U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Report on File Transfer Protocol Client and Network Setup

Computer Networks 2025/26

Class 04
Casemiro Melo Jorge de Medeiros 202301897
Mykyta Melnykovych 202309183

Summary

This report analyzes the development of a FTP download client application and also analyzes the necessary steps for fully configuring a network that uses one MikroTik router, one MikroTik switch and three debian-based computers. By the end of this process, it is possible to use the developed client to download a file from a FTP server, it can be from an outside server or from Netlab server, from any of the three computers. The client can also be executed from any computer to download files from public FTP servers.

Table of Contents

Introduction

During the second part of the semester, efforts were directed towards the second practical assignment. The project was divided into two distinct phases. The initial phase was the development of a download application, while the second was the setup of a network in the Netlab room. This network was configured using three computers (tux92, tux93, and tux94) one router and one switch. The final product consists of a tux computer running the application, accessing the Netlab FTP server through the said network, and downloading a file.

Part 1 - Download Application

As previously mentioned, the initial phase of the project consisted in developing a simple program in the C language that implements the File Transfer Protocol (FTP). In summary, the application receives an URL (in the format *ftp://[<user>:<pass>@]<host>/<url-path>*) as an argument, then it parses it, extracting all necessary information. Finally, it establishes a binary, passive-mode connection (authenticated if needed) with the desired host. After the file is successfully transferred, both TCP sockets that were previously opened, one for control and the other for data, are closed. In other words, the connection is closed.

Architecture

The download client is composed of three files, one being the main file and the other two auxiliary files, plus two header files to better organize the FTP response codes and define a struct called *Url* for storing the information extracted from the URL. The three said files are *download.c*, *parser.c* and *utils.c*.

The two auxiliary files are *parser.c* and *utils.c*. The *utils* file contains only one function that is the function used by *parser* to retrieve the IP address of the target server by its host name. Meanwhile, the *parser* file has the logic that reads the URL passed and stores it in the previously said *Url* struct.

The main file is *download.c*. This file contains the *main function*, which handles the TCP socket creation, managing the connection, communication with the server, and file transfer. It also contains the definition of five auxiliary functions, *is_msg_complete*, *get_message*, *send_message*, *parse_ip*, and *get_file_size*. The *get_message* and *send_message* functions are used for communication between client and server, and *is_msg_complete* is used inside *get_message* to check if the message was fully received. The function *parse_ip* is used to compute the new IP address that will be used in the data TCP socket. Lastly, the *get_file_size* function is used before the file transfer itself starts to retrieve the size of the desired file.

What happens in a successful download?

In the event of a successful download, the user can observe the messages exchanged between the client and the server in the terminal. This means that the user can see the messages “transfer complete” and “goodbye”, indicating that the transfer was

successful and that the connection was closed. The received file will be available in the same directory as the project. The FTP packets can be verified in annex 1.

Part 2 - Network configuration and analysis

The subsequent phase of the project consisted in performing several experiments. Each experiment was associated with a specific component of the network setup. For each of these experiments extra steps were done, including capturing Wireshark logs, to better understand how the network operates.

Experiment 1 - Configure an IP Network

The main objective of the experiment was to set IPs of two different computers, namely tux93 and tux 94, and test the communication between them. The log of this experiment is available in annex 2.1 and the network diagram is available in annex 2.2.

Questions

» What are the ARP packets and what are they used for?

The Address Resolution Protocol (ARP) packets are used for associating MAC addresses to IP addresses. A request is broadcasted asking who has a certain IP address and a reply is a response containing the MAC address that will be assigned to the IP.

» What are the MAC and IP addresses of ARP packets and why?

First the tux93, with MAC ec:75:0c:c2:17:7a and IP 172.16.90.1, broadcasts a request to find out the MAC address of a device with IP 172.16.90.254. Then tux94 sends a response, saying that 172.16.90.254 (tux94) has the following MAC address: ec:75:0c:c2:26:42.

» What packets does the ping command generate?

The ping command generates Internet Control Message Protocol (ICMP) packets by default, but it can also generate ARP packets if the IP address is not related to a cached MAC address.

» What are the MAC and IP addresses of the ping packets?

Requests are sent from tux93 (MAC ec:75:0c:c2:17:7a IP 172.16.90.1) to tux94. On the other hand, responses are sent from tux94 (MAC ec:75:0c:c2:26:42 IP 172.16.90.254) to tux93.

» How to determine if a receiving Ethernet frame is ARP, IP, ICMP?

Using Wireshark, it is possible to check the "Protocol" column. However, it is possible to inspect the Ethernet header

» How to determine the length of a receiving frame?

Using Wireshark, it is possible to check the "Length" column. However, similar to determining the protocol, it is possible to inspect the frame header and find the frame length.

» What is the loopback interface and why is it important?

The loopback interface is an interface for locally testing applications that interact with the network (e.g. php servers), inter-process communication

Experiment 2 - Implement two bridges in a switch

The main objective of this experiment was to add a new device, tux92, and set two distinct bridges in the switch, therefore creating two different subnets inside the main network. The final communication diagram can be seen in annex 3.8 and verified in the log available in annex 3.1.

Questions

» How to configure bridgeY0?

In our case, to configure the bridge90, we first created the bridge with */interface bridge add name=bridge90*. After that, considering that tux93 was connected to switch port 16 and tux94 to port 14, we needed to remove the default bridge configuration with */interface bridge port remove [find interface=etherXX]*. Finally, with the ports not belonging to any bridges, we could add them to our new bridge90 with */interface bridge port add bridge=bridge90 interface=etherXX*. It is important to highlight that these commands need to be performed in GTK while being connected to SW CONS.

» How many broadcast domains are there? How can you conclude it from the logs?

Having two different bridges, bridge90 and bridge91, we have two different broadcast domains. Those two bridges work like two subnets inside the general network we want to set up. Through the logs we can conclude this because when pinging 172.16.90.255 broadcast, tux93 and tux94 can see the ICMP packets, but tux92 cannot see any. However, when pinging 172.16.16.91.255 broadcast, it occurs the opposite, only tux92 sees ICMP packets. Detailed logs in annexes, also the lack of ICMP packets are easily concluded by the lack of the pink color on the scroll.

Experiment 3 - Configure a Route in Linux

The main objective of this experiment is to transform tux94 into a router. This way, there will be a way to subnet 90 and subnet 91 interact. Network diagram is available in annex 4.1.

Questions

» What routes are there in the tuxes? What are their meaning?

After setting the new routes, we observe that tux92 has a route to network 172.16.90.0/24 using 172.16.91.253 (tux94 if_e2) as gateway, as well as a route to 172.16.91.0/24 through 0.0.0.0, that means that any traffic destined to that LAN, stays within. On tux93 we observe almost the same, but destined to 172.16.91.0/24 through 172.16.90.254 (tux94 if_e1). Lastly, on tux94 we observe that it only contains routes to itself since it is responsible for doing the redirect between networks. All routes are available in annexes.

» What information does an entry of the forwarding table contain?

A forwarding table contains the destination, gateway, genmask, flags, metric and iface. *Destination* is the target network/host, *gateway* is the next hop to reach the *destination* (if it is 0.0.0.0 the packet does not need to change network or it is a direct route), *genmask* is the subnet mask, *flags* is the status/type of a certain route, *metric* is a value assigned to a network path that helps the operating system/router choose the best route when multiple options exist to the same destination, lower metric values indicates a higher preference, *iface* is is the interface associated to that route.

» What ARP messages, and associated MAC addresses, are observed and why?

When inspecting the tux94 (if_e1), we observe that there is a broadcast message from tux93 (ec:75:0c:c2:17:7a) asking who owns IP 172.16.90.254 (ec:75:0c:c2:26:42). After that, when inspecting the tux (if_e2), we observe that tux94 (ec:75:0c:c2:31:93) broadcast an ARP message asking who has IP 172.16.91.1 (a2:05:02:f6:c6:5c). Finally, it is possible to see ICMP packets from tux93 (172.16.90.1) reaching tux92 (172.16.91.1) and it being replied.

Those ARP packets occur because the ARP tables were cleaned. All related logs can be found in annexes.

» What ICMP packets are observed and why?

As previously mentioned, we observe ICMP packets going from tux93 and reaching tux92 on both tux94 network interfaces.

» What are the IP and MAC addresses associated to ICMP packets and why?

In an ICMP packet we have the Source P address, who is pinging, and the Destination IP address, who is being pinged. Similarly, it happens the same with MAC addresses. There is the Source MAC address, the address of the sending network interface, and the Destination MAC address, the address of the receiving network interface.

The IP addresses ensure communication between different devices in a more general way. In contrast, the MAC addresses are used for local communication, that is, inside the same subnetwork. This can also be observed after each hop, since the MAC addresses change, but not the IP addresses.

Experiment 4 - Configure a Commercial Router and Implement NAT

The main goal of this experiment is to connect our current network to the lab network, so that our computers can access the Netlab FTP server. The network diagram can be found in annexes.

Questions

» How to configure a static route in a commercial router?

For configuring a static route in our router, we first needed to do the router configuration. First, add the router (switch port ether10) to bridge91. Then, set IPs for the router interfaces with commands */ip address add address=172.16.1.91/24 interface=ether1* and */ip address add address=172.16.91.254/24 interface=ether2*. Finally, we can run */ip*

route add dst-address=172.16.90.0/24 gateway=172.16.91.253. All these commands must be executed in GTK, the bridge related commands on SW CONS mode and then on ROUTER CONS.

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

Inspecting the packets captured by Wireshark and traceroutes (all available in annexes), we observe that with ICMP redirect disabled (they appear on Wireshark because they are being rejected) the packets need to travel a longer (tux92 -> router -> tux94 -> tux93) path than with ICMP redirect enabled (tux92 -> tux94 -> tux93). In conclusion, ICMP redirect messages are responsible for informing the source about a shorter path to the destination.

» How to configure NAT in a commercial router?

By default it comes activated, but for certainty we can run */ip firewall nat disable 0* for turning it off and */ip firewall nat enable 0* for turning it on. Both commands need to be run in GTK.

» What does NAT do?

NAT (Network Address Translation) is responsible for translating IP addresses from a local network to a single public IP address, and the opposite. For example, when tux93 (172.16.90.1) tries to reach the FTP server (172.16.1.10), the router changes the Source IP address to its own (172.16.1.91). When the FTP sends a response, it is destined to the router IP address, but then NAT translates it in order to be properly sent to tux93.

This way, there are less public IP addresses in a network and artificially more available IPs, since one single public IP can receive messages on behalf of a lot more.

» What happens when tuxY3 pings the FTP server with the NAT disabled? Why?

When the NAT is disabled, the Source IP of an ICMP packet does not get translated to the router's IP address. Also, even if the packet got to the FTP server, it would not reach tux93, since NAT is not there to translate the IP address to the correct recipient.

Experiment 5 - DNS

The main purpose of this experiment is to configure a DNS (Domain Name System) so that we can use human-readable names instead of IP addresses.

Questions

» How to configure the DNS service in a host?

For setting the DNS service, it is necessary to check the file */etc/resolv.conf*. Once we opened the file, we added *nameserver 10.227.20.3*. After that, we are able to use hostname instead of IPs.

» What packets are exchanged by DNS and what information is transported?

When pinging a server that our device has never seen before, the DNS message is seen in wireshark. In this message we verify that the device is asking the DNS server for the

IP address associated with the domain google.com. Then, it responds with the IP used in ICMP packets. This log can be accessed in the annexes.

Experiment 6 - TCP Connections

The main objective of this last experiment is to test our developed application and to test the network setup.

Questions

» How many TCP connections are opened by your FTP application?

Two connections are opened - one for exchanging messages with the FTP server, and one for downloading the file in passive mode.

» In what connection is transported the FTP control information?

In the first one, which is used for exchanging messages with the server.

» What are the phases of a TCP connection?

First, there is the connection establishment (3-way handshake), then data transfer (client and server sending packages to each other). Finally, there is connection termination.

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

ARQ mechanism ensures that the data is delivered by retransmitting lost/corrupted packets. When data is sent the receiver responds with an acknowledgment (ACK). If the sender fails to receive an ACK in time, he retransmits the packet. TCP packets include the sequence number and the acknowledgment number.

In logs we can see how the connection is being established for both the control and data connections, how messages are sent and connections are closed.

» 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 mechanism ensures that the sender does not overwhelm the network with too much data. Sender starts with sending a small amount of data, gradually increasing the rate. At a certain threshold the sender starts to increase the rate more slowly to avoid sudden spikes in network congestion. If a packet is lost (e.g. sender receives duplicate ACKs), sender retransmits the lost packet quickly, without waiting for a timeout. After the loss the congestion window is reduced. After a loss, the sending rate is still being slowly increased.

Relevant fields are:

Congestion window (Win) - determines how much data can be sent by sender before waiting for ACKs. It is adjusted based on the current network congestion.

Acknowledgment number(Ack) - is used to confirmed received packets

Sequence number (Seq) - helps track the order of the packets to make sure nothing is lost.

» Is the throughput of a TCP data connections disturbed by the appearance of a

second TCP connection? How?

Yes, it is. We can observe with the help of a graph that when a second download client is executed in tux92, it makes the total number of packets go up since it was not on the channel limit. However, it is not possible to view in the graph the total speed going down because the second download stayed up for not enough time.

Conclusions

In conclusion, we successfully developed a robust FTP client for Linux-based systems capable of downloading files from both Netlab server and public FTP repositories. Additionally, we were able to successfully configure a complete network infrastructure that uses a router and a switch to manage traffic. Lastly, we used Wireshark to capture packets and inspect what was happening during our setup and see how our FTP client behaved under different circumstances.

References

The only materials we used were the material available in RCOM's moodle.

Annexes

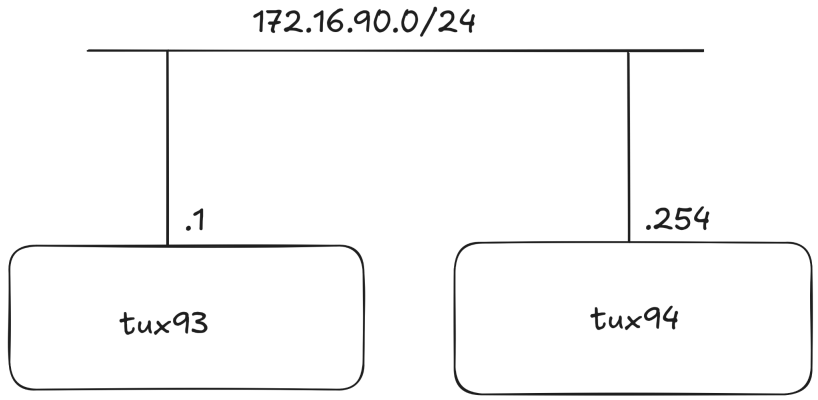
Annex 1.1 - FTP Client successful download

No.	Time	Source	Destination	Protocol	Length	Info
4	0.005244724	172.16.1.10	172.16.90.1	FTP	86	Response: 220 (vsFTPd 3.0.5)
6	0.005332539	172.16.90.1	172.16.1.10	FTP	82	Request: USER anonymous
8	0.005810158	172.16.1.10	172.16.90.1	FTP	100	Response: 331 Please specify the password.
9	0.005865509	172.16.90.1	172.16.1.10	FTP	81	Request: PASS password
10	0.011262290	172.16.1.10	172.16.90.1	FTP	89	Response: 230 Login successful.
11	0.011309666	172.16.90.1	172.16.1.10	FTP	74	Request: TYPE I
12	0.011898997	172.16.1.10	172.16.90.1	FTP	97	Response: 200 Switching to Binary mode.
13	0.011961227	172.16.90.1	172.16.1.10	FTP	82	Request: SIZE /pipe.txt
14	0.012514301	172.16.1.10	172.16.90.1	FTP	75	Response: 213 191
15	0.012576927	172.16.90.1	172.16.1.10	FTP	72	Request: PASV
16	0.013431235	172.16.1.10	172.16.90.1	FTP	115	Response: 227 Entering Passive Mode (172,16,1,10,165,69).
20	0.013998700	172.16.90.1	172.16.1.10	FTP	82	Request: RETR /pipe.txt
21	0.014929987	172.16.1.10	172.16.90.1	FTP	134	Response: 150 Opening BINARY mode data connection for /pipe.t
27	0.058758991	172.16.1.10	172.16.90.1	FTP	90	Response: 226 Transfer complete.
29	0.058882913	172.16.90.1	172.16.1.10	FTP	72	Request: QUIT
30	0.059344481	172.16.1.10	172.16.90.1	FTP	80	Response: 221 Goodbye.

