to retrieve the associated MAC address. That is why the Target MAC is unknown because it is what we are looking for.

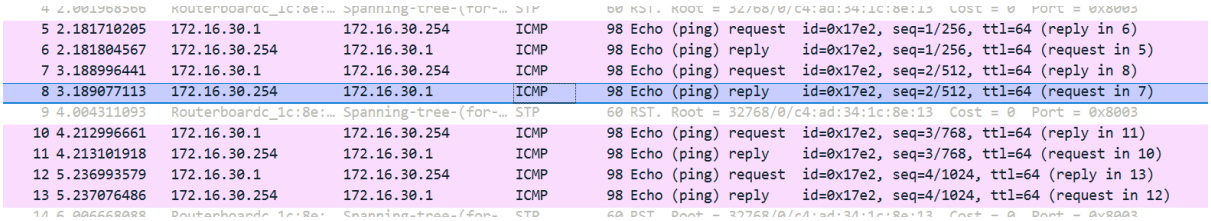
ARP Request Packet	Sender	Target
IP address	172.16.30.1/24	172.16.30.254/24
MAC address	00:08:54:71:74:10	Not known

Then, the ARP Reply packet is sent with the IP and MAC addresses of both the sender and the target. The sender's MAC address corresponds to the value we wanted to find.

ARP Reply Packet	Sender	Target
IP address	172.16.30.254/24	172.16.30.1/24
MAC address	00:c0:df:25:24:5b	00:08:54:71:74:10

3. **What packets does the ping command generate?**

The ping command generates ICMP (Internet Control Message Protocol) packets, including request and reply messages. These packets are used to send control and error messages in network communication. We used the ping command on tux33 to ping tux34 with the command ping 172.16.30.254, which corresponds to the IP address of tux34.



5	2.181710205	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0x17e2, seq=1/256, ttl=64 (reply in 6)
6	2.181804567	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x17e2, seq=1/256, ttl=64 (request in 5)
7	3.188996441	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0x17e2, seq=2/512, ttl=64 (reply in 8)
8	3.189077113	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x17e2, seq=2/512, ttl=64 (request in 7)
9	4.004311093	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8e:13 Cost = 0 Port = 0x8003
10	4.212996661	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0x17e2, seq=3/768, ttl=64 (reply in 11)
11	4.213101918	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x17e2, seq=3/768, ttl=64 (request in 10)
12	5.236993579	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0x17e2, seq=4/1024, ttl=64 (reply in 13)
13	5.237076486	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x17e2, seq=4/1024, ttl=64 (request in 12)

4. **What are the MAC and IP addresses of the ping packets?**

Ping Request Packet	Sender	Target
IP address	172.16.30.1/24	172.16.30.254/24
MAC address	00:08:54:71:74:10	00:c0:df:25:24:5b

Ping Request Packet	Sender	Target
IP address	172.16.30.254/24	172.16.30.1/24
MAC address	00:c0:df:25:24:5b	00:08:54:71:74:10

5. **How to determine if a receiving Ethernet frame is ARP, IP, ICMP?**

You can determine if a receiving Ethernet frame is ARP or IP by looking at the **Ethernet type** field in the Ethernet header:

ARP: 0x0806

IP: 0x0800

If it is an IP frame, you can determine if the protocol is ICMP by looking at the protocol field in the IP header. If its value is 1, then it is an ICMP frame.

6. **How to determine the length of a receiving frame?**

If the received frame is of type **IP**, you can determine its size by looking at the **Total Datagram Length field** in the IP header. This field specifies the length of the entire IP packet (including the IP header and the payload). To get the total frame size, you need to add the Ethernet header size (14 bytes) and the FCS (4 bytes).

For **ARP frames**, the total length corresponds to the fixed size of the **ARP payload** (28 bytes for standard ARP) plus the Ethernet header size (14 bytes). If the FCS (4 bytes) is included, you should add that as well.

7. **What is the loopback interface and why is it important?**

The **loopback interface** is a virtual network interface that allows a computer to communicate with itself. It is important because it enables internal communication between applications and services on the same machine, without using external network resources.

Experience 2

1. **How to configure bridgeY0?**

   1.Create the bridge

   First, you need to create the new bridge (bridgeY0) by running the following command: **/interface bridge add name=bridge30**

   2.Remove Ports from the Default Bridge:

   Next, you must remove the ports where tuxY3 and tuxY4 are connected from the default bridge. Using the following commands:

   /interface bridge port remove [find interface= port_number where tuxY3 is connected to the switch]

   /interface bridge port remove [find interface=port_number where tuxY4 is connected]

   3.Add Ports to the New Bridge:

   Finally, add the corresponding ports to bridgeY0. For example, if TUX33 is connected to port 3 and TUX34 is connected to port 4, use the following commands:

   /interface bridge port add bridge=bridgeY0 interface=ether3

   /interface bridge port add bridge=bridgeY0 interface=ether4

2. **How many broadcast domains are there? How can you conclude it from the logs?**

   There are two broadcast domains - one for each bridge created - whose ip addresses are 171.16.30.255 171.16.31.255.

   We can verify this, because, as it is seen in the logs, each device only receives the packages broadcasted from the same domain. When TUX33 broadcasts, only itself and TUX34 receive the packages. In the same way, when TUX32 broadcasts, only itself receives the packages.

Experience 3

1. **What routes are there in the tuxes? What are their meaning?**

   In TUXY3, there is a route to the 172.16.Y1.0/24 subnet with the gateway 172.16.Y0.254, which is TUXY4's network interface for its subnet. This means that when TUXY3 wants to communicate with a device in the 172.16.Y1.0/24 subnet, it will send the traffic to TUXY4, which acts as the router for that subnet.

Similarly, in TUXY2, there is a route to the 172.16.Y0.0/24 subnet with the gateway 172.16.Y1.253, which is also TUXY4's network interface for its subnet. This means that when TUXY2 wants to communicate with a device in the 172.16.Y0.0/24 subnet, it will send the traffic to TUXY4, which routes it appropriately.

