Computer Networks

Second Project

Bruno Oliveira
Rodrigo Silva

Bachelor's in Informatics and Computing Engineering
Computer Networks

Professor: Helder Fontes

January 2025

Contents

1 Summary

The aim of this project in Computer Networks was twofold: to implement a program for downloading files using the FTP protocol using common C and Unix libraries, as well as to configure an entire computer network step-by-step in the laboratory at FEUP.

By creating the download application, we became familiar with the FTP protocol, how to establish connections using it, and also learned more about different libraries that exist for these purposes. Besides the code itself, setting up the network to work with different devices allowed us to gain insights about how they work and the commands used to configure networks, as well as apply the knowledge studied in class regarding Medium Access Control, Network Layer and Transport Layer.

2 Introduction

This report presents the goals, design, and results of the second project in the Computer Networks course. The project focused on configuring a computer network and developing a download application using FTP to download a file from a server on the laboratory computers.

The report is divided into the following sections:

- **Download Application**: Describing the FTP download application developed for this project.
- **Configuration of Networks**: Describing the steps followed for each experiment of the project with commands, Wireshark logs, and theoretical analysis.
  - **Experiment 1 - Configure an IP Network**
  - **Experiment 2 - Implement two Bridges in a Switch**
  - **Experiment 3 - Configure a Router in Linux**
  - **Experiment 4 - Configure a Commercial Router and Implement NAT**
  - **Experiment 5 - Domain Name System (DNS)**
  - **Experiment 6 - TCP Connections**
- **Conclusion**: A synthesis the information presented and reflection on the objectives achieved.

Besides these main sections, the report also includes an **Appendix** section, containing the source code of the download application, as well as the commands and logs used in the different experiments of the project.

3 Download Application

The first part of this project consisted of developing a simple FTP client written in C language capable of downloading a single file, given the corresponding URL (following the RFC 1738 standard).

3.1 Architecture

Our implementation was done in a single C source code file, `download.c`. This file contains multiple auxiliary functions and structures, of which the most relevant are listed below:

- `FtpUrl` - Structure that stores each component of the FTP URL, more specifically, the username and password (used for the login), the server domain, and the file path.
- `Message` - Structure that stores information about a server response's line, such as the response code and whether the line is final or not.
- `parse_url()` - Builds an FtpUrl instance using the provided URL, carefully checking its structure.
- `get_socket_fd_addr()` and `get_socket_fd_host()` - Connects the client to the server, using the provided address/host and port, and returns the socket file descriptor.
- `read_message()` - Fills a `Message` object with a line read from the server's response.
- `read_end()` - Reads all the lines of the server's response until the final line is reached, storing it in one of its arguments.
- `check_code()` - Verifies if the given `Message` contains one of the expected codes, printing the error if a mismatch occurs.
- `send_command()` - Sends a command to the socket that matches the given file descriptor.
- `parse_pasv_response()` - Parses the text of a successful response to the `PASV` command to extract the IP address and port from where to execute the file data transfer.

3.2 Application Execution and Flow

After compiling the application (using the provided Makefile) with the `make` command, the program can be executed with `./download <URL>`.

The normal flow of the application follows the steps described below. If, at any step, the application encounters an error, whether it is an unparseable URL or a bad response code from the server, it stops execution and provides an adequate error message.

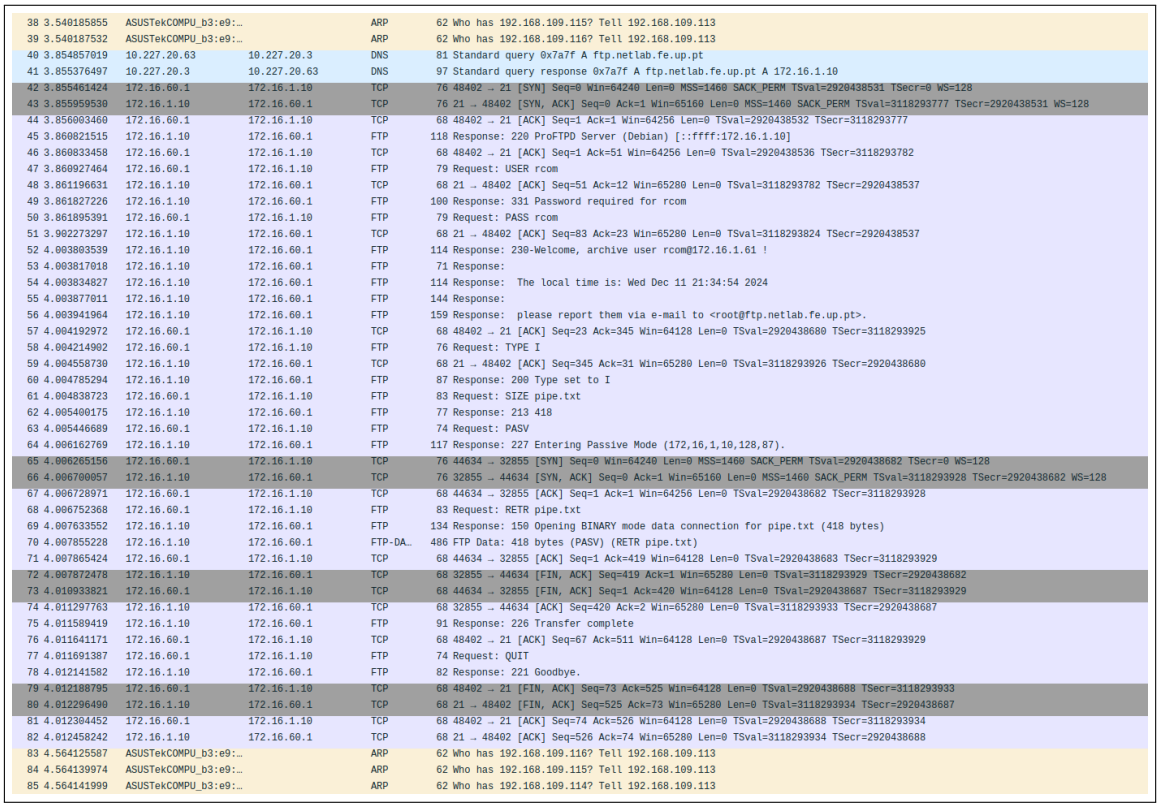
1. **URL Parsing** - The URL provided in the command line is analyzed and parsed, giving a proper contextual error message on failure. If the username and/or password are omitted from the provided URL, they are replaced with "anonymous".
2. **Connection Establishment** - The application establishes a socket connection to the server through the default FTP port 21. This is the control connection which allows the application to send FTP commands to the server.
3. **Login and File Information Retrieval** - The credentials are sent to authenticate the session. If the login is successful, the application sets the binary transfer mode (using the `TYPE I` command) and retrieves the file size (with the `SIZE` command). This last step is not necessary, but our application uses it to track the transfer progress.

4. **Passive Mode and Data Transfer** - The application enters passive mode (with the `PASV` command), creating a new socket connection to the provided port and IP address from the server. This is the data connection which receives the data from the server and writes it to a file with the same filename.

5. **Termination** - After the transfer is complete, the data connection is closed and only after that the main connection (control) is closed with the `QUIT` command, and some statistics about the transfer are presented.

3.3 Download Report

The application was tested successfully with all the provided FTP URLs. In the case of the file `pipe.txt` at ftp://ftp.netlab.fe.up.pt/, the respective Wireshark logs can be observed in Figure 1 and the terminal output is also shown in **this example**.



38	3.540185855	ASUSTekCOMPU_b3:e9:…		ARP	62 Who has 192.168.109.115? Tell 192.168.109.113
39	3.540187532	ASUSTekCOMPU_b3:e9:…		ARP	62 Who has 192.168.109.116? Tell 192.168.109.113
40	3.854857019	10.227.20.63	10.227.20.3	DNS	81 Standard query 0x7a7f A ftp.netlab.fe.up.pt
41	3.855376497	10.227.20.3	10.227.20.63	DNS	97 Standard query response 0x7a7f A ftp.netlab.fe.up.pt A 172.16.1.10
42	3.855461424	172.16.60.1	172.16.1.10	TCP	76 48402 → 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2920438531 TSecr=0 WS=128
43	3.855959530	172.16.1.10	172.16.60.1	TCP	76 21 → 48402 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=3118293777 TSecr=2920438531 WS=128
44	3.856003460	172.16.60.1	172.16.1.10	TCP	68 48402 → 21 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2920438532 TSecr=3118293777
45	3.860821515	172.16.1.10	172.16.60.1	FTP	118 Response: 220 ProFTPD Server (Debian) [::ffff:172.16.1.10]
46	3.860833458	172.16.60.1	172.16.1.10	TCP	68 48402 → 21 [ACK] Seq=1 Ack=51 Win=64256 Len=0 TSval=2920438536 TSecr=3118293782
47	3.860927464	172.16.60.1	172.16.1.10	FTP	79 Request: USER rcom
48	3.861196631	172.16.1.10	172.16.60.1	TCP	68 21 → 48402 [ACK] Seq=51 Ack=12 Win=65280 Len=0 TSval=3118293782 TSecr=2920438537
49	3.861827226	172.16.1.10	172.16.60.1	FTP	100 Response: 331 Password required for rcom
50	3.861895391	172.16.60.1	172.16.1.10	FTP	79 Request: PASS rcom
51	3.902273297	172.16.1.10	172.16.60.1	TCP	68 21 → 48402 [ACK] Seq=83 Ack=23 Win=65280 Len=0 TSval=3118293824 TSecr=2920438537
52	4.003803539	172.16.1.10	172.16.60.1	FTP	114 Response: 230-Welcome, archive user rcom@172.16.1.61 !
53	4.003817018	172.16.1.10	172.16.60.1	FTP	71 Response:
54	4.003834827	172.16.1.10	172.16.60.1	FTP	114 Response: The local time is: Wed Dec 11 21:34:54 2024
55	4.003877011	172.16.1.10	172.16.60.1	FTP	144 Response:
56	4.003941964	172.16.1.10	172.16.60.1	FTP	159 Response: please report them via e-mail to <root@ftp.netlab.fe.up.pt>.
57	4.004192972	172.16.60.1	172.16.1.10	TCP	68 48402 → 21 [ACK] Seq=23 Ack=345 Win=64128 Len=0 TSval=2920438680 TSecr=3118293925
58	4.004214902	172.16.60.1	172.16.1.10	FTP	76 Request: TYPE I
59	4.004558730	172.16.1.10	172.16.60.1	TCP	68 21 → 48402 [ACK] Seq=345 Ack=31 Win=65280 Len=0 TSval=3118293926 TSecr=2920438680
60	4.004785294	172.16.1.10	172.16.60.1	FTP	87 Response: 200 Type set to I
61	4.004838723	172.16.60.1	172.16.1.10	FTP	83 Request: SIZE pipe.txt
62	4.005400175	172.16.1.10	172.16.60.1	FTP	77 Response: 213 418
63	4.005446689	172.16.60.1	172.16.1.10	FTP	74 Request: PASV
64	4.006162769	172.16.1.10	172.16.60.1	FTP	117 Response: 227 Entering Passive Mode (172,16,1,10,128,87).
65	4.006265156	172.16.60.1	172.16.1.10	TCP	76 44634 → 32855 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2920438682 TSecr=0 WS=128
66	4.006700057	172.16.1.10	172.16.60.1	TCP	76 32855 → 44634 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=3118293928 TSecr=2920438682 WS=128
67	4.006728971	172.16.60.1	172.16.1.10	TCP	68 44634 → 32855 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2920438682 TSecr=3118293928
68	4.006752368	172.16.60.1	172.16.1.10	FTP	83 Request: RETR pipe.txt
69	4.007633552	172.16.1.10	172.16.60.1	FTP	134 Response: 150 Opening BINARY mode data connection for pipe.txt (418 bytes)
70	4.007855228	172.16.1.10	172.16.60.1	FTP-DA…	486 FTP Data: 418 bytes (PASV) (RETR pipe.txt)
71	4.007865424	172.16.60.1	172.16.1.10	TCP	68 44634 → 32855 [ACK] Seq=1 Ack=419 Win=64128 Len=0 TSval=2920438683 TSecr=3118293929
72	4.007872478	172.16.1.10	172.16.60.1	TCP	68 32855 → 44634 [FIN, ACK] Seq=419 Ack=1 Win=65280 Len=0 TSval=3118293929 TSecr=2920438682
73	4.010933821	172.16.60.1	172.16.1.10	TCP	68 44634 → 32855 [FIN, ACK] Seq=1 Ack=420 Win=64128 Len=0 TSval=2920438687 TSecr=3118293929
74	4.011297763	172.16.1.10	172.16.60.1	TCP	68 32855 → 44634 [ACK] Seq=420 Ack=2 Win=65280 Len=0 TSval=3118293933 TSecr=2920438687
75	4.011589419	172.16.1.10	172.16.60.1	FTP	91 Response: 226 Transfer complete
76	4.011641171	172.16.60.1	172.16.1.10	TCP	68 48402 → 21 [ACK] Seq=67 Ack=511 Win=64128 Len=0 TSval=2920438687 TSecr=3118293929
77	4.011691387	172.16.60.1	172.16.1.10	FTP	74 Request: QUIT
78	4.012141582	172.16.1.10	172.16.60.1	FTP	82 Response: 221 Goodbye.
79	4.012188795	172.16.60.1	172.16.1.10	TCP	68 48402 → 21 [FIN, ACK] Seq=73 Ack=525 Win=64128 Len=0 TSval=2920438688 TSecr=3118293933
80	4.012296490	172.16.1.10	172.16.60.1	TCP	68 21 → 48402 [FIN, ACK] Seq=525 Ack=73 Win=65280 Len=0 TSval=3118293934 TSecr=2920438687
81	4.012304452	172.16.60.1	172.16.1.10	TCP	68 48402 → 21 [ACK] Seq=74 Ack=526 Win=64128 Len=0 TSval=2920438688 TSecr=3118293934
82	4.012458242	172.16.1.10	172.16.60.1	TCP	68 21 → 48402 [ACK] Seq=526 Ack=74 Win=65280 Len=0 TSval=3118293934 TSecr=2920438688
83	4.564125587	ASUSTekCOMPU_b3:e9:…		ARP	62 Who has 192.168.109.116? Tell 192.168.109.113
84	4.564139974	ASUSTekCOMPU_b3:e9:…		ARP	62 Who has 192.168.109.115? Tell 192.168.109.113
85	4.564141999	ASUSTekCOMPU_b3:e9:…		ARP	62 Who has 192.168.109.114? Tell 192.168.109.113

Figure 1: Wireshark Logs of Successful Download

4 Configuration of Networks

This following subsections will describe each of the experiments performed in the practical classes focusing the network architecture, configuration commands, relevant logs, as well as explaining the objectives. Since each experiment also has some questions, we will provide answers and explanations for each of them.

