

# Download application & Configuration and Study of a computer network

João António Morgado Parada ([up201405280@up.pt](mailto:up201405280@up.pt))  
Tiago Alexandre Rebelo Oliveira ([up202009302@up.pt](mailto:up202009302@up.pt))



**Redes de Computadores - 3LEIC11**

23 December 2024

# Summary

This report describes the development and analysis of an FTP client application in C. The application enables downloading files from FTP servers using standard FTP commands, including authentication, passive mode, and binary file transfer. The report also includes network configuration, packet analysis using Wireshark, and an evaluation of the application and experiments conducted during the project.

## Introduction

The primary objective of this work is to deepen understanding of network protocols and their practical applications through two main components: developing an FTP download application and conducting network experiments. The development of the FTP client emphasizes the implementation of core networking concepts such as the client-server model, socket programming, and the FTP protocol, providing hands-on experience with TCP/IP communication and protocol behaviors outlined in RFC standards.

The experiments complement this by exploring real-world network configurations, including routing, NAT, and bridge implementations, while analyzing packet flows and protocol interactions using tools like Wireshark.

## Part 1 - Download application

### Architecture of the Download Application

The FTP client application is designed to allow users to download files from FTP servers using the File Transfer Protocol (FTP) as defined in RFC 959. The architecture of the application consists of the following key components:

- **DNS Resolution:** The hostname of the FTP server is resolved to an IP address using the `gethostbyname` function.
- **Control Connection:** A TCP control connection is established with the server on port 21. This connection is used to send FTP commands (`USER`, `PASS`, `PASV`, `RETR`) and receive server responses.
- **Data Connection:** The client switches to passive mode (`PASV` command), and a second TCP connection is established to transfer file data. The data connection uses the IP and port provided by the server in response to the `PASV` command.
- **File Transfer:** The client sends the `RETR` command to initiate the file download. Files are downloaded in binary mode (`TYPE I`) and saved locally.

### Report of a successful download

To download a file the user has to use the command `download {ftp_url}`, where `{ftp_url}` can have one of the following formats:

- `ftp://<user>:<password>@<host>/<path>;`
- `ftp://<host>/<path>;`

In the case of the anonymous login, the application assumes anonymous login credentials  
`user = anonymous, password = guest`

Here is the screenshot of the Wireshark logs showing the FTP packets captured when we run the command `download ftp://rcom:rcom@ftp.netlab.fe.up.pt/pipe.txt`

192	195.558014799	172.16.1.10	172.16.10.1	FTP	116 Response: 220 ProFTPD Server (Debian) [::ffff:172.16.1.10]
194	195.558142958	172.16.1.10	172.16.1.10	FTP	77 Request: USER rcom
196	195.559104598	172.16.1.10	172.16.1.10	FTP	98 Response: 331 Password required for rcom
197	195.559160611	172.16.1.10	172.16.1.10	FTP	77 Request: PASS rcom
199	195.734850715	172.16.1.10	172.16.1.10	FTP	112 Response: 230-Welcome, archive user rcom@172.16.1.19 !
200	195.734865730	172.16.1.10	172.16.1.10	FTP	69 Response:
201	195.734872016	172.16.1.10	172.16.1.10	FTP	112 Response: The local time is: Sat Dec 14 08:34:02 2024
202	195.734932149	172.16.1.10	172.16.1.10	FTP	142 Response:
203	195.734938924	172.16.1.10	172.16.1.10	FTP	129 Response: please report them via e-mail to <root@ftp.netlab.fe.up.pt>.
204	195.734983762	172.16.1.10	172.16.1.10	FTP	94 Response:
206	195.735241405	172.16.1.10	172.16.1.10	FTP	72 Request: PASV
208	195.736025021	172.16.1.10	172.16.1.10	FTP	115 Response: 227 Entering Passive Mode (172,16,1,10,168,55).
212	195.736616155	172.16.1.10	172.16.1.10	FTP	74 Request: TYPE I
213	195.737177118	172.16.1.10	172.16.1.10	FTP	85 Response: 200 Type set to I
214	195.737219302	172.16.1.10	172.16.1.10	FTP	81 Request: RETR pipe.txt
215	195.738084073	172.16.1.10	172.16.1.10	FTP	132 Response: 150 Opening BINARY mode data connection for pipe.txt (418 bytes)
216	195.738306377	172.16.1.10	172.16.1.10	FTP-DA...	484 FTP Data: 418 bytes (PASV) (TYPE I)
229	206.539869664	172.16.1.10	172.16.1.10	FTP	89 Response: 226 Transfer complete
232	206.540015492	172.16.1.10	172.16.1.10	FTP	72 Request: QUIT
233	206.540643782	172.16.1.10	172.16.1.10	FTP	80 Response: 221 Goodbye.

By analyzing these logs, we can clearly see the entire flow of a file download using the FTP protocol. The process begins with the client establishing a control connection and sending the `USER` and `PASS` commands to authenticate with the server. Once the authentication is successful, as indicated by the `331` and `230` responses, the client switches to passive mode using the `PASV` command. The server responds with the `227` code, providing the IP address and port for the data connection. The client then sets binary mode with the `TYPE I` command (response `200`) to ensure accurate file transfer for non-text files (in this case this step would not be necessary, but we found that other files would become corrupted if we did not include it). The `RETR` command is issued to request the file (`pipe.txt`), and the server responds with `150`, indicating that the data connection is open and ready for transfer. The file data is transmitted (418 bytes in this case), and the server confirms the completion of the transfer with the `226` response. Finally, the client sends the `QUIT` command to terminate the session, and the server acknowledges with the `221` response, gracefully closing the connection.

## Part 2 - Network configuration analysis

### Experience 1 - Configure an IP Network

**Network Architecture:** The network consists of two devices (`tux13` and `tux14`) connected via a switch within the subnet `172.16.10.0/24`. Both devices are configured with unique IP addresses.

**Experiment Objectives:** Configure the IP network interfaces on `tux13` using `ifconfig eth1 172.16.10.1/24` and `tux14` using `ifconfig eth1 172.16.10.254/24`, verify connectivity using `ping`, and analyze ARP table behavior using `arp -a`.

**Relevant Logs:** Captured ARP requests and replies for resolving MAC addresses, as well as ICMP packets (echo requests and replies) for verifying connectivity. The logs demonstrate proper device communication over the network.

**Analysis:** The ARP requests captured in the logs show how IP-to-MAC address resolution occurs before initiating communication. ICMP packets confirm successful connectivity between `tux13` and `tux14`. The experiment highlights the importance of ARP in facilitating communication within a subnet. ([See annexed logs for detailed packet captures.](#))

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

- ARP (Address Resolution Protocol) is a protocol used to map an IP address into a MAC address inside the same local network (LAN). When a device wants to send a packet to another device on the same network he needs its MAC address. If he does not have it however, he can send an ARP Request packet where he can find out the MAC address corresponding to the IP.

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

- ARP Request packets have the MAC address of the device that sent the request as the source, and FF:FF:FF:FF:FF:FF (broadcast) as the destination. The source IP address used is the transmitter's address while the destination IP address is the one which we wanna map.
- An ARP Reply packet includes the MAC address of the responding device as the source, while the destination MAC address is that of the requester. The source IP is the responder's IP, and the destination IP is the requester's address.

#### **What packets does the ping command generate?**

- The ping command uses the ICMP (Internet Control Message Protocol) protocol and it generates two types of packets:
  - Echo Request: Sent by the device who 'pings' to the destination;
  - Echo Reply: Answer sent by the destination device to the one who sent the ping;

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

- The Echo Request packet uses as the source MAC address the one of the device who sends the ping and as the MAC destination address, the MAC of the destination's device on the same network (or of the gateway if in separate networks). The source IP address corresponds to the address who has sent the ping and the IP destination is the one specified on the ping command.
- For an Echo Reply, the source MAC and IP addresses are those of the responding device, and the destination MAC and IP addresses belong to the original sender.

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

- Using the tool, Wireshark it's easy since you just need to check the "Protocol" field of the receiving packets.

#### **How to determine the length of a receiving frame?**

- Also using Wireshark, just see the "Length" field.

#### **What is the loopback interface and why is it important?**

- A loopback interface is a virtual network that a system uses to talk to itself. It's really important since it allows the device to test his own network stack without a physical interface being active. Additionally, it is used by services that need local connectivity and plays a critical role in software development and diagnostics.

### **Experience 2 - Implement two bridges in a switch**

**Network Architecture:** The experiment involved a switch configured with two separate bridges (`bridge10` and `bridge11`), connecting devices in isolated networks. `Tux13` and

Tux14 were connected to bridge10, while Tux12 was connected to bridge11. This setup was used to analyze traffic isolation and the behavior of broadcast packets across bridges.