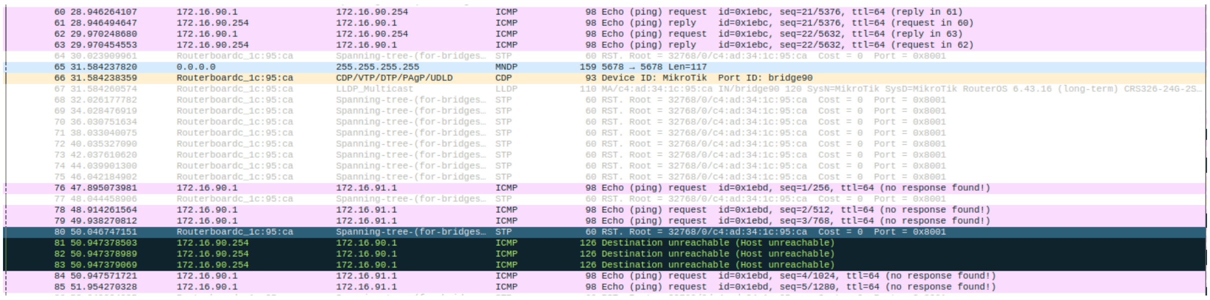
Annex 2.1 - Experiment 1: Log

7	10.62637…	TPLink_c2:17:7a	Broadcast	ARP	42	Who has 172.16.90.254? Tell 172.16.90.1
8	10.62654…	TPLink_c2:26:42	TPLink_c2:17:7a	ARP	60	172.16.90.254 is at ec:75:0c:c2:26:42
9	10.62656…	172.16.90.1	172.16.90.254	ICMP	98	Echo (ping) request id=0x127b, seq=1/256, ttl=64 (reply in 10)
10	10.62672…	172.16.90.254	172.16.90.1	ICMP	98	Echo (ping) reply id=0x127b, seq=1/256, ttl=64 (request in 9)
11	11.65234…	172.16.90.1	172.16.90.254	ICMP	98	Echo (ping) request id=0x127b, seq=2/512, ttl=64 (reply in 12)
12	11.65250…	172.16.90.254	172.16.90.1	ICMP	98	Echo (ping) reply id=0x127b, seq=2/512, ttl=64 (request in 11)

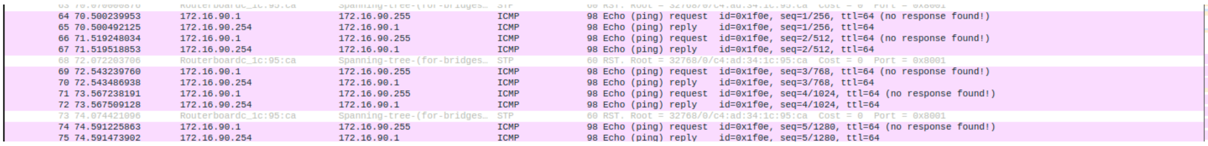
Annex 2.2 - Experiment 1: Network



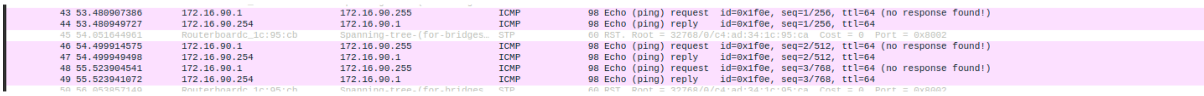
Annex 3.1 - Experiment 2: Step 5



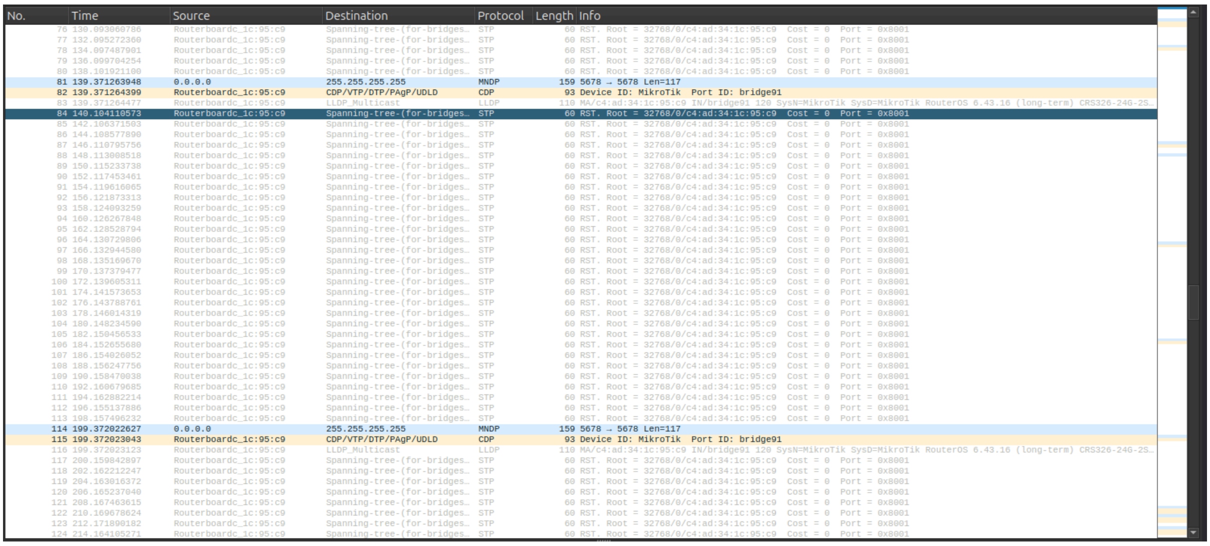
Annex 3.2 - Experiment 2: Pinging 172.16.90.255 broadcast in tux93



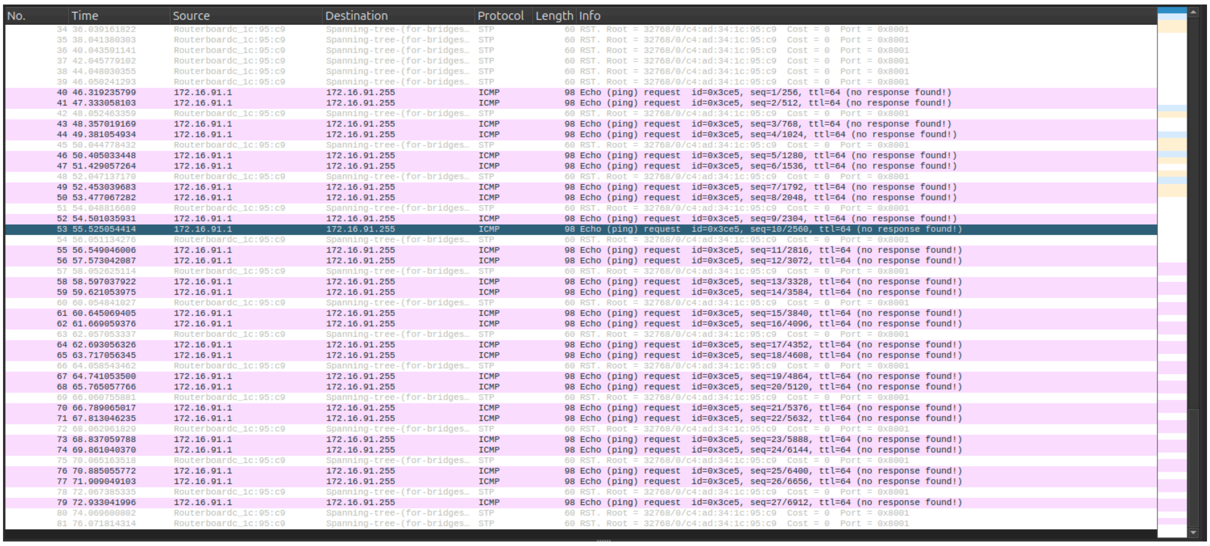
Annex 3.3 - Experiment 2: Pinging 172.16.90.255 broadcast in tux94



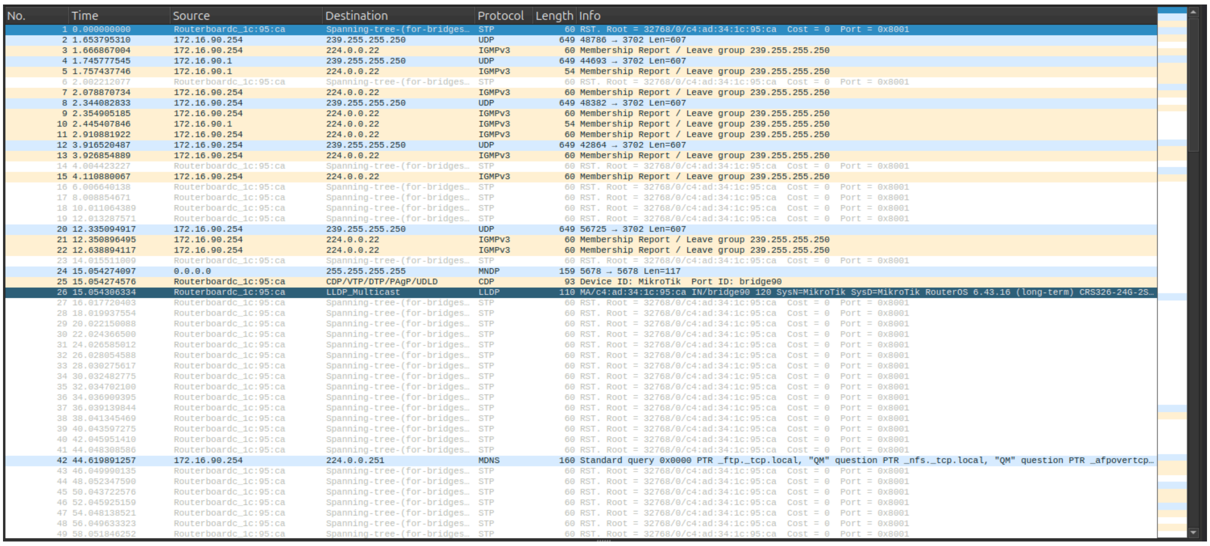
Annex 3.4 - Experiment 2: Pinging 172.16.90.255 broadcast in tux92



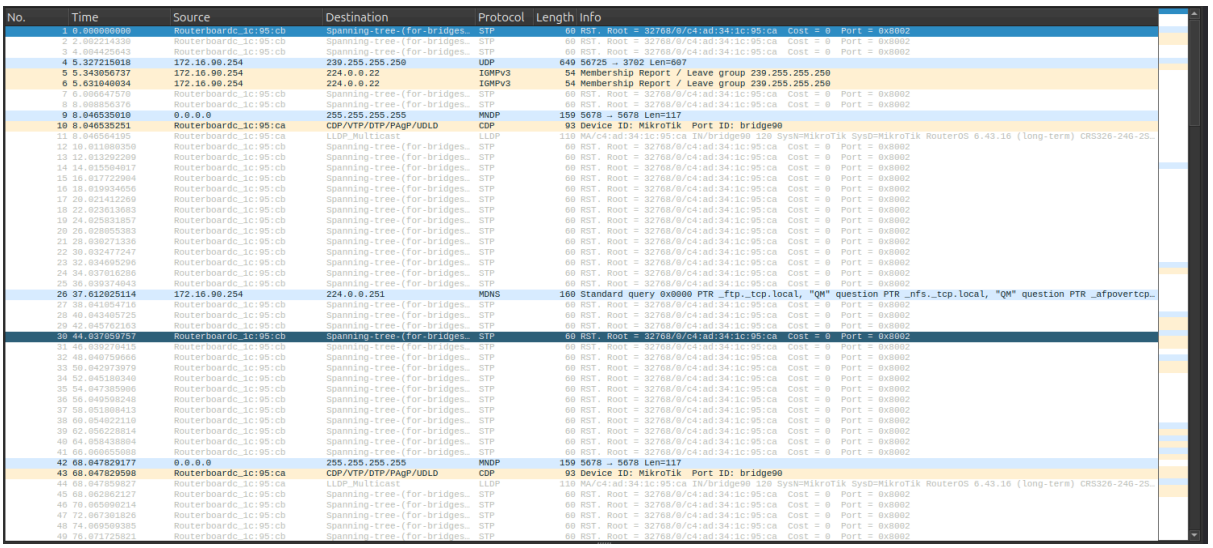
Annex 3.5 - Experiment 2: Pinging 172.16.91.255 broadcast in tux92