For reference, the **image** below illustrates the final computer network setup which we replicated throughout the experiments. Note that, we did the experiments and captured the logs in workbench 6, therefore, the logs will show a 6 where "Y" is present in the figure. Throughout the report, we aimed to keep the explanations general but some examples are also mentioned for better clarity.

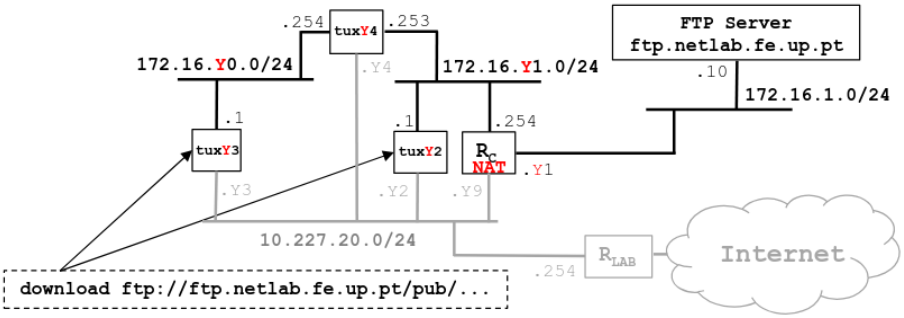


Figure 2: Computer Network Setup

4.1 Experiment 1 - Configure an IP Network

The goal of this first experiment is to configure the network between two different computers, namely Tux63 and Tux64, using a Mikrotik switch. This allows us to become more familiar with the development environment, configure IP addresses, and analyze ARP tables.

The configuration commands used can be found in **Experiment 1 Commands**.

The first step is to connect the desired ethernet ports from both computers to the switch. After that, using `ifconfig`, we specify the desired IP addresses for each one of them in the subnetwork. Each of the devices also has a specific MAC address that uniquely identifies it. Both of these values can also be checked with `ifconfig`.

The setup is now complete and we can test it by running the `ping` command between the two devices. Using Wireshark to capture the logs, we can see that ICMP Echo Request/Reply packets will be generated containing the IP addresses of the source and destination, as well as the MAC address of the source and destination of the next device which the packet should follow (also known as the next-hop).

However, since in the beginning the computers don't know the MAC addresses of each other, the source machine will broadcast an ARP request with the origin (itself) and destination IP address. This ARP request will be answered by the machine with the destination IP address responding with its MAC address which is then saved in the ARP table. After the value is saved in the ARP table, the ARP request is not needed in future connections. These ARP requests are processed using ARP packets which map IP addresses to MAC addresses.

To distinguish between the type of request sent, we can analyze the EtherType header in the sent packet: when it is `0x0806`, it corresponds to an ARP packet and when it is `0x0800`, it is an IP packet. When we have an IP packet, if the "Protocol" field is 1, we have an ICMP packet. In order to know the length of a receiving frame, we just need to check the payload size in the Ethernet header.

Finally, we also noticed that the ARP table contains a special value for the loopback interface, which is a software interface that allows a machine to communicate with itself.

This interface is essential for testing and communications internal to the device.

These conclusions can be taken from the **Experiment 1 Logs**.

4.2 Experiment 2 - Implement two Bridges in a Switch

The goal of this second experiment is to implement two bridges in the Mikrotik switch: the first one that connects Tux63 and Tux64 and the second one that connects Tux62. This allows us to learn more about creating bridges using the Mikrotik switch console and also to verify that the devices in different subnetworks cannot communicate with each other (at least for now).

The configuration commands used can be found in **Experiment 2 Commands**.

The first step is to configure the connection in Tux62 using the `ifconfig` commands, similarly to the first experiment. After that, we can access the Mikrotik switch console and setup the two bridges: `bridge60` and `bridge61`. To do this, we start by creating the bridges themselves and then assigning each device accordingly by specifying the Ethernet ports connected to the switch.

To check the connections between the bridges, we can capture the logs in all the computers and run ping broadcasts in both Tux63 and then on Tux62. As expected, we will receive responses from the devices in the same bridge (in this case, Tux63 can connect to Tux64), but not from the device in the other bridge (Tux63 cannot reach Tux62 and vice-versa). From this, we can conclude that there are two different broadcast domains: one for `bridge60` (Tux63 and Tux64) and another for `bridge61` (Tux62).

The logs for this experiment can be found in **Experiment 2 Logs**.

4.3 Experiment 3 - Configure a Router in Linux

The goal of this third experiment is to extend the configuration of the two bridges from the previous experiment, using Tux64 as a router between the two bridges. This will give us some insights on how routing works, how to use it to connect different subnetworks and also analyze the IP and MAC addresses and ARP and routing tables of this network.

The configuration commands used can be found in **Experiment 3 Commands**.

The setup for this experiment is very similar to previous ones, we first configure a new Ethernet port on Tux64 and add it to `bridge61` with Tux62 (specifying the new port used on the switch). Now, in order to allow Tux64 to work as a router between the two subnetworks, we must enable IP forwarding and disable ICMP echo-ignore-broadcast.

Last but not least, we must add the routes that connect: in Tux63 to the subnetwork containing Tux62 (172.16.61.0/24), and in Tux62 to the subnetwork containing Tux63 (172.16.60.0/24), using Tux64 as a gateway in both cases (in the first case, 172.16.60.254 and in the second one 172.16.61.253). Due to the way we configured Tux64, when Tux63 tries to connect to Tux62, it will connect to Tux64 (since it is defined as the gateway) which will then connect to Tux62. The same will happen the other way around when Tux62 tries to connect to Tux63.

The setup for this experiment is now complete and we can now successfully ping Tux62 from Tux63 and vice-versa, as shown **in the logs**. As an additional test, we also tried to establish these connections after clearing the ARP tables in all the computers. As expected, the results were similar, but, just like in Experiment 1, there were initial broadcast ARP requests sent to retrieve the MAC addresses of the machines, before actually starting the connection, as shown **in the logs**.

The routes of all Tuxes are shown **in the logs** and correspond to the path the packets take to reach other devices. An entry on this forwarding table specifies the IP address of the destination (device or subnetwork), the IP address next-hop of the connection (gateway) and the network interface to use for forwarding. In the case of Tux64, it shows how packets are forwarded between the two subnetworks.

The ARP messages observed at the start of the connection are broadcasts that allow each machine to resolve the MAC addresses of their next-hop connections. In this case, the ARP requests occur between Tux63 and the Tux64 (the gateway) and between Tux64 and Tux62.

By analyzing the subsequent ICMP Echo Requests/Replies packets transmitted from the ping commands, we can see that they have the initial and destination IP addresses (for Tux63 and Tux62), but the MAC addresses always correspond to direct connections (for instance, Tux63 to Tux64 and Tux64 to Tux62). This is because MAC addresses work on the Data Link layer with direct physical connections between machines, while the IP addresses are on the Network layer.

The logs for this experiment can be found in **Experiment 3 Logs**.

4.4 Experiment 4 - Configure a Commercial Router and Implement NAT

The goal of this fourth experiment is to continue with the configuration of the two bridges from the previous experiment but now we will add a Mikrotik router connected to the `bridge61` of the Mikrotik switch (containing Tux62 and Tux64). This Mikrotik router will also connect the network to the Internet using NAT, and the routes on all Tuxes should be updated accordingly. This allows us to learn how to configure the Mikrotik router, connect our network to the Internet, implement NAT and further expand the knowledge about routing.

The configuration commands used can be found in **Experiment 4 Commands**.

The first step is to connect the router to the lab network, with NAT enabled by default, and then connect it to the Mikrotik switch. After that, we use the switch console to connect the router to the existing `bridge61` with Tux62 and Tux64, similar to what we did on previous experiments.

Now, we must configure/check all routes in the network, as shown in **Experiment 4 Commands**. Following the **Computer Network Setup** from left to right:

- **Tux63**: connection to subnetworks 172.16.61.0 (bridge61) and 172.16.1.0 (external) should use gateway 172.16.60.254 (Tux64)
- **Tux64**: connection to subnetwork 172.16.1.0 (external) should use gateway 172.16.61.254 (Router)
- **Tux62**: connection to subnetwork 172.16.1.0 (external) should use gateway 172.16.61.254 (Router) and to subnetwork 172.16.60.0 (bridge60) should use gateway 172.16.61.253 (Tux64)
- **Router**: connection to subnetwork 172.16.60.0 (bridge60) should use gateway 172.16.61.253 (Tux64)

This setup is the goal of the experiment, and we are now able to ping any connections in the network from any machine. The rest of this section contains additional tests and analysis of their results.

The first test we did consisted of changing the configuration of Tux62, as shown in **Experiment 4 Commands**. Initially, we disable ICMP redirects and changed the route to subnetwork 172.16.60.0 (bridge60) from gateway 172.16.61.253 (Tux64) to gateway 172.16.61.254 (Router). With this setup, when we ping Tux63 from Tux62, the packets will go through the following route: Tux62 → Router → Tux64 → Tux63. This shows that despite existing a shorter route, since redirects are disabled, the packets will follow the longer path from Tux63 to Tux62. After this, if we enable the redirects back again, the packets will change route and go directly from Tux62 → Tux64 → Tux63. These conclusions can be analyzed with `traceroute`.

The other test consists on pinging the FTP server from Tux63 with and without Network Address Translation (NAT), which can be seen in the **Experiment 4 Commands**. The NAT mechanism is what allows a set of IP addresses in a private network to be translated into a single public IP address. This public IP address can then be accessed, for example, from the Internet without any collisions. As expected, with NAT enabled, the ping works correctly, however, with NAT disabled, the packet is able to reach the server, but the response does not reach Tux63 since it cannot route the private IP address.

The logs for this experiment can be found in **Experiment 4 Logs**.

4.5 Experiment 5 - Domain Name System (DNS)

The goal of this fifth experiment is to configure the Domain Name System (DNS) in all machines within the network. This makes it possible to understand how DNS works and how hostnames are resolved into IP addresses based on the netlab DNS service.

The configuration commands used can be found in **Experiment 5 Commands**.

The steps for this experiment are very simple and consist of adding `services.netlab.fe.up.pt 10.227.20.3` to the `/etc/resolv.conf` file in all of the Tuxes. By doing this, we have configured DNS on all the machines which allows translating between human-readable names (like google.com or archlinux.org) into the public IP addresses that are used in the Network level.

When an address in human-readable format is accessed, it makes a DNS Query request to the nameservers specified in the `/etc/resolv.conf` to translate it. The response is then sent in a DNS Query Response with the result. Both of these packets follow a structure with the same Header (Transaction ID, Flags, Questions, Answers, Authority, Additional), followed by a Queries section which is divided into Name (domain name), Type (DNS record type) and Class (allows domain names to be used for arbitrary objects). Additionally, the DNS Query Response contains the Answers section which is similar to the Queries, but with 3 extra sections: Time To Live (seconds the record can live), Data Length and Data (the result of the query itself).

By analyzing our attempts to ping different hosts we concluded that `google.com` has IP `142.250.200.142` and `archlinux.com` has IP `95.217.163.246`. The complete logs can be analyzed in **Experiment 5 Logs**.

4.6 Experiment 6 - TCP Connections

The goal of this sixth experiment is to test the **Download Application** developed by downloading a file on Tux63 from the Netlab FTP server. This not only allows us to review the concepts from the FTP download application, but also verify that the network config-

uration is correct and understand how the final architecture is configured and what TCP packets are exchanged.

The configuration commands used can be found in **Experiment 6 Commands**.

The first part of this experiment consists of compiling the download application in Tux63 and downloading a file from the FTP server and verifying that it arrives correctly. By analyzing the logs, we notice that, initially, one TCP connection is established (for control). The application then exchanges some commands (login, obtain file size,...) with the server and eventually receives the information needed to connect a second port. A new TCP connection is then established to this port which is responsible for the actual file transfer. We can also easily view the 3 TCP phases through the **logs**: the Connection Establishment, the Data Transfer and the Connection Termination.