2. **What information does an entry of the forwarding table contain?**

   Each entry in the forwarding table represents a route configured on the device. It typically contains information such as the destination network or subnet address, the subnet mask, and the gateway (next-hop address) to reach that destination. Additionally, it may include the network interface to use for forwarding traffic and other routing metrics, such as route cost or priority.

3. **What ARP messages, and associated MAC addresses, are observed and why?**

In Experiment 3, we observed both ARP Request and ARP Reply packets during the capture at TUXY3. These ARP messages were not needed initially, as the MAC addresses were already cached. However, once the ARP cache expired, new ARP messages were sent to resolve the IP addresses to MAC addresses again.

Specifically, TUXY3 sent ARP Requests to discover the MAC address corresponding to the pinged IP address (e.g., 172.16.Y0.254), and the pinged device responded with ARP Replies containing its MAC address. Similarly, the pinged device also sent ARP Requests to resolve the MAC address of TUXY3 to complete the communication.

ARP Request Packet	Sender	Target
IP address	172.16.60.1	172.16.60.254
MAC address	00:c0:df:25:40:66	Not known

ARP Reply Packet	Sender	Target
IP address	172.16.60.254	172.16.60.1
MAC address	**00:01:02:a1:35:69**	00:c0:df:25:40:66

ARP Request Packet	Sender	Target
IP address	172.16.60.254	172.16.60.1
MAC address	00:01:02:a1:35:69	Not known

ARP Reply Packet	Sender	Target
IP address	172.16.60.1	172.16.60.254
MAC address	**00:c0:df:25:40:66**	00:01:02:a1:35:69

4. **What ICMP packets are observed and why?**

We observed ICMP Echo Request and Echo Reply packets generated during the ping tests to verify connectivity between network interfaces (172.16.Y0.254, 172.16.Y1.253, 172.16.Y1.1) and TUXY3. These packets are used to confirm proper communication and routing through TUXY4.

5. **What are the IP and MAC addresses associated to ICMP packets and why?**

In Experiment 3, ICMP packets were observed as ping requests and replies sent from TUXY3 to other network interfaces, such as 172.16.Y0.254, 172.16.Y1.253, and 172.16.Y1.1. The observed IP and MAC addresses associated with these ICMP packets were as follows:

- **IP Addresses**:
  - 172.16.60.1 (TUXY3)
  - 172.16.60.254 (ether1 of TUXY4)
  - 172.16.61.253 (ether2 of TUXY4)
  - 172.16.61.1 (TUXY2)
- **MAC Addresses**:
  - 00:c0:df:25:40:66 (TUXY3)
  - 00:01:02:a1:35:69 (for 172.16.60.254, 172.16.61.253, and 172.16.61.1)

When performing a capture on TUXY4 and pinging from TUXY3 to TUXY2, the same IP and MAC addresses were observed.

The repeated MAC address 00:01:02:a1:35:69 corresponds to TUXY4, which is configured as a router. This MAC address represents the interface of TUXY4 that is forwarding packets between the networks connected to bridge Y0 and bridge Y1. Since the devices in the different subnets require routing, the observed MAC address corresponds to TUXY4's interface acting as the next hop for the ICMP packets directed to different subnets.

In routed networks, when a packet is destined for a different subnet, it is first forwarded to the router. The router then rewrites the Ethernet frame and forwards it to the destination. Therefore, the observed MAC address in the captured packets corresponds to the router's interface on the segment where the traffic is being captured.

Experience 4

1. **How to configure a static route in a commercial router?**

The configuration of a static route can simply be done with a command in the router configuration terminal (GTKTerm in our case). Below is an example of the command for the destination address of 172.16.Y0.0/24 using the gateway address of 172.16.Y1.253.

```
Unset
/ip route add dst-address=172.16.Y0.0/24 gateway=172.16.Y1.253
```

2. **What are the paths followed by the packets, with and without ICMP redirect enabled, in the experiments carried out and why?**

Without ICMP redirect enabled, packets from TUXY2 are sent to the default route (172.16.31.254) RC and forwarded to TUXY3 each time.

When ICMP redirect is enabled, the router sends a redirect to TUXY2, informing it of a more direct route to TUXY3. After receiving the redirect, TUXY2 updates its routing table to use the new route, sending packets directly to TUXY3 and bypassing the default gateway. This reduces unnecessary hops and optimizes the communication path.

3. **How to configure NAT in a commercial router?**

In a commercial router, the NAT can be disabled with the following command on the router terminal:

```
Unset
/ip firewall nat disable 0
```

It can also be enabled again with the following command:

```
Unset
/ip firewall nat add chain=srcnat action=masquerade out-interface=ether1
```

4. **What does NAT do?**

NAT (Network Address Translation) allows multiple devices on a private network to share a single public IP address for accessing external networks like the internet. It translates private IP addresses to a public IP address when sending data out and vice versa when receiving responses, ensuring that the data reaches the correct device within the private network.

5. **What happens when tuxY3 pings the FTP server with the NAT disabled? Why?**

When NAT is disabled on the router, the private IP address of TUXY3 is used for the request. Since private IP addresses are not routable on the internet, the FTP server, which typically expects public IP addresses, will not be able to send a response back to TUXY3. As a result, the reply is lost because the server doesn't know how to route the response to the private IP of TUXY3.

Experience 5

1. **How to configure the DNS service in a host?**

To configure DNS on a host, edit the /etc/resolv.conf file to include the DNS server's IP address, using the line nameserver <DNS_IP>. This points the host to the specified DNS server for name resolution.

2. **What packets are exchanged by DNS and what information is transported**

DNS works by exchanging two types of packets:

1. **DNS Query (Request):** This packet is sent by the host to the configured DNS server to request the IP address for a domain name. The query includes the domain name the host wants to resolve (e.g.,santander.pt) and is directed to the specified DNS server’s IP address.
2. **DNS Response (Reply):** This packet is sent by the DNS server back to the host, providing the IP address corresponding to the domain name in the query. For example, it might reply with 2.16.8.42 as the IP address for santander.pt.