Annex 3.6 - Experiment 2: Pinging 172.16.91.255 broadcast in tux93

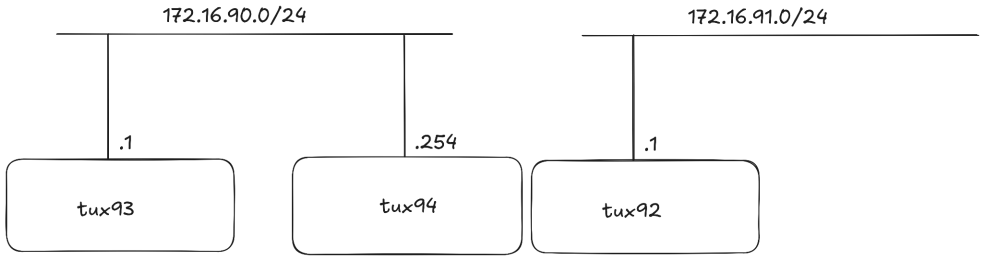


Annex 3.7 - Experiment 2: Pinging 172.16.91.255 broadcast in tux94

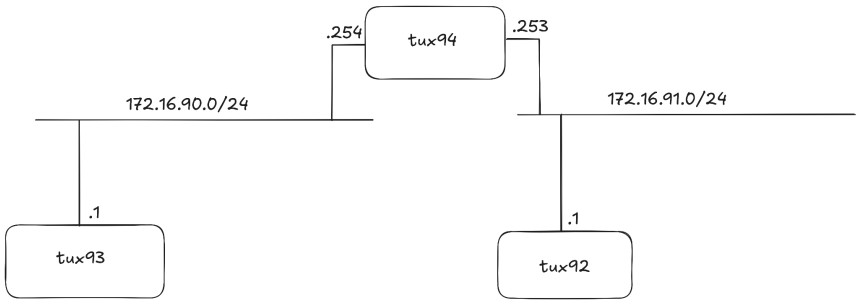


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
2	2.002214330	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
3	4.004425643	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
4	5.327215018	172.16.90.254	239.255.255.250	UDP	649	56725 → 3702 Len=607
5	5.343056737	172.16.90.254	224.0.0.22	IGMPv3	54	Membership Report / Leave group 239.255.255.250
6	5.631040034	172.16.90.254	224.0.0.22	IGMPv3	54	Membership Report / Leave group 239.255.255.250
7	6.006647570	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
8	8.008655070	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
9	8.046535010	0.0.0.0	255.255.255.255	MNDP	159	5678 → 5678 Len=117
10	8.046535251	Routerboardc_1c:95:ca	CDP/VTP/DTP/PAgP/UDLD	CDP	93	Device ID: MikroTik Port ID: bridge90
11	8.046564195	Routerboardc_1c:95:ca	LLDP_Multicast	LLDP	110	MA/c4:ad:34:1c:95:ca IN/bridge90 120 SysN=MikroTik SysD=MikroTik RouterOS 6.43.16 (long-term) CRS326-24G-2S…
12	10.011080350	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
13	12.013202209	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
14	14.015559417	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
15	16.017722904	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
16	18.019934656	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
17	20.021412269	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
18	22.023613683	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
19	24.025831857	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
20	26.028055383	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
21	28.030271336	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
22	30.032477247	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
23	32.034695296	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
24	34.037016286	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
25	36.039374943	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
26	37.612025114	172.16.90.254	224.0.0.251	MDNS	160	Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR _nfs._tcp.local, "QM" question PTR _afpovertcp…
27	38.041054716	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
28	40.043405725	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
29	42.045762163	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
30	44.037059757	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
31	46.039270415	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
32	48.040759666	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
33	50.042973979	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
34	52.045189340	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
35	54.047385906	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
36	56.049598248	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
37	58.051808413	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
38	60.054022110	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
39	62.056228814	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
40	64.058438804	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
41	66.060655008	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
42	68.047829177	0.0.0.0	255.255.255.255	MNDP	159	5678 → 5678 Len=117
43	68.047829598	Routerboardc_1c:95:ca	CDP/VTP/DTP/PAgP/UDLD	CDP	93	Device ID: MikroTik Port ID: bridge90
44	68.047859827	Routerboardc_1c:95:ca	LLDP_Multicast	LLDP	110	MA/c4:ad:34:1c:95:ca IN/bridge90 120 SysN=MikroTik SysD=MikroTik RouterOS 6.43.16 (long-term) CRS326-24G-2S…
45	68.062862127	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
46	70.065090214	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
47	72.067301826	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
48	74.069509385	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002
49	76.071725821	Routerboardc_1c:95:cb	Spanning-tree-(for-bridges…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x8002

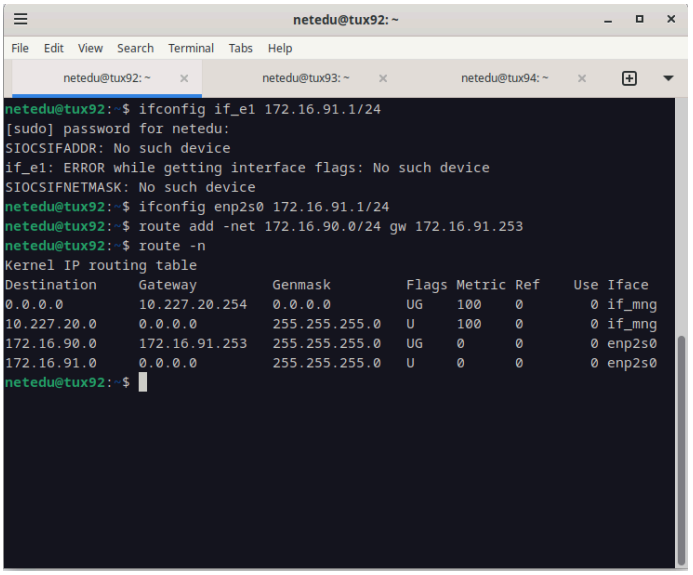
Annex 3.8 - Experiment 2: Network



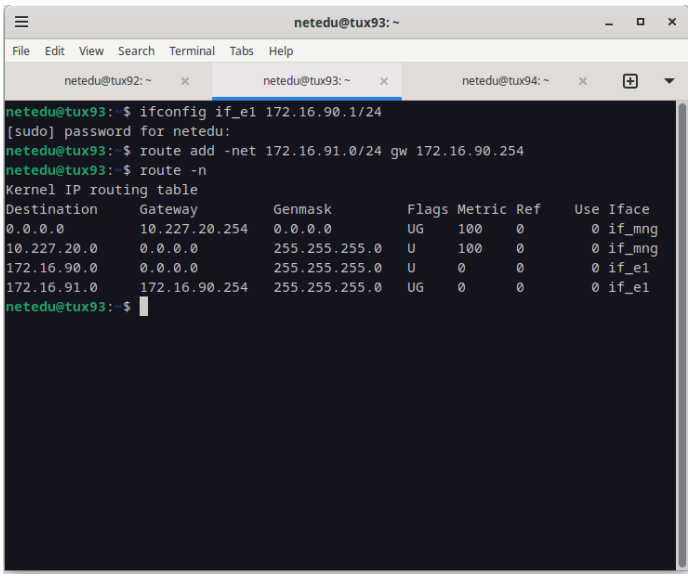
Annex 4.1 - Experiment 3: Network



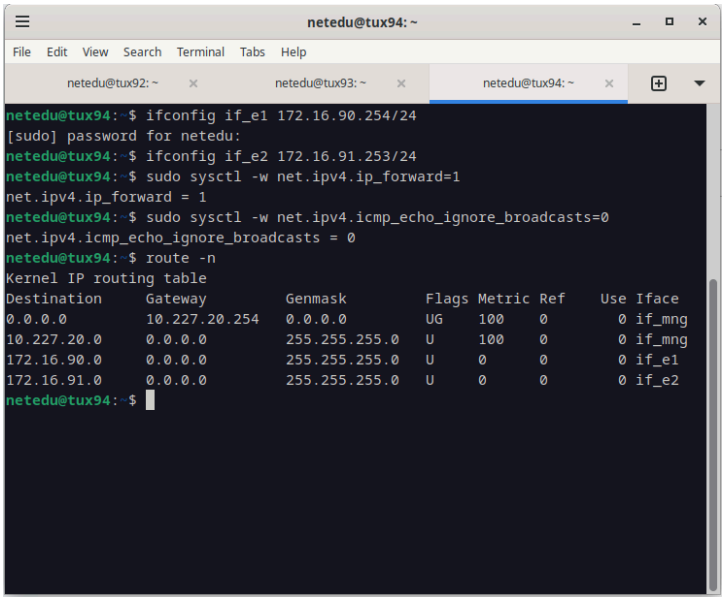
Annex 4.2 - Experiment 3: Routes on tux92



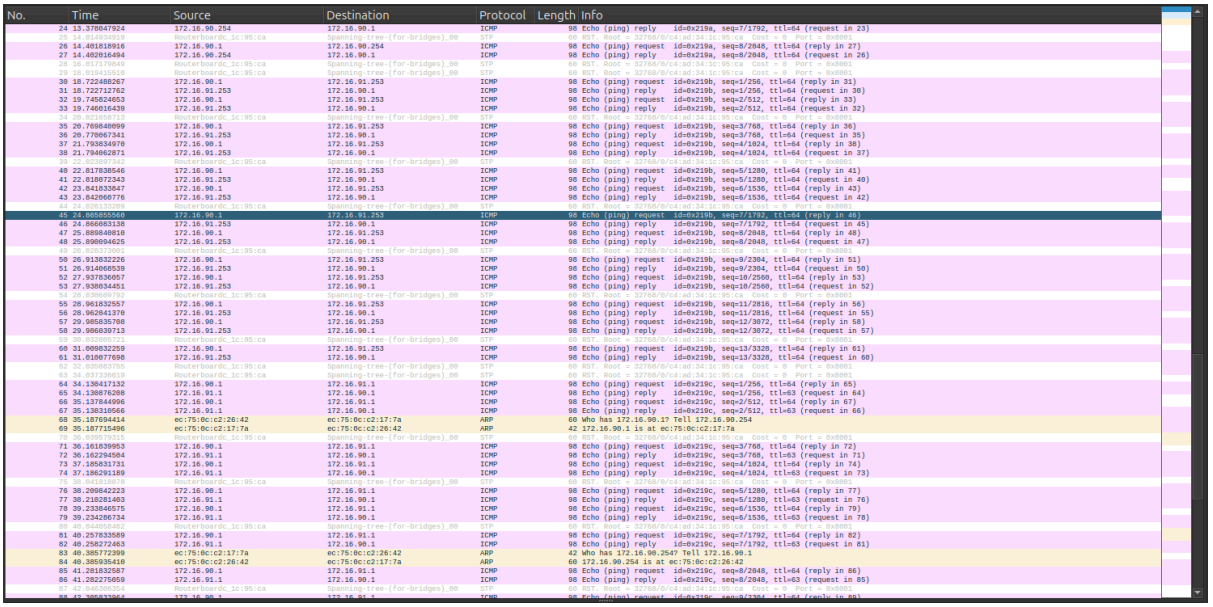
Annex 4.3 - Experiment 3: Routes on tux93



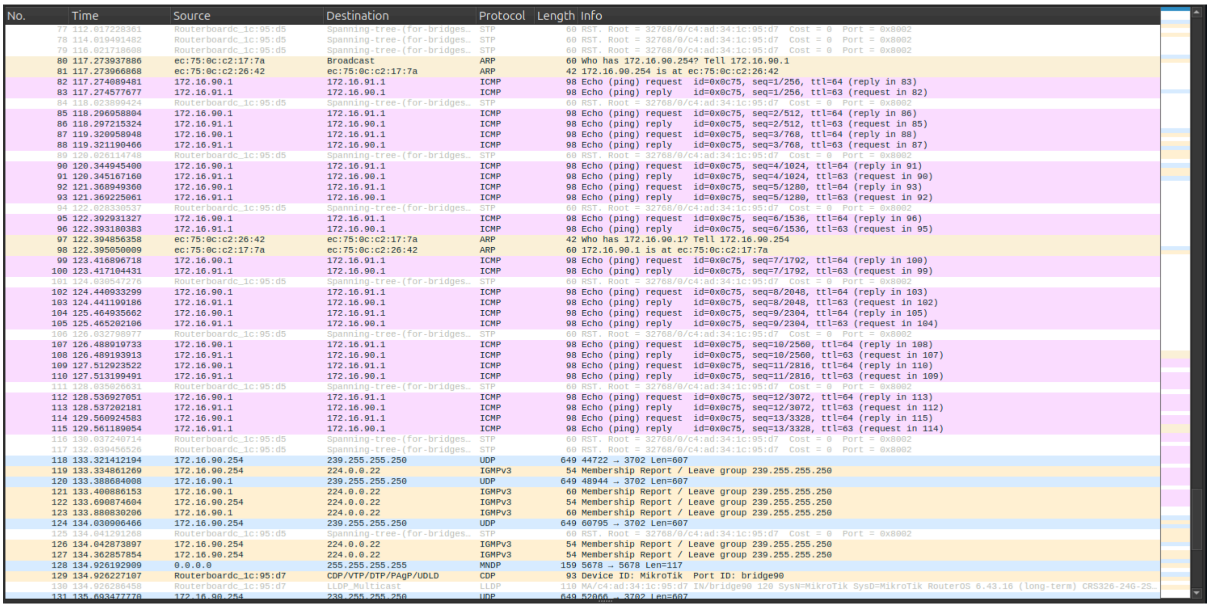
Annex 4.4 - Experiment 3: Routes on tux94



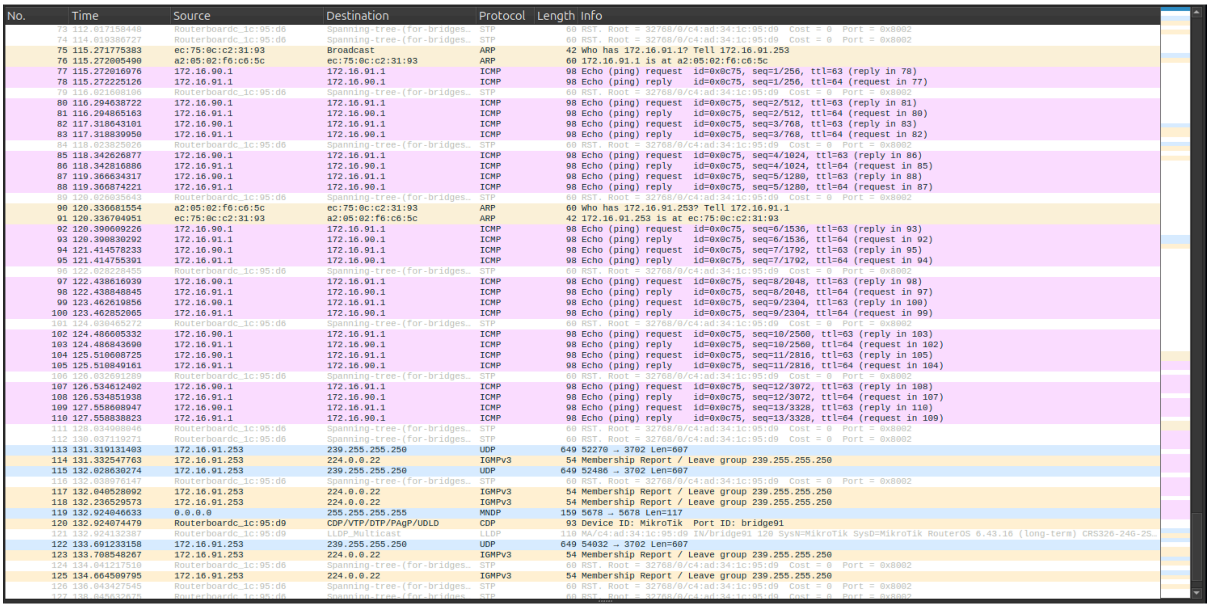
Annex 4.5 - Experiment 3: Tux93 pinging all network interfaces