### Experiment Objectives:

1. Verify connectivity between devices in the same bridge (e.g., Tux13 to Tux14).
2. Demonstrate lack of connectivity between devices in different bridges (e.g., Tux13 to Tux12).
3. Analyze the propagation of broadcast pings across bridges and evaluate received responses.

### Relevant Logs:

1. **Tux13 pinging Tux14:** Successful communication observed with ICMP echo requests and replies ([see logs](#)).
2. **Tux12 Unreachable:** "Network is unreachable" message confirms lack of connectivity between devices in separate bridges ([see logs](#)).
3. **Broadcast Pings:** Captured ARP requests and broadcast ICMP packets, showing that devices within the same bridge ([Tux13 ping broadcast logs](#), [Tux14 broadcast reception](#)) receive and respond to broadcasts, while devices in separate bridges do not ([Tux12 ping broadcast logs](#)).

### Analysis:

The experiment demonstrates the functionality of bridges in isolating traffic between subnets while allowing intra-bridge communication. Broadcast traffic is limited to devices within the same bridge, confirming proper bridge isolation. These findings validate the use of bridges in segmenting network traffic and improving security by limiting broadcast domain sizes.

### How to configure bridge10?

- In order to configure a bridge we have to use the Mikrotik switch. A bridge connects multiple interfaces in one single broadcast. To create a bridge we just need to execute:
  - `/interface bridge add name=bridge10`
- Next, we need to connect an interface to our bridge10:
  - `/interface bridge port add bridge=bridge10 interface=ether1`
- In the end, we can confirm if our job is done by executing two commands:
  - Check the created bridges with: `/interface bridge print`
  - Check the ports associated with the bridge: `/interface bridge port print`

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

- A broadcast domain is the set of devices that are receiving packets sent by broadcast (ARP or ICMP packets) without needing a router.
- The interfaces were divided into different bridges (tux13 and tux14 in bridge10, tuxY2 in bridge11). So, right now there are two broadcast domains: bridge10 and bridge11.
- What we can conclude from the logs is:
  - Packets that were sent over a bridge were not showing up in interfaces associated with other bridges.

### Experiment 3 - Configure a Router in Linux

**Network Architecture:** In this experiment, Tux14 was configured as a router connecting two subnets, 172.16.10.0/24 and 172.16.11.0/24. Devices Tux13 and Tux12 were placed in different subnets and used the router (Tux14) as a gateway to enable communication between them.

#### **Experiment Objectives:**

1. Configure Tux14 as a router by enabling IP forwarding.
2. Test connectivity between Tux13 and other devices in different subnets.
3. Capture traffic on Tux14 to analyze the forwarding behavior across its interfaces.

#### **Relevant Logs:**

1. Tux13 Pinging Other Interfaces:
  - Successful ICMP requests and replies demonstrate routing through Tux14.
  - ARP packets confirm gateway resolution for packets destined to other subnets.
2. Tux14 Capturing Both Network Interfaces:
  - Captured packets on eth1 and eth2 of Tux14 show routing behavior, with incoming traffic from one subnet being forwarded to the other.
  - Logs highlight that Tux14 modifies the source/destination MAC addresses as part of the forwarding process.

#### **Analysis:**

The experiment validates the use of Tux14 as a router between two subnets. IP forwarding enables communication between devices in different subnets by directing traffic through the gateway. The logs confirm correct routing behavior, with ICMP packets being forwarded across interfaces. ARP logs further demonstrate how Tux14 resolves and updates MAC addresses for devices in each subnet.

#### **What routes are there in the tuxes? What are their meaning?**

- There are **directly connected networks** which are routes for the subnets, to which the interfaces are directly connected.
  - 172.16.10.0/24 -> associated with tuxY3 and tuxY4
  - 172.16.11.0/24 -> associated with tuxY2
- There is the **default route**, which is a route pointing to the next router looking for packets destined for networks outside the directly connected ones.

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

- A forward table entry contains the following information:
  - **Destination IP or Network:** IP address or subnet the entry applies to
  - **Gateway:** Address which we want to send packets to (if necessary)
  - **Interface:** The network interface to use for forwarding packets
  - **Metric:** A value that tells the preferred route
  - **Flag:** Displays specific characteristics such as if the route is up or if it is a gateway route

- **Genmask:** Defines the size

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

- **ARP Request:** Is sent when a device wants to resolve a MAC address of another device's IP address. The sender's MAC address and IP are included, while the destination MAC address is broadcast (FF:FF:FF:FF:FF:FF).
  - Example: A tux sends "Who has 172.16.Y0.1? Tell 172.16.Y0.2".
- **ARP Reply:** Sent by the device that owns the desired IP address, and provided its MAC address. The source MAC and IP address belong to the one who answers, the destination MAC and IP address belong to the one who requested.
  - Example: 172.16.Y0.1 answers "It is at AA:BB:CC:DD:EE:FF".

#### **What ICMP packets are observed and why?**

ICMP is used to make diagnoses and communicate errors between devices.

- **Echo Request packet:**
  - Sent by the tux who starts a ping command
  - Motive: To verify the connectivity with another device.
- **Echo Reply packet:**
  - Sent by the destination answering the Echo Request packet.
  - Motive: To confirm that the communication was established.

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

- **Echo Request:**
  - **Source MAC/IP:** From the device that started the ping command.
  - **Destination MAC/IP:** Depends on the network configuration:
    - If in different subnets, the destination MAC is the same as the router's interface MAC;
    - If within the same subnet, the destination MAC is the recipient device's MAC;

### **Experiment 4 - Configure a Commercial Router and Implement NAT**

**Network Architecture:** In this experiment, **Rc** (a commercial router) connects two subnets: **172.16.10.0/24** and **172.16.11.0/24**. NAT (Network Address Translation) is enabled on the router to facilitate communication between the subnets.

#### **Experiment Objectives:**

1. Configure **Rc** to act as a router between the subnets and enable NAT.
2. Verify connectivity between **Tux12** and **Tux13** through the router.
3. Analyze the routing behavior with tools like **ping** and **traceroute**.
4. Demonstrate how NAT affects communication between subnets.

#### **Relevant Logs:**

1. **Tux13 Pinging All Network Interfaces:**
  - ICMP echo requests and replies from different interfaces confirm connectivity within the network.
2. **Tux12 Pinging Tux13 Through Rc:**
  - ICMP echo requests and replies confirm successful communication across the subnets via the router.



- Each response includes a "Redirect Host" message from 172.16.11.254, indicating that routing adjustments are being made by the router..
3. **Tux12 Traceroutes:**
- Traceroute logs display the 2 routing paths: One using Rc as the gateway to the other subnet, and one using Tux14.

**Analysis:** The experiment validates the role of NAT in enabling communication between private subnets and external networks. The captured logs confirm proper routing and address translation through the commercial router. NAT ensures that devices in 172.16.11.0/24 can communicate with devices in 172.16.10.0/24 while preserving network security and address space efficiency.

### **How to configure a static route in a commercial router?**

- In order to configure a static route we need to following a few steps:
  1. Define the destination subnet;
  2. Specify the next-hop IP address or gateway for that subnet;
  3. Associate the route with an interface;
- With the GTKterm:
  1. Add a static route:
    - /ip route add dst-address=<DESTINATION>/<MASK> gateway=<GATEWAY>
  2. List configured routes:
    - /ip route print
- We can test the connectivity with commands like ping and traceroute.

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

- **Without ICMP Redirect:** The commercial router's default route setting routes packets from tux13 to tux12. Since tux14 is specifically designated as the gateway, the router handles the packets even though tux13 and tux12 can connect directly through it. The road gains one additional hop as a result.
- **With ICMP Redirect:** The router notifies tux13 that tux14 offers a more direct path when ICMP redirect is enabled. Consequently, packets are delivered through tux14 directly between tux13 and tux12, avoiding the commercial router. This behavior lowers latency and optimizes the path.

### **How to configure NAT in a commercial router?**

- Enabling NAT on the interface:
  - /ip firewall nat add chain=srcnat action=masquerade out-interface=ether1
- Verify NAT rules:
  - /ip firewall nat print

### **What does NAT do?**

- Network address translation, or NAT, converts private IP addresses used on internal networks into distinct public IP addresses so that external networks can communicate with one another.

### **What happens when tux13 pings the FTP server with the NAT disabled? Why?**

- When NAT gets disabled tux13 cannot receive replies from the FTP server. That's because the packets sent by it use the private IP address as the source. On top of



this, the ftp server responds to the private IP, which cannot be routable to the public. Without NAT, routers cannot translate private IPs into public IPs to establish communication with external sources.

## **Experiment 5 - DNS**

### **Network Architecture:**

This experiment focuses on enabling and testing DNS functionality within the network.

Tux12 is configured to use a specific DNS server (10.227.20.3) for resolving domain names. The functionality is validated by testing name resolution for internal and external hosts (e.g., google.com).