Experience 6

1. **How many TCP connections are opened by your FTP application?**

Our FTP application opens two TCP connections: one for the FTP control connection, on port 21, and another for the data connection used to transfer the requested file (via passive mode).

2. **In what connection is transported the FTP control information?**

The FTP control information (e.g., commands such as user, pass, pasv, retr) is transported over the **FTP control connection** (first connection), which uses TCP port 21.

3. **What are the phases of a TCP connection?**

A TCP connection goes through three main phases:

- **Connection establishment**: Initiated by the exchange of SYN, SYN-ACK, and ACK packets (Three-Way Handshake).
- **Data transfer**: The connection is used to transmit data between the client and server, with continuous acknowledgment of received packets.
- **Connection termination**: This is done through the exchange of FIN and ACK packets from both sides to gracefully close the connection.

4. **How does the ARQ TCP mechanism work? What are the relevant TCP fields? What relevant information can be observed in the logs?**

The ARQ (Automatic Repeat reQuest) mechanism in TCP ensures reliable data delivery by using acknowledgments and retransmissions. Lost or out-of-order packets are retransmitted based on received acknowledgments (ACKs). TCP uses sequence numbers to track data and determine which packets need to be retransmitted.

Relevant TCP Fields:

- **Sequence Number**: Identifies the position of data.
- **Acknowledgment Number**: Indicates the next expected byte.
- **ACK Flag**: Marks acknowledgment packets.
- **Window Size**: Controls flow by showing available buffer space.

Based on the logs we could extract relevant information such as:

- **ACKs**: Logs show ACK packets confirming data receipt.
- **Sequence/Acknowledgment Numbers**: Track data flow..
- **No Retransmissions**: The logs show normal ACK behavior without retransmissions.

The logs confirm normal ARQ operation with successful acknowledgments and no packet loss.

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?**

TCP congestion control adjusts the rate of data transmission to avoid network congestion. It includes:

- **Slow Start**: The congestion window increases exponentially until a threshold is reached.
- **Congestion Avoidance**: After reaching the threshold, the window grows linearly.
- **Fast Retransmit/Recovery**: If packet loss is detected, the sender reduces the window and retransmits lost packets.

Relevant fields for congestion control include:

- **Congestion Window (cwnd)**: Limits the amount of data sent before receiving an acknowledgment.
- **Round Trip Time (RTT)**: Affects the rate at which the window size grows.

The throughput remained steady throughout the data transfer, indicating consistent data flow without congestion. This is in line with the normal operation of TCP congestion control, which manages the rate to avoid sudden drops or congestion, as seen in the absence of significant throughput variations during the session.

6. **Is the throughput of a TCP data connections disturbed by the appearance of a second TCP connection? How?**

Yes, the throughput of a TCP data connection can be affected by the appearance of a second TCP connection. This is because both connections share the same network resources, such as bandwidth. If a new connection is established, it can cause congestion, leading to packet loss, retransmissions, and reduced throughput in the existing connection. This leads to a decrease in the throughput of the original connection when the second connection starts transferring data.