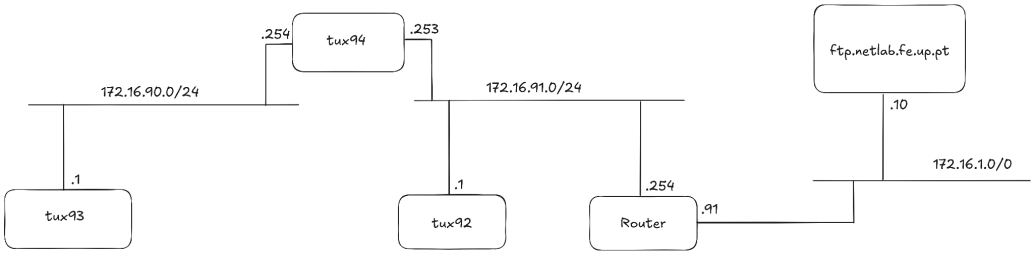
Annex 4.6 - Experiment 3: Tux94 (if_e1) log when tux93 pings tux92 (post ARP table clean)



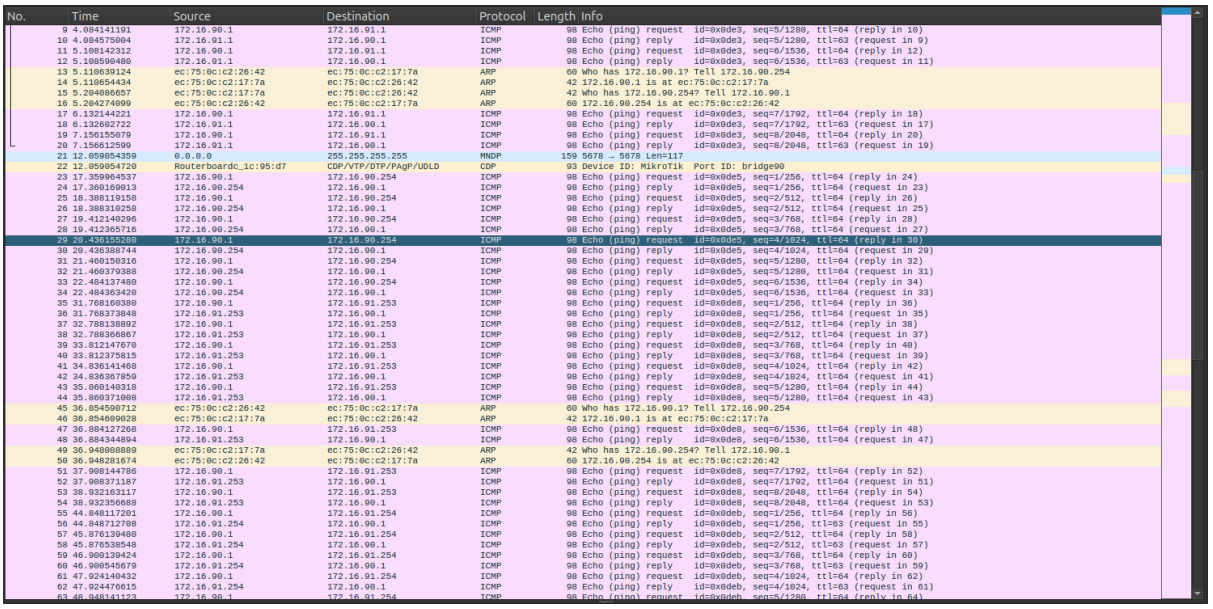
Annex 4.7 - Experiment 3: Tux94 (if_e2) log when tux93 pings tux92 (post ARP table clean)



Annex 5.1 - Experiment 4: Network

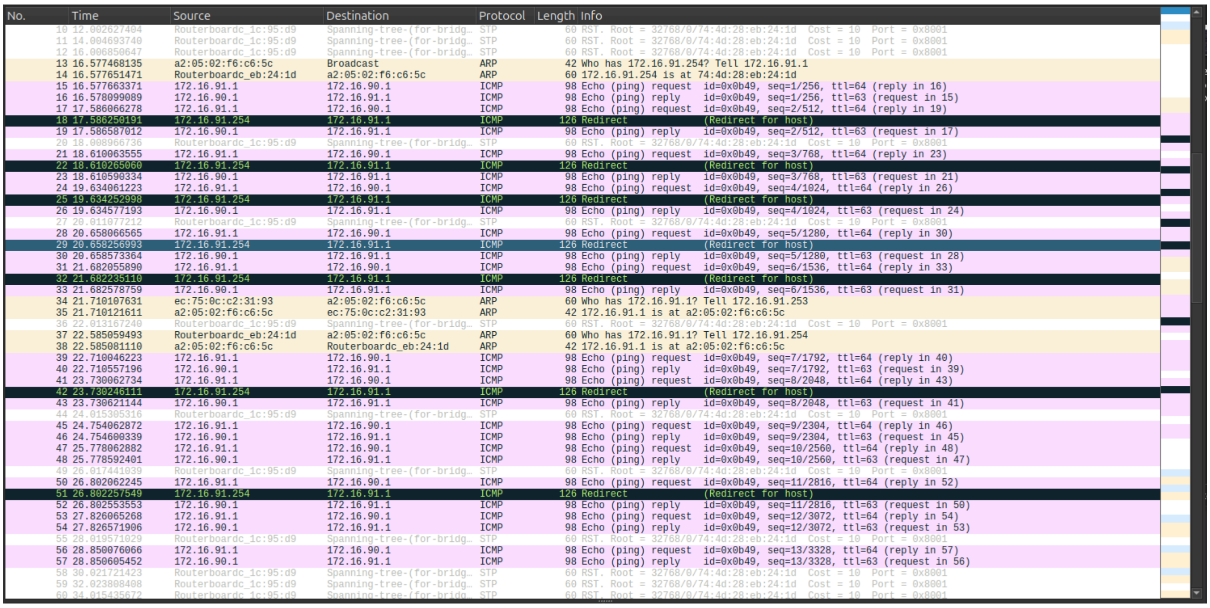


Annex 5.2 - Experiment 4: Tux93 pinging all interfaces

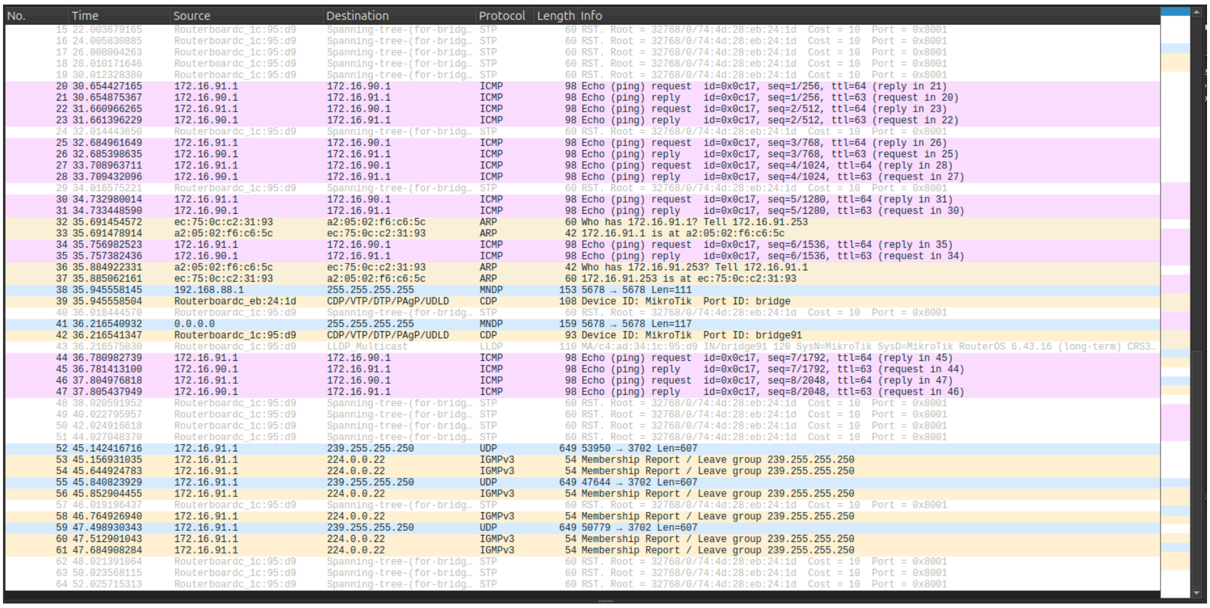


No.	Time	Source	Destination	Protocol	Length	Info
9	4.08414119	172.16.90.1	172.16.91.1	ICMP	98	Echo (ping) request id=0x0de3, seq=5/1280, ttl=64 (reply in 10)
10	4.084575004	172.16.91.1	172.16.90.1	ICMP	98	Echo (ping) reply id=0x0de3, seq=5/1280, ttl=63 (request in 9)
11	5.108142312	172.16.90.1	172.16.91.1	ICMP	98	Echo (ping) request id=0x0de3, seq=6/1536, ttl=64 (reply in 12)
12	5.108590480	172.16.91.1	172.16.90.1	ICMP	98	Echo (ping) reply id=0x0de3, seq=6/1536, ttl=63 (request in 11)
13	5.110639124	ec:75:0c:c2:26:42	ec:75:0c:c2:17:7a	ARP	60	Who has 172.16.90.1? Tell 172.16.90.254
14	5.110654434	ec:75:0c:c2:17:7a	ec:75:0c:c2:26:42	ARP	42	172.16.90.1 is at ec:75:0c:c2:17:7a
15	5.204086657	ec:75:0c:c2:17:7a	ec:75:0c:c2:26:42	ARP	42	Who has 172.16.90.254? Tell 172.16.90.1
16	5.204274090	ec:75:0c:c2:26:42	ec:75:0c:c2:17:7a	ARP	60	172.16.90.254 is at ec:75:0c:c2:26:42
17	6.132144221	172.16.90.1	172.16.91.1	ICMP	98	Echo (ping) request id=0x0de3, seq=7/1792, ttl=64 (reply in 18)
18	6.132602722	172.16.91.1	172.16.90.1	ICMP	98	Echo (ping) reply id=0x0de3, seq=7/1792, ttl=63 (request in 17)
19	7.156155079	172.16.90.1	172.16.91.1	ICMP	98	Echo (ping) request id=0x0de3, seq=8/2048, ttl=64 (reply in 20)
20	7.156612599	172.16.91.1	172.16.90.1	ICMP	98	Echo (ping) reply id=0x0de3, seq=8/2048, ttl=63 (request in 19)
21	12.059054359	0.0.0.0	255.255.255.255	MNDP	159	5678 → 5678 Len=117
22	12.059054720	Routerboardc_1c:95:d7	CDP/VTP/DTP/PAgP/UDLD	CDP	93	Device ID: MikroTik Port ID: bridge90
23	17.359964537	172.16.90.1	172.16.90.254	ICMP	98	Echo (ping) request id=0x0de5, seq=1/256, ttl=64 (reply in 24)
24	17.360160013	172.16.90.254	172.16.90.1	ICMP	98	Echo (ping) reply id=0x0de5, seq=1/256, ttl=64 (request in 23)
25	18.388119158	172.16.90.1	172.16.90.254	ICMP	98	Echo (ping) request id=0x0de5, seq=2/512, ttl=64 (reply in 26)
26	18.388310258	172.16.90.254	172.16.90.1	ICMP	98	Echo (ping) reply id=0x0de5, seq=2/512, ttl=64 (request in 25)
27	19.412140296	172.16.90.1	172.16.90.254	ICMP	98	Echo (ping) request id=0x0de5, seq=3/768, ttl=64 (reply in 28)
28	19.412365716	172.16.90.254	172.16.90.1	ICMP	98	Echo (ping) reply id=0x0de5, seq=3/768, ttl=64 (request in 27)
29	20.436155280	172.16.90.1	172.16.90.254	ICMP	98	Echo (ping) request id=0x0de5, seq=4/1024, ttl=64 (reply in 30)
30	20.436388744	172.16.90.254	172.16.90.1	ICMP	98	Echo (ping) reply id=0x0de5, seq=4/1024, ttl=64 (request in 29)
31	21.460150316	172.16.90.1	172.16.90.254	ICMP	98	Echo (ping) request id=0x0de5, seq=5/1280, ttl=64 (reply in 32)
32	21.460379388	172.16.90.254	172.16.90.1	ICMP	98	Echo (ping) reply id=0x0de5, seq=5/1280, ttl=64 (request in 31)
33	22.484137480	172.16.90.1	172.16.90.254	ICMP	98	Echo (ping) request id=0x0de5, seq=6/1536, ttl=64 (reply in 34)
34	22.484363420	172.16.90.254	172.16.90.1	ICMP	98	Echo (ping) reply id=0x0de5, seq=6/1536, ttl=64 (request in 33)
35	31.768160380	172.16.90.1	172.16.91.253	ICMP	98	Echo (ping) request id=0x0de8, seq=1/256, ttl=64 (reply in 36)
36	31.768373848	172.16.91.253	172.16.90.1	ICMP	98	Echo (ping) reply id=0x0de8, seq=1/256, ttl=64 (request in 35)
37	32.788138892	172.16.90.1	172.16.91.253	ICMP	98	Echo (ping) request id=0x0de8, seq=2/512, ttl=64 (reply in 38)
38	32.788366867	172.16.91.253	172.16.90.1	ICMP	98	Echo (ping) reply id=0x0de8, seq=2/512, ttl=64 (request in 37)
39	33.812147670	172.16.90.1	172.16.91.253	ICMP	98	Echo (ping) request id=0x0de8, seq=3/768, ttl=64 (reply in 40)
40	33.812375815	172.16.91.253	172.16.90.1	ICMP	98	Echo (ping) reply id=0x0de8, seq=3/768, ttl=64 (request in 39)
41	34.836141468	172.16.90.1	172.16.91.253	ICMP	98	Echo (ping) request id=0x0de8, seq=4/1024, ttl=64 (reply in 42)
42	34.836367859	172.16.91.253	172.16.90.1	ICMP	98	Echo (ping) reply id=0x0de8, seq=4/1024, ttl=64 (request in 41)
43	35.860140318	172.16.90.1	172.16.91.253	ICMP	98	Echo (ping) request id=0x0de8, seq=5/1280, ttl=64 (reply in 44)
44	35.860371008	172.16.91.253	172.16.90.1	ICMP	98	Echo (ping) reply id=0x0de8, seq=5/1280, ttl=64 (request in 43)
45	36.854590712	ec:75:0c:c2:26:42	ec:75:0c:c2:17:7a	ARP	60	Who has 172.16.90.1? Tell 172.16.90.254
46	36.854609028	ec:75:0c:c2:17:7a	ec:75:0c:c2:26:42	ARP	42	172.16.90.1 is at ec:75:0c:c2:17:7a
47	36.884127268	172.16.90.1	172.16.91.253	ICMP	98	Echo (ping) request id=0x0de8, seq=6/1536, ttl=64 (reply in 48)
48	36.884344894	172.16.91.253	172.16.90.1	ICMP	98	Echo (ping) reply id=0x0de8, seq=6/1536, ttl=64 (request in 47)
49	36.948088889	ec:75:0c:c2:17:7a	ec:75:0c:c2:26:42	ARP	42	Who has 172.16.90.254? Tell 172.16.90.1
50	36.948281674	ec:75:0c:c2:26:42	ec:75:0c:c2:17:7a	ARP	60	172.16.90.254 is at ec:75:0c:c2:26:42
51	37.908144786	172.16.90.1	172.16.91.253	ICMP	98	Echo (ping) request id=0x0de8, seq=7/1792, ttl=64 (reply in 52)
52	37.908371187	172.16.91.253	172.16.90.1	ICMP	98	Echo (ping) reply id=0x0de8, seq=7/1792, ttl=64 (request in 51)
53	38.932163117	172.16.90.1	172.16.91.253	ICMP	98	Echo (ping) request id=0x0de8, seq=8/2048, ttl=64 (reply in 54)
54	38.932356688	172.16.91.253	172.16.90.1	ICMP	98	Echo (ping) reply id=0x0de8, seq=8/2048, ttl=64 (request in 53)
55	44.848117201	172.16.90.1	172.16.91.254	ICMP	98	Echo (ping) request id=0x0deb, seq=1/256, ttl=64 (reply in 56)
56	44.848712708	172.16.91.254	172.16.90.1	ICMP	98	Echo (ping) reply id=0x0deb, seq=1/256, ttl=63 (request in 55)
57	45.876139480	172.16.90.1	172.16.91.254	ICMP	98	Echo (ping) request id=0x0deb, seq=2/512, ttl=64 (reply in 58)
58	45.876538548	172.16.91.254	172.16.90.1	ICMP	98	Echo (ping) reply id=0x0deb, seq=2/512, ttl=63 (request in 57)
59	46.900139424	172.16.90.1	172.16.91.254	ICMP	98	Echo (ping) request id=0x0deb, seq=3/768, ttl=64 (reply in 60)
60	46.900545679	172.16.91.254	172.16.90.1	ICMP	98	Echo (ping) reply id=0x0deb, seq=3/768, ttl=63 (request in 59)
61	47.924140432	172.16.90.1	172.16.91.254	ICMP	98	Echo (ping) request id=0x0deb, seq=4/1024, ttl=64 (reply in 62)
62	47.924476615	172.16.91.254	172.16.90.1	ICMP	98	Echo (ping) reply id=0x0deb, seq=4/1024, ttl=63 (request in 61)
63	48.948141123	172.16.90.1	172.16.91.254	ICMP	98	Echo (ping) request id=0x0deb, seq=5/1280, ttl=64 (reply in 64)