### **Experiment Objectives:**

1. Configure Tux12 to use the correct DNS server.
2. Verify DNS functionality by resolving hostnames to IP addresses.
3. Test connectivity to external domains using ping to validate DNS resolution.

### **Relevant Logs:**

1. **DNS Configuration:**
  - The /etc/resolv.conf file confirms the correct setup of the nameserver (10.227.20.3) and search domains.
  - This configuration ensures that DNS queries can be processed by the designated server.
2. **Testing Name Resolution:**
  - Pinging google.com demonstrates successful hostname resolution, with the hostname translating into its corresponding IP address.
3. **Pinging External Domain:**
  - Logs show successful DNS resolution for google.com, followed by ICMP echo requests and replies.
  - The Wireshark capture highlights DNS query and response packets exchanged between Tux12 and the DNS server (10.227.20.3), confirming proper DNS operation.

### **Analysis:**

This experiment validates the role of DNS in facilitating name resolution within a network. The logs and packet captures confirm that DNS queries are correctly routed to the specified server and responses are processed as expected. This ensures communication with hosts using domain names rather than IP addresses.

### **How to configure the DNS service in a host?**

1. Locate and edit the /etc/resolv.conf:
  - Example: sudo nano /etc/resolv.conf
2. Add the DNS server addresses:
  - nameserver <DNS-SERVER>

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

- DNS Query: Client sends a query to a DNS server to resolve a domain name into IP address (or vice versa). “What is the IP for [www.example.com](http://www.example.com)?”
- DNS Response: DNS server replies with the requested information. “The IP address for [www.example.com](http://www.example.com)” is 01.234.567.89.
- Information passed:
  - **Header:** Contains the query ID, flags and other sections;
  - **Question Section:** Has the domain name that was queried and the type of record requested;
  - **Answer Section:** Displays the resolved data, such as the IP address of the domain;
  - **Authority Section:** Lists authoritative name servers for the queried domain;
  - **Additional Section:** Provides extra information;

## **Experiment 6 - TCP connections**

### **Network Architecture:**

In this experiment, the FTP client developed earlier is tested in a multi-device network setup, involving **tux13** (FTP client), **Rc** (router with NAT enabled), and **ftp.netlab.fe.up.pt** (FTP server). The network spans multiple subnets interconnected via NAT. The goal is to analyze TCP connections, their phases, and mechanisms, including ARQ and congestion control, during a file download and in scenarios with competing traffic.

### **Experiment Objectives:**

1. Test the FTP download application in a real-world network environment.
2. Analyze the TCP phases—connection establishment, data transfer, and termination—using Wireshark.
3. Observe and understand TCP mechanisms, including ARQ and congestion control.
4. Examine the impact of competing downloads on TCP throughput and behavior.

**Wireshark Capture:** TCP control and data connections are captured:

- **Connection Establishment:** The three-way handshake (**SYN**, **SYN-ACK**, **ACK**) is clearly observed, marking the initiation of the TCP connection.
- **Data Transfer:** Logs display FTP commands (e.g., **USER**, **PASS**, **RETR**) in the control connection and the subsequent data transfer.
- **Connection Termination:** The four-segment connection termination (**FIN**, **ACK**) is logged, indicating proper teardown of the TCP session.

**Competing Downloads:** The files that were available on **ftp.netlab.fe.up.pt** were either too small (crab.mp4 download finished before we could start it on the other tux) or too large (ubuntu.iso produced very large logs that we were unable to retrieve to an external device or upload for future examination) so we could not conclude anything. We tried to reproduce this at home and the conclusion we reached was that the transfer speed of the first connection gets approximately halved by the TCP congestion control mechanism.

### **Analysis:**

This experiment highlights TCP's ability to reliably transfer data even under adverse conditions such as packet loss or congestion.

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

- There are two TCP connections on a FTP application:
  1. **Control Connection:** Used to send and receive FTP commands;
  2. **Data Connection:** Used to transfer files between the client and server;

### In what connection is transported the FTP control information?

- The FTP control information is transported in the **Control Connection**. This remains open during the session and is used to handle commands such as USER,PASS,RETR,etc...

### What are the phases of a TCP connection?

1. **Connection Establishment (3-way Handshake):**
  - **SYN:** Client send a synchronization packet;
  - **SYN-ACK:** Server acknowledges and sends a SYN;
  - **ACK:** Client acknowledges completing the "handshake";
2. **Data Transfer:**
  - Data is traded between the client and the server.
3. **Connection Termination (4-way Handshake):**
  - **FIN:** Request to terminate the connection;
  - **ACK:** The other side acknowledges the FIN;
  - **FIN:** The second side sends a FIN;
  - **ACK:** The first side acknowledges, completing the termination;

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

1.
  - TCP used ARQ (Automatic Repeat Request) to reliably deliver the data;
  - Its sent an ACK for every packet that is received;
  - If there is not ACK being received within the timeout, the sender will retransmitter the packet;
2.
  - Sequence Number:** Identifies the order of bytes in the data stream;
  - Acknowledgment Number:** Confirms the receipt of data;
  - Window Size:** Indicates the buffer capacity available
  - Flag:** ACK,FIN,SYN,etc...
3.
  - Adjustments in window size during flow control;
  - Duplicate ACKs signaling packet loss;
  - Retransmissions due to timeout;

### How does the TCP congestion control mechanism work? What are the relevant fields?

- TCP congestion control mechanism operates in four phases:
  1. **Slow Start:** The congestion window grows exponentially until it reached a threshold or a packet loss occurs;
  2. **Congestion Avoidance:** After the threshold is reached, the window increases linearly trying not to overload the network;
  3. **Fast Retransmit:** If three duplicate ACKs are received, TCP assumes a packet is lost and retransmits it;
  4. **Fast Recovery:** The congestion window is reduced, and TCP shifts to congestion avoidance mode to gradually recover;

### How did the throughput of the data connection evolve along the time? Is it according to the TCP congestion control mechanism?

- At the beginning, during the Slow Start phase, throughput grows rapidly (exponentially) as the congestion window (cwnd) increases. Once the connection enters the Congestion Avoidance phase, throughput grows more slowly (linearly). If the connection detects packet loss, throughput drops sharply because the congestion window is reduced (often halved). Afterward, throughput gradually recovers. This pattern of growth and occasional drops can be observed in graphs of throughput over time.

### **Is the throughput of a TCP data connection disturbed by the appearance of a second TCP connection? How?**

- Yes, the throughput of a TCP data connection is affected when a second connection starts sharing the same network. Both compete for the existing bandwidth, while TCP adjusts the congestion window for each to ensure fairness, typically the result is each connection being reduced to half of the bandwidth. Over time, the network is stabilized and both connections reach a steady state of share throughput as seen in performance graphs.

## Conclusions

In conclusion, this lab provided a comprehensive understanding of networking principles and protocols, emphasizing the development of a simple FTP client and its interaction with TCP/IP. Through practical experiments, we analyzed network configurations, routing, and bridging while observing key TCP mechanisms such as ARQ and congestion control. The successful execution and analysis of FTP file transfers demonstrated the reliability of TCP and the importance of precise network configurations. Overall, the lab reinforced theoretical concepts with hands-on experience, enhancing our ability to diagnose and optimize network communication.

## Annexes

### **Download Application Code:**

1.1 - main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <stdbool.h>
#include <libgen.h>
#include "ftp_client.h"

int main(int argc, char *argv[]) {
    if (argc != 2) {
```

```

        fprintf(stderr, "Usage: %s
ftp://[<user>:<password>@]<host>/<url-path>\n", argv[0]);
        return -1;
    }

    char user[64], password[64], host[256], path[256];
    if (parse_ftp_url(argv[1], user, password, host, path) < 0) {
        fprintf(stderr, "Error: Invalid FTP URL format.\n");
        return -1;
    }

    printf("Starting FTP download...\n");

    // Connect to FTP server
    int control_sock = connect_to_server(host, 21);
    if (control_sock < 0) {
        fprintf(stderr, "Error: Unable to connect to server.\n");
        return -1;
    }
    printf("Connected to server: %s\n", host);

    // Read initial greeting
    char greeting[2048];
    if (read_ftp_response(control_sock, greeting, sizeof(greeting)) <=
0) {
        fprintf(stderr, "Error: Failed to read server greeting.\n");
        close(control_sock);
        return -1;
    }
    printf("Server greeting: %s", greeting);

    // Login to FTP server
    if (ftp_login(control_sock, user, password) < 0) {
        fprintf(stderr, "Error: Login failed.\n");
        close(control_sock);
        return -1;
    }
    printf("Logged in as: %s\n", user);

    // Enter passive mode
    char ip[16];
    int port;
    if (ftp_enter_passive_mode(control_sock, ip, &port) < 0) {

```

```

        fprintf(stderr, "Error: Failed to enter passive mode.\n");
        close(control_sock);
        return -1;
    }
    printf("Entered passive mode: %s:%d\n", ip, port);