From the **logs**, we can also observe the Automatic Repeat Request (ARQ) mechanism which is an error control process used in TCP connections to reliably transmit data packets. The ARQ mechanism is based on acknowledgments, timeouts and packet retransmissions, hence we are able to verify this mechanism in the Wireshark logs. This mechanism uses parameters like Sequence Number, Acknowledgment Number and Window Size to make sure the packets are transmitted correctly and, otherwise, retransmit certain packets.

We were also able to view how the TCP Congestion Control Mechanism works. Initially, the transfer window starts small and starts to gradually increase by 1 segment (additive increase) while no errors occur. As soon as one error occurs, the window size is halved (multiplicative decrease), to reduce congestion and packet loss. The best way to analyze this is through the **throughput graph** which has saw-tooth pattern due to the steady increases and drastic decreases.

The second part of this experiment investigates how the throughput is disturbed when two machines download files simultaneously. For this, we started a download in Tux63 and, in the middle of the transfer, started another on in Tux62. By analyzing the logs, we can conclude that the throughput of the server is slightly reduced when the second TCP connection is initiated, as seen in the **throughput graph**.

5 Conclusion

In conclusion, this project allowed us to deepen our understanding of fundamental computer network concepts, such as MAC and IP addresses, bridges, switches, routers, ARP and route tables, Network Address Translation, Domain Name Systems, TCP protocol and a general overview of Computer Network configuration with focus on the Network layer. Moreover, developing a FTP download application provided hands-on experience with sockets, C libraries for network interfacing and FTP server connections.

This project was also very useful to gain more practical skills by configuring the network with appropriate commands and connections. Furthermore, we also believe using Wireshark to capture the logs was essential to gain insights with common development tools in this field.

Overall, the project successfully met its objectives, giving us practical insights into the network protocols and computer network configuration.

A Appendix

A.1 Download Application

A.1.1 Source Code

```c
#include <stdarg.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#include <arpa/inet.h>
#include <fcntl.h>
#include <libgen.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/stat.h>
#include <time.h>
#include <unistd.h>

#define URL_MAX_LEN 2048  // Maximum length of an URL
#define BUFFER_LEN 2048   // Maximum length of a buffer
#define FTP_PORT 21       // FTP default port number
#define BAR_WIDTH 24      // Width of the progress bar

// Structure to store the FTP URL components
typedef struct {
    char username[BUFFER_LEN + 1];
    char password[BUFFER_LEN + 1];
    char domain[BUFFER_LEN + 1];
    char path[BUFFER_LEN + 1];
} FtpUrl;

// Structure to store a line from a response received from the server
typedef struct {
    int code;
    char content[BUFFER_LEN + 1];
    bool final;
} Message;


// Print an error message
void print_error(const char *function_name, const char *message_format, ...) {
    va_list args;
    va_start(args, message_format);

    fprintf(stderr, "Error in %s: ", function_name);
    vfprintf(stderr, message_format, args);
    fprintf(stderr, "\n");

    va_end(args);
}

// Parse an URL, storing its components in a FtpUrl structure
// Return 0 on success and non-zero otherwise
int parse_url(char *url, FtpUrl *ftp_url) {
    if (strlen(url) > URL_MAX_LEN) {
        print_error(__func__, "URL max length exceeded");
        return 1;
    }

    char *scheme = strtok(url, ":"); // Remove scheme
    char *schemeless_url = strtok(NULL, "");
```

```c
    if (scheme == NULL || strncmp(schemeless_url, "//", 2) != 0) {
        print_error(__func__, "Bad URL");
        return 1;
    }

    char *username_password, *domain;
    if (strchr(schemeless_url + 2, '@') != NULL) {
        username_password = strtok(schemeless_url + 2, "@");
        domain = strtok(NULL, "/");
    } else {
        username_password = NULL;
        domain = strtok(schemeless_url + 2, "/");
    }
    char *path = strtok(NULL, "");

    if (domain == NULL || path == NULL) {
        print_error(__func__, "Bad URL");
        return 1;
    }

    if (strcmp(scheme, "ftp") != 0) {
        print_error(__func__, "Invalid schema %s", scheme);
        return 1;
    }

    char *username, *password;
    if (username_password == NULL) {
        username = "anonymous";
        password = "anonymous";
    } else if (strchr(username_password, ':') == NULL) {
        username = username_password;
        password = "anonymous";
    } else {
        username = strtok(username_password, ":");
        password = strtok(NULL, "");
    }

    strncpy(ftp_url->username, username, BUFFER_LEN);
    strncpy(ftp_url->password, password, BUFFER_LEN);
    strncpy(ftp_url->domain, domain, BUFFER_LEN);
    strncpy(ftp_url->path, path, BUFFER_LEN);

    return 0;
}
// Get a socket file descriptor connected to a given address and port
// Return the socket file descriptor on success and -1 otherwise
int get_socket_fd_addr(const char *addr, uint16_t port) {
    struct sockaddr_in server_addr;
    int sockfd;

    bzero((char *) &server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr(addr);
    server_addr.sin_port = htons(port);

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        print_error(__func__, "socket() failed");
        return -1;
    }

    if (connect(sockfd, (struct sockaddr *) &server_addr, sizeof(server_addr)) < 0) {
        print_error(__func__, "connect() failed");
        return -1;
    }

    printf("Connected to %s:%d\n", addr, port);
    return sockfd;
```

```c
}

// Returns a socket file descriptor connected to a specific host and port
// Return the socket file descriptor on success and -1 otherwise
int get_socket_fd_host(const struct hostent *host, uint16_t port) {
    const char *addr = inet_ntoa(*((struct in_addr *) host->h_addr));
    return get_socket_fd_addr(addr, port);
}

// Read a line from a socket, storing the corresponding info in a Message structure
// Return 0 on success and non-zero otherwise
int read_message(int sockfd, Message *message) {
    char buffer[BUFFER_LEN + 1], separator;

    while (true) {
        int i = 0;
        char c;
        while (i < BUFFER_LEN && read(sockfd, &c, 1) > 0 && c != '\r') {
            buffer[i++] = c;
        }
        buffer[i] = '\0';
        read(sockfd, &c, 1);  // Consume '\n'

        printf("    \x1B[2;37m%s\x1B[0m\n", buffer);

        int res, n;
        if ((res = sscanf(buffer, "%d%c%n", &message->code, &separator, &n)) == 2) {  //
        ↪  %n does not count for the argument count
            strcpy(message->content, buffer + n);
        } else {
            continue;
        }
        break;
    }

    switch (separator) {
        case '-':
            message->final = false;
            break;
        case ' ':
            message->final = true;
            break;
        default:
            print_error(__func__, "Invalid response (Unknown separator)\n");
            return 1;
    }

    return 0;
}

// Read the final line of a socket response, ignoring all the previous ones
// Return 0 on success and non-zero otherwise
int read_end(int sockfd, Message *message) {
    do {
        if (read_message(sockfd, message))
            return 1;
    } while (!message->final);

    return 0;
}

// Check if a message has a specific code
// Return 0 if the code is the expected one and non-zero otherwise
int check_code(Message *message, ...) {
    va_list args;
    va_start(args, message);
```

```c
    for (int expected_code = va_arg(args, int); expected_code != 0; expected_code =
    ↪  va_arg(args, int)) {
        if (message->code == expected_code) {
            return 0;
        }
    }

    va_end(args);

    fprintf(stderr, "Error in %s: Invalid response (code %d, expected ", __func__,
    ↪  message->code);

    va_start(args, message);
    for (int i = 0, expected_code = va_arg(args, int); expected_code != 0; i++,
    ↪  expected_code = va_arg(args, int)) {
        if (i > 0) {
            fprintf(stderr, "/");
        }
        fprintf(stderr, "%d", expected_code);
    }
    va_end(args);

    fprintf(stderr, ")\n");

    return 1;
}

// Send a command to a socket
// Return 0 on success and non-zero otherwise
int send_command(int sockfd, const char *command_format, ...) {
    char command[BUFFER_LEN + 1];

    va_list args;
    va_start(args, command_format);

    int len = vsnprintf(command, BUFFER_LEN - 2, command_format, args);
    sprintf(command + len, "\r\n");

    va_end(args);

    printf("  > \x1B[1;2;37m%.*s\x1B[0m\n", len, command);
    if (write(sockfd, command, len + 2) < 0) {
        print_error(__func__, "write() failed");
        return 1;
    }

    return 0;
}

// Print a progress bar
void print_progress(size_t current, size_t total) {
    int width = current * BAR_WIDTH / total;

    if (width < 24) {
        double percentage = current * 100.0 / total;
        printf("\x1B[2KDownloading...      [%.*s>%*s] %.1f%%\r", width,
        ↪  "=======================", 23 - width, "", percentage);
    } else {
        printf("\x1B[2KDownload complete!  [%s] 100.0%%\n", "=======================");
    }

    fflush(stdout);
}

// Parse the address and port from a "Passive Mode" response
// Return 0 on success and non-zero otherwise
int parse_pasv_response(const char *response, char *addr, uint16_t *port) {
```

```c
    uint8_t addr_bytes[4], port_bytes[2];

    int res = sscanf(
        response,
        "Entering Passive Mode (%hhu,%hhu,%hhu,%hhu,%hhu,%hhu)",
        addr_bytes, addr_bytes + 1, addr_bytes + 2, addr_bytes + 3, port_bytes,
        ↪  port_bytes + 1
    );
    if (res ≠ 6) {
        print_error(__func__, "Invalid \"Passive Mode\" response");
        return 1;
    }

    sprintf(addr, "%hhu.%hhu.%hhu.%hhu", addr_bytes[0], addr_bytes[1], addr_bytes[2],
    ↪  addr_bytes[3]);
    *port = port_bytes[0] * 256 + port_bytes[1];

    return 0;
}

// Reduce a size to the most appropriate unit
// Gives the most appropriate unit and the corresponding size modifier
void reduce_unit(size_t size, char **unit, size_t *size_modifier) {
    if (size > 1024 * 1024) {
        *size_modifier = 1024 * 1024;
        *unit = "MiB";
    } else if (size > 1024) {
        *size_modifier = 1024;
        *unit = "KiB";
    } else {
        *size_modifier = 1;
        *unit = "B";
    }
}

// Print transfer statistics
void print_transfer_stats(size_t bytes, struct timespec start, struct timespec end) {
    double total_time = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) /
    ↪  1e9;
    double speed = bytes / total_time;
    char* speed_unit, *size_unit;
    size_t speed_modifier, size_modifier;

    reduce_unit(speed, &speed_unit, &speed_modifier);
    reduce_unit(bytes, &size_unit, &size_modifier);

    printf("\n============ STATISTICS ============\n");
    printf("Total transfer time: %.3f s\n", total_time);
    printf("Total transferred bytes: %.2f %s\n", (double)bytes / size_modifier,
    ↪  size_unit);
    printf("Transfer speed: %.2f %s/s\n", speed / speed_modifier, speed_unit);
    printf("====================================\n\n");
}

int main(int argc, char **argv) {
    if (argc ≠ 2) {
        printf("Usage: %s <FTP URL>\n", argv[0]);
        return 1;
    }

    char *url = argv[1];
    FtpUrl ftp_url;
    size_t filesize;

    if (parse_url(url, &ftp_url)) {
        return 1;
    }
```

```c
    struct hostent *host = gethostbyname(ftp_url.domain);
    if (host == NULL) {
        print_error(__func__, "Could not resolve host");
        return 1;
    }

    Message message;
    int control_sockfd = get_socket_fd_host(host, FTP_PORT);
    if (control_sockfd < 0) {
        print_error(__func__, "get_socket_fd_addr() failed");
        return 1;
    }
    if (read_end(control_sockfd, &message)
        || check_code(&message, 220, NULL)) {
        return 1;
    }

    if (send_command(control_sockfd, "USER %s", ftp_url.username)
        || read_end(control_sockfd, &message)
        || check_code(&message, 331, NULL)) {
        return 1;
    }

    if (send_command(control_sockfd, "PASS %s", ftp_url.password)
        || read_end(control_sockfd, &message)
        || check_code(&message, 230, NULL)) {
        return 1;
    }

    if (send_command(control_sockfd, "TYPE I")
        || read_end(control_sockfd, &message)
        || check_code(&message, 200, NULL)) {
        return 1;
    }

    if (send_command(control_sockfd, "SIZE %s", ftp_url.path)
        || read_end(control_sockfd, &message)
        || check_code(&message, 213, NULL)) {
        return 1;
    }
    if (sscanf(message.content, "%lu", &filesize) != 1) {
        print_error(__func__, "Invalid \"SIZE\" response");
        return 1;
    }

    if (send_command(control_sockfd, "PASV")
        || read_end(control_sockfd, &message)
        || check_code(&message, 227, NULL)) {
        return 1;
    }

    char addr[15 + 1];
    uint16_t port;
    if (parse_pasv_response(message.content, addr, &port)) {
        return 1;
    }
    int data_sockfd = get_socket_fd_addr(addr, port);
    if (data_sockfd < 0) {
        print_error(__func__, "get_socket_fd_addr() failed");
        return 1;
    }

    if (send_command(control_sockfd, "RETR %s", ftp_url.path)
        || read_end(control_sockfd, &message)
        || check_code(&message, 150, 125, NULL)) {
        return 1;
```

```
    }

    int file_fd = open(basename(ftp_url.path), O_WRONLY | O_CREAT, 0640);
    uint8_t buffer[BUFFER_LEN];
    size_t response_len, total_bytes = 0;

    struct timespec start, end;
    clock_gettime(CLOCK_MONOTONIC, &start);
    while ((response_len = read(data_sockfd, buffer, BUFFER_LEN)) > 0) {
        write(file_fd, buffer, response_len);

        total_bytes += response_len;
        print_progress(total_bytes, filesize);
    }
    clock_gettime(CLOCK_MONOTONIC, &end);

    if (close(file_fd) || close(data_sockfd)) {
        print_error(__func__, "close() failed");
        return 1;
    }

    if (read_message(control_sockfd, &message) || check_code(&message, 226)) {
        return 1;
    }

    if (send_command(control_sockfd, "QUIT")
        || read_message(control_sockfd, &message)
        || check_code(&message, 221, NULL)) {
        return 1;
    }

    if (close(control_sockfd)) {
        print_error(__func__, "close() failed");
        return 1;
    }

    print_transfer_stats(total_bytes, start, end);

    return 0;
}
```