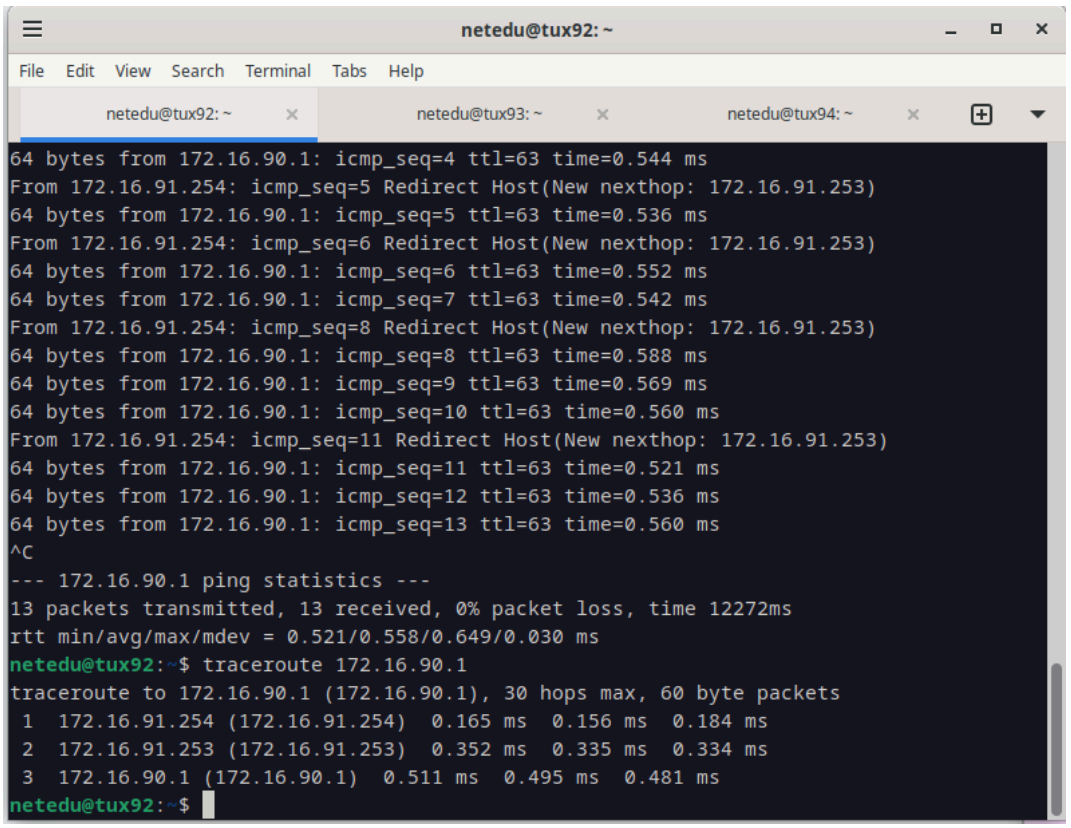
Annex 5.3 - Experiment 4: Tux92 pinging tux93 using router as gateway and no redirect



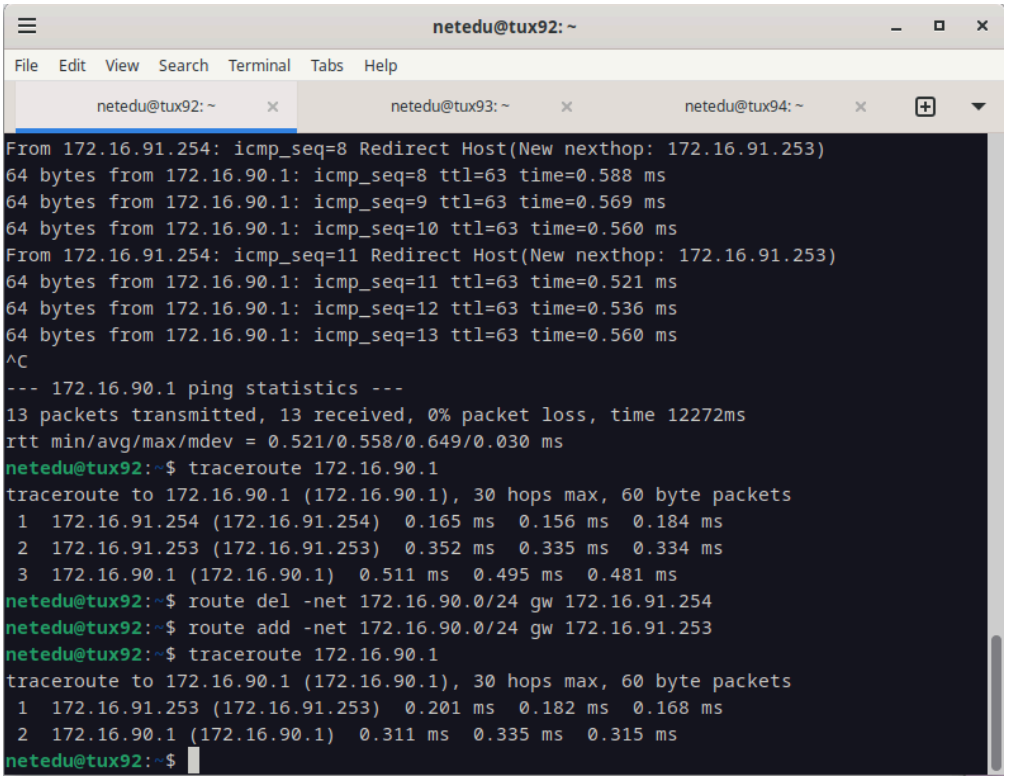
Annex 5.4 - Experiment 4: Tux92 pinging tux93 using tux94 as gateway and redirects



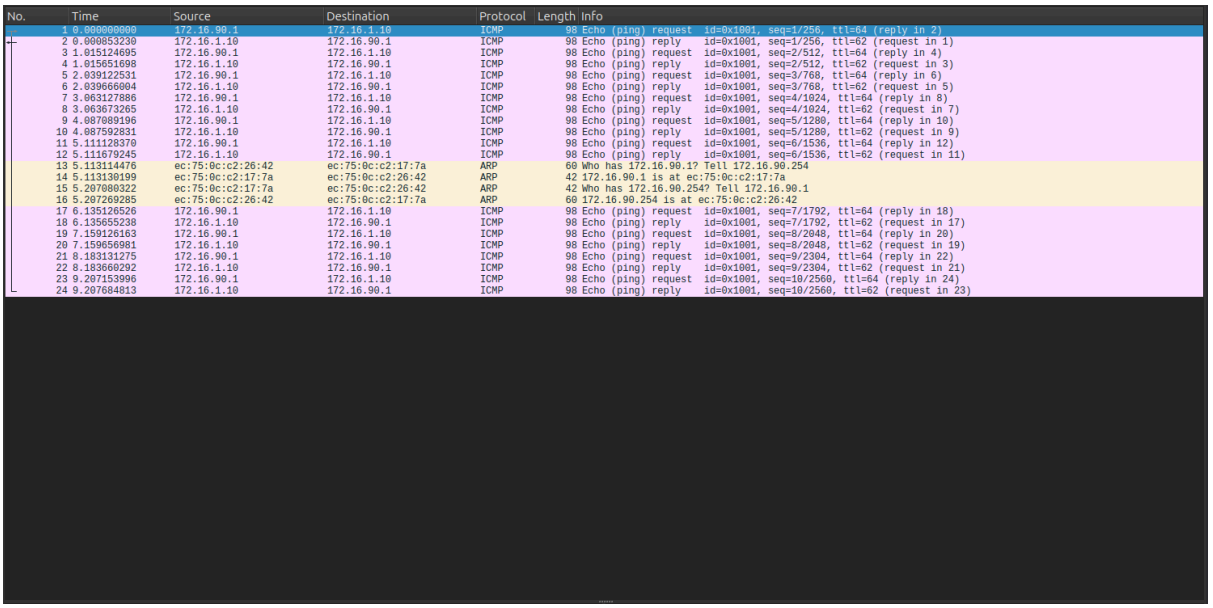
Annex 5.5 - Experiment 4: Traceroute from tux92 to tux93 using router as gateway



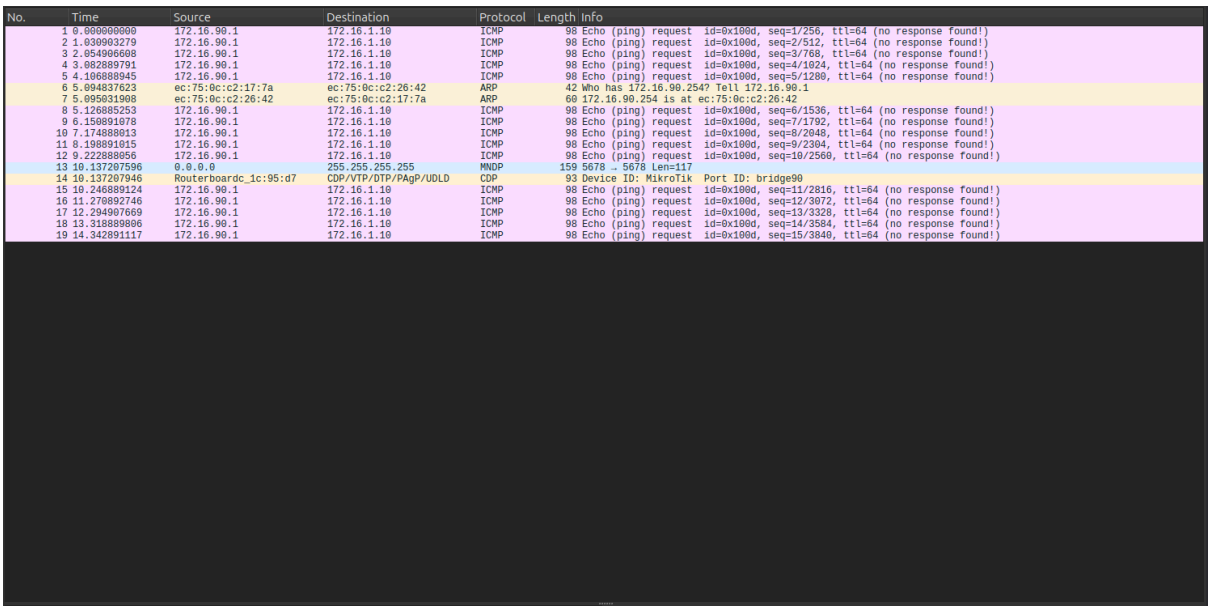
Annex 5.6 - Experiment 4: Traceroute from tux92 to tux93 using tux94 as gateway