    // Connect to data socket
    int data_sock = connect_to_server(ip, port);
    if (data_sock < 0) {
        fprintf(stderr, "Error: Unable to establish data
connection.\n");
        close(control_sock);
        return -1;
    }
    printf("Data connection established.\n");
    char *filename = basename(path);

    // Retrieve file
    if (ftp_retrieve_file(control_sock, data_sock, path, filename) < 0)
    {
        fprintf(stderr, "Error: Failed to retrieve file.\n");
        close(data_sock);
        close(control_sock);
        return -1;
    }
    printf("File downloaded successfully: %s\n", filename);

    // Close connections
    close(data_sock);
    ftp_quit(control_sock);
    close(control_sock);
    printf("FTP session closed.\n");

    return 0;
}

```

## 1.2 - ftp\_client.h

```

#ifndef FTP_CLIENT_H
#define FTP_CLIENT_H

```

```

#include <stddef.h>

int read_ftp_response(int sock, char *buf, size_t size);
int parse_ftp_url(const char *url, char *user, char *password, char
*host, char *path);
int connect_to_server(const char *hostname, int port);
int ftp_login(int control_sock, const char *user, const char
*password);
int ftp_enter_passive_mode(int control_sock, char *ip, int *port);
int ftp_retrieve_file(int control_sock, int data_sock, const char
*remote_path, const char *local_filename);
void ftp_quit(int control_sock);

#endif

```

### 1.3 - ftp\_client.c

```

#include "ftp_client.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netdb.h>

// Parses the FTP URL and extracts user, password, host, and path
int parse_ftp_url(const char *url, char *user, char *password, char
*host, char *path)
{
    // Try full format: ftp://user:pass@host/path
    if (sscanf(url, "ftp://%[^:]:%[^@]@%[/]/%s", user, password, host,
path) == 4)
    {
        return 0;
    }
    // Try anonymous format: ftp://host/path
    if (sscanf(url, "ftp://%[/]/%s", host, path) == 2)
    {

```



```

        strcpy(user, "anonymous");
        strcpy(password, "guest");
        return 0;
    }
    return -1;
}

// Connects to the server and returns the socket descriptor
int connect_to_server(const char *hostname, int port)
{
    struct sockaddr_in server_addr;
    struct hostent *host_entry;

    if ((host_entry = gethostbyname(hostname)) == NULL)
    {
        perror("gethostbyname");
        return -1;
    }

    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
    {
        perror("socket");
        return -1;
    }

    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    memcpy(&server_addr.sin_addr, host_entry->h_addr,
host_entry->h_length);
    server_addr.sin_port = htons(port);

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

    return sockfd;
}

```

```

// Reads a complete FTP response, handling multi-line replies.
// FTP multi-line replies format:
// xyz-... (one or more lines)
// xyz ... (final line)
int read_ftp_response(int sock, char *buf, size_t size)
{
    memset(buf, 0, size);
    int total = 0;
    char line[1024];
    bool multiline = false;
    char code[4] = {0};

    while (1)
    {
        memset(line, 0, sizeof(line));
        int n = 0;
        // Read one line at a time
        while (n < (int)sizeof(line) - 1)
        {
            int r = read(sock, &line[n], 1);
            if (r <= 0)
            {
                // Connection closed or error
                break;
            }
            if (line[n] == '\n')
            {
                n++;
                break;
            }
            n++;
        }
        if (n == 0)
        {
            // No data read, possibly connection closed
            break;
        }
        line[n] = '\0';

        // Append this line to buf if space allows
        if ((int)strlen(buf) + n < (int)size - 1)
        {
            strcat(buf, line);

```

```

    }

    // If this is the first line, determine if it's multiline
    if (total == 0 && strlen(line) >= 4)
    {
        strncpy(code, line, 3);
        code[3] = '\0';
        // Check if multiline
        if (line[3] == '-')
        {
            multiline = true;
        }
    }

    total += n;

    // Check for end of multiline
    if (strlen(code) == 3 && !multiline)
    {
        // Single line response
        break;
    }

    if (multiline && strncmp(line, code, 3) == 0 && line[3] == ' ')
    {
        // End of multiline response
        break;
    }

    if (!multiline)
    {
        // If not multiline, we read only one line
        break;
    }
}

return total;
}

// Logs in to the FTP server
int ftp_login(int control_sock, const char *user, const char *password)
{
    char buffer[2048];

```

```

    // If user is anonymous and password is "guest", try a common
anonymous password
    char actual_password[256];
    if (strcmp(user, "anonymous") == 0 && strcmp(password, "guest") == 0)
    {
        strcpy(actual_password, "anonymous@example.com");
    }
    else
    {
        strcpy(actual_password, password);
    }

    snprintf(buffer, sizeof(buffer), "USER %s\r\n", user);
    write(control_sock, buffer, strlen(buffer));
    read_ftp_response(control_sock, buffer, sizeof(buffer));

    // Expect a 331 code here
    if (strncmp(buffer, "331", 3) != 0)
    {
        fprintf(stderr, "Login failed (USER step): %s", buffer);
        return -1;
    }

    snprintf(buffer, sizeof(buffer), "PASS %s\r\n", actual_password);
    write(control_sock, buffer, strlen(buffer));
    read_ftp_response(control_sock, buffer, sizeof(buffer));

    // Expect a 230 code here for success
    if (strncmp(buffer, "230", 3) != 0)
    {
        fprintf(stderr, "Login failed (PASS step): %s", buffer);
        return -1;
    }

    return 0;
}

// Enters passive mode and retrieves IP and port for data connection
int ftp_enter_passive_mode(int control_sock, char *ip, int *port)
{
    char buffer[2048];
    char *start;
    int ip1, ip2, ip3, ip4, p1, p2;

```

```

snprintf(buffer, sizeof(buffer), "PASV\r\n");
write(control_sock, buffer, strlen(buffer));
read_ftp_response(control_sock, buffer, sizeof(buffer));
if (strncmp(buffer, "227", 3) != 0)
{
    fprintf(stderr, "Passive mode failed: %s", buffer);
    return -1;
}

// Parse the response to extract IP and port
start = strchr(buffer, '(');
if (!start || sscanf(start, "(%d,%d,%d,%d,%d,%d)", &ip1, &ip2, &ip3,
&ip4, &p1, &p2) != 6)
{
    fprintf(stderr, "Failed to parse passive mode response: %s",
buffer);
    return -1;
}

snprintf(ip, 16, "%d.%d.%d.%d", ip1, ip2, ip3, ip4);
*port = p1 * 256 + p2;
return 0;
}

// Retrieves a file from the server
int ftp_retrieve_file(int control_sock, int data_sock, const char
*remote_path, const char *local_filename)
{
    char buffer[2048];
    FILE *file;

    // Set binary mode using TYPE I command
    snprintf(buffer, sizeof(buffer), "TYPE I\r\n");
    write(control_sock, buffer, strlen(buffer));
    read_ftp_response(control_sock, buffer, sizeof(buffer));
    if (strncmp(buffer, "200", 3) != 0)
    {
        fprintf(stderr, "Failed to set binary mode: %s", buffer);
        return -1;
    }

    // Send RETR command to retrieve the file

```

```
snprintf(buffer, sizeof(buffer), "RETR %s\r\n", remote_path);
write(control_sock, buffer, strlen(buffer));
read_ftp_response(control_sock, buffer, sizeof(buffer));

// Expecting 150 (File status okay; about to open data connection)
if (strncmp(buffer, "150", 3) != 0 && strncmp(buffer, "125", 3) != 0)
{
    fprintf(stderr, "Failed to retrieve file: %s", buffer);
    return -1;
}

// Open file for writing
if ((file = fopen(local_filename, "wb")) == NULL)
{
    perror("fopen");
    return -1;
}

// Read data from the data socket and write to file
int bytes;
char data_buf[1024];
while ((bytes = read(data_sock, data_buf, sizeof(data_buf))) > 0)
{
    if (fwrite(data_buf, 1, bytes, file) != (size_t)bytes)
    {
        perror("fwrite");
        fclose(file);
        return -1;
    }
}

// Check if reading from data socket encountered an error
if (bytes < 0)
{
    perror("read");
    fclose(file);
    return -1;
}

fclose(file);

// After data transfer, server should send a 226 response (Transfer
complete)
```