A.1.2 Download Example

```
root@tux63:/media/root/rcom/proj2/ftp-app/bin# ./download
↪  ftp://rcom:rcom@ftp.netlab.fe.up.pt/pipe.txt
Connected to 172.16.1.10:21
    220 ProFTPD Server (Debian) [::ffff:172.16.1.10]
  > USER rcom
    331 Password required for rcom
  > PASS rcom
    230-Welcome, archive user rcom@172.16.1.61 !

     The local time is: Wed Dec 11 20:52:08 2024

     This is an experimental FTP server.  If you have any unusual problems,
     please report them via e-mail to <root@ftp.netlab.fe.up.pt>.

    230 User rcom logged in
  > TYPE I
    200 Type set to I
  > SIZE pipe.txt
    213 418
  > PASV
    227 Entering Passive Mode (172,16,1,10,167,15).
Connected to 172.16.1.10:42767
  > RETR pipe.txt
    150 Opening BINARY mode data connection for pipe.txt (418 bytes)
Download complete!  [=======================] 100.0%
    226 Transfer complete
  > QUIT
    221 Goodbye.

============ STATISTICS ============
Total transfer time: 0.003 s
Total transferred bytes: 418.00 B
Transfer speed: 158.33 KiB/s
====================================
```

Figure 3: Terminal usage and output example of download application

A.2 Experiment 1

A.2.1 Experiment 1 Commands

```
// tuxY3
ifconfig eth1 up
ifconfig eth1 172.16.60.1/24
ifconfig                // Verify IP -> 172.16.60.1 & MAC -> 00:c0:df:25:40:66

// tuxY4
ifconfig eth1 up
ifconfig eth1 172.16.60.254/24
ifconfig                // Verify IP -> 172.16.60.254 & MAC -> 00:01:02:a1:35:69
ping 172.16.60.1        // Exists connectivity

// tuxY3 and tuxY4
route -n                // Check routes
arp -a                  // Check ARP table

// tuxY3
arp -d 172.16.60.254
ping 172.16.60.254      // Exists connectivity, first packet takes more time
```

Figure 4: Experiment 1 Commands

A.2.2 Experiment 1 Logs

11	20.021286229	Routerboardc_1c:8b:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
12	21.781563951	KYE_25:40:66	Broadcast	ARP	42 Who has 172.16.60.254? Tell 172.16.60.1
13	21.781663196	3Com_a1:35:69	KYE_25:40:66	ARP	60 172.16.60.254 is at 00:01:02:a1:35:69
14	21.781680865	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x0a60, seq=1/256, ttl=64 (reply in 15)
15	21.781785208	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x0a60, seq=1/256, ttl=64 (request in 14)
16	22.023346968	Routerboardc_1c:8b:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
17	22.782378983	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x0a60, seq=2/512, ttl=64 (reply in 18)
18	22.782478576	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x0a60, seq=2/512, ttl=64 (request in 17)
19	23.806376430	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x0a60, seq=3/768, ttl=64 (reply in 20)
20	23.806473509	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x0a60, seq=3/768, ttl=64 (request in 19)
21	24.025626366	Routerboardc_1c:8b:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
22	24.830376196	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x0a60, seq=4/1024, ttl=64 (reply in 23)
23	24.830472508	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x0a60, seq=4/1024, ttl=64 (request in 22)
24	25.854381146	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x0a60, seq=5/1280, ttl=64 (reply in 25)
25	25.854478295	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x0a60, seq=5/1280, ttl=64 (request in 24)
26	26.027909662	Routerboardc_1c:8b:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001

Figure 5: Experiment 1 Logs

A.3 Experiment 2

A.3.1 Experiment 2 Commands

```
// tuxY3
ifconfig eth1 up
ifconfig eth1 172.16.61.1/24
ifconfig                // Verify IP -> 172.16.61.1 & MAC -> 00:e0:7d:b5:8c:8e

// Switch Console
/interface bridge add name=bridge60
/interface bridge add name=bridge61
/interface bridge port remove [find interface=ether1]
/interface bridge port remove [find interface=ether8]
/interface bridge port remove [find interface=ether9]
/interface bridge port add bridge=bridge60 interface=ether1
/interface bridge port add bridge=bridge60 interface=ether8
/interface bridge port add bridge=bridge61 interface=ether9
/interface bridge port print brief  // Verify ports

// tuxY3
ping 172.16.60.254      // No packet loss
ping 172.16.61.1        // All packets lost
ping -b 172.16.60.255   // tuxY3 and tuxY4 receive packets with empty response; tuxY2
↪  does not

// tuxY2
ping -b 172.16.61.255   // tuxY2 receives packets with empty response; tuxY3 and tuxY4 do
↪  not
```

Figure 6: Experiment 2 Commands

A.3.2 Experiment 2 Logs

3	3.993994636	Routerboardc_1c:8b:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
4	4.037812599	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x0bed, seq=1/256, ttl=64 (reply in 5)
5	4.037964085	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x0bed, seq=1/256, ttl=64 (request in 4)
6	5.069935030	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x0bed, seq=2/512, ttl=64 (reply in 7)
7	5.070071640	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x0bed, seq=2/512, ttl=64 (request in 6)
8	5.996435760	Routerboardc_1c:8b:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
9	6.093936929	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x0bed, seq=3/768, ttl=64 (reply in 10)
10	6.094068440	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x0bed, seq=3/768, ttl=64 (request in 9)
11	7.117938272	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x0bed, seq=4/1024, ttl=64 (reply in 12)
12	7.118076768	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x0bed, seq=4/1024, ttl=64 (request in 11)
13	7.998888208	Routerboardc_1c:8b:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
14	8.141934240	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x0bed, seq=5/1280, ttl=64 (reply in 15)
15	8.142065402	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x0bed, seq=5/1280, ttl=64 (request in 14)
16	9.165938312	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x0bed, seq=6/1536, ttl=64 (reply in 17)
17	9.166099366	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x0bed, seq=6/1536, ttl=64 (request in 16)
18	9.261901014	KYE_25:40:66	3Com_a1:35:69	ARP	42 Who has 172.16.60.254? Tell 172.16.60.1
19	9.262016741	3Com_a1:35:69	KYE_25:40:66	ARP	60 172.16.60.254 is at 00:01:02:a1:35:69
20	10.001334170	Routerboardc_1c:8b:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
21	10.189938125	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x0bed, seq=7/1792, ttl=64 (reply in 22)
22	10.190079973	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x0bed, seq=7/1792, ttl=64 (request in 21)
23	11.213932145	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x0bed, seq=8/2048, ttl=64 (reply in 24)
24	11.214074831	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x0bed, seq=8/2048, ttl=64 (request in 23)
25	11.993373068	Routerboardc_1c:8b:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
26	12.237931824	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x0bed, seq=9/2304, ttl=64 (reply in 27)
27	12.238068993	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x0bed, seq=9/2304, ttl=64 (request in 26)
28	13.995298311	Routerboardc_1c:8b:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
29	15.391978930	3Com_a1:35:69	KYE_25:40:66	ARP	60 Who has 172.16.60.1? Tell 172.16.60.254
30	15.391997508	KYE_25:40:66	3Com_a1:35:69	ARP	42 172.16.60.1 is at 00:c0:df:25:40:66
31	15.997745978	Routerboardc_1c:8b:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001