Code of the Download Application

```C/C++
#include <stdio.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

#include <string.h>

#define SERVER_PORT 6000
#define SERVER_ADDR "192.168.28.96"

//Different Answers
#define FTP_PORT 21
#define WELCOME_AN 220
#define PASSWORD_ANS 331
#define LOGIN_ANS 230
#define PASV_ANS 227
#define OPEN_BINARY_ANS 150
#define OPEN_BINARY_ANS2 125
#define TRANSFER_COMP_ANS 226
#define QUIT_ANS 221

#define PARAM_SIZE 512

struct parameters {
       char user[PARAM_SIZE];
       char password[PARAM_SIZE];
       char ip[40]; // ipv4: 16; ipv6: 40.
       char host[PARAM_SIZE];
       char path[PARAM_SIZE];
       unsigned file;
};

int parseURL(char *input, struct parameters *param){

       if (strncmp(input, "ftp://", 6) != 0) {
       printf("Error: URL must start with 'ftp://'\n");
       return -1;
       }
       input += 6;
       memset(param,0,sizeof(struct parameters)); //initialize struct values
to empty string
```

```
        char *at_sign = strchr(input, '@'); //look for @ in the url
        if (at_sign){
        char *delimiter = strchr(input,':');
        strncpy(param->user,input,delimiter-input); //extract the user
        param->user[delimiter-input] = '\0';

        strncpy(param->password,delimiter+1,at_sign-delimiter-1);    //extract
the password
        param->password[at_sign-delimiter-1] = '\0';

        input = at_sign + 1;
        }
        else{
        strncpy(param->user,"anonymous",PARAM_SIZE);
        strncpy(param->password,"anonymous",PARAM_SIZE);
        }
        struct hostent *h;

        char *slash = strchr(input,'/');
        strncpy(param->host,input,slash-input); //extract the host

        strncpy(param->path,slash+1,PARAM_SIZE); //extract the path

        char * p = strrchr(param->path, '/');
        param->file = p - param->path + 1;
        if (p == NULL) param->file = 0;

        if ((h = gethostbyname(param->host)) == NULL) {
        printf("Invalid hostname '%s'\n", param->host);
        exit(-1);
        }
        strcpy(param->ip,   inet_ntoa(*((struct   in_addr   *)   h->h_addr)));
//trasnform the host into an ip address

        return 0;
}

int openSocket(char *IP,int port){

        int sockfd;
        struct sockaddr_in server_addr;

        /*server address handling*/
        bzero((char *) &server_addr, sizeof(server_addr));
        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr = inet_addr(IP);   /*32    bit    Internet
address network byte ordered*/
```

```
        server_addr.sin_port = htons(port);

        /*open a TCP socket*/
        if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket()");
        exit(-1);
        }


        /*connect to the server*/
        if         (connect(sockfd,         (struct         sockaddr         *)
&server_addr,sizeof(server_addr)) < 0) {
        perror("connect()");
        exit(-1);
        }

        return sockfd;
}

/**
 * @brief Reads answer of the socket.
 *
 * @param sockfd Socket file descriptor.
 * @param answer Buffer to store answer.
 * @param size Answer buffer size.
  * @return int - Returns the code of the answer if successful and -1
otherwise.
 */
int readAnswer(int sockfd, char *answer, unsigned size){
        int ret = -1;
        char byte;
        int i = 0;
        while (read(sockfd, &byte, 1) > 0) {
        if (i < size - 1) {
                answer[i] = byte;
                i++;
        }
        if (byte == '\n') {
                answer[i] = '\0';
                printf("A: %s", answer);
                if ( // If last line, break
                (answer[0] >= '0' && answer[0] <= '9') &&
                (answer[1] >= '0' && answer[1] <= '9') &&
                (answer[2] >= '0' && answer[2] <= '9') &&
                answer[3] == ' '
                ) {break;}
                i = 0;
        }
```

```
        }
        sscanf(answer, "%d", &ret);

        return ret;
}

int main(int argc, char **argv) {

        struct parameters param;

        if        (argc        !=        2)        printf("Usage:        %s
ftp://[<user>:<password>@]<host>/<url-path>\n", argv[0]);

        if(parseURL(argv[1],&param) != 0){
        printf("How to call: ftp://[<user>:<password>@]<host>/<url-path>\n");
        exit(1);
        }
        printf("user:   %s\n",param.user);
        printf("pw:  %s\n",param.password);
        printf("file i: %d\n",param.file);
        printf("file:   %s\n",param.path + param.file);
        printf("host:   %s\n",param.host);
        printf("ip:  %s\n",param.ip);
        printf("path:   %s\n",param.path);

        char answer[1024] = {0};
        int socket_A = openSocket(param.ip,FTP_PORT);
        int code = readAnswer(socket_A, answer, 1024);

        if(code != WELCOME_AN){
        printf("code=%d\n", code);
        printf("Error: Wrong answer received, supposed to be welcome!\n");
        exit(1);
        }

        char cmd[6 + PARAM_SIZE] = {0};
        strcpy(cmd, "user ");
        strcat(cmd, param.user);
        strcat(cmd, "\r\n");

        write(socket_A,cmd,strlen(cmd));
        printf("cmd: %s", cmd);
        memset(answer, 0, 1024);
        code = readAnswer(socket_A,answer, 1024);
        if(code != PASSWORD_ANS){
        printf("code:%d\n",code);
        printf("Error:   Wrong   answer   received,   supposed   to   request
password!\n");
```

```
        exit(1);
        }

        strcpy(cmd, "pass ");
        strcat(cmd, param.password);
        strcat(cmd, "\r\n");

        write(socket_A, cmd, strlen(cmd));
        printf("cmd: %s", cmd);
        memset(answer, 0, 1024);

        code = readAnswer(socket_A,answer, 1024);
        if(code != LOGIN_ANS){
        printf("Error: Wrong answer received, supposed to be login!\n");
        exit(1);
        }
        strcpy(cmd, "pasv\r\n");

        write(socket_A,cmd,6);
        printf("cmd: %s", cmd);
        memset(answer, 0, 1024);

        code = readAnswer(socket_A,answer, 1024);

        if(code != PASV_ANS){
        printf("Error: Wrong answer received, supposed to be pasv info!\n");
        exit(1);
        }

        unsigned char _i[4] = {0};
        unsigned char _p[2] = {0};
        char * beg = strchr(answer, '(');
        sscanf(beg,  "(%hhd,%hhd,%hhd,%hhd,%hhd,%hhd)",  _i,  _i+1,  _i+2,  _i+3,
_p, _p+1);

        int port = _p[0] * 256 + _p[1];
        char ip[40] = {0};
        sprintf(ip, "%d.%d.%d.%d", _i[0], _i[1], _i[2], _i[3]);
        int socket_B = openSocket(ip, port);

        strcpy(cmd, "retr ");
        strcat(cmd, param.path);
        strcat(cmd, "\r\n");

        write(socket_A, cmd, strlen(cmd));
        printf("cmd: %s", cmd);
        memset(answer, 0, 1024);
        code = readAnswer(socket_A,answer, 1024);
```

```
        printf("code:%d\n",code);
        if(code != OPEN_BINARY_ANS && code != OPEN_BINARY_ANS2){
        printf("Error:  Wrong  answer  received.  File  probably  does  not
exist.\n");
        exit(1);
        }

        FILE *file = fopen(param.path + param.file,"w");
        if(file == NULL){
        printf("Error, Could not open the file\n");
        exit(1);
        }
        int n_bytes;
        char temp[1024];

        while(1){
        n_bytes = read(socket_B,temp,1024);
        fwrite(temp,1, n_bytes, file);
        if(n_bytes <= 0) break;
        }
        fclose(file);
        if(close(socket_B) != 0) {
        printf("Error: Could not close socket B");
        exit(1);
        }

        memset(answer, 0, 1024);

        code = readAnswer(socket_A,answer, 1024);

        if(code != TRANSFER_COMP_ANS){
        printf("Error:  Wrong  answer  received,  supposed  to  be  transfer
completed!\n");
        exit(1);
        }
        strcpy(cmd, "quit\r\n");
        write(socket_A,cmd,6);

        printf("cmd: %s", cmd);

        memset(answer, 0, 1024);
        code = readAnswer(socket_A,answer, 1024);
        printf("code:%d\n",code);
        if(code != QUIT_ANS){
        printf("Error:  Wrong  answer  received,  supposed  to  be  transfer
completed!\n");
        exit(1);
        }
```

```
        if (close(socket_A) != 0){
        printf("Error: Could not close socket A\n");
        exit(1);
        }


        return 0;
}
```

Logs Captured

Part1:

No.	Time	Source	Destination	Protocol	Lengtl	Info
18	26.415068911	172.16.1.10	172.16.60.1	FTP	116	Response: 220 ProFTPD Server (Debian) [::ffff:172.16.1.10]
20	26.415202517	172.16.60.1	172.16.1.10	FTP	77	Request: user rcom
22	26.416227855	172.16.1.10	172.16.60.1	FTP	98	Response: 331 Password required for rcom
23	26.416295740	172.16.60.1	172.16.1.10	FTP	77	Request: pass rcom
25	26.590890537	172.16.1.10	172.16.60.1	FTP	112	Response: 230-Welcome, archive user rcom@172.16.1.61 !
26	26.590903527	172.16.1.10	172.16.60.1	FTP	115	Response:
27	26.591050612	172.16.1.10	172.16.60.1	FTP	233	Response:
29	26.591305044	172.16.60.1	172.16.1.10	FTP	72	Request: pasv
31	26.592186857	172.16.1.10	172.16.60.1	FTP	115	Response: 227 Entering Passive Mode (172,16,1,10,163,41).
35	26.592632164	172.16.60.1	172.16.1.10	FTP	87	Request: retr files/crab.mp4
36	26.593568314	172.16.1.10	172.16.60.1	FTP	142	Response: 150 Opening ASCII mode data connection for files/crab.mp4 (29803194 bytes)
31498	29.149098496	172.16.1.10	172.16.60.1	FTP	89	Response: 226 Transfer complete
31500	29.149171200	172.16.60.1	172.16.1.10	FTP	72	Request: quit
31501	29.149668678	172.16.1.10	172.16.60.1	FTP	80	Response: 221 Goodbye.

Exp1:

19	7.380949823	Netronix_71:74:10	KYE_25:24:5b	ARP	42	Who has 172.16.30.254? Tell 172.16.30.1
20	7.381010659	KYE_25:24:5b	Netronix_71:74:10	ARP	60	172.16.30.254 is at 00:c0:df:25:24:5b
21	8.008700726	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:13 Cost = 0 Port = 0x8003
22	8.308987570	172.16.30.1	172.16.30.254	ICMP	98	Echo (ping) request id=0x17e2, seq=7/1792, ttl=64 (reply in 23)
23	8.309064819	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x17e2, seq=7/1792, ttl=64 (request in 22)
24	9.332987065	172.16.30.1	172.16.30.254	ICMP	98	Echo (ping) request id=0x17e2, seq=8/2048, ttl=64 (reply in 25)
25	9.333081916	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x17e2, seq=8/2048, ttl=64 (request in 24)
26	10.011042583	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:13 Cost = 0 Port = 0x8003
27	10.360988575	172.16.30.1	172.16.30.254	ICMP	98	Echo (ping) request id=0x17e2, seq=9/2304, ttl=64 (reply in 28)
28	10.361072251	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x17e2, seq=9/2304, ttl=64 (request in 27)
29	12.013004489	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:13 Cost = 0 Port = 0x8003

Exp2

No.	Time	Source	Destination	Protocol	Length	Info
20	13.095649988	Netronix_71:74:10	KYE_13:20:0c	ARP	42	Who has 172.16.30.254? Tell 172.16.30.1
21	13.095738337	KYE_13:20:0c	Netronix_71:74:10	ARP	60	172.16.30.254 is at 00:c0:df:13:20:0c
22	13.991687559	172.16.30.1	172.16.30.254	ICMP	98	Echo (ping) request id=0x7102, seq=7/1792, ttl=64 (reply in 23)
23	13.991795604	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x7102, seq=7/1792, ttl=64 (request in 22)
24	14.014678576	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:15 Cost = 0 Port = 0x8001
25	15.015676341	172.16.30.1	172.16.30.254	ICMP	98	Echo (ping) request id=0x7102, seq=8/2048, ttl=64 (reply in 26)
26	15.015776354	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x7102, seq=8/2048, ttl=64 (request in 25)
27	16.017000126	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:15 Cost = 0 Port = 0x8001
28	16.039677066	172.16.30.1	172.16.30.254	ICMP	98	Echo (ping) request id=0x7102, seq=9/2304, ttl=64 (reply in 29)
29	16.039776450	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x7102, seq=9/2304, ttl=64 (request in 28)

No.	Time	Source	Destination	Protocol	Length	Info
64	50.053878495	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:15 Cost = 0 Port = 0x8001
65	50.791652624	Netronix_71:74:10	Broadcast	ARP	42	Who has 172.16.31.1? Tell 172.16.30.1
66	51.815650206	Netronix_71:74:10	Broadcast	ARP	42	Who has 172.16.31.1? Tell 172.16.30.1
67	52.056085505	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:15 Cost = 0 Port = 0x8001
68	52.839714765	Netronix_71:74:10	Broadcast	ARP	42	Who has 172.16.31.1? Tell 172.16.30.1
69	53.863661922	Netronix_71:74:10	Broadcast	ARP	42	Who has 172.16.31.1? Tell 172.16.30.1
70	54.058302432	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:15 Cost = 0 Port = 0x8001
71	54.887648259	Netronix_71:74:10	Broadcast	ARP	42	Who has 172.16.31.1? Tell 172.16.30.1
72	55.911679504	Netronix_71:74:10	Broadcast	ARP	42	Who has 172.16.31.1? Tell 172.16.30.1
73	56.060516915	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:15 Cost = 0 Port = 0x8001

No.	Time	Source	Destination	Protocol	Length	Info
49	88.096886216	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:15 Cost = 0 Port = 0x8001
50	88.917284006	172.16.30.1	172.16.30.255	ICMP	98	Echo (ping) request id=0x7698, seq=1/256, ttl=64 (no response found!)
51	88.917444362	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x7698, seq=1/256, ttl=64
52	89.939434836	172.16.30.1	172.16.30.255	ICMP	98	Echo (ping) request id=0x7698, seq=2/512, ttl=64 (no response found!)
53	89.939576614	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x7698, seq=2/512, ttl=64
54	90.099099929	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:15 Cost = 0 Port = 0x8001
55	90.963426341	172.16.30.1	172.16.30.255	ICMP	98	Echo (ping) request id=0x7698, seq=3/768, ttl=64 (no response found!)
56	90.963565884	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x7698, seq=3/768, ttl=64
57	91.987430557	172.16.30.1	172.16.30.255	ICMP	98	Echo (ping) request id=0x7698, seq=4/1024, ttl=64 (no response found!)
58	91.987572684	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x7698, seq=4/1024, ttl=64