```

read ftp_response(control_sock, buffer, sizeof(buffer));
if (strncmp(buffer, "226", 3) != 0)
{
    fprintf(stderr, "File transfer incomplete: %s", buffer);
    return -1;
}

return 0;
}

// Sends QUIT command to close the session
void ftp_quit(int control_sock)
{
    char buffer[2048];
    snprintf(buffer, sizeof(buffer), "QUIT\r\n");
    write(control_sock, buffer, strlen(buffer));
    read ftp_response(control_sock, buffer, sizeof(buffer));
    printf("Server response: %s", buffer);
}

```

## Configuration and Study of a Network Experience Logs and Screenshots

### 2.1.1 - Tux13 pinging Tux14

The screenshot shows a Wireshark network capture titled "Capturing on eth1". The display filter is "Apply a display filter... <Ctrl-F>". The packet list shows 61 packets, with the first 50 being ICMP Echo (ping) requests and replies. The packet details pane shows the selected packet (No. 50) as an ICMP Echo (ping) request from 172.16.10.254 to 172.16.10.1. The packet bytes pane shows the raw data of the packet. The logical-link control section shows the Spanning Tree Protocol (STP) configuration for the network.

No.	Time	Source	Destination	Protocol	Length	Info
16	29.893771624	EdimaxTe_ed:bb:37	EdimaxTe_ed:bb:37	Broadcast	ARP	42 Who has 172.16.10.254? Tell 172.16.10.1
17	29.893812772	Kye_04:20:99	EdimaxTe_ed:bb:37	ARP	60	172.16.10.254 is at 00:00:0f:04:20:99
18	29.893930442	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x58aa, seq=1/256, ttl=64 (request in 19)
19	29.894048543	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x58aa, seq=1/256, ttl=64 (request in 18)
20	29.915233719	RouterBo_1c:8c:9b	Spanning-tree-(for-- STP	60	RST, Root = 32768/0/c4:ad:34:1c:8c:9b Cost = 0 Port = 0x8001	
21	30.915108215	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x58aa, seq=2/512, ttl=64 (reply in 22)
22	30.915233719	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x58aa, seq=2/512, ttl=64 (request in 21)
23	31.939106097	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x58aa, seq=3/768, ttl=64 (reply in 24)
24	31.939231182	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x58aa, seq=3/768, ttl=64 (request in 23)
25	32.635143531	RouterBo_1c:8c:9b	Spanning-tree-(for-- STP	60	RST, Root = 32768/0/c4:ad:34:1c:8c:9b Cost = 0 Port = 0x8001	
26	32.963103769	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x58aa, seq=4/1024, ttl=64 (reply in 27)
27	32.963225813	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x58aa, seq=4/1024, ttl=64 (request in 26)
28	33.987114362	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x58aa, seq=5/1280, ttl=64 (reply in 29)
29	33.987268222	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x58aa, seq=5/1280, ttl=64 (request in 28)
30	34.067242485	RouterBo_1c:8c:9b	Spanning-tree-(for-- STP	60	RST, Root = 32768/0/c4:ad:34:1c:8c:9b Cost = 0 Port = 0x8001	
31	34.982730682	Kye_04:20:99	EdimaxTe_ed:bb:37	ARP	60	Who has 172.16.10.1? Tell 172.16.10.254
32	34.982749888	EdimaxTe_ed:bb:37	Kye_04:20:99	ARP	42	172.16.10.1 is at 00:00:0f:04:20:99
33	35.011113780	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x58aa, seq=6/1536, ttl=64 (reply in 34)
34	35.011230804	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x58aa, seq=6/1536, ttl=64 (request in 33)
35	36.635110475	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x58aa, seq=7/1792, ttl=64 (reply in 36)
36	36.635238214	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x58aa, seq=7/1792, ttl=64 (request in 35)
37	36.635231223	RouterBo_1c:8c:9b	Spanning-tree-(for-- STP	60	RST, Root = 32768/0/c4:ad:34:1c:8c:9b Cost = 0 Port = 0x8001	
38	37.059111080	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x58aa, seq=8/2048, ttl=64 (reply in 39)
39	37.059229391	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x58aa, seq=8/2048, ttl=64 (request in 38)
40	38.611732437	RouterBo_1c:8c:9b	Spanning-tree-(for-- STP	60	RST, Root = 32768/0/c4:ad:34:1c:8c:9b Cost = 0 Port = 0x8001	
41	38.603105051	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x58aa, seq=9/2304, ttl=64 (reply in 42)
42	38.603222105	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x58aa, seq=9/2304, ttl=64 (request in 41)
43	39.107107403	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x58aa, seq=10/2560, ttl=64 (reply in 44)
44	39.107214850	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x58aa, seq=10/2560, ttl=64 (request in 43)
45	39.170229688	0.0.0.0	255.255.255.255	MNPD	159	5678 - 5678 Len=117
46	39.170258672	RouterBo_1c:8c:9b	CDP/VTP/DTP/PagP/UD...	CDP	93	Device ID: Mikrotik Port ID: bridge10