Figure 7: Experiment 2, Steps 4-6 Logs

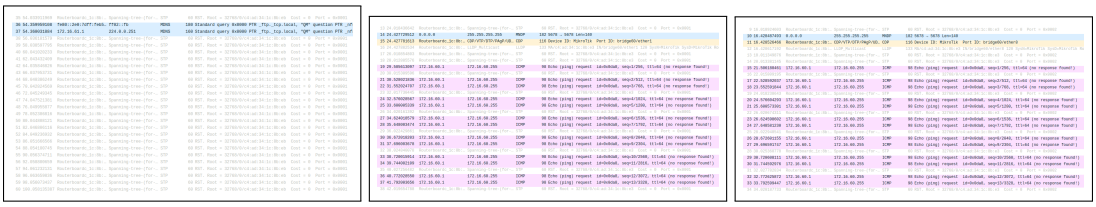


Figure 8: Experiment 2, Steps 7-9 Logs, for tux62, tux63 and tux64, respectively

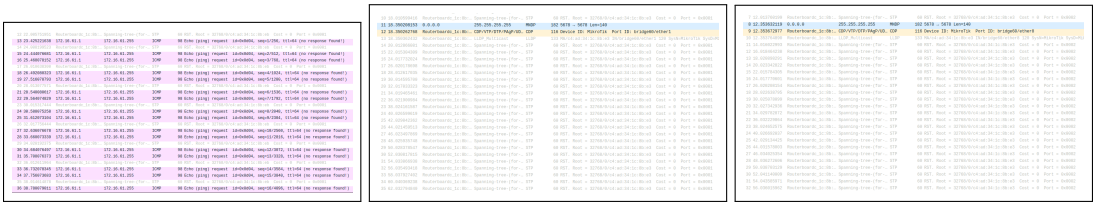


Figure 9: Experiment 2, Steps 10 Logs, for tux62, tux63 and tux64, respectively

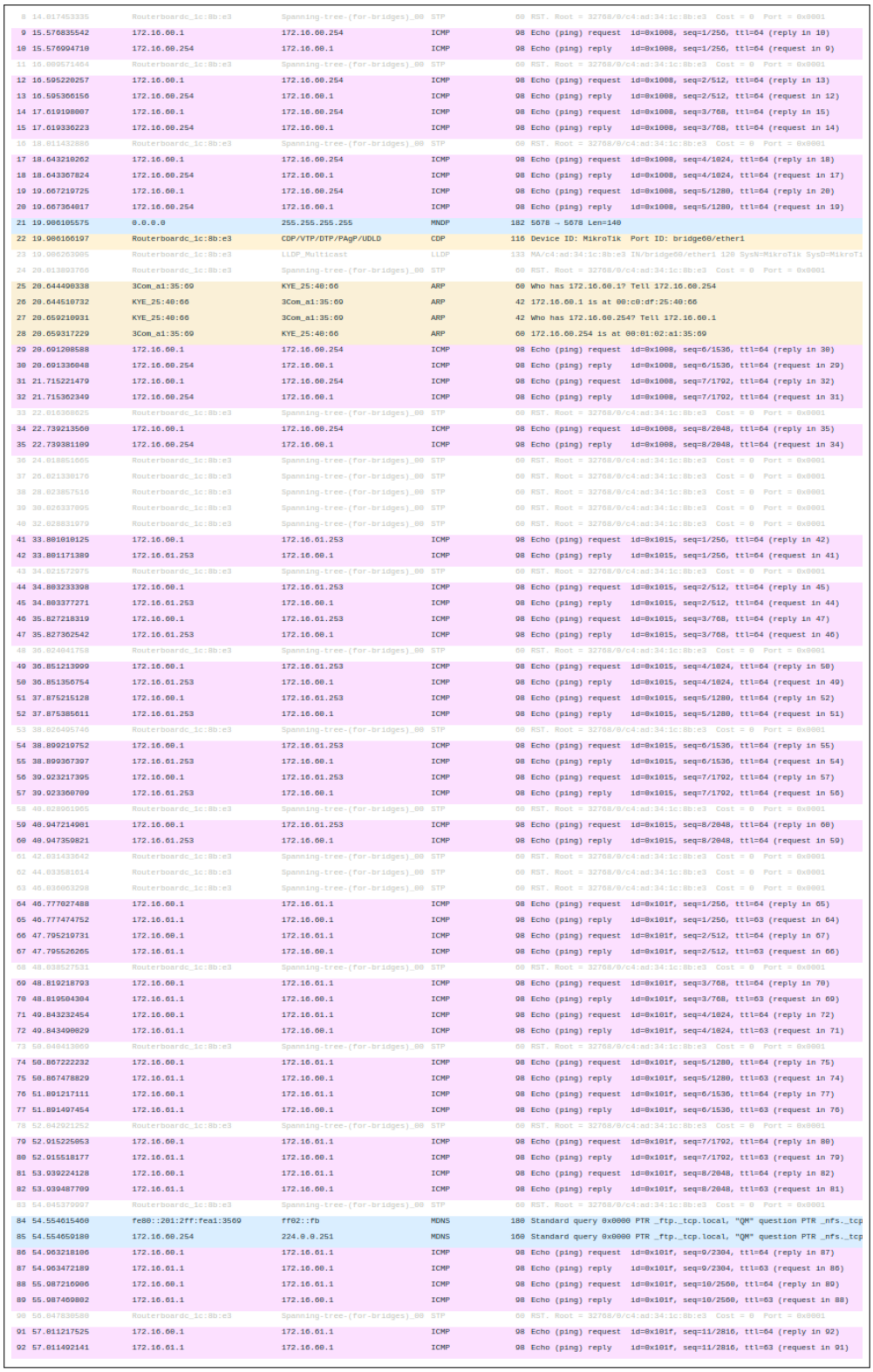
A.4 Experiment 3

A.4.1 Experiment 3 Commands

```
// tuxY4
ifconfig eth2 up
ifconfig eth2 172.16.61.253/24

// Switch Console
/interface bridge port remove [find interface=ether16]
/interface bridge port add bridge=bridge61 interface=ether16
/interface bridge port print brief  // Verify ports

// tuxY4
sysctl net.ipv4.ip_forward=1
sysctl net.ipv4.icmp_echo_ignore_broadcasts=0
ifconfig
// Verify eth1 IP -> 172.16.60.254 & MAC -> 00:01:02:a1:35:69
// Verify eth2 IP -> 172.16.61.253 & MAC -> 00:c0:df:04:20:8c

// tuxY2
route add -net 172.16.60.0/24 gw 172.16.61.253

// tuxY3
route add -net 172.16.61.0/24 gw 172.16.60.254
ping 172.16.61.1

route -n  // tuxY2, route to 172.16.60.0/24 with 172.16.61.253 as the gateway
route -n  // tuxY3, route to 172.16.61.0/24 with 172.16.60.254 as the gateway
route -n  // tuxY4, routes to 172.16.60.0/24 and 172.16.61.0/24 through default gateway

// tuxY3
ping 172.16.60.254  // Exists connectivity
ping 172.16.61.253  // Exists connectivity
ping 172.16.61.1    // Exists connectivity

// tuxY4
arp -d 172.16.60.1
arp -d 172.16.61.1
arp -a  // Verify ARP table

// tuxY2
arp -d 172.16.61.254
arp -a  // Verify ARP table

// tuxY3
arp -d 172.16.60.254
arp -a  // Verify ARP table
ping 172.16.61.1    // Exists connectivity, first packet takes more time
```

Figure 10: Experiment 3 Commands

A.4.2 Experiment 3 Logs



8	14.017453335	Routerboardc_1c:8b:e3	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
9	15.576835542	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x1008, seq=1/256, ttl=64 (reply in 10)
10	15.576994710	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1008, seq=1/256, ttl=64 (request in 9)
11	16.009571464	Routerboardc_1c:8b:e3	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
12	16.595220257	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x1008, seq=2/512, ttl=64 (reply in 13)
13	16.595366156	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1008, seq=2/512, ttl=64 (request in 12)
14	17.619198007	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x1008, seq=3/768, ttl=64 (reply in 15)
15	17.619336223	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1008, seq=3/768, ttl=64 (request in 14)
16	18.011432886	Routerboardc_1c:8b:e3	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
17	18.643216262	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x1008, seq=4/1024, ttl=64 (reply in 18)
18	18.643367824	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1008, seq=4/1024, ttl=64 (request in 17)
19	19.667219725	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x1008, seq=5/1280, ttl=64 (reply in 20)
20	19.667364017	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1008, seq=5/1280, ttl=64 (request in 19)
21	19.906105575	0.0.0.0	255.255.255.255	MNDP	182 5678 → 5678 Len=140
22	19.906166197	Routerboardc_1c:8b:e3	CDP/VTP/DTP/PAgP/UDLD	CDP	116 Device ID: MikroTik Port ID: bridge60/ether1
23	19.906263005	Routerboardc_1c:8b:e3	LLDP_Multicast	LLDP	133 MA/c4:ad:34:1c:8b:e3 IN/bridge60/ether1 120 SysN=MikroTik SysD=MikroTi
24	20.013893766	Routerboardc_1c:8b:e3	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
25	20.644498338	3Com_a1:35:69	KYE_25:40:66	ARP	60 Who has 172.16.60.1? Tell 172.16.60.254
26	20.644510732	KYE_25:40:66	3Com_a1:35:69	ARP	42 172.16.60.1 is at 00:c0:df:25:40:66
27	20.659210031	KYE_25:40:66	3Com_a1:35:69	ARP	42 Who has 172.16.60.254? Tell 172.16.60.1
28	20.659317229	3Com_a1:35:69	KYE_25:40:66	ARP	60 172.16.60.254 is at 00:01:02:a1:35:69
29	20.691208588	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x1008, seq=6/1536, ttl=64 (reply in 30)
30	20.691336048	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1008, seq=6/1536, ttl=64 (request in 29)
31	21.715221479	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x1008, seq=7/1792, ttl=64 (reply in 32)
32	21.715362349	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1008, seq=7/1792, ttl=64 (request in 31)
33	22.016368625	Routerboardc_1c:8b:e3	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
34	22.739213560	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x1008, seq=8/2048, ttl=64 (reply in 35)
35	22.739381109	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1008, seq=8/2048, ttl=64 (request in 34)
36	24.018851665	Routerboardc_1c:8b:e3	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
37	26.021338176	Routerboardc_1c:8b:e3	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
38	28.023857516	Routerboardc_1c:8b:e3	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
39	30.026337095	Routerboardc_1c:8b:e3	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
40	32.028831979	Routerboardc_1c:8b:e3	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
41	33.801010125	172.16.60.1	172.16.61.253	ICMP	98 Echo (ping) request id=0x1015, seq=1/256, ttl=64 (reply in 42)
42	33.801171389	172.16.61.253	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1015, seq=1/256, ttl=64 (request in 41)
43	34.021572975	Routerboardc_1c:8b:e3	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
44	34.803233398	172.16.60.1	172.16.61.253	ICMP	98 Echo (ping) request id=0x1015, seq=2/512, ttl=64 (reply in 45)
45	34.803377271	172.16.61.253	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1015, seq=2/512, ttl=64 (request in 44)
46	35.827218319	172.16.60.1	172.16.61.253	ICMP	98 Echo (ping) request id=0x1015, seq=3/768, ttl=64 (reply in 47)
47	35.827362542	172.16.61.253	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1015, seq=3/768, ttl=64 (request in 46)
48	36.024041758	Routerboardc_1c:8b:e3	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
49	36.851213999	172.16.60.1	172.16.61.253	ICMP	98 Echo (ping) request id=0x1015, seq=4/1024, ttl=64 (reply in 50)
50	36.851356754	172.16.61.253	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1015, seq=4/1024, ttl=64 (request in 49)
51	37.875215128	172.16.60.1	172.16.61.253	ICMP	98 Echo (ping) request id=0x1015, seq=5/1280, ttl=64 (reply in 52)
52	37.875385611	172.16.61.253	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1015, seq=5/1280, ttl=64 (request in 51)
53	38.026495746	Routerboardc_1c:8b:e3	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
54	38.899219752	172.16.60.1	172.16.61.253	ICMP	98 Echo (ping) request id=0x1015, seq=6/1536, ttl=64 (reply in 55)
55	38.899367307	172.16.61.253	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1015, seq=6/1536, ttl=64 (request in 54)
56	39.923217395	172.16.60.1	172.16.61.253	ICMP	98 Echo (ping) request id=0x1015, seq=7/1792, ttl=64 (reply in 57)
57	39.923360709	172.16.61.253	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1015, seq=7/1792, ttl=64 (request in 56)
58	40.028961965	Routerboardc_1c:8b:e3	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
59	40.947214001	172.16.60.1	172.16.61.253	ICMP	98 Echo (ping) request id=0x1015, seq=8/2048, ttl=64 (reply in 60)
60	40.947359821	172.16.61.253	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1015, seq=8/2048, ttl=64 (request in 59)
61	42.031433642	Routerboardc_1c:8b:e3	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
62	44.033581614	Routerboardc_1c:8b:e3	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
63	46.036063298	Routerboardc_1c:8b:e3	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
64	46.777027488	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request id=0x101f, seq=1/256, ttl=64 (reply in 65)
65	46.777474752	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply id=0x101f, seq=1/256, ttl=63 (request in 64)
66	47.795219731	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request id=0x101f, seq=2/512, ttl=64 (reply in 67)
67	47.795526265	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply id=0x101f, seq=2/512, ttl=63 (request in 66)
68	48.038527531	Routerboardc_1c:8b:e3	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
69	48.819218793	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request id=0x101f, seq=3/768, ttl=64 (reply in 70)
70	48.819504304	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply id=0x101f, seq=3/768, ttl=63 (request in 69)
71	49.843232454	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request id=0x101f, seq=4/1024, ttl=64 (reply in 72)
72	49.843490829	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply id=0x101f, seq=4/1024, ttl=63 (request in 71)
73	50.040413069	Routerboardc_1c:8b:e3	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
74	50.867222232	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request id=0x101f, seq=5/1280, ttl=64 (reply in 75)
75	50.867478829	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply id=0x101f, seq=5/1280, ttl=63 (request in 74)
76	51.891217111	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request id=0x101f, seq=6/1536, ttl=64 (reply in 77)
77	51.891497454	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply id=0x101f, seq=6/1536, ttl=63 (request in 76)
78	52.042021252	Routerboardc_1c:8b:e3	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
79	52.915225053	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request id=0x101f, seq=7/1792, ttl=64 (reply in 80)
80	52.915518177	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply id=0x101f, seq=7/1792, ttl=63 (request in 79)
81	53.939224128	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request id=0x101f, seq=8/2048, ttl=64 (reply in 82)
82	53.939487709	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply id=0x101f, seq=8/2048, ttl=63 (request in 81)
83	54.045379997	Routerboardc_1c:8b:e3	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
84	54.554615460	fe80::201:2ff:fea1:3569	ff02::fb	MDNS	180 Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR _nfs._tcp
85	54.554659180	172.16.60.254	224.0.0.251	MDNS	160 Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR _nfs._tcp
86	54.963218106	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request id=0x101f, seq=9/2304, ttl=64 (reply in 87)
87	54.963472189	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply id=0x101f, seq=9/2304, ttl=63 (request in 86)
88	55.987216006	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request id=0x101f, seq=10/2560, ttl=64 (reply in 89)
89	55.987460802	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply id=0x101f, seq=10/2560, ttl=63 (request in 88)
90	56.047830580	Routerboardc_1c:8b:e3	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
91	57.011217525	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request id=0x101f, seq=11/2816, ttl=64 (reply in 92)
92	57.011492141	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply id=0x101f, seq=11/2816, ttl=63 (request in 91)

Figure 11: Experiment 3, steps 5-7 logs

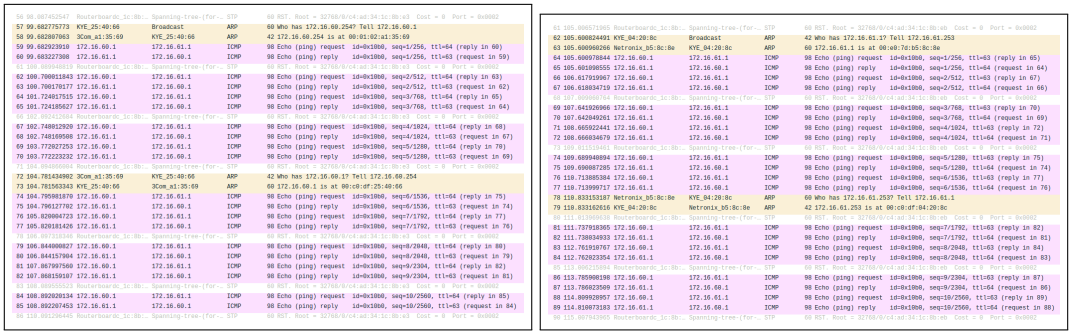


Figure 12: Experiment 3, steps 8-11 logs, captured on tux64's eth1 and eth2 interfaces, respectively

A.4.3 Experiment 3 Routes

```
root@tux62:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.227.20.254   0.0.0.0         UG    0      0        0 eth0
10.227.20.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
172.16.60.0     172.16.61.253   255.255.255.0   UG    0      0        0 eth1
172.16.61.0     0.0.0.0         255.255.255.0   U     0      0        0 eth1
```

Figure 13: Experiment 3 Tux62 routes

```
root@tux63:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.227.20.254   0.0.0.0         UG    0      0        0 eth0
10.227.20.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
172.16.60.0     0.0.0.0         255.255.255.0   U     0      0        0 eth1
172.16.61.0     172.16.60.254   255.255.255.0   UG    0      0        0 eth1
```

Figure 14: Experiment 3 Tux63 routes

```
root@tux64:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.227.20.254   0.0.0.0         UG    0      0        0 eth0
10.227.20.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
172.16.60.0     0.0.0.0         255.255.255.0   U     0      0        0 eth1
172.16.61.0     0.0.0.0         255.255.255.0   U     0      0        0 eth2
```