Annex 5.7 - Experiment 4: Tux93 pinging Netlab FTP Server with NAT on

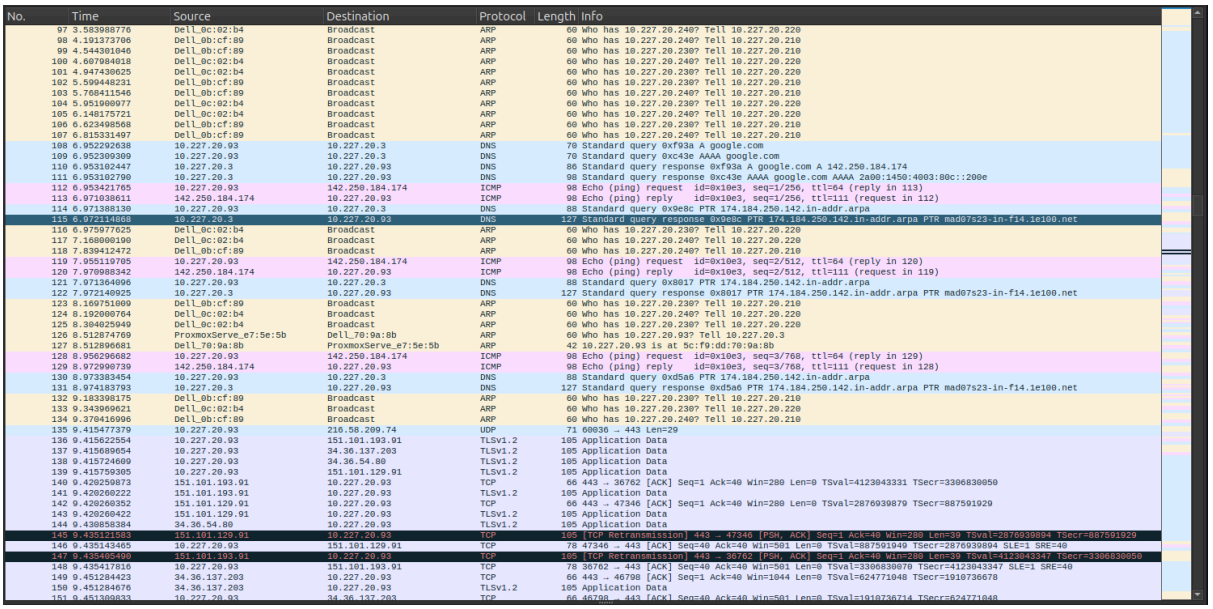


Annex 5.8 - Experiment 4: Tux93 pinging Netlab FTP Server with NAT off



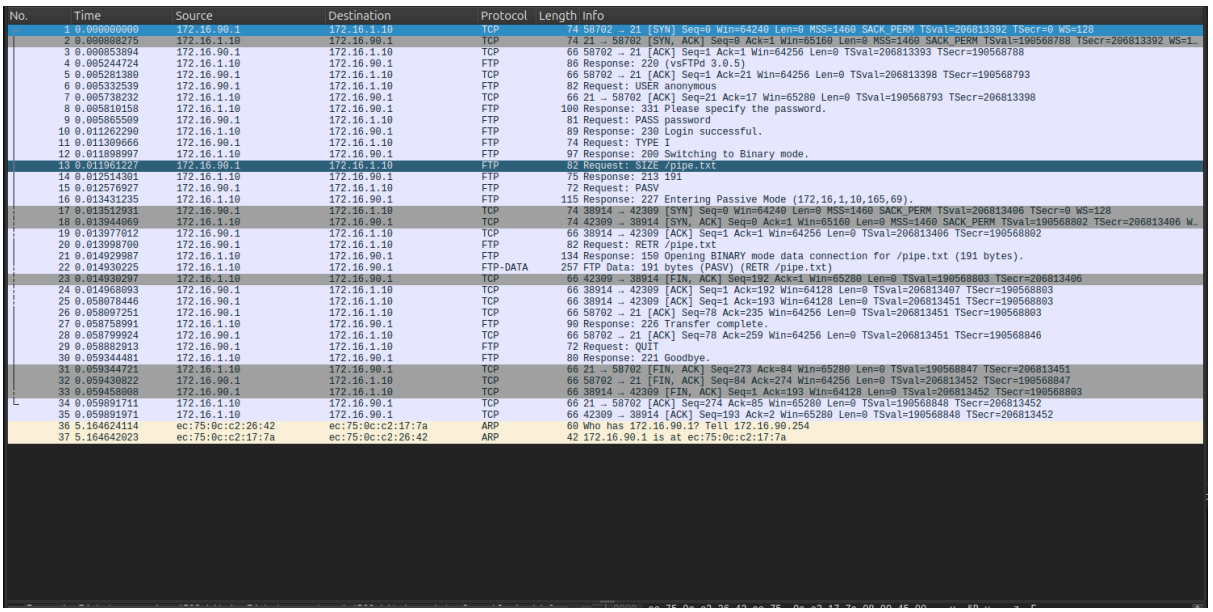
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.16.90.1	172.16.1.10	ICMP	98	Echo (ping) request id=0x100d, seq=1/256, ttl=64 (no response found!)
2	1.030903279	172.16.90.1	172.16.1.10	ICMP	98	Echo (ping) request id=0x100d, seq=2/512, ttl=64 (no response found!)
3	2.054906608	172.16.90.1	172.16.1.10	ICMP	98	Echo (ping) request id=0x100d, seq=3/768, ttl=64 (no response found!)
4	3.082889791	172.16.90.1	172.16.1.10	ICMP	98	Echo (ping) request id=0x100d, seq=4/1024, ttl=64 (no response found!)
5	4.106888945	172.16.90.1	172.16.1.10	ICMP	98	Echo (ping) request id=0x100d, seq=5/1280, ttl=64 (no response found!)
6	5.094837623	ec:75:0c:c2:17:7a	ec:75:0c:c2:26:42	ARP	42	Who has 172.16.90.254? Tell 172.16.90.1
7	5.095031908	ec:75:0c:c2:26:42	ec:75:0c:c2:17:7a	ARP	60	172.16.90.254 is at ec:75:0c:c2:26:42
8	5.126885253	172.16.90.1	172.16.1.10	ICMP	98	Echo (ping) request id=0x100d, seq=6/1536, ttl=64 (no response found!)
9	6.150891078	172.16.90.1	172.16.1.10	ICMP	98	Echo (ping) request id=0x100d, seq=7/1792, ttl=64 (no response found!)
10	7.174888013	172.16.90.1	172.16.1.10	ICMP	98	Echo (ping) request id=0x100d, seq=8/2048, ttl=64 (no response found!)
11	8.198891015	172.16.90.1	172.16.1.10	ICMP	98	Echo (ping) request id=0x100d, seq=9/2304, ttl=64 (no response found!)
12	9.222888056	172.16.90.1	172.16.1.10	ICMP	98	Echo (ping) request id=0x100d, seq=10/2560, ttl=64 (no response found!)
13	10.137207596	0.0.0.0	255.255.255.255	MNDP	159	5678 → 5678 Len=117
14	10.137207946	Routerboardc_1c:95:d7	CDP/VTP/DTP/PAgP/UDLD	CDP	93	Device ID: MikroTik Port ID: bridge90
15	10.246889124	172.16.90.1	172.16.1.10	ICMP	98	Echo (ping) request id=0x100d, seq=11/2816, ttl=64 (no response found!)
16	11.270892746	172.16.90.1	172.16.1.10	ICMP	98	Echo (ping) request id=0x100d, seq=12/3072, ttl=64 (no response found!)
17	12.294907669	172.16.90.1	172.16.1.10	ICMP	98	Echo (ping) request id=0x100d, seq=13/3328, ttl=64 (no response found!)
18	13.318889806	172.16.90.1	172.16.1.10	ICMP	98	Echo (ping) request id=0x100d, seq=14/3584, ttl=64 (no response found!)
19	14.342891117	172.16.90.1	172.16.1.10	ICMP	98	Echo (ping) request id=0x100d, seq=15/3840, ttl=64 (no response found!)

Annex 6.1 - Experiment 5: Tux93 pinging google.com



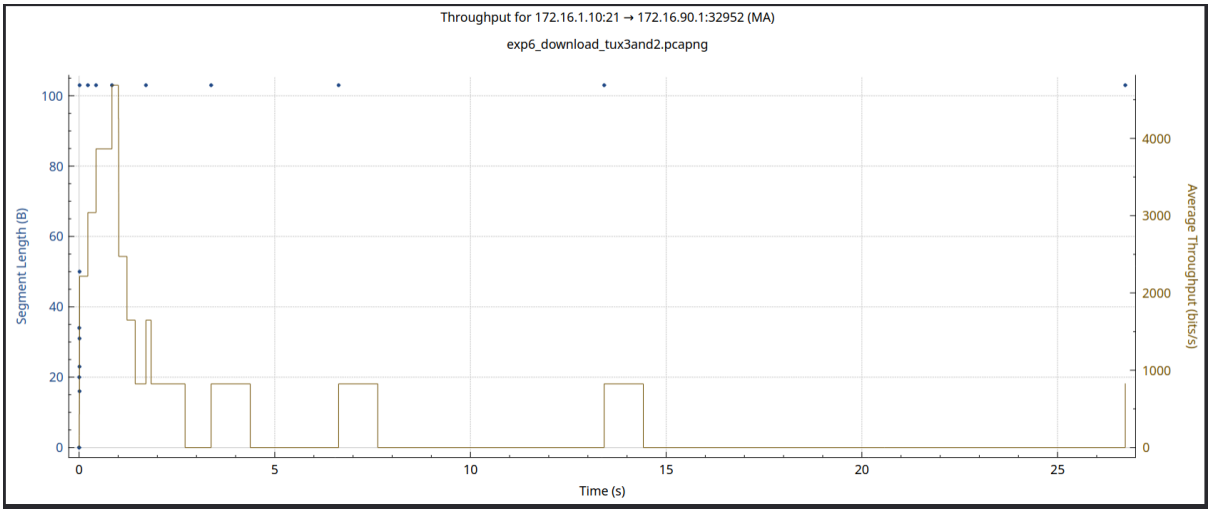
No.	Time	Source	Destination	Protocol	Length	Info
97	3.583988776	Dell_0c:02:b4	Broadcast	ARP	60	Who has 10.227.20.240? Tell 10.227.20.220
98	4.191373706	Dell_0b:cf:89	Broadcast	ARP	60	Who has 10.227.20.240? Tell 10.227.20.210
99	4.544301846	Dell_0b:cf:89	Broadcast	ARP	60	Who has 10.227.20.230? Tell 10.227.20.210
100	4.607984018	Dell_0c:02:b4	Broadcast	ARP	60	Who has 10.227.20.240? Tell 10.227.20.220
101	4.947430625	Dell_0c:02:b4	Broadcast	ARP	60	Who has 10.227.20.230? Tell 10.227.20.220
102	5.599448231	Dell_0b:cf:89	Broadcast	ARP	60	Who has 10.227.20.230? Tell 10.227.20.210
103	5.768411546	Dell_0b:cf:89	Broadcast	ARP	60	Who has 10.227.20.240? Tell 10.227.20.210
104	5.951090977	Dell_0c:02:b4	Broadcast	ARP	60	Who has 10.227.20.230? Tell 10.227.20.220
105	6.148175721	Dell_0c:02:b4	Broadcast	ARP	60	Who has 10.227.20.240? Tell 10.227.20.220
106	6.623498568	Dell_0b:cf:89	Broadcast	ARP	60	Who has 10.227.20.230? Tell 10.227.20.210
107	6.815331497	Dell_0b:cf:89	Broadcast	ARP	60	Who has 10.227.20.240? Tell 10.227.20.210
108	6.952292638	10.227.20.93	10.227.20.3	DNS	70	Standard query 0xf93a A google.com
109	6.952309309	10.227.20.93	10.227.20.3	DNS	70	Standard query 0xc43e AAAA google.com
110	6.953102447	10.227.20.3	10.227.20.93	DNS	86	Standard query response 0xf93a A google.com A 142.250.184.174
111	6.953102769	10.227.20.3	10.227.20.93	DNS	98	Standard query response 0xc43e AAAA google.com AAAA 2a00:1450:4003:80c::200e
112	6.953421765	10.227.20.93	142.250.184.174	ICMP	98	Echo (ping) request id=0x10e3, seq=1/256, ttl=64 (reply in 113)
113	6.971038611	142.250.184.174	10.227.20.93	ICMP	98	Echo (ping) reply id=0x10e3, seq=1/256, ttl=111 (request in 112)
114	6.971388130	10.227.20.93	10.227.20.3	DNS	88	Standard query 0x9e8c PTR 174.184.250.142.in-addr.arpa
115	6.972114660	10.227.20.3	10.227.20.93	DNS	127	Standard query response 0x9e8c PTR 174.184.250.142.in-addr.arpa PTR mad07s23-in-f14.1e100.net
116	6.975977625	Dell_0c:02:b4	Broadcast	ARP	60	Who has 10.227.20.230? Tell 10.227.20.220
117	7.168000190	Dell_0c:02:b4	Broadcast	ARP	60	Who has 10.227.20.240? Tell 10.227.20.220
118	7.839412472	Dell_0b:cf:89	Broadcast	ARP	60	Who has 10.227.20.240? Tell 10.227.20.210
119	7.955119705	10.227.20.93	142.250.184.174	ICMP	98	Echo (ping) request id=0x10e3, seq=2/512, ttl=64 (reply in 120)
120	7.970988342	142.250.184.174	10.227.20.93	ICMP	98	Echo (ping) reply id=0x10e3, seq=2/512, ttl=111 (request in 119)
121	7.971364096	10.227.20.93	10.227.20.3	DNS	88	Standard query 0x8017 PTR 174.184.250.142.in-addr.arpa
122	7.972140925	10.227.20.3	10.227.20.93	DNS	127	Standard query response 0x8017 PTR 174.184.250.142.in-addr.arpa PTR mad07s23-in-f14.1e100.net
123	8.169751009	Dell_0b:cf:89	Broadcast	ARP	60	Who has 10.227.20.230? Tell 10.227.20.210
124	8.192000764	Dell_0c:02:b4	Broadcast	ARP	60	Who has 10.227.20.240? Tell 10.227.20.220
125	8.384025949	Dell_0c:02:b4	Broadcast	ARP	60	Who has 10.227.20.230? Tell 10.227.20.220
126	8.512874769	ProxmoxServe_e7:5e:5b	Dell_70:9a:8b	ARP	60	Who has 10.227.20.93? Tell 10.227.20.3
127	8.512896681	Dell_70:9a:8b	ProxmoxServe_e7:5e:5b	ARP	42	10.227.20.93 is at 5c:f9:dd:70:9a:8b
128	8.956296682	10.227.20.93	142.250.184.174	ICMP	98	Echo (ping) request id=0x10e3, seq=3/768, ttl=64 (reply in 129)
129	8.972990739	142.250.184.174	10.227.20.93	ICMP	98	Echo (ping) reply id=0x10e3, seq=3/768, ttl=111 (request in 128)
130	8.973383454	10.227.20.93	10.227.20.3	DNS	88	Standard query 0xd5a6 PTR 174.184.250.142.in-addr.arpa
131	8.974183793	10.227.20.3	10.227.20.93	DNS	127	Standard query response 0xd5a6 PTR 174.184.250.142.in-addr.arpa PTR mad07s23-in-f14.1e100.net
132	9.183398175	Dell_0b:cf:89	Broadcast	ARP	60	Who has 10.227.20.230? Tell 10.227.20.210
133	9.343969621	Dell_0c:02:b4	Broadcast	ARP	60	Who has 10.227.20.230? Tell 10.227.20.220
134	9.370416996	Dell_0b:cf:89	Broadcast	ARP	60	Who has 10.227.20.240? Tell 10.227.20.210
135	9.415477379	10.227.20.93	216.58.209.74	UDP	71	60036 → 443 Len=29
136	9.415622554	10.227.20.93	151.101.193.91	TLSv1.2	105	Application Data
137	9.415689654	10.227.20.93	34.36.137.203	TLSv1.2	105	Application Data
138	9.415724609	10.227.20.93	34.36.54.80	TLSv1.2	105	Application Data
139	9.415759305	10.227.20.93	151.101.129.91	TLSv1.2	105	Application Data
140	9.420259873	151.101.193.91	10.227.20.93	TCP	66	443 → 36762 [ACK] Seq=1 Ack=40 Win=280 Len=0 TSval=4123043331 TSecr=3306830050
141	9.420260222	151.101.193.91	10.227.20.93	TLSv1.2	105	Application Data
142	9.420260352	151.101.129.91	10.227.20.93	TCP	66	443 → 47346 [ACK] Seq=1 Ack=40 Win=280 Len=0 TSval=2876939879 TSecr=887591929
143	9.420260422	151.101.129.91	10.227.20.93	TLSv1.2	105	Application Data
144	9.430858384	34.36.54.80	10.227.20.93	TLSv1.2	105	Application Data
145	9.435121583	151.101.129.91	10.227.20.93	TCP	105	[TCP Retransmission] 443 → 47346 [PSH, ACK] Seq=1 Ack=40 Win=280 Len=39 TSval=2876939894 TSecr=887591929
146	9.435143465	10.227.20.93	151.101.129.91	TCP	78	47346 → 443 [ACK] Seq=40 Ack=40 Win=501 Len=0 TSval=887591949 TSecr=2876939894 SLE=1 SRE=40
147	9.435405490	151.101.193.91	10.227.20.93	TCP	105	[TCP Retransmission] 443 → 36762 [PSH, ACK] Seq=1 Ack=40 Win=280 Len=39 TSval=4123043347 TSecr=3306830050
148	9.435417816	10.227.20.93	151.101.193.91	TCP	78	36762 → 443 [ACK] Seq=40 Ack=40 Win=501 Len=0 TSval=3306830070 TSecr=4123043347 SLE=1 SRE=40
149	9.451284423	34.36.137.203	10.227.20.93	TCP	66	443 → 46798 [ACK] Seq=1 Ack=40 Win=1044 Len=0 TSval=624771048 TSecr=1910736678
150	9.451284676	34.36.137.203	10.227.20.93	TLSv1.2	105	Application Data
151	9.451309833	10.227.20.93	34.36.137.203	TCP	66	46798 → 443 [ACK] Seq=40 Ack=40 Win=501 Len=0 TSval=1910736714 TSecr=624771048