47	39.170308373	RouterBo_1c:8c:9b	LLDP-Multicast	LLDP	110	TTL = 120 System Name = Mikrotik System Description = Mikrotik RouterOS 6.43.10 (long-term) CRS326-24G-25+
48	40.643029805	RouterBo_1c:8c:9b	Spanning-tree-(for-- STP	60	RST, Root = 32768/0/c4:ad:34:1c:8c:9b Cost = 0 Port = 0x8001	
49	40.13105704	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x58aa, seq=11/2816, ttl=64 (reply in 50)
50	40.131225760	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x58aa, seq=11/2816, ttl=64 (request in 49)
51	41.155105051	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x58aa, seq=12/3072, ttl=64 (reply in 52)
52	41.155231182	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x58aa, seq=12/3072, ttl=64 (request in 51)

Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0

IEEE 802.3 Ethernet

Logical-Link Control

Spanning Tree Protocol

0000 01 00 c2 00 00 c4 ad 34 1c 8c 9b 00 27 42 46 .....4....

0010 00 00 02 02 3c 00 00 c4 ad 34 1c 8c 9b 00 00 .....<...4....

0020 00 00 00 c4 ad 34 1c 8c 9b 00 01 00 00 14 00 .....4:.....

0030 02 00 0f 00 00 00 00 00 00 00 00 00 00 00 00 .....<.....

Logical-Link Control (LLC) 3 bytes

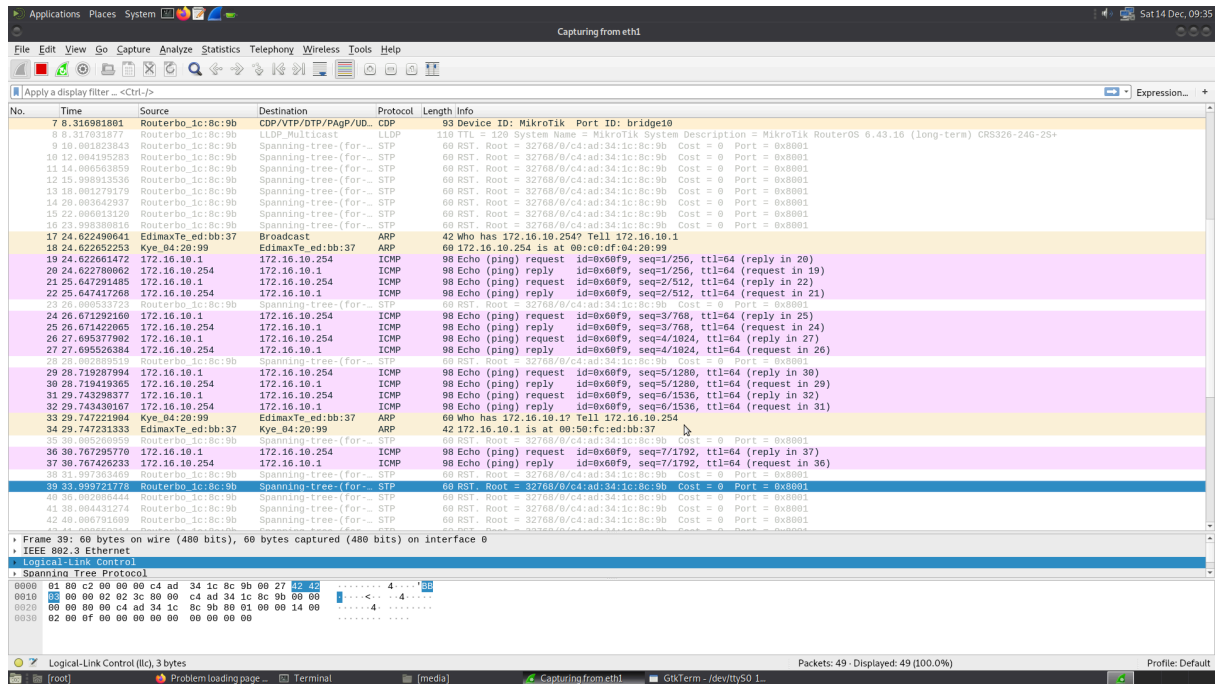
Packets: 61 · Displayed: 61 (100.0%)

Profile: Default

[root] [Problem loading page...] [Terminal] [GitTerm - /dev/tty50...] [media] [Capturing from eth1]



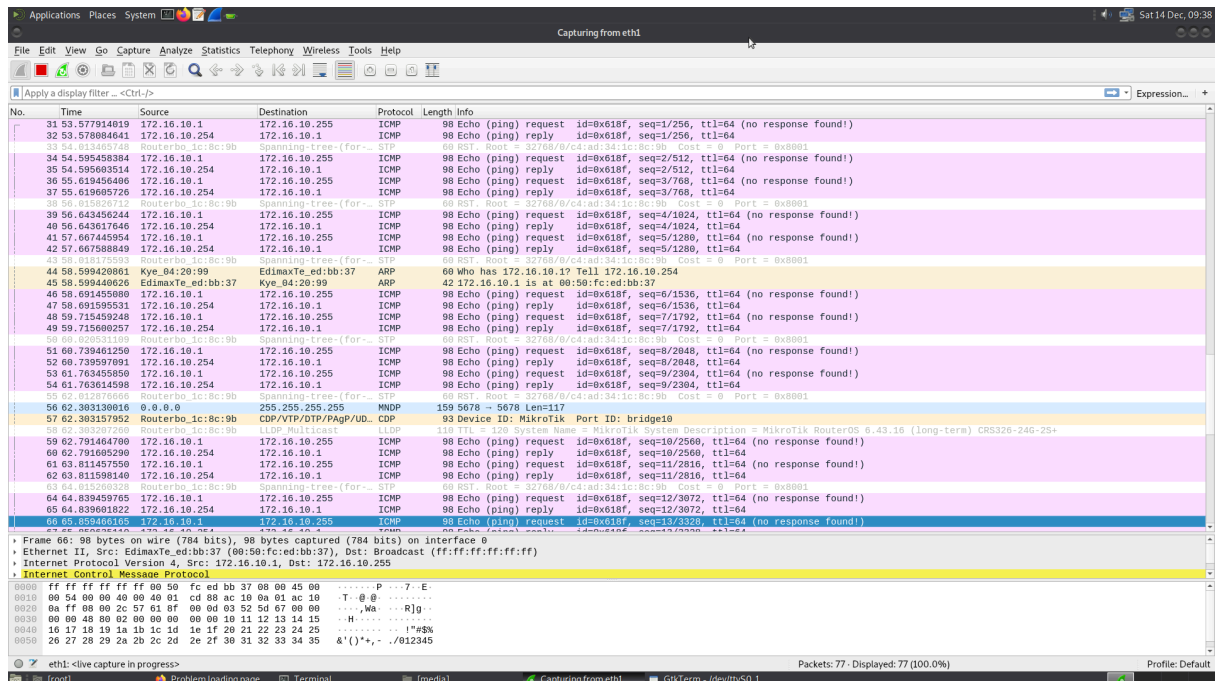
## 2.2.1 - Tux13 pinging Tux14



## 2.2.2 - Tux12 unreachable



## 2.2.3 - Tux13 ping broadcast logs



## 2.2.4 - Tux12 ping broadcast logs

Applications Places System pt Sat 14 Dec, 09:41

\*eth1

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/> Expression... +

No.	Time	Source	Destination	Protocol	Length	Info
8	11.579372765	172.16.11.1	172.16.11.255	ICMP	98	Echo (ping) request id=0x392c, seq=2/512, ttl=64 (no response found!)
9	12.013627556	Routerbo_ic:8c:9a	Spanning-tree-(for_	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
10	12.603375713	172.16.11.1	172.16.11.255	ICMP	98	Echo (ping) request id=0x392c, seq=3/768, ttl=64 (no response found!)
11	13.627373771	172.16.11.1	172.16.11.255	ICMP	98	Echo (ping) request id=0x392c, seq=4/1024, ttl=64 (no response found!)
12	14.015921338	Routerbo_ic:8c:9a	Spanning-tree-(for_	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
13	14.651373154	172.16.11.1	172.16.11.255	ICMP	98	Echo (ping) request id=0x392c, seq=5/1280, ttl=64 (no response found!)
14	15.675382871	172.16.11.1	172.16.11.255	ICMP	98	Echo (ping) request id=0x392c, seq=6/1536, ttl=64 (no response found!)
15	16.018226625	Routerbo_ic:8c:9a	Spanning-tree-(for_	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
16	16.699380016	172.16.11.1	172.16.11.255	ICMP	98	Echo (ping) request id=0x392c, seq=7/1792, ttl=64 (no response found!)
17	17.727380450	172.16.11.1	172.16.11.255	ICMP	98	Echo (ping) request id=0x392c, seq=8/2048, ttl=64 (no response found!)
18	18.020533676	Routerbo_ic:8c:9a	Spanning-tree-(for_	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
19	18.747377722	172.16.11.1	172.16.11.255	ICMP	98	Echo (ping) request id=0x392c, seq=9/2304, ttl=64 (no response found!)
20	19.771374792	172.16.11.1	172.16.11.255	ICMP	98	Echo (ping) request id=0x392c, seq=10/2560, ttl=64 (no response found!)
21	20.022827648	Routerbo_ic:8c:9a	Spanning-tree-(for_	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
22	20.795374234	172.16.11.1	172.16.11.255	ICMP	98	Echo (ping) request id=0x392c, seq=11/2816, ttl=64 (no response found!)
23	21.810384081	172.16.11.1	172.16.11.255	ICMP	98	Echo (ping) request id=0x392c, seq=12/3072, ttl=64 (no response found!)
24	22.025119797	Routerbo_ic:8c:9a	Spanning-tree-(for_	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
25	22.843380608	172.16.11.1	172.16.11.255	ICMP	98	Echo (ping) request id=0x392c, seq=13/3328, ttl=64 (no response found!)
26	23.867382957	172.16.11.1	172.16.11.255	ICMP	98	Echo (ping) request id=0x392c, seq=14/3584, ttl=64 (no response found!)
27	24.027410473	Routerbo_ic:8c:9a	Spanning-tree-(for_	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
28	24.891376805	172.16.11.1	172.16.11.255	ICMP	98	Echo (ping) request id=0x392c, seq=15/3840, ttl=64 (no response found!)
29	26.029726355	Routerbo_ic:8c:9a	Spanning-tree-(for_	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
30	28.032819322	Routerbo_ic:8c:9a	Spanning-tree-(for_	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001

Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0

- IEEE 802.3 Ethernet
- Logical-Link Control