Figure 15: Experiment 3 Tux64 routes

A.5 Experiment 4

A.5.1 Experiment 4 Commands

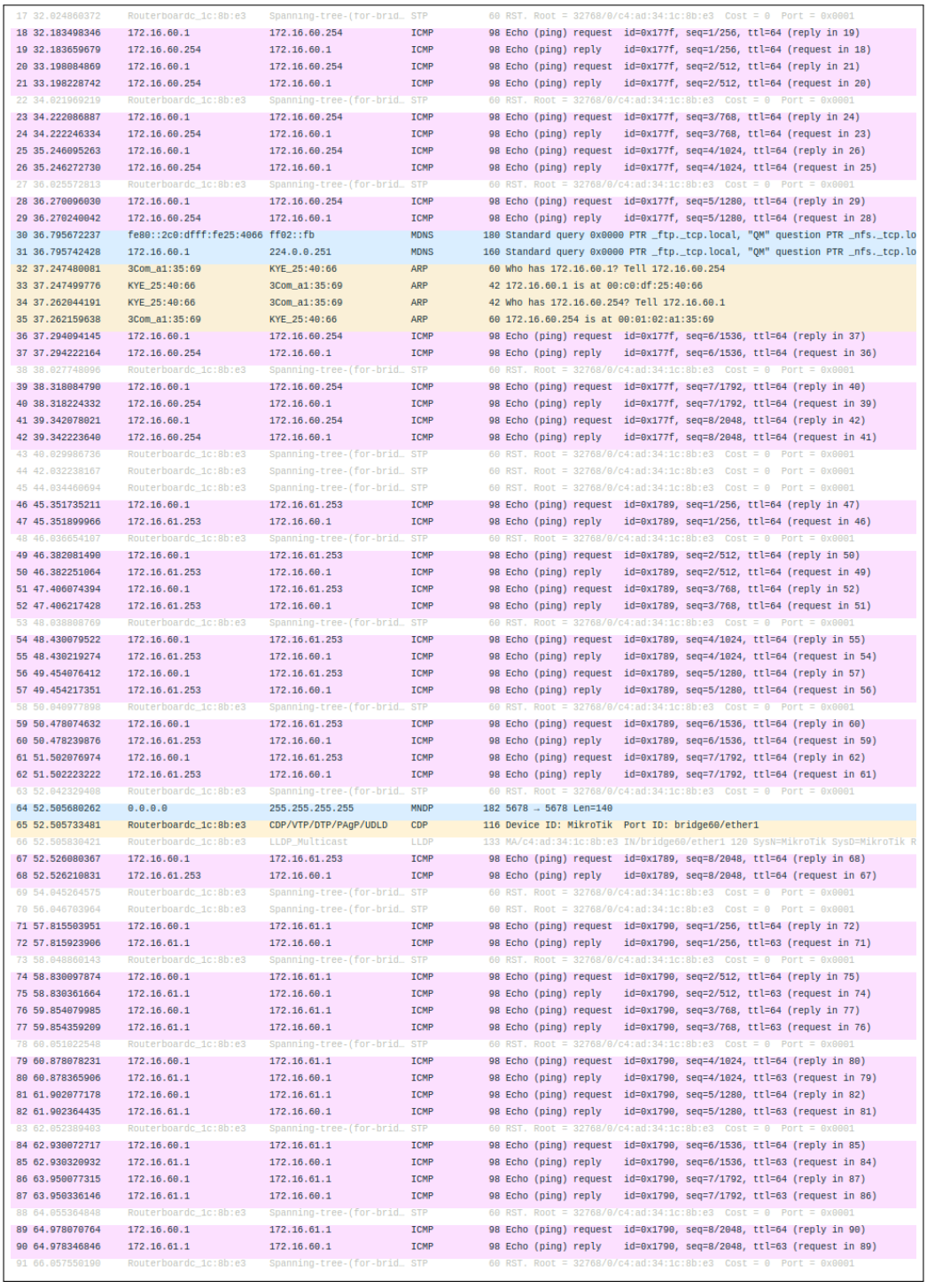
```
// Switch Console
/interface bridge port remove [find interface=ether12]
/interface bridge port add bridge=bridge61 interface=ether12
/interface bridge port print brief  // Verify ports

// Router Console
/ip address add address=172.16.1.61/24 interface=ether1
/ip address add address=172.16.61.254/24 interface=ether2
/ip address print  // Verify addresses

// tuxY3
route add -net 172.16.61.0/24 gw 172.16.60.254
route add -net 172.16.1.0/24 gw 172.16.60.254
route -n  // Verify routes

// tuxY4
route add -net 172.16.1.0/24 gw 172.16.61.254
route -n  // Verify routes

// tuxY2
route add -net 172.16.60.0/24 gw 172.16.61.253
route add -net 172.16.1.0/24 gw 172.16.61.254
route -n  // Verify routes

// Router Console
/ip route add dst-address=172.16.60.0/24 gateway=172.16.61.253
/ip route print  // Verify routes

// tuxY3
ping 172.16.60.254  // Exists
ping 172.16.61.253
ping 172.16.61.1
ping 172.16.61.254
ping 172.16.1.61
// Exists connectivity in all of them
```

Figure 16: Experiment 4 Setup Commands

```
// tuxY2
sysctl net.ipv4.conf.eth1.accept_redirects=0
sysctl net.ipv4.conf.all.accept_redirects=0
route del -net 172.16.60.0/24 gw 172.16.61.253
route add -net 172.16.60.0/24 gw 172.16.61.254
route -n  // Verify routes
ping 172.16.60.1  // Exists connectivity, with multiple "Redirect Host" messages
traceroute 172.16.60.1  // 172.16.61.254 -> 172.16.61.253 -> 172.16.60.1

route del -net 172.16.60.0/24 gw 172.16.61.254
route add -net 172.16.60.0/24 gw 172.16.61.253
route -n  // Verify routes
traceroute 172.16.60.1  // 172.16.61.253 -> 172.16.60.1

route del -net 172.16.60.0/24 gw 172.16.61.253
route add -net 172.16.60.0/24 gw 172.16.61.254
route -n  // Verify routes
sysctl net.ipv4.conf.eth1.accept_redirects=1
sysctl net.ipv4.conf.all.accept_redirects=1
ping 172.16.60.1  // Exists connectivity, only one "Redirect Host"
traceroute 172.16.60.1  // 172.16.61.253 -> 172.16.60.1

// tuxY3
ping 172.16.1.10  // Exists connectivity

// Router Console
/ip firewall nat disable 0

// tuxY3
ping 172.16.1.10  // No connectivity

// Route Console
/ip firewall nat enable 0  // To re-enable NAT
```

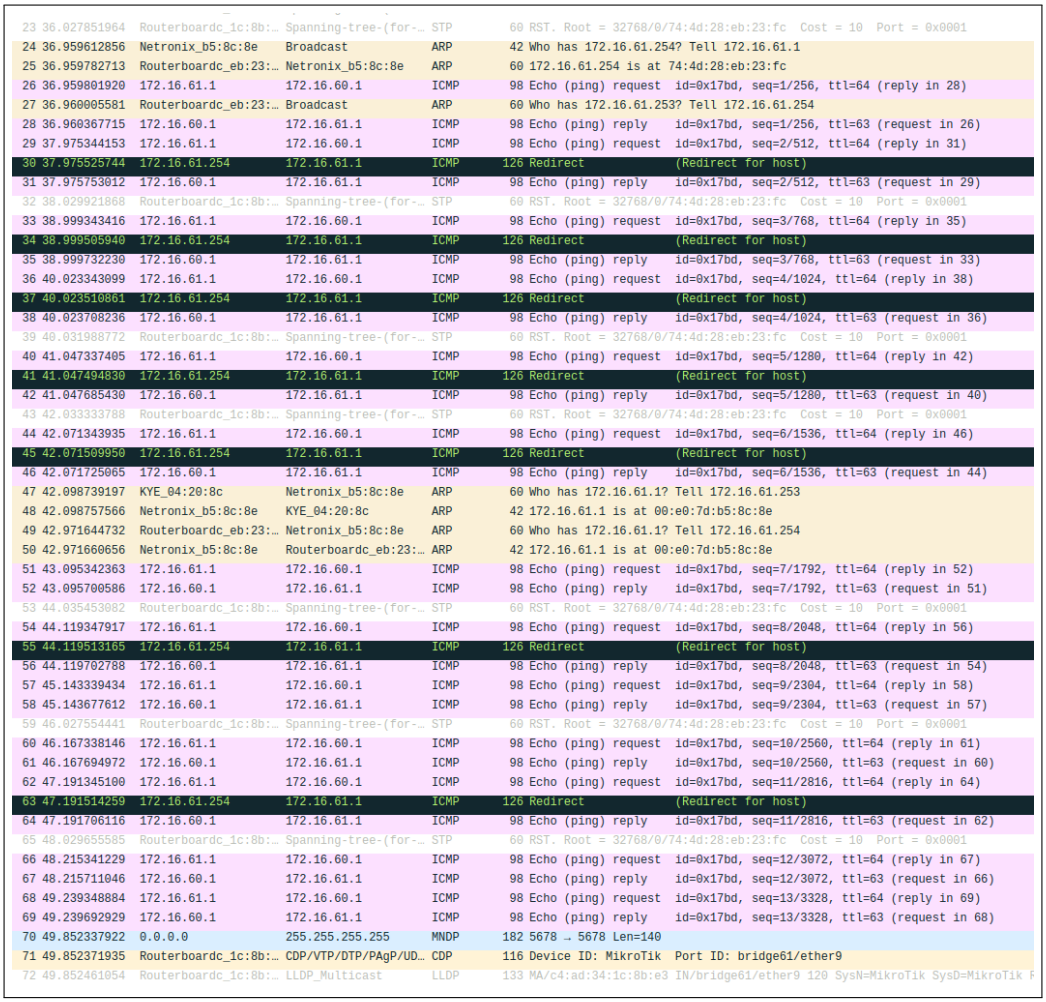
Figure 17: Experiment 4 Testing Commands