Annex 7.1 - Experiment 6: Successful download with all packets

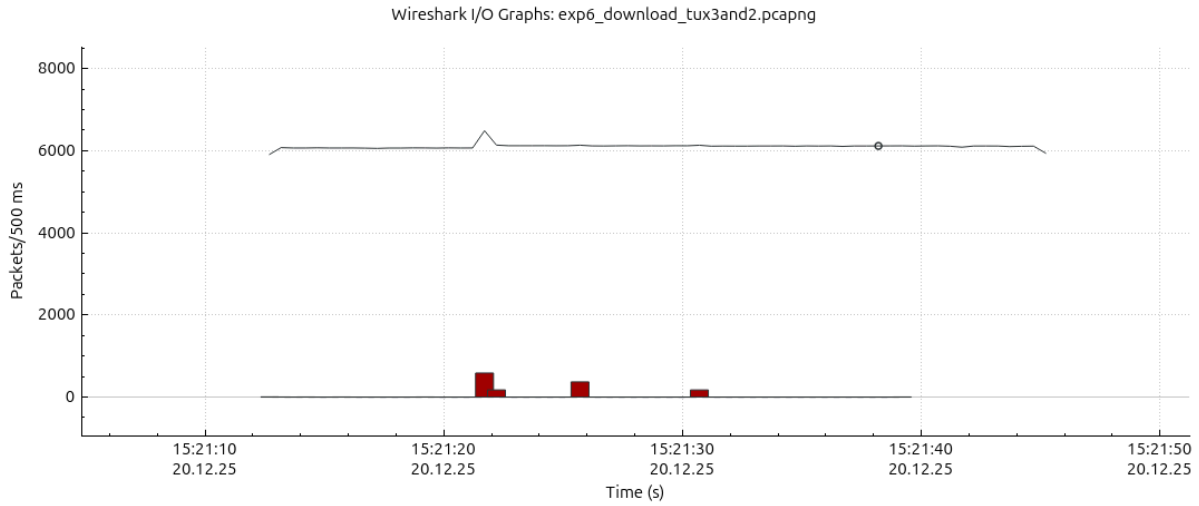


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.16.90.1	172.16.1.10	TCP	74	58702 → 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=206813392 TSecr=0 WS=128
2	0.000808275	172.16.1.10	172.16.90.1	TCP	74	21 → 58702 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=190568788 TSecr=206813392 WS=1…
3	0.000853894	172.16.90.1	172.16.1.10	TCP	66	58702 → 21 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=206813393 TSecr=190568788
4	0.005244724	172.16.1.10	172.16.90.1	FTP	86	Response: 220 (vsFTPd 3.0.5)
5	0.005281380	172.16.90.1	172.16.1.10	TCP	66	58702 → 21 [ACK] Seq=1 Ack=21 Win=64256 Len=0 TSval=206813398 TSecr=190568793
6	0.005332539	172.16.90.1	172.16.1.10	FTP	82	Request: USER anonymous
7	0.005738232	172.16.1.10	172.16.90.1	TCP	66	21 → 58702 [ACK] Seq=21 Ack=17 Win=65280 Len=0 TSval=190568793 TSecr=206813398
8	0.005810158	172.16.1.10	172.16.90.1	FTP	100	Response: 331 Please specify the password.
9	0.005865509	172.16.90.1	172.16.1.10	FTP	81	Request: PASS password
10	0.011262290	172.16.1.10	172.16.90.1	FTP	89	Response: 230 Login successful.
11	0.011309666	172.16.90.1	172.16.1.10	FTP	74	Request: TYPE I
12	0.011898997	172.16.1.10	172.16.90.1	FTP	97	Response: 200 Switching to Binary mode.
13	0.011961227	172.16.90.1	172.16.1.10	FTP	82	Request: SIZE /pipe.txt
14	0.012514301	172.16.1.10	172.16.90.1	FTP	75	Response: 213 191
15	0.012576927	172.16.90.1	172.16.1.10	FTP	72	Request: PASV
16	0.013431235	172.16.1.10	172.16.90.1	FTP	115	Response: 227 Entering Passive Mode (172,16,1,10,165,69).
17	0.013512931	172.16.90.1	172.16.1.10	TCP	74	38914 → 42309 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=206813406 TSecr=0 WS=128
18	0.013944069	172.16.1.10	172.16.90.1	TCP	74	42309 → 38914 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=190568802 TSecr=206813406 W…
19	0.013977012	172.16.90.1	172.16.1.10	TCP	66	38914 → 42309 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=206813406 TSecr=190568802
20	0.013998700	172.16.90.1	172.16.1.10	FTP	82	Request: RETR /pipe.txt
21	0.014929987	172.16.1.10	172.16.90.1	FTP	134	Response: 150 Opening BINARY mode data connection for /pipe.txt (191 bytes).
22	0.014930225	172.16.1.10	172.16.90.1	FTP-DATA	257	FTP Data: 191 bytes (PASV) (RETR /pipe.txt)
23	0.014930297	172.16.1.10	172.16.90.1	TCP	66	42309 → 38914 [FIN, ACK] Seq=192 Ack=1 Win=65280 Len=0 TSval=190568803 TSecr=206813406
24	0.014968093	172.16.90.1	172.16.1.10	TCP	66	38914 → 42309 [ACK] Seq=1 Ack=192 Win=64128 Len=0 TSval=206813407 TSecr=190568803
25	0.058078446	172.16.90.1	172.16.1.10	TCP	66	38914 → 42309 [ACK] Seq=1 Ack=193 Win=64128 Len=0 TSval=206813451 TSecr=190568803
26	0.058097251	172.16.90.1	172.16.1.10	TCP	66	58702 → 21 [ACK] Seq=78 Ack=235 Win=64256 Len=0 TSval=206813451 TSecr=190568803
27	0.058758991	172.16.1.10	172.16.90.1	FTP	90	Response: 226 Transfer complete.
28	0.058799924	172.16.90.1	172.16.1.10	TCP	66	58702 → 21 [ACK] Seq=78 Ack=259 Win=64256 Len=0 TSval=206813451 TSecr=190568846
29	0.058882913	172.16.90.1	172.16.1.10	FTP	72	Request: QUIT
30	0.059344481	172.16.1.10	172.16.90.1	FTP	80	Response: 221 Goodbye.
31	0.059344721	172.16.1.10	172.16.90.1	TCP	66	21 → 58702 [FIN, ACK] Seq=273 Ack=84 Win=65280 Len=0 TSval=190568847 TSecr=206813451
32	0.059430822	172.16.90.1	172.16.1.10	TCP	66	58702 → 21 [FIN, ACK] Seq=84 Ack=274 Win=64256 Len=0 TSval=206813452 TSecr=190568847
33	0.059458008	172.16.90.1	172.16.1.10	TCP	66	38914 → 42309 [FIN, ACK] Seq=1 Ack=193 Win=64128 Len=0 TSval=206813452 TSecr=190568803
34	0.059891711	172.16.1.10	172.16.90.1	TCP	66	21 → 58702 [ACK] Seq=274 Ack=85 Win=65280 Len=0 TSval=190568848 TSecr=206813452
35	0.059891971	172.16.1.10	172.16.90.1	TCP	66	42309 → 38914 [ACK] Seq=193 Ack=2 Win=65280 Len=0 TSval=190568848 TSecr=206813452
36	5.164624114	ec:75:0c:c2:26:42	ec:75:0c:c2:17:7a	ARP	60	Who has 172.16.90.1? Tell 172.16.90.254
37	5.164642023	ec:75:0c:c2:17:7a	ec:75:0c:c2:26:42	ARP	42	172.16.90.1 is at ec:75:0c:c2:17:7a

Annex 7.2 - Experiment 6: Simultaneous download on tux92 and on tux93



Annex 7.3 - Experiment 6: Simultaneous download on tux92 and on tux93



Annex 8.1 - download.c

```C/C++
// main entrypoint for the program

#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <stdbool.h>
#include <ctype.h>
#include <assert.h>

#include "parser.h"
#include "ftp_response.h"

#define FTP_PORT 21
#define MAX_BUF_SIZE 10000

/**
 * Checks if the message was complete
 * @param buf Buffer where the message is stored
 * @param buf_size Buffer size
 * @return true if complete, false otherwise
 */
```

```
bool is_msg_complete(const char* buf, int buf_size) {
    int i = 0;
    bool final_line = false;

    while(i+1 < buf_size) {
        // buffer safety check
        if (i+4 >= buf_size && !final_line)
            return false;

        if (!final_line && (i == 0 || buf[i-1] == '\n')
            && isdigit(buf[i]) && isdigit(buf[i+1]) && isdigit(buf[i+2])
            && buf[i+3] == ' ') {
            // final, skip all until CRLF
            final_line = true;
        }

        if (final_line && buf[i] == '\r' && buf[i+1] == '\n') {
            return true;
        }

        i++;
    }

    return false;
}

/**
 * Reads message pieces and writes them to buf until CRLF is encountered
 * @param socket Socket being read
 * @param buf Buffer where the message is stored
 * @param buf_size Buffer size
 * @return the response code or -1 if something's wrong (e.g. buffer too
small)
 */
int get_message(int socket, char* buf, int buf_size) {
    int total_bytes = 0;

    while (total_bytes < buf_size) {
        long int bytes = recv(socket, buf + total_bytes, buf_size -
total_bytes, 0);
        if (bytes > 0) {
            printf("Bytes read %ld\n", bytes);
        }
        else {
            perror("recv() failed");
            return -1;
        }
```

```
        total_bytes += bytes;

        if (is_msg_complete(buf, total_bytes)) {
            break;
        }
    }

    if (total_bytes >= buf_size)
        return -1;

    buf[total_bytes] = '\0';

    // get message code
    int res = (buf[0] - '0')*100 + (buf[1] - '0')*10 + (buf[2] - '0');
    return res;
}

/**
 * Sends a message via socket
 * @param socket Socket used to send a message
 * @param buf Where message was
 * @param buf_size Buffer size
 * @return 0 if succeeded, -1 otherwise
 */
int send_message(int socket, char* buf, int buf_size) {
    long int bytes = send(socket, buf, buf_size, 0);
    if (bytes > 0) {
        printf("Bytes sent %ld\n", bytes);
    } else {
        perror("send()");
        return -1;
    }

    return 0;
}

/**
 * Parses the 227 message (PASSIVE MODE ON)
 * @param buf Message to be parsed
 * @param ip New ip
 * @return Port for data socket
 */
int parse_ip(const char* buf, char* ip) {
    // get to opening paren and skip it
    while (*buf != '(') {
        buf++;
```

```
}
buf++;

int ip_index = 0;
// copy first octet to the ip address
ip[ip_index] = *buf;
buf++;
ip_index++;
if (isdigit(*buf)) {
    ip[ip_index] = *buf;
    buf++;
    ip_index++;
    if (isdigit(*buf)) {
        ip[ip_index] = *buf;
        buf++;
        ip_index++;
    }
}

assert(*buf == ',');

ip[ip_index] = '.';
buf++;
ip_index++;

// copy second octet to the ip address
ip[ip_index] = *buf;
buf++;
ip_index++;
if (isdigit(*buf)) {
    ip[ip_index] = *buf;
    buf++;
    ip_index++;
    if (isdigit(*buf)) {
        ip[ip_index] = *buf;
        buf++;
        ip_index++;
    }
}
assert(*buf == ',');
ip[ip_index] = '.';
buf++;
ip_index++;
// copy third octet to the ip address
ip[ip_index] = *buf;
buf++;
ip_index++;
```

```
    if (isdigit(*buf)) {
        ip[ip_index] = *buf;
        buf++;
        ip_index++;
        if (isdigit(*buf)) {
            ip[ip_index] = *buf;
            buf++;
            ip_index++;
        }
    }

    assert(*buf == ',');
    ip[ip_index] = '.';
    buf++;
    ip_index++;
    // copy fourth octet to the ip address
    ip[ip_index] = *buf;
    buf++;
    ip_index++;
    if (isdigit(*buf)) {
        ip[ip_index] = *buf;
        buf++;
        ip_index++;
        if (isdigit(*buf)) {
            ip[ip_index] = *buf;
            buf++;
            ip_index++;
        }
    }
    assert(*buf == ',');
    buf++;
    ip[ip_index] = '\0';

    // Now the port
    int port_oct1 = *buf - '0';
    buf++;
    if (isdigit(*buf)) {
        port_oct1 *= 10;
        port_oct1 += *buf - '0';
        buf++;
        if (isdigit(*buf)) {
            port_oct1 *= 10;
            port_oct1 += *buf - '0';
            buf++;
        }
    }
    assert(*buf == ',');
```

```
    buf++;
    // second octet
    int port_oct2 = *buf - '0';
    buf++;
    if (isdigit(*buf)) {
        port_oct2 *= 10;
        port_oct2 += *buf - '0';
        buf++;
        if (isdigit(*buf)) {
            port_oct2 *= 10;
            port_oct2 += *buf - '0';
            buf++;
        }
    }
    assert(*buf == ')');
    printf("oct1 %d oct2 %d\n", port_oct1, port_oct2);

    return port_oct1 * 256 + port_oct2;
}

size_t get_file_size(const char* buf) {
    // skip code and space
    buf += 4;

    size_t size = *buf - '0';
    buf++;

    while (isdigit(*buf)) {
        size *= 10;
        size += *buf - '0';
        buf++;
    }

    return size;
}

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

    if (argc != 2) {
        printf("Usage: ftp://[<user>:<password>@]<host>/<url-path>\n");
        return -1;
    }

    Url url;

    // get information from param
    if (parse(argv[1], &url) != 0) {
```

```
        printf("Error parsing url\n");
        return -1;
    }

    printf( "Host: %s\n"
            "IP Address: %s\n"
            "User: %s\n"
            "Password: %s\n"
            "Resource: %s\n"
            "File: %s\n",

            url.hostname,
            url.host_ip,
            url.user,
            url.password,
            url.resource,
            url.file
    );

    // creating the ctrl socket

    // ctrl_socket
    int sockfd;
    struct sockaddr_in server_addr;
    size_t bytes;

    /*server address handling*/
    bzero((char *) &server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr(url.host_ip);    /*32 bit
Internet address network byte ordered*/
    server_addr.sin_port = htons(FTP_PORT);        /*server TCP port must be
network byte ordered */

    /*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);
    }
```

```
char buf[MAX_BUF_SIZE];
// get welcome message
if (get_message(sockfd, buf, MAX_BUF_SIZE) != WELCOME_CODE) {
    printf("Error: did not receive welcome message.\n");
    return -1;
}
printf("%s\n", buf);

// send USER message for authentication
sprintf(buf, "USER %s\r\n", url.user);
if (send_message(sockfd, buf, strlen(buf)) != 0) {
    printf("Failed sending USER message.\n");
    return -1;
}
// (try to) receive 331 (request for password)
if (get_message(sockfd, buf, MAX_BUF_SIZE) != PSSWRD_REQUEST_CODE) {
    printf("Error: did not receive password request\n");
    return -1;
}
printf("%s\n", buf);

// send PASS message
sprintf(buf, "PASS %s\r\n", url.password);
if (send_message(sockfd, buf, strlen(buf)) != 0) {
    printf("Error: could not send the PASS\n");
    return -1;
}
// receive 230 (login successful)
if (get_message(sockfd, buf, MAX_BUF_SIZE) != LOGIN_SUCCESS_CODE) {
    printf("Error: couldn't login\n");
    return -1;
}
printf("%s\n", buf);

// send TYPE message (switching to binary mode)
sprintf(buf, "TYPE I\r\n");
if (send_message(sockfd, buf, strlen(buf)) != 0) {
    printf("Error: couldn't switch to binary mode\n");
    return -1;
}
// receive 200 (switched to binary mode)
if (get_message(sockfd, buf, MAX_BUF_SIZE) != CMD_OK_CODE) {
    printf("Error: couldn't switch to binary mode\n");
    return -1;
}
printf("%s\n", buf);
```

```
    // send SIZE message
    sprintf(buf, "SIZE %s\r\n", url.resource);
    if (send_message(sockfd, buf, strlen(buf)) != 0) {
        printf("Error: couldn't send SIZE message\n");
        return -1;
    }
    // receive 213 (file size in bytes)
    if (get_message(sockfd, buf, MAX_BUF_SIZE) != FILE_STAT_CODE) {
        printf("Error: couldn't get file size\n");
        return -1;
    }
    printf("%s\n", buf);
    // parse the file size
    size_t file_size = get_file_size(buf);
    printf("file size is %ld\n", file_size);

    // send PASV message
    sprintf(buf, "PASV\r\n");
    if (send_message(sockfd, buf, strlen(buf))) {
        printf("Error: couldn't send PASV message\n");
    }
    // receive 227 (passive mode entered, plus the new address)
    if (get_message(sockfd, buf, MAX_BUF_SIZE) != PASSIVE_MODE_CODE) {
        printf("Error: couldn't enter in PASSIVE mode\n");
        return -1;
    }
    printf("%s\n", buf);

    // compute the new address
    char data_ip[] = "255.255.255.255";
    int data_port = parse_ip(buf, data_ip);
    printf("%s:%d\n", data_ip, data_port);

    // create a separate connection to the new address
    int data_socket;
    struct sockaddr_in data_addr;
    bzero((char *) &data_addr, sizeof(data_addr));
    data_addr.sin_family = AF_INET;
    data_addr.sin_addr.s_addr = inet_addr(data_ip);    /*32 bit Internet
address network byte ordered*/
    data_addr.sin_port = htons(data_port);        /*server TCP port must be
network byte ordered */

    /*open a TCP socket*/
    if ((data_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("Error: can't create data socket\n");
```

```
        return -1;
    }

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

    // send RETR message
    sprintf(buf, "RETR %s\r\n", url.resource);
    if (send_message(sockfd, buf, strlen(buf)) != 0) {
        printf("Error: couldn't send RETR message\n");
        return -1;
    }
    // receive 150 (transfer started)
    if (get_message(sockfd, buf, MAX_BUF_SIZE) != OPEN_DATA_CONNECTION_CODE)
{
        printf("Error: couldn't start transfer\n");
        return -1;
    }
    printf("%s\n", buf);

    // read from data socket, write to the file each time
    FILE* file = fopen(url.file, "w");
    size_t total_bytes = 0;
    while (total_bytes < file_size) {
        bytes = recv(data_socket, buf, MAX_BUF_SIZE, 0);
        if (bytes > 0) {
            //printf("Bytes read %ld\n", bytes);
            // comment out for a cleaner download process
        }
        else {
            perror("recv() failed");
            return -1;
        }
        total_bytes += bytes;
        fwrite(buf, 1, bytes, file);
    }
    if (total_bytes != file_size) {
        printf("Error: transfer was problematic\n");
        return -1;
    }

    // receive 226 (transfer complete)
```

```
    if (get_message(sockfd, buf, MAX_BUF_SIZE) != TRANSFER_COMPLETE_CODE) {
        printf("Error: transfer was not done\n");
        return -1;
    }
    printf("%s\n", buf);

    // send QUIT message
    sprintf(buf, "QUIT\r\n");
    if (send_message(sockfd, buf, strlen(buf)) != 0) {
        printf("Error: couldn't send QUIT message");
        return -1;
    }
    // receive 221 (bye)
    if (get_message(sockfd, buf, MAX_BUF_SIZE) != BYE_CODE) {
        printf("Error: couldn't get bye\n");
        return -1;
    }
    printf("%s\n", buf);

    // close connectios

    if (close(sockfd) < 0) {
        printf("Error: couldn't close ctrl_socket\n");
        return -1;
    }
    if (close(data_socket) < 0) {
        printf("Error: couldn't close data_socket\n");
        return -1;
    }

    return 0;
}
```