```

0000 01 00 c2 00 00 00 c4 ad 34 1c 8c 9a 00 27 42 42 .....4...BB
0010 03 00 00 02 02 3c 80 00 c4 ad 34 1c 8c 9a 00 00 .....4...
0020 00 00 00 00 c4 ad 34 1c 8c 9a 00 01 00 00 14 00 .....4...
0030 02 00 0f 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

wireshark\_eth1\_20241214094123\_uFzIRZ.pcapng Packets: 30 · Displayed: 30 (100.0%) Profile: Default

Terminal \*eth1

## 2.2.5 - Tux13 ping broadcast logs received in Tux14

Applications Places System Sat 14 Dec, 09:39

Capturing from eth1

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter -<Ctrl-F> Expression...

No.	Time	Source	Destination	Protocol	Length	Info
13	22.561689130	172.16.10.1	172.16.10.255	ICMP	98	Echo (ping) request id=0x618f, seq=1/256, ttl=64 (no response found!)
14	22.56171416	172.16.10.1	172.16.10.1	ICMP	98	Echo (ping) reply id=0x618f, seq=1/256, ttl=64
15	23.579238806	172.16.10.1	172.16.10.255	ICMP	98	Echo (ping) request id=0x618f, seq=2/512, ttl=64 (no response found!)
16	23.579238828	172.16.10.1	172.16.10.1	ICMP	98	Echo (ping) reply id=0x618f, seq=2/512, ttl=64
17	23.683558338	Routerbo.1c:8c:9c	Spanning-tree-(for-br...	STP	80	RST, Root = 32768/0/c4:ad:34:1c:8c:9b Cost = 0 Port = 0x0002
18	24.683221937	172.16.10.1	172.16.10.255	ICMP	98	Echo (ping) request id=0x618f, seq=3/768, ttl=64 (no response found!)
19	24.683240826	172.16.10.1	172.16.10.1	ICMP	98	Echo (ping) reply id=0x618f, seq=3/768, ttl=64
20	25.627225811	172.16.10.1	172.16.10.255	ICMP	98	Echo (ping) request id=0x618f, seq=4/1824, ttl=64 (no response found!)
21	25.627252389	172.16.10.1	172.16.10.1	ICMP	98	Echo (ping) reply id=0x618f, seq=4/1824, ttl=64
22	26.686705770	Routerbo.1c:8c:9c	Spanning-tree-(for-br...	STP	80	RST, Root = 32768/0/c4:ad:34:1c:8c:9b Cost = 0 Port = 0x0002
23	26.686714256	172.16.10.1	172.16.10.255	ICMP	98	Echo (ping) request id=0x618f, seq=5/1280, ttl=64 (no response found!)
24	26.686722266	172.16.10.1	172.16.10.1	ICMP	98	Echo (ping) reply id=0x618f, seq=5/1280, ttl=64
25	27.583096874	Kye.04:20:99	EdimaxTe.ed:bb:37	ARP	42	Who has 172.16.10.1? Tell 172.16.10.254
26	27.583170095	EdimaxTe.ed:bb:37	Kye.04:20:99	ARP	60	172.16.10.1 is at 00:50:fc:ed:bb:37
27	27.675223267	172.16.10.1	172.16.10.255	ICMP	98	Echo (ping) request id=0x618f, seq=6/1536, ttl=64 (no response found!)
28	27.675236661	172.16.10.1	172.16.10.1	ICMP	98	Echo (ping) reply id=0x618f, seq=6/1536, ttl=64
29	28.693054543	Routerbo.1c:8c:9c	Spanning-tree-(for-br...	STP	80	RST, Root = 32768/0/c4:ad:34:1c:8c:9b Cost = 0 Port = 0x0002
30	28.699225434	172.16.10.1	172.16.10.255	ICMP	98	Echo (ping) request id=0x618f, seq=7/1792, ttl=64 (no response found!)
31	28.699234513	172.16.10.1	172.16.10.1	ICMP	98	Echo (ping) reply id=0x618f, seq=7/1792, ttl=64
32	29.723223968	172.16.10.1	172.16.10.255	ICMP	98	Echo (ping) request id=0x618f, seq=8/2048, ttl=64 (no response found!)
33	29.723231232	172.16.10.1	172.16.10.1	ICMP	98	Echo (ping) reply id=0x618f, seq=8/2048, ttl=64
34	29.948481839	Routerbo.1c:8c:9c	Spanning-tree-(for-br...	STP	80	RST, Root = 32768/0/c4:ad:34:1c:8c:9b Cost = 0 Port = 0x0002
35	30.747223760	172.16.10.1	172.16.10.255	ICMP	98	Echo (ping) request id=0x618f, seq=9/2304, ttl=64 (no response found!)
36	30.747248065	172.16.10.1	172.16.10.1	ICMP	98	Echo (ping) reply id=0x618f, seq=9/2304, ttl=64
37	31.286795718	0.0.0.0	255.255.255.255	MNPD	159	5678 - 5678 Len=117
38	31.286833563	Routerbo.1c:8c:9b	CDP/VTP/DTP/PagP/UDLD	CDP	93	Device ID: Mikrotik Port ID: bridge10
39	31.286833510	Routerbo.1c:8c:9b	LLDP Multicast	LLDP	119	TLV = 129 System Name Description = Mikrotik RouterOS 6.43.16 (long-term) CRS326...
40	31.775231038	172.16.10.1	172.16.10.255	ICMP	98	Echo (ping) request id=0x618f, seq=10/2560, ttl=64 (no response found!)
41	31.775238720	172.16.10.1	172.16.10.1	ICMP	98	Echo (ping) reply id=0x618f, seq=10/2560, ttl=64
42	31.997760122	Routerbo.1c:8c:9c	Spanning-tree-(for-br...	STP	80	RST, Root = 32768/0/c4:ad:34:1c:8c:9b Cost = 0 Port = 0x0002
43	32.795223343	172.16.10.1	172.16.10.255	ICMP	98	Echo (ping) request id=0x618f, seq=11/2816, ttl=64 (no response found!)
44	32.795231654	172.16.10.1	172.16.10.1	ICMP	98	Echo (ping) reply id=0x618f, seq=11/2816, ttl=64
45	33.823226430	172.16.10.1	172.16.10.255	ICMP	98	Echo (ping) request id=0x618f, seq=12/3072, ttl=64 (no response found!)
46	33.823234462	172.16.10.1	172.16.10.1	ICMP	98	Echo (ping) reply id=0x618f, seq=12/3072, ttl=64
47	34.088142710	Routerbo.1c:8c:9c	Spanning-tree-(for-br...	STP	80	RST, Root = 32768/0/c4:ad:34:1c:8c:9b Cost = 0 Port = 0x0002
48	34.843237244	172.16.10.1	172.16.10.255	ICMP	98	Echo (ping) request id=0x618f, seq=13/3328, ttl=64 (no response found!)
49	34.843267985	172.16.10.1	172.16.10.1	ICMP	98	Echo (ping) reply id=0x618f, seq=13/3328, ttl=64
50	35.867200598	172.16.10.1	172.16.10.255	ICMP	98	Echo (ping) request id=0x618f, seq=14/3584, ttl=64 (no response found!)
51	35.867208448	172.16.10.1	172.16.10.1	ICMP	98	Echo (ping) reply id=0x618f, seq=14/3584, ttl=64

Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0

- IEEE 802.3 Ethernet
- Logical-Link Control
- Spanning Tree Protocol

eth1: <live capture in progress> Packets: 104 - Displayed: 104 (100.0%) Profile: Default

Terminal Capturing from eth1

### 2.3.1 - Tux13 pinging other interfaces

Applications Places System Sat 14 Dec, 09:53

File Edit View Search Terminal Help

```
root@tux13:~# route add -net 172.16.11.0/24 gw 172.16.10.254
root@tux13:~# route -n
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
0.0.0.0 0.0.0.0 0.0.0.0 U 0 0 0 eth0
172.16.10.0 0.0.0.0 255.255.255.0 U 0 0 0 eth1
172.16.11.0 172.16.10.254 255.255.255.0 UG 0 0 0 eth1

root@tux13:~# ping 172.16.10.254
PING 172.16.10.254 (172.16.10.254) 56(84) bytes of data.
64 bytes from 172.16.10.254: icmp_seq=1 ttl=64 time=0.172 ms
64 bytes from 172.16.10.254: icmp_seq=2 ttl=64 time=0.181 ms
64 bytes from 172.16.10.254: icmp_seq=3 ttl=64 time=0.190 ms
64 bytes from 172.16.10.254: icmp_seq=4 ttl=64 time=0.172 ms
64 bytes from 172.16.10.254: icmp_seq=5 ttl=64 time=0.153 ms
^C
--- 172.16.10.254 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 81ms
rtt min/avg/max/mdev = 0.153/0.173/0.190/0.018 ms

root@tux13:~# ping 172.16.11.253
PING 172.16.11.253 (172.16.11.253) 56(84) bytes of data.
64 bytes from 172.16.11.253: icmp_seq=1 ttl=64 time=0.178 ms
64 bytes from 172.16.11.253: icmp_seq=2 ttl=64 time=0.165 ms
64 bytes from 172.16.11.253: icmp_seq=3 ttl=64 time=0.181 ms
64 bytes from 172.16.11.253: icmp_seq=4 ttl=64 time=0.178 ms
64 bytes from 172.16.11.253: icmp_seq=5 ttl=64 time=0.161 ms
^C
--- 172.16.11.253 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 89ms
rtt min/avg/max/mdev = 0.161/0.171/0.181/0.007 ms

root@tux13:~# ping 172.16.11.1
PING 172.16.11.1 (172.16.11.1) 56(84) bytes of data.
64 bytes from 172.16.11.1: icmp_seq=1 ttl=63 time=0.417 ms
64 bytes from 172.16.11.1: icmp_seq=2 ttl=63 time=0.320 ms
64 bytes from 172.16.11.1: icmp_seq=3 ttl=63 time=0.298 ms
64 bytes from 172.16.11.1: icmp_seq=4 ttl=63 time=0.297 ms
64 bytes from 172.16.11.1: icmp_seq=5 ttl=63 time=0.296 ms
^C
--- 172.16.11.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 89ms
rtt min/avg/max/mdev = 0.296/0.325/0.417/0.050 ms

root@tux13:~#
```

eth1: <live capture in progress> Packets: 65 - Displayed: 35 (53.8%) - Dropped: 0 (0.0%) Profile: Default

Terminal Capturing from eth1

### 2.3.2 - Tux 14 capturing both network interfaces

Applications Places System

Sat 14 Dec, 09:57

\*eth2

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Expression... +

Apply a display filter - <Ctrl-F>

Expression... +

Frame 29: 110 bytes on wire (880 bits), 110 bytes captured (880 bits) on interface 0  
Ethernet II, Src: Routerbo\_1c:8c:9a (c4:ad:34:1c:8c:9a), Dst: LLDP\_Multicast (01:80:c2:00:00:00)  
Link Layer Discovery Protocol

Frame 59: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0  
IEEE 802.3 Ethernet  
Logical-Link Control  
Spanning Tree Protocol

wireshark.fpcapng Packets: 87 - Displayed: 22 (25.3%) - Dropped: 0 (0.0%) Profile: Default