A.5.2 Experiment 4 Logs



17	32.024860372	Routerboardc_1c:8b:e3	Spanning-tree-(for-brid…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
18	32.183498346	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x177f, seq=1/256, ttl=64 (reply in 19)
19	32.183659679	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x177f, seq=1/256, ttl=64 (request in 18)
20	33.198084869	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x177f, seq=2/512, ttl=64 (reply in 21)
21	33.198228742	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x177f, seq=2/512, ttl=64 (request in 20)
22	34.021969219	Routerboardc_1c:8b:e3	Spanning-tree-(for-brid…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
23	34.222086887	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x177f, seq=3/768, ttl=64 (reply in 24)
24	34.222246334	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x177f, seq=3/768, ttl=64 (request in 23)
25	35.246095263	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x177f, seq=4/1024, ttl=64 (reply in 26)
26	35.246272730	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x177f, seq=4/1024, ttl=64 (request in 25)
27	36.025572813	Routerboardc_1c:8b:e3	Spanning-tree-(for-brid…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
28	36.270096030	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x177f, seq=5/1280, ttl=64 (reply in 29)
29	36.270240042	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x177f, seq=5/1280, ttl=64 (request in 28)
30	36.795672237	fe80::2c0:dfff:fe25:4066	ff02::fb	MDNS	180 Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR _nfs._tcp.lo
31	36.795742428	172.16.60.1	224.0.0.251	MDNS	160 Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR _nfs._tcp.lo
32	37.247480081	3Com_a1:35:69	KYE_25:40:66	ARP	60 Who has 172.16.60.1? Tell 172.16.60.254
33	37.247499776	KYE_25:40:66	3Com_a1:35:69	ARP	42 172.16.60.1 is at 00:c0:df:25:40:66
34	37.262044191	KYE_25:40:66	3Com_a1:35:69	ARP	42 Who has 172.16.60.254? Tell 172.16.60.1
35	37.262159638	3Com_a1:35:69	KYE_25:40:66	ARP	60 172.16.60.254 is at 00:01:02:a1:35:69
36	37.294094145	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x177f, seq=6/1536, ttl=64 (reply in 37)
37	37.294222164	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x177f, seq=6/1536, ttl=64 (request in 36)
38	38.027748096	Routerboardc_1c:8b:e3	Spanning-tree-(for-brid…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
39	38.318084790	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x177f, seq=7/1792, ttl=64 (reply in 40)
40	38.318224332	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x177f, seq=7/1792, ttl=64 (request in 39)
41	39.342078021	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x177f, seq=8/2048, ttl=64 (reply in 42)
42	39.342223640	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x177f, seq=8/2048, ttl=64 (request in 41)
43	40.029986736	Routerboardc_1c:8b:e3	Spanning-tree-(for-brid…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
44	42.032238167	Routerboardc_1c:8b:e3	Spanning-tree-(for-brid…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
45	44.034460694	Routerboardc_1c:8b:e3	Spanning-tree-(for-brid…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
46	45.351735211	172.16.60.1	172.16.61.253	ICMP	98 Echo (ping) request id=0x1789, seq=1/256, ttl=64 (reply in 47)
47	45.351899966	172.16.61.253	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1789, seq=1/256, ttl=64 (request in 46)
48	46.036654107	Routerboardc_1c:8b:e3	Spanning-tree-(for-brid…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
49	46.382081490	172.16.60.1	172.16.61.253	ICMP	98 Echo (ping) request id=0x1789, seq=2/512, ttl=64 (reply in 50)
50	46.382251064	172.16.61.253	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1789, seq=2/512, ttl=64 (request in 49)
51	47.406074394	172.16.60.1	172.16.61.253	ICMP	98 Echo (ping) request id=0x1789, seq=3/768, ttl=64 (reply in 52)
52	47.406217428	172.16.61.253	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1789, seq=3/768, ttl=64 (request in 51)
53	48.038808769	Routerboardc_1c:8b:e3	Spanning-tree-(for-brid…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
54	48.430079522	172.16.60.1	172.16.61.253	ICMP	98 Echo (ping) request id=0x1789, seq=4/1024, ttl=64 (reply in 55)
55	48.430219274	172.16.61.253	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1789, seq=4/1024, ttl=64 (request in 54)
56	49.454076412	172.16.60.1	172.16.61.253	ICMP	98 Echo (ping) request id=0x1789, seq=5/1280, ttl=64 (reply in 57)
57	49.454217351	172.16.61.253	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1789, seq=5/1280, ttl=64 (request in 56)
58	50.040977898	Routerboardc_1c:8b:e3	Spanning-tree-(for-brid…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
59	50.478074632	172.16.60.1	172.16.61.253	ICMP	98 Echo (ping) request id=0x1789, seq=6/1536, ttl=64 (reply in 60)
60	50.478239876	172.16.61.253	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1789, seq=6/1536, ttl=64 (request in 59)
61	51.502076974	172.16.60.1	172.16.61.253	ICMP	98 Echo (ping) request id=0x1789, seq=7/1792, ttl=64 (reply in 62)
62	51.502223222	172.16.61.253	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1789, seq=7/1792, ttl=64 (request in 61)
63	52.042329408	Routerboardc_1c:8b:e3	Spanning-tree-(for-brid…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
64	52.505680262	0.0.0.0	255.255.255.255	MNDP	182 5678 → 5678 Len=140
65	52.505733481	Routerboardc_1c:8b:e3	CDP/VTP/DTP/PAgP/UDLD	CDP	116 Device ID: MikroTik Port ID: bridge60/ether1
66	52.505830421	Routerboardc_1c:8b:e3	LLDP_Multicast	LLDP	133 MA/c4:ad:34:1c:8b:e3 IN/bridge60/ether1 120 SysN=MikroTik SysD=MikroTik R
67	52.526080367	172.16.60.1	172.16.61.253	ICMP	98 Echo (ping) request id=0x1789, seq=8/2048, ttl=64 (reply in 68)
68	52.526210831	172.16.61.253	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1789, seq=8/2048, ttl=64 (request in 67)
69	54.045264575	Routerboardc_1c:8b:e3	Spanning-tree-(for-brid…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
70	56.046703964	Routerboardc_1c:8b:e3	Spanning-tree-(for-brid…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
71	57.815503951	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request id=0x1790, seq=1/256, ttl=64 (reply in 72)
72	57.815923906	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1790, seq=1/256, ttl=63 (request in 71)
73	58.048860143	Routerboardc_1c:8b:e3	Spanning-tree-(for-brid…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
74	58.830097874	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request id=0x1790, seq=2/512, ttl=64 (reply in 75)
75	58.830361664	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1790, seq=2/512, ttl=63 (request in 74)
76	59.854079985	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request id=0x1790, seq=3/768, ttl=64 (reply in 77)
77	59.854359209	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1790, seq=3/768, ttl=63 (request in 76)
78	60.051022548	Routerboardc_1c:8b:e3	Spanning-tree-(for-brid…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
79	60.878078231	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request id=0x1790, seq=4/1024, ttl=64 (reply in 80)
80	60.878365906	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1790, seq=4/1024, ttl=63 (request in 79)
81	61.902077178	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request id=0x1790, seq=5/1280, ttl=64 (reply in 82)
82	61.902364435	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1790, seq=5/1280, ttl=63 (request in 81)
83	62.052389403	Routerboardc_1c:8b:e3	Spanning-tree-(for-brid…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
84	62.930072717	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request id=0x1790, seq=6/1536, ttl=64 (reply in 85)
85	62.930320932	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1790, seq=6/1536, ttl=63 (request in 84)
86	63.950077315	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request id=0x1790, seq=7/1792, ttl=64 (reply in 87)
87	63.950336146	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1790, seq=7/1792, ttl=63 (request in 86)
88	64.055364848	Routerboardc_1c:8b:e3	Spanning-tree-(for-brid…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001
89	64.970070764	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request id=0x1790, seq=8/2048, ttl=64 (reply in 90)
90	64.978346846	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1790, seq=8/2048, ttl=63 (request in 89)
91	66.057550190	Routerboardc_1c:8b:e3	Spanning-tree-(for-brid…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8b:e3 Cost = 0 Port = 0x0001

Figure 18: Experiment 4, step 3 logs



23 36.027851964	Routerboardc_1c:8b:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/74:4d:28:eb:23:fc Cost = 10 Port = 0x0001
24 36.959612856	Netronix_b5:8c:8e	Broadcast	ARP	42	Who has 172.16.61.254? Tell 172.16.61.1
25 36.959782713	Routerboardc_eb:23:…	Netronix_b5:8c:8e	ARP	60	172.16.61.254 is at 74:4d:28:eb:23:fc
26 36.959801920	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) request id=0x17bd, seq=1/256, ttl=64 (reply in 28)
27 36.960005581	Routerboardc_eb:23:…	Broadcast	ARP	60	Who has 172.16.61.253? Tell 172.16.61.254
28 36.960367715	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) reply id=0x17bd, seq=1/256, ttl=63 (request in 26)
29 37.975344153	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) request id=0x17bd, seq=2/512, ttl=64 (reply in 31)
30 37.975525744	172.16.61.254	172.16.61.1	ICMP	126	Redirect (Redirect for host)
31 37.975753012	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) reply id=0x17bd, seq=2/512, ttl=63 (request in 29)
32 38.029921868	Routerboardc_1c:8b:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/74:4d:28:eb:23:fc Cost = 10 Port = 0x0001
33 38.999343416	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) request id=0x17bd, seq=3/768, ttl=64 (reply in 35)
34 38.999505940	172.16.61.254	172.16.61.1	ICMP	126	Redirect (Redirect for host)
35 38.999732230	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) reply id=0x17bd, seq=3/768, ttl=63 (request in 33)
36 40.023343099	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) request id=0x17bd, seq=4/1024, ttl=64 (reply in 38)
37 40.023510861	172.16.61.254	172.16.61.1	ICMP	126	Redirect (Redirect for host)
38 40.023708236	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) reply id=0x17bd, seq=4/1024, ttl=63 (request in 36)
39 40.031988772	Routerboardc_1c:8b:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/74:4d:28:eb:23:fc Cost = 10 Port = 0x0001
40 41.047337405	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) request id=0x17bd, seq=5/1280, ttl=64 (reply in 42)
41 41.047494830	172.16.61.254	172.16.61.1	ICMP	126	Redirect (Redirect for host)
42 41.047685430	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) reply id=0x17bd, seq=5/1280, ttl=63 (request in 40)
43 42.033333788	Routerboardc_1c:8b:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/74:4d:28:eb:23:fc Cost = 10 Port = 0x0001
44 42.071343935	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) request id=0x17bd, seq=6/1536, ttl=64 (reply in 46)
45 42.071509950	172.16.61.254	172.16.61.1	ICMP	126	Redirect (Redirect for host)
46 42.071725065	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) reply id=0x17bd, seq=6/1536, ttl=63 (request in 44)
47 42.098739197	KYE_04:20:8c	Netronix_b5:8c:8e	ARP	60	Who has 172.16.61.1? Tell 172.16.61.253
48 42.098757566	Netronix_b5:8c:8e	KYE_04:20:8c	ARP	42	172.16.61.1 is at 00:e0:7d:b5:8c:8e
49 42.971644732	Routerboardc_eb:23:…	Netronix_b5:8c:8e	ARP	60	Who has 172.16.61.1? Tell 172.16.61.254
50 42.971660656	Netronix_b5:8c:8e	Routerboardc_eb:23:…	ARP	42	172.16.61.1 is at 00:e0:7d:b5:8c:8e
51 43.095342363	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) request id=0x17bd, seq=7/1792, ttl=64 (reply in 52)
52 43.095700586	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) reply id=0x17bd, seq=7/1792, ttl=63 (request in 51)
53 44.035453082	Routerboardc_1c:8b:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/74:4d:28:eb:23:fc Cost = 10 Port = 0x0001
54 44.119347917	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) request id=0x17bd, seq=8/2048, ttl=64 (reply in 56)
55 44.119513165	172.16.61.254	172.16.61.1	ICMP	126	Redirect (Redirect for host)
56 44.119702788	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) reply id=0x17bd, seq=8/2048, ttl=63 (request in 54)
57 45.143339434	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) request id=0x17bd, seq=9/2304, ttl=64 (reply in 58)
58 45.143677612	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) reply id=0x17bd, seq=9/2304, ttl=63 (request in 57)
59 46.027554441	Routerboardc_1c:8b:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/74:4d:28:eb:23:fc Cost = 10 Port = 0x0001
60 46.167338146	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) request id=0x17bd, seq=10/2560, ttl=64 (reply in 61)
61 46.167694972	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) reply id=0x17bd, seq=10/2560, ttl=63 (request in 60)
62 47.191345100	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) request id=0x17bd, seq=11/2816, ttl=64 (reply in 64)
63 47.191514259	172.16.61.254	172.16.61.1	ICMP	126	Redirect (Redirect for host)
64 47.191706116	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) reply id=0x17bd, seq=11/2816, ttl=63 (request in 62)
65 48.029655585	Routerboardc_1c:8b:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/74:4d:28:eb:23:fc Cost = 10 Port = 0x0001
66 48.215341229	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) request id=0x17bd, seq=12/3072, ttl=64 (reply in 67)
67 48.215711046	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) reply id=0x17bd, seq=12/3072, ttl=63 (request in 66)
68 49.239348884	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) request id=0x17bd, seq=13/3328, ttl=64 (reply in 69)
69 49.239692929	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) reply id=0x17bd, seq=13/3328, ttl=63 (request in 68)
70 49.852337922	0.0.0.0	255.255.255.255	MNDP	182	5678 → 5678 Len=140
71 49.852371935	Routerboardc_1c:8b:…	CDP/VTP/DTP/PAgP/UD…	CDP	116	Device ID: MikroTik Port ID: bridge61/ether9
72 49.852461054	Routerboardc_1c:8b:…	LLDP_Multicast	LLDP	133	MA/c4:ad:34:1c:8b:e3 IN/bridge61/ether9 120 SysN=MikroTik SysD=MikroTik F

Figure 19: Experiment 4, step 4.4 logs

A.6 Experiment 5

A.6.1 Experiment 5 Commands

```
// tuxY2, tuxY3 & tuxY4
nano /etc/resolv.conf
// Write "services.netlab.fe.up.pt 10.227.20.3"
ping google.com      // Exists connectivity, IP -> 142.250.200.142
ping archlinux.org   // Exists connectivity, IP -> 95.217.163.246
```

Figure 20: Experiment 5 Commands

A.6.2 Experiment 5 Logs

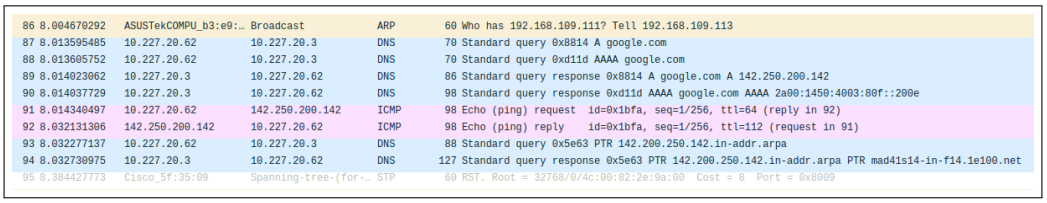


Figure 21: Experiment 5, step 3 logs

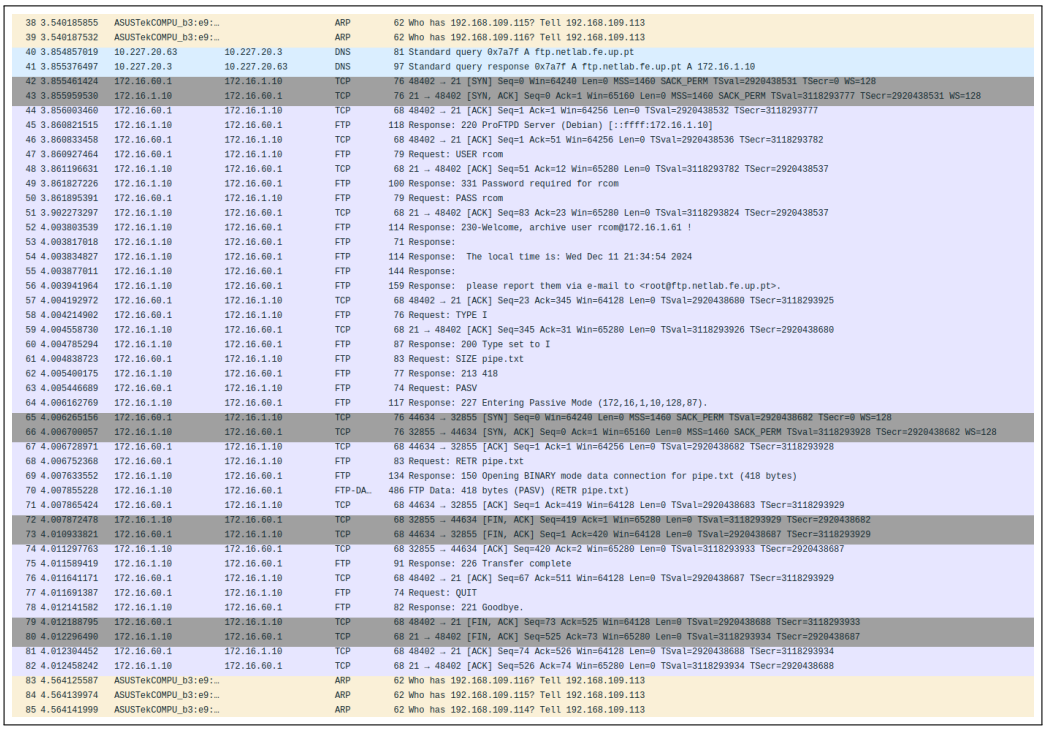
A.7 Experiment 6

A.7.1 Experiment 6 Commands

```
// tuxY3
make
cd bin
./download ftp://rcom:rcom@ftp.netlab.fe.up.pt/pipe.txt   // Download successful
./download ftp://rcom:rcom@ftp.netlab.fe.up.pt/ubuntu.iso // Download successful
```