Annex 8.2 - parser.h

```
C/C++
#ifndef PARSER_H
#define PARSER_H

#include "url.h"

/**
 * Parses the received to url
 * @param raw_url Url in string format
```

```
 * @param url Url after being parsed in url format
 * @return 0 if succeded, -1 otherwise
 */
int parse(char *raw_url, Url *url);

#endif
```

Annex 8.3 - parser.c

```C/C++
#include "parser.h"

#include <string.h>
#include <stdio.h>

#include "utils.h"

int parse(char *raw_url, Url *url) {

    // pointer to update curr url element while preserving original string
    char *ptr = raw_url;

    if (strncmp(ptr, "ftp://", 6) != 0) {
        printf("Invalid protocol in url\n");
        return -1;
    }

    ptr += 6;

    char *at = strchr(ptr, '@');

    if (at != NULL) {
        char *colon = strchr(ptr, ':');

        // get user
        strncpy(url->user, ptr, colon-ptr);
        ptr += colon-ptr+1;

        // get password
        strncpy(url->password, ptr, at-ptr);
        ptr += at-ptr+1;
    } else {
        strcpy(url->user, "anonymous");
```

```
        strcpy(url->password, "password");
    }

    char *slash = strchr(ptr, '/');
    strncpy(url->hostname, ptr, slash-ptr);
    ptr += slash-ptr;   // dont skip the / char, because it can cause
problem if the resource is the file itself

    strcpy(url->resource, ptr);

    if (getHostIp(url) != 0) return -1;

    // creates a pointer to the last occurence of /, then copies the file
name to url struct
    char *f = strrchr(ptr, '/');
    strcpy(url->file, f+1);

    return 0;
}
```

Annex 8.4 - utils.h

```
C/C++
#ifndef UTILS_H
#define UTILS_H

#include "url.h"

/**
 * Gets Host IP from a given Url Struct
 * @return 0 if succeeded, -1 otherwise
 */
int getHostIp(Url *url);

#endif
```

Annex 8.5 - utils.c

```C/C++
#include "utils.h"

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int getHostIp(Url *url) {
    struct hostent *h;

    if ( (h = gethostbyname(url->hostname)) == NULL) {
        printf("error getting host information\n");
        return -1;
    }

    strcpy(url->host_ip, inet_ntoa(*((struct in_addr *) h->h_addr)) );

    return 0;
}
```

Annex 8.6 - url.h

```C/C++
#ifndef URL_H
#define URL_H

#define MAX_LENGTH 500

typedef struct Url {
    char user[MAX_LENGTH];
    char password[MAX_LENGTH];
    char hostname[MAX_LENGTH];
    char resource[MAX_LENGTH];

    char host_ip[MAX_LENGTH];
    char file[MAX_LENGTH];
} Url;

#endif
```

Annex 8.7 - ftp_response.h

```C/C++
#ifndef FTP_RESPONSE_H
#define FTP_RESPONSE_H

#define WELCOME_CODE 220
#define PSSWRD_REQUEST_CODE 331
#define LOGIN_SUCCESS_CODE 230
#define CMD_OK_CODE 200
#define FILE_STAT_CODE 213
#define PASSIVE_MODE_CODE 227
#define OPEN_DATA_CONNECTION_CODE 150
#define TRANSFER_COMPLETE_CODE 226
#define BYE_CODE 221

#endif
```

Annex 8.8 - Makefile

```None
CC = gcc
CFLAGS = -Wall -Iinclude

SRC = src
INCLUDE = include
BIN = bin

# URL for debugging
#URL1 = ftp://ftp.up.pt/pub/gnu/emacs/elisp-manual-21-2.8.tar.gz
URL1 = ftp://ftp.up.pt/pub/debian/README.html
URL2 = ftp://demo:password@test.rebex.net/readme.txt
URL3 = ftp://anonymous:anonymous@ftp.bit.nl/speedtest/100mb.bin

.PHONY: all
all: main

main: $(SRC)/*.c
    $(CC) $(CFLAGS) -o $(BIN)/download $^

# simple command for easier debugging
.PHONY: run_client_debug
run_client_debug: main
```

```
    @echo ""
    @echo "Testing 1st case"
    ./$(BIN)/download $(URL1)
#     @echo ""
#     @echo "Testing 2nd case"
#     ./$(BIN)/download $(URL2)
    @echo ""
    @echo "Testing 3rd case"
    ./$(BIN)/download $(URL3)

.PHONY: clean
clean:
    rm -rf $(BIN)/download
    rm -rf 100mb.bin
    rm -rf readme.txt
    rm -rf README.html
```

Annex 9.1 - Configuration commands

None

```
# Part 2 - Practical experiments

## Step 0

1. Reset switch and router (in GTK).
    1. Do the router, then move the leftmost cable to SW CON and reset again
    2. Login: admin Password: <blank>

```bash
/system reset-configuration
```

2. Reset networking services

```bash
sudo systemctl restart networking
```

Exp 1 - Configure an IP Network

Steps

1. Connect cables from E1_tuxY3 [ether16] and E1_tuxY4 [ether14] to switch
2. Set ip and masks
```

```bash
# on tux Y3
ifconfig if_e1 up
ifconfig if_e1 <IP> [172.16.Y0.1/24]    # this sets the ip and the mask at
the same time

# on tux Y4
ifconfig if_e1 up
ifconfig if_e1 <IP> ?? [172.16.Y0.254/24]
```

3. Verify connection with ping. i.e, try to ping tuxY3 on tuxY4 and vice versa

Exp 2 - Implement two bridges in a switch

Steps

1. Connect E1_tuxY2 [ether18] to switch and set its ip/mask

```bash
# on tuxY2; in tux92 if_e1 is enp2s0
ifconfig if_e1 up
ifconfig if_e1 <IP> [172.16.Y1.1/24]
```

2. Assure that the cables are correctly connected. Left cable should be in SW CON since we are setting up the switch
3. Add new bridges

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

4. Remove ports default bridge from tuxY3, tuxY4 and tuxY2.

```bash
# these ether interface are not fixed. is just to be easier when following
the guide

# port for tux93
/interface bridge port remove [find interface=ether16]

# port for tux94
/interface bridge port remove [find interface=ether14]
```

```
# port for tux92
/interface bridge port remove [find interface=ether18]
```

5. Add ports to new bridge

```bash
# port for tux93
/interface bridge port add bridge=bridge90 interface=ether16

# port for tux94
/interface bridge port add bridge=bridge90 interface=ether14

# port for tux92
/interface bridge port add bridge=bridge91 interface=ether18
```

6. Verify with ping. Pinging from tuxY3 to tuxY4 must succeed, but fail to tuxY2

Exp 3 - Configure a Router in Linux

Steps

1. Plug E2_tux94 [ether15] to swtich and set its ip/mask

```bash
# on tux Y4
ifconfig if_e2 up
ifconfig if_e2 <IP> ?? [172.16.Y1.253/24]
```

2. Add port to tux92 bridge

```bash
# removing default port
/interface bridge port remove [find interface=ether15]

# port for tux94_e2
/interface bridge port add bridge=bridge91 interface=ether15
```

3. Enable IP forwarding

```bash
sudo sysctl -w net.ipv4.ip_forward=1
```

```

4. Disable ICMP echo-ignore-broadcast

```bash
sudo sysctl -w net.ipv4.icmp_echo_ignore_broadcasts=0
```

5. Add new routing

```bash
# on tuxY2; add a "path" to tuxY3 via tuxY4(if_e2)
route add -net 172.16.Y0.0/24 gw 172.16.Y1.253

# on tuxY3; add a "path" to tuxY3 via tuxY4(if_e1)
route add -net 172.16.Y1.0/24 gw 172.16.Y0.254
```

6. From tuxY3, ping the other network interfaces (172.16.Y0.254,
172.16.Y1.253,
172.16.Y1.1) and verify if there is connectivity

>> clean arp tables with “sudo ip neigh flush all”

Exp 4 - Configure a Commercial Router and Implement NAT

1. Plug Rc ether2 to the switch (ether10)
2. Add Rc port to bridge91

```bash
# remove default port
/interface bridge port remove [find interface=ether10]

# port for router
/interface bridge port add bridge=bridge91 interface=ether10
```

3. Change from SW CONS to ROUTER CONS
4. Set router ips via GTK

```bash
/ip address add address=172.16.1.Y1/24 interface=ether1

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

5. Add new routes
```

```bash
# on tuxY2
#route add default gw 172.16.Y1.254
route add -net 172.16.1.0/24 gw 172.16.Y1.254

# on tuxY3
#route add default gw 172.16.Y0.254
route add -net 172.16.1.0/24 gw 172.16.Y0.254

# on tuxY4
#route add default gw 172.16.Y1.254
route add -net 172.16.1.0/24 gw 172.16.Y1.254

# on Router serial terminal
/ip route add dst-address=172.16.Y0.0/24 gateway=172.16.Y1.253
```

6. Verify with ping on tuxY3. Try to ping 172.16.Y0.254 (tuxY4 e1), 172.16.Y1.253 (tuxY4 e2), 172.16.Y1.1 (tuxY2 e1), 172.16.Y1.254 (Rc)

Exp 5 - DNS

1. In each tux access the file and add nameserver 10.227.20.254

```bash
sudo nano /etc/resolv.conf
```

2. Some checking and problem resolution

```bash
# check
ping google.com
ping ftp.netlab.fe.up.pt
```

Exp 6 - TCP Connections

Important info

e1_tuxY2

MAC: a2:05:02:f6:c6:5c
IP: 172.16.91.1

ether18

```
### e1_tuxY3

MAC: ec:75:0c:c2:17:7a
IP: 172.16.90.1

ether16

### e1_tuxY4

MAC: ec:75:0c:c2:26:42
IP: 172.16.90.254

ether14

### e2_tuxY4

MAC: ec:75:0c:c2:31:93
IP: 172.16.91.253

ether15

### Rc

IP: 192.168.88.1 (generic)

ip_lab: 172.16.1.99

ip_sw: 172.16.91.254

ether10
```