No.	Time	Source	Destination	Protocol	Length	Info
32	56.061405708	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:15 Cost = 0 Port = 0x8002
33	56.881852141	172.16.30.1	172.16.30.255	ICMP	98	Echo (ping) request id=0x7698, seq=1/256, ttl=64 (no response found!)
34	56.881881823	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x7698, seq=1/256, ttl=64
35	57.903993687	172.16.30.1	172.16.30.255	ICMP	98	Echo (ping) request id=0x7698, seq=2/512, ttl=64 (no response found!)
36	57.904017642	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x7698, seq=2/512, ttl=64
37	58.063616772	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:15 Cost = 0 Port = 0x8002
38	58.927985947	172.16.30.1	172.16.30.255	ICMP	98	Echo (ping) request id=0x7698, seq=3/768, ttl=64 (no response found!)
39	58.928008505	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x7698, seq=3/768, ttl=64
40	59.951995946	172.16.30.1	172.16.30.255	ICMP	98	Echo (ping) request id=0x7698, seq=4/1024, ttl=64 (no response found!)
41	59.952015362	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x7698, seq=4/1024, ttl=64

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:14 Cost = 0 Port = 0x8001
2	2.002217848	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:14 Cost = 0 Port = 0x8001
3	2.016929322	172.16.31.1	172.16.31.255	ICMP	98	Echo (ping) request id=0x06ee, seq=1/256, ttl=64 (no response found!)
4	3.037224182	172.16.31.1	172.16.31.255	ICMP	98	Echo (ping) request id=0x06ee, seq=2/512, ttl=64 (no response found!)
5	4.004448616	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:14 Cost = 0 Port = 0x8001
6	4.061226417	172.16.31.1	172.16.31.255	ICMP	98	Echo (ping) request id=0x06ee, seq=3/768, ttl=64 (no response found!)
7	5.085224672	172.16.31.1	172.16.31.255	ICMP	98	Echo (ping) request id=0x06ee, seq=4/1024, ttl=64 (no response found!)
8	6.006666534	Routerboardc_1c:8e:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:14 Cost = 0 Port = 0x8001
9	6.109222856	172.16.31.1	172.16.31.255	ICMP	98	Echo (ping) request id=0x06ee, seq=5/1280, ttl=64 (no response found!)
10	7.133224254	172.16.31.1	172.16.31.255	ICMP	98	Echo (ping) request id=0x06ee, seq=6/1536, ttl=64 (no response found!)

Exp3:

No.	Time	Source	Destination	Protocol	Length	Info
13	22.764180808	172.16.60.1	172.16.60.254	ICMP	98	Echo (ping) request id=0x356f, seq=1/256, ttl=64 (reply in 14)
14	22.764348986	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x356f, seq=1/256, ttl=64 (request in 13)
15	23.790730230	172.16.60.1	172.16.60.254	ICMP	98	Echo (ping) request id=0x356f, seq=2/512, ttl=64 (reply in 16)
16	23.790900992	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x356f, seq=2/512, ttl=64 (request in 15)

No.	Time	Source	Destination	Protocol	Length	Info
39	36.750716714	172.16.60.1	172.16.61.253	ICMP	98	Echo (ping) request id=0x3576, seq=2/512, ttl=64 (reply in 40)
40	36.750886289	172.16.61.253	172.16.60.1	ICMP	98	Echo (ping) reply id=0x3576, seq=2/512, ttl=64 (request in 39)
41	37.774716443	172.16.60.1	172.16.61.253	ICMP	98	Echo (ping) request id=0x3576, seq=3/768, ttl=64 (reply in 42)
42	37.774892931	172.16.61.253	172.16.60.1	ICMP	98	Echo (ping) reply id=0x3576, seq=3/768, ttl=64 (request in 41)

No.	Time	Source	Destination	Protocol	Length	Info
52	50.316114988	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) request id=0x3580, seq=1/256, ttl=64 (reply in 53)
53	50.316435838	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) reply id=0x3580, seq=1/256, ttl=63 (request in 52)
54	51.342699889	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) request id=0x3580, seq=2/512, ttl=64 (reply in 55)
55	51.342964587	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) reply id=0x3580, seq=2/512, ttl=63 (request in 54)

No.	Time	Source	Destination	Protocol	Length	Info
148	263.502649315	KYE_25:40:66	Broadcast	ARP	60	Who has 172.16.60.254? Tell 172.16.60.1
149	263.502682141	3Com_a1:35:69	KYE_25:40:66	ARP	42	172.16.60.254 is at 00:01:02:a1:35:69
150	263.502799198	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) request id=0x36a5, seq=1/256, ttl=64 (reply in 151)
151	263.503102107	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) reply id=0x36a5, seq=1/256, ttl=63 (request in 150)

No.	Time	Source	Destination	Protocol	Length	Info
289	251.010725792	KYE_04:20:8c	Broadcast	ARP	42	Who has 172.16.61.1? Tell 172.16.61.253
290	251.010866176	Netronix_b5:8c:8e	KYE_04:20:8c	ARP	60	172.16.61.1 is at 00:e0:7d:b5:8c:8e
291	251.010879376	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) request id=0x36a5, seq=1/256, ttl=63 (reply in 292)
292	251.011000205	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) reply id=0x36a5, seq=1/256, ttl=64 (request in 291)

Exp4:

No.	Time	Source	Destination	Protocol	Length	Info
5	6.264195164	172.16.30.1	172.16.30.254	ICMP	98	Echo (ping) request id=0x36df, seq=1/256, ttl=64 (reply in 6)
6	6.264324161	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x36df, seq=1/256, ttl=64 (request in 5)