Figure 22: Experiment 6 Commands

A.7.2 Experiment 6 Logs



38	3.540185855	ASUSTekCOMPU_b3:e9:…		ARP	62 Who has 192.168.109.115? Tell 192.168.109.113
39	3.540187532	ASUSTekCOMPU_b3:e9:…		ARP	62 Who has 192.168.109.116? Tell 192.168.109.113
40	3.854857019	10.227.20.63	10.227.20.3	DNS	81 Standard query 0x7a7f A ftp.netlab.fe.up.pt
41	3.855376497	10.227.20.3	10.227.20.63	DNS	97 Standard query response 0x7a7f A ftp.netlab.fe.up.pt A 172.16.1.10
42	3.855461424	172.16.60.1	172.16.1.10	TCP	76 48402 → 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2920438531 TSecr=0 WS=128
43	3.855959530	172.16.1.10	172.16.60.1	TCP	76 21 → 48402 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=3118293777 TSecr=2920438531 WS=128
44	3.856003460	172.16.60.1	172.16.1.10	TCP	68 48402 → 21 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2920438532 TSecr=3118293777
45	3.860821515	172.16.1.10	172.16.60.1	FTP	118 Response: 220 ProFTPD Server (Debian) [::ffff:172.16.1.10]
46	3.860833458	172.16.60.1	172.16.1.10	TCP	68 48402 → 21 [ACK] Seq=1 Ack=51 Win=64256 Len=0 TSval=2920438536 TSecr=3118293782
47	3.860927464	172.16.60.1	172.16.1.10	FTP	79 Request: USER rcom
48	3.861196631	172.16.1.10	172.16.60.1	TCP	68 21 → 48402 [ACK] Seq=51 Ack=12 Win=65280 Len=0 TSval=3118293782 TSecr=2920438537
49	3.861827226	172.16.1.10	172.16.60.1	FTP	100 Response: 331 Password required for rcom
50	3.861895391	172.16.60.1	172.16.1.10	FTP	79 Request: PASS rcom
51	3.902273297	172.16.1.10	172.16.60.1	TCP	68 21 → 48402 [ACK] Seq=83 Ack=23 Win=65280 Len=0 TSval=3118293824 TSecr=2920438537
52	4.003803539	172.16.1.10	172.16.60.1	FTP	114 Response: 230-Welcome, archive user rcom@172.16.1.61 !
53	4.003817018	172.16.1.10	172.16.60.1	FTP	71 Response:
54	4.003834827	172.16.1.10	172.16.60.1	FTP	114 Response: The local time is: Wed Dec 11 21:34:54 2024
55	4.003877011	172.16.1.10	172.16.60.1	FTP	144 Response:
56	4.003941964	172.16.1.10	172.16.60.1	FTP	159 Response: please report them via e-mail to <root@ftp.netlab.fe.up.pt>.
57	4.004192972	172.16.60.1	172.16.1.10	TCP	68 48402 → 21 [ACK] Seq=23 Ack=345 Win=64128 Len=0 TSval=2920438680 TSecr=3118293925
58	4.004214902	172.16.60.1	172.16.1.10	FTP	76 Request: TYPE I
59	4.004558730	172.16.1.10	172.16.60.1	TCP	68 21 → 48402 [ACK] Seq=345 Ack=31 Win=65280 Len=0 TSval=3118293926 TSecr=2920438680
60	4.004785294	172.16.1.10	172.16.60.1	FTP	87 Response: 200 Type set to I
61	4.004838723	172.16.60.1	172.16.1.10	FTP	83 Request: SIZE pipe.txt
62	4.005400175	172.16.1.10	172.16.60.1	FTP	77 Response: 213 418
63	4.005446689	172.16.60.1	172.16.1.10	FTP	74 Request: PASV
64	4.006162769	172.16.1.10	172.16.60.1	FTP	117 Response: 227 Entering Passive Mode (172,16,1,10,128,87).
65	4.006265156	172.16.60.1	172.16.1.10	TCP	76 44634 → 32855 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2920438682 TSecr=0 WS=128
66	4.006700057	172.16.1.10	172.16.60.1	TCP	76 32855 → 44634 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=3118293928 TSecr=2920438682 WS=128
67	4.006728971	172.16.60.1	172.16.1.10	TCP	68 44634 → 32855 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2920438682 TSecr=3118293928
68	4.006752368	172.16.60.1	172.16.1.10	FTP	83 Request: RETR pipe.txt
69	4.007633552	172.16.1.10	172.16.60.1	FTP	134 Response: 150 Opening BINARY mode data connection for pipe.txt (418 bytes)
70	4.007855228	172.16.1.10	172.16.60.1	FTP-DA…	486 FTP Data: 418 bytes (PASV) (RETR pipe.txt)
71	4.007865424	172.16.60.1	172.16.1.10	TCP	68 44634 → 32855 [ACK] Seq=1 Ack=419 Win=64128 Len=0 TSval=2920438683 TSecr=3118293929
72	4.007872478	172.16.1.10	172.16.60.1	TCP	68 32855 → 44634 [FIN, ACK] Seq=419 Ack=1 Win=65280 Len=0 TSval=3118293929 TSecr=2920438682
73	4.010933821	172.16.60.1	172.16.1.10	TCP	68 44634 → 32855 [FIN, ACK] Seq=1 Ack=420 Win=64128 Len=0 TSval=2920438687 TSecr=3118293929
74	4.011297763	172.16.1.10	172.16.60.1	TCP	68 32855 → 44634 [ACK] Seq=420 Ack=2 Win=65280 Len=0 TSval=3118293933 TSecr=2920438687
75	4.011589419	172.16.1.10	172.16.60.1	FTP	91 Response: 226 Transfer complete
76	4.011641171	172.16.60.1	172.16.1.10	TCP	68 48402 → 21 [ACK] Seq=67 Ack=511 Win=64128 Len=0 TSval=2920438687 TSecr=3118293929
77	4.011691387	172.16.60.1	172.16.1.10	FTP	74 Request: QUIT
78	4.012141582	172.16.1.10	172.16.60.1	FTP	82 Response: 221 Goodbye.
79	4.012188795	172.16.60.1	172.16.1.10	TCP	68 48402 → 21 [FIN, ACK] Seq=73 Ack=525 Win=64128 Len=0 TSval=2920438688 TSecr=3118293933
80	4.012296490	172.16.1.10	172.16.60.1	TCP	68 21 → 48402 [FIN, ACK] Seq=525 Ack=73 Win=65280 Len=0 TSval=3118293934 TSecr=2920438687
81	4.012304452	172.16.60.1	172.16.1.10	TCP	68 48402 → 21 [ACK] Seq=74 Ack=526 Win=64128 Len=0 TSval=2920438688 TSecr=3118293934
82	4.012458242	172.16.1.10	172.16.60.1	TCP	68 21 → 48402 [ACK] Seq=526 Ack=74 Win=65280 Len=0 TSval=3118293934 TSecr=2920438688
83	4.564125587	ASUSTekCOMPU_b3:e9:…		ARP	62 Who has 192.168.109.116? Tell 192.168.109.113
84	4.564139974	ASUSTekCOMPU_b3:e9:…		ARP	62 Who has 192.168.109.115? Tell 192.168.109.113
85	4.564141999	ASUSTekCOMPU_b3:e9:…		ARP	62 Who has 192.168.109.114? Tell 192.168.109.113

Figure 23: Experiment 6, step 4 logs, obtained by transferring the file `pipe.txt`

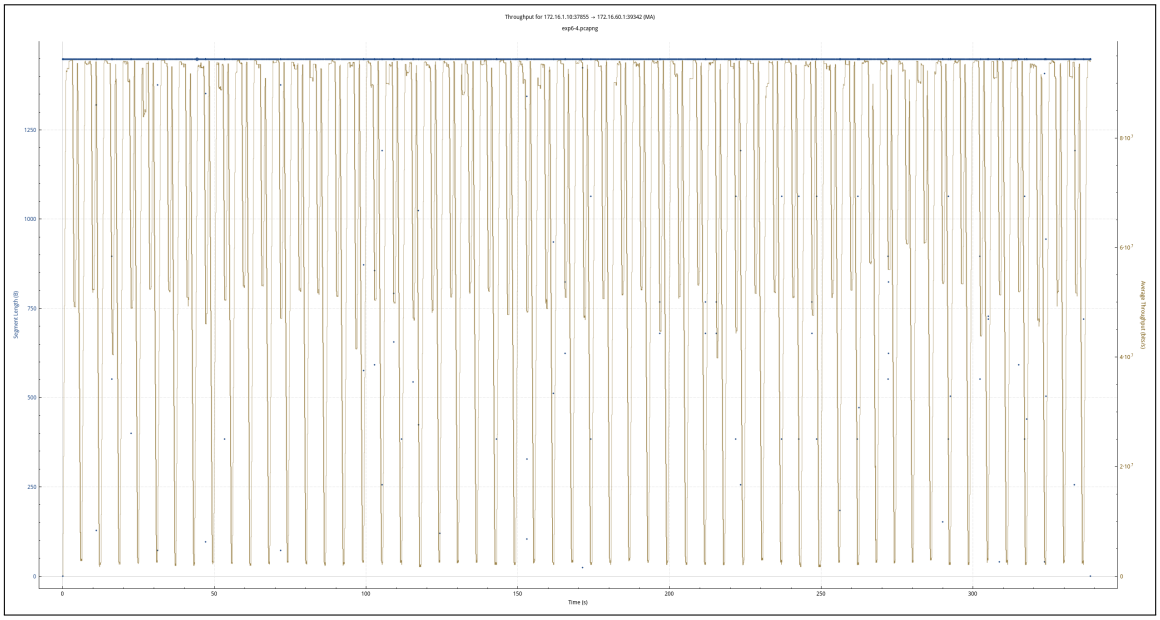


Figure 24: Throughput graph of TCP connection in experiment 6, step 4, obtained by transferring the file `ubuntu.iso` from the server

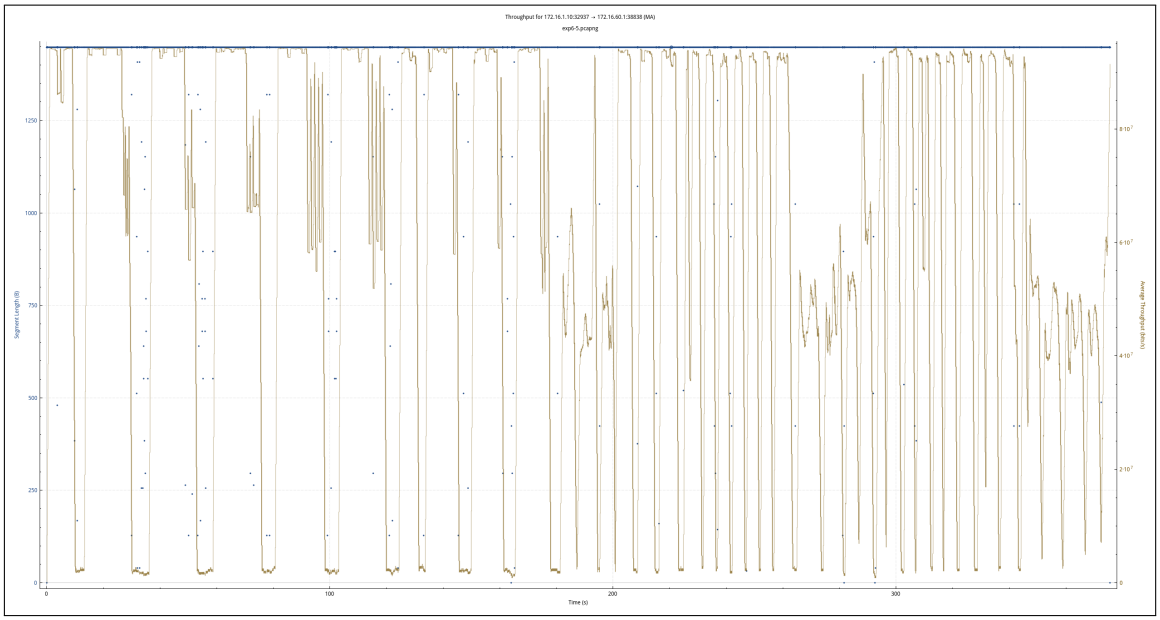


Figure 25: Throughput graph of TCP connection in experiment 6, step 5, obtained by transferring the file `ubuntu.iso` from the server in both tuxes and starting the second transfer around the 180s timestamp