No.	Time	Source	Destination	Protocol	Length	Info
31	25.944274474	172.16.30.1	172.16.31.253	ICMP	98	Echo (ping) request id=0x36ec, seq=1/256, ttl=64 (reply in 32)
32	25.944411015	172.16.31.253	172.16.30.1	ICMP	98	Echo (ping) reply id=0x36ec, seq=1/256, ttl=64 (request in 31)

No.	Time	Source	Destination	Protocol	Length	Info
53	39.896474594	172.16.30.1	172.16.31.1	ICMP	98	Echo (ping) request id=0x36f6, seq=1/256, ttl=64 (reply in 54)
54	39.896801802	172.16.31.1	172.16.30.1	ICMP	98	Echo (ping) reply id=0x36f6, seq=1/256, ttl=63 (request in 53)

No.	Time	Source	Destination	Protocol	Length	Info
70	52.072501089	172.16.30.1	172.16.31.254	ICMP	98	Echo (ping) request id=0x3700, seq=1/256, ttl=64 (reply in 71)
71	52.072822361	172.16.31.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x3700, seq=1/256, ttl=63 (request in 70)

No.	Time	Source	Destination	Protocol	Length	Info
4	1.022475848	172.16.60.1	172.16.1.61	ICMP	98	Echo (ping) request id=0x3a8c, seq=2/512, ttl=64 (reply in 5)
5	1.022760172	172.16.1.61	172.16.60.1	ICMP	98	Echo (ping) reply id=0x3a8c, seq=2/512, ttl=63 (request in 4)

No.	Time	Source	Destination	Protocol	Length	Info
36	50.769490854	172.16.31.1	172.16.30.1	ICMP	98	Echo (ping) request id=0x2517, seq=2/512, ttl=64 (reply in 38)
37	50.769642476	172.16.31.254	172.16.31.1	ICMP	126	Redirect (Redirect for host)
38	50.769818892	172.16.30.1	172.16.31.1	ICMP	98	Echo (ping) reply id=0x2517, seq=2/512, ttl=63 (request in 36)
39	51.793483265	172.16.31.1	172.16.30.1	ICMP	98	Echo (ping) request id=0x2517, seq=3/768, ttl=64 (reply in 41)
40	51.793645782	172.16.31.254	172.16.31.1	ICMP	126	Redirect (Redirect for host)
41	51.793814027	172.16.30.1	172.16.31.1	ICMP	98	Echo (ping) reply id=0x2517, seq=3/768, ttl=63 (request in 39)
43	52.817481683	172.16.31.1	172.16.30.1	ICMP	98	Echo (ping) request id=0x2517, seq=4/1024, ttl=64 (reply in 45)
44	52.817633654	172.16.31.254	172.16.31.1	ICMP	126	Redirect (Redirect for host)
45	52.817828926	172.16.30.1	172.16.31.1	ICMP	98	Echo (ping) reply id=0x2517, seq=4/1024, ttl=63 (request in 43)

No.	Time	Source	Destination	Protocol	Length	Info
11	7.394510069	172.16.31.1	172.16.30.1	ICMP	98	Echo (ping) request id=0x2888, seq=2/512, ttl=64 (reply in 13)
12	7.394696262	172.16.31.254	172.16.31.1	ICMP	126	Redirect (Redirect for host)
13	7.394871140	172.16.30.1	172.16.31.1	ICMP	98	Echo (ping) reply id=0x2888, seq=2/512, ttl=63 (request in 11)
15	8.418512272	172.16.31.1	172.16.30.1	ICMP	98	Echo (ping) request id=0x2888, seq=3/768, ttl=64 (reply in 16)
16	8.418761531	172.16.30.1	172.16.31.1	ICMP	98	Echo (ping) reply id=0x2888, seq=3/768, ttl=63 (request in 15)
17	9.442510076	172.16.31.1	172.16.30.1	ICMP	98	Echo (ping) request id=0x2888, seq=4/1024, ttl=64 (reply in 18)
18	9.442739081	172.16.30.1	172.16.31.1	ICMP	98	Echo (ping) reply id=0x2888, seq=4/1024, ttl=63 (request in 17)
20	10.466504597	172.16.31.1	172.16.30.1	ICMP	98	Echo (ping) request id=0x2888, seq=5/1280, ttl=64 (reply in 21)
21	10.466764262	172.16.30.1	172.16.31.1	ICMP	98	Echo (ping) reply id=0x2888, seq=5/1280, ttl=63 (request in 20)

Exp5:

No.	Time	Source	Destination	Protocol	Lengtl	Info
78	7.783739788	10.227.20.53	10.227.20.3	DNS	72	Standard query 0x89ef A santander.pt
79	7.783749706	10.227.20.53	10.227.20.3	DNS	72	Standard query 0x06fa AAAA santander.pt
80	7.784318845	10.227.20.3	10.227.20.53	DNS	104	Standard query response 0x89ef A santander.pt A 2.16.8.96 A 2.16.8.42
81	7.784347969	10.227.20.3	10.227.20.53	DNS	149	Standard query response 0x06fa AAAA santander.pt SOA eur2.akam.net
85	7.808377962	10.227.20.53	10.227.20.3	DNS	82	Standard query 0x9882 PTR 96.8.16.2.in-addr.arpa
86	7.808850720	10.227.20.3	10.227.20.53	DNS	143	Standard query response 0x9882 PTR 96.8.16.2.in-addr.arpa PTR a2-16-8-96.deploy.static.akamaitechno

Exp6:

No.	Time	Source	Destination	Protocol	Lengtl	Info
18	26.415068911	172.16.1.10	172.16.60.1	FTP	116	Response: 220 ProFTPD Server (Debian) [::ffff:172.16.1.10]
20	26.415202517	172.16.60.1	172.16.1.10	FTP	77	Request: user rcom
22	26.416227855	172.16.1.10	172.16.60.1	FTP	98	Response: 331 Password required for rcom
23	26.416295740	172.16.60.1	172.16.1.10	FTP	77	Request: pass rcom
25	26.590890537	172.16.1.10	172.16.60.1	FTP	112	Response: 230-Welcome, archive user rcom@172.16.1.61 !
26	26.590903527	172.16.1.10	172.16.60.1	FTP	115	Response:
27	26.591050612	172.16.1.10	172.16.60.1	FTP	233	Response:
29	26.591305044	172.16.60.1	172.16.1.10	FTP	72	Request: pasv
31	26.592186857	172.16.1.10	172.16.60.1	FTP	115	Response: 227 Entering Passive Mode (172,16,1,10,163,41).
35	26.592632164	172.16.60.1	172.16.1.10	FTP	87	Request: retr files/crab.mp4
36	26.593568314	172.16.1.10	172.16.60.1	FTP	142	Response: 150 Opening ASCII mode data connection for files/crab.mp4 (29803194 bytes)
31498	29.149098496	172.16.1.10	172.16.60.1	FTP	89	Response: 226 Transfer complete
31500	29.149171200	172.16.60.1	172.16.1.10	FTP	72	Request: quit
31501	29.149668678	172.16.1.10	172.16.60.1	FTP	80	Response: 221 Goodbye.

Configuration Commands

Beginning

Restart networking in all the tuxes:

```
systemctl restart networking
```

Restart the **switch** and the **router**:

```
/system reset-configuration
```

The user is **admin** and the password is blank.

Exp 1

Connect eth1 of tuxY3 and tuxY4 to the switch (ether3 & ether4).

tuxY3

Configure eth1 interface:

```
ifconfig eth1 up
ifconfig eth1 172.16.Y0.1/24
```

tuxY4

Configure eth1 interface:

```
ifconfig eth1 up
ifconfig eth1 172.16.Y0.254/24
```

Exp 2

Connect eth1 of tuxY2 to the switch (ether2).

tuxY2

Configure eth1 interface:

```
ifconfig eth1 up
ifconfig eth1 172.16.Y1.1/24
```

Switch

Remove ports from default bridge:
Ether2:

```
/interface bridge port print
/interface bridge port remove [find interface=ether2]
```

Ether3:

```
/interface bridge port print
/interface bridge port remove [find interface=ether3]
```

Ether4:

```
/interface bridge port print
/interface bridge port remove [find interface=ether4]
```

Create bridges **bridgeY0** and **bridgeY1**:

```
/interface bridge add name=bridgeY0
/interface bridge add name=bridgeY1
```

Assuming tuxY2, tuxY3, and tuxY4 are connected to ports ether2, ether3 and ether4, respectively. Add ports to bridges:

```
/interface bridge port add bridge=bridgeY0 interface=ether3
/interface bridge port add bridge=bridgeY0 interface=ether4
/interface bridge port add bridge=bridgeY1 interface=ether2
```

Exp 3

Connect eth2 of tuxY4 to the switch (ether10).

tuxY4

Configure eth2 interface:

```
ifconfig eth2 up
ifconfig eth2 172.16.Y1.253/24
```

Enable IP forwarding:

```
sysctl net.ipv4.ip_forward=1
```

Disable ICMP echo-ignore-broadcast:

```
sysctl net.ipv4.icmp_echo_ignore_broadcasts=0
```

Switch

Assuming eth2 interface of tuxY4 is connected to the switch on ether10. Remove ether10 from default bridge:

```
/interface bridge port remove [find interface=ether10]
```

Add eth2 of tuxY4 to bridgeY1:

```
/interface bridge port add bridge=bridgeY1 interface=ether10
```

tuxY2

Add route to subnetwork 172.16.Y0.0/24 via eth2 of tuxY4:

```
route add -net 172.16.Y0.0/24 gw 172.16.Y1.253
```

tuxY3

Add route to subnetwork 172.16.Y1.0/24 via eth1 of tuxY4:

```
route add -net 172.16.Y1.0/24 gw 172.16.Y0.254
```

Exp 4

IMPORTANT: Reset router, if not done yet!!

Connect ether1 of Rc to PY.12. Connect ether2 of RC to the switch (ether15).

Switch

Assuming ether2 of Rc is connected to the switch on ether15. Remove ether15 from default bridge:

```
/interface bridge port remove [find interface=ether15]
```

Add ether2 of Rc to bridgeY1:

```
/interface bridge port add bridge=bridgeY1 interface=ether15
```

Router

Configure IP addresses of Rc:

```
/ip address add address=172.16.1.Y1/24 interface=ether1
/ip address add address=172.16.Y1.254/24 interface=ether2
```

tuxY3

Route to subnetwork 172.16.Y1.0/24 is already configured in Exp 3.

Add route to 172.16.1.0/24:

```
route add -net 172.16.1.0/24 gw 172.16.Y0.254
```

tuxY4

Add route to 172.16.1.0/24:

```
route add -net 172.16.1.0/24 gw 172.16.Y1.254
```

tuxY2

Route to subnetwork 172.16.Y0.0/24 is already configured in Exp 3.

Add route to 172.16.1.0/24:

```
route add -net 172.16.1.0/24 gw 172.16.Y1.254
```

Router

Add route to 172.16.Y0.0/24:

```
/ip route add dst-address=172.16.Y0.0/24 gateway=172.16.Y1.253
```

Exp 5

tuxY2

Configure the DNS (with ip address of 10.225.20.3):

```
echo nameserver 10.227.20.3 >> /etc/resolv.conf
```

tuxY3

Configure the DNS (with ip address of 10.225.20.3):

```
echo nameserver 10.227.20.3 >> /etc/resolv.conf
```

tuxY4

Configure the DNS (with ip address of 10.225.20.3):

```
echo nameserver 10.227.20.3 >> /etc/resolv.conf
```

Exp 6

Compile the download application:

```
gcc download.c -o download
```

Run the download application:

```
./download <URL>
```

This URL is in the format ftp://[<user>:<password>@]<host>/<url-path>