Terminal

\*eth1

\*eth2

## 2.4.1 - Tux13 pinging all network interfaces

Applications Places System

Sat 14 Dec, 10:44

Terminal

File Edit View Search Terminal Help

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Expression... +

Apply a display filter - <Ctrl-F>

Expression... +

Frame 4: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0  
Ethernet II, Src: EdimaxTe\_ed:bb:37 (00:50:fc:04:20:99) (00:50:fc:04:20:99)  
Internet Protocol Version 4, Src: 172.16.10.1, Dst: 172.16.10.254  
Internet Control Message Protocol

eth1: live capture in progress Packets: 82 - Displayed: 41 (50.0%) Profile: Default



## 2.4.2 - Tux12 pinging Tux13 through Rc

The screenshot shows a Linux terminal window on the left and a Wireshark network capture window on the right. The terminal window displays the following commands and output:

```
root@tux12:~# route del -net 172.16.10.0 gw 172.16.11.253 netmask 255.255.255.0
root@tux12:~# ping 172.16.10.1
PING 172.16.10.1 (172.16.10.1) 56(84) bytes of data:
^C
--- 172.16.10.1 ping statistics ---
25 packets transmitted, 0 received, 100% packet loss, time 606ms

root@tux12:~# route -n
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use
0.0.0.0 10.227.20.254 0.0.0.0 UG 0 0
10.227.20.0 0.0.0.0 255.255.255.0 U 0 0
172.16.1.0 172.16.11.254 255.255.255.0 UG 0 0
172.16.11.0 0.0.0.0 255.255.255.0 U 0 0

root@tux12:~# route add -net 172.16.10.0/24 gw 172.16.11.254
root@tux12:~# ping 172.16.10.1
PING 172.16.10.1 (172.16.10.1) 56(84) bytes of data:
64 bytes from 172.16.10.1: icmp_seq=1 ttl=63 time=0.456 ms
From 172.16.11.254: icmp_seq=2 Redirect Host(New nexthop: 172.16.11.253)
64 bytes from 172.16.10.1: icmp_seq=2 ttl=63 time=0.425 ms
From 172.16.11.254: icmp_seq=3 Redirect Host(New nexthop: 172.16.11.253)
64 bytes from 172.16.10.1: icmp_seq=3 ttl=63 time=0.372 ms
From 172.16.11.254: icmp_seq=4 Redirect Host(New nexthop: 172.16.11.253)
64 bytes from 172.16.10.1: icmp_seq=4 ttl=63 time=0.526 ms
From 172.16.11.254: icmp_seq=5 Redirect Host(New nexthop: 172.16.11.253)
64 bytes from 172.16.10.1: icmp_seq=5 ttl=63 time=0.401 ms
From 172.16.11.254: icmp_seq=6 Redirect Host(New nexthop: 172.16.11.253)
64 bytes from 172.16.10.1: icmp_seq=6 ttl=63 time=0.351 ms
^C
--- 172.16.10.1 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 106ms
rtt min/avg/max/mdev = 0.351/0.421/0.526/0.063 ms
root@tux12:~#
```

The Wireshark window shows a capture from the eth1 interface. The packet list on the left shows several ICMP Echo (ping) requests and responses. The packet details on the right show the structure of an ICMP Echo request, including the type (8), code (0), identifier (0), and sequence number (1). The packet bytes on the right show the raw data of the ICMP Echo request.

## 2.4.3 - Tux12 traceroutes

```
root@tux12:~# traceroute -n 172.16.10.1
traceroute to 172.16.10.1 (172.16.10.1), 30 hops max, 60 byte packets
 1 172.16.11.254 0.178 ms 0.172 ms 0.172 ms
 2 172.16.11.253 0.271 ms 0.264 ms 0.266 ms
 3 172.16.10.1 0.410 ms 0.411 ms 0.393 ms
root@tux12:~# route add -net 172.16.10.0/24 gw 172.16.11.253
root@tux12:~# traceroute -n 172.16.10.1
traceroute to 172.16.10.1 (172.16.10.1), 30 hops max, 60 byte packets
 1 172.16.11.253 0.180 ms 0.180 ms 0.153 ms
 2 172.16.10.1 0.273 ms 0.266 ms 0.271 ms
```

## 2.4.4 - Exp 4 step 7

The screenshot shows a terminal window on the left and a Wireshark packet capture window on the right. The terminal window displays the output of several commands, including route and ping, and shows the configuration of a network interface. The Wireshark window shows a list of captured packets, with the selected packet being an ICMP Echo (ping) request from 172.16.1.10 to 172.16.1.10. The packet details pane shows the ICMP Echo (ping) request with a sequence number of 11 and a time-to-live of 62. The packet bytes pane shows the raw data of the packet.

```
64 bytes from 172.16.11.254: icmp_seq=2 ttl=63 time=0.288 ms
64 bytes from 172.16.11.254: icmp_seq=3 ttl=63 time=0.286 ms
64 bytes from 172.16.11.254: icmp_seq=4 ttl=63 time=0.288 ms
64 bytes from 172.16.11.254: icmp_seq=5 ttl=63 time=0.311 ms
^C
GkTerm - /dev/tty50 115200-8-N-1

--- 172.16.1.10 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 256ms
rtt min/avg/max/mdev = 0.375/0.434/0.549/0.062 ms
root@tux13:~# ping 172.16.1.10
PING 172.16.1.10 (172.16.1.10) 56(84) bytes of data:
^C
--- 172.16.1.10 ping statistics ---
18 packets transmitted, 0 received, 100% packet loss, time 43ms
root@tux13:~#
```

Wireshark packet capture details:

No.	Time	Source	Destination	Protocol	Length	Info
20	8.515519617	Kye_B4:20:99	EdimaxTe_ed:bb:37	ARP	60	172.16.10.254 is at 00:c0:00:00:00:00
21	8.520691063	Kye_B4:20:99	EdimaxTe_ed:bb:37	ARP	60	Who has 172.16.10.17 Tell me
22	8.520783565	EdimaxTe_ed:bb:37	Kye_B4:20:99	ARP	42	172.16.10.1 is at 00:50:00:00:00:00
23	8.579443866	172.16.10.1	172.16.1.10	ICMP	98	Echo (ping) request id=0
24	8.579863631	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) reply id=0
25	9.603443724	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) request id=0
26	9.603887316	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) reply id=0
28	10.627446564	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) request id=0
29	10.627904791	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) reply id=0
30	11.651449754	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) request id=0
31	11.651804616	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) reply id=0
33	12.675447008	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) request id=0
34	12.675863958	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) reply id=0
35	13.699445308	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) request id=0
36	13.699811764	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) reply id=0
77	80.621865550	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) request id=0
78	80.651442638	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) reply id=0
80	88.675442085	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) request id=0
81	89.699444199	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) request id=0
83	90.723443198	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) request id=0
84	91.715412873	EdimaxTe_ed:bb:37	Kye_B4:20:99	ARP	42	Who has 172.16.10.254? Tell me
85	91.715519837	Kye_B4:20:99	EdimaxTe_ed:bb:37	ARP	60	172.16.10.254 is at 00:c0:00:00:00:00
86	91.747438496	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) request id=0
88	92.771443882	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) request id=0
89	93.795443898	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) request id=0
91	94.819442199	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) request id=0
92	95.843443814	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) request id=0
94	96.897448966	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) request id=0
95	97.891448355	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) request id=0
97	98.915442666	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) request id=0
98	99.939456262	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) request id=0
100	100.963447788	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) request id=0
101	101.987443226	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) request id=0
103	103.811447533	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) request id=0
104	104.835442591	172.16.1.10	172.16.1.10	ICMP	98	Echo (ping) request id=0

## 2.5.1 - DNS configuration

The screenshot shows a terminal window with the GNU nano 3.2 text editor open, editing the /etc/resolv.conf file. The file contains the following content:

```
domain netlab.fe.up.pt
search netlab.fe.up.pt
nameserver 10.227.20.3
```

## 2.5.2 - Tux12 testing if names can be used in hosts

The screenshot displays a Linux desktop environment with two windows open. The left window is a terminal titled 'Terminal' showing network configuration and testing commands. The right window is titled 'Capturing from eth1' and shows a packet capture interface with a list of captured packets and a detailed view of the selected packet.

**Terminal Window:**

```
root@tux12:~# route add -net 172.16.10.0/24 gw 172.16.11.253
root@tux12:~# traceroute -n 172.16.10.1
traceroute to 172.16.10.1 (172.16.10.1), 30 hops max, 60 byte packets
 1 172.16.11.253 0.180 ms 0.170 ms 0.153 ms
 2 172.16.10.1 0.273 ms 0.266 ms 0.271 ms
root@tux12:~# sysctl net.ipv4.conf.all.accept_redirects=1
net.ipv4.conf.all.accept_redirects = 1
root@tux12:~# sysctl net.ipv4.conf.eth1.accept_redirects=1
net.ipv4.conf.eth1.accept_redirects = 1
root@tux12:~# route del -net 172.16.10.0 gw 172.16.11.253 netmask 255.255.255.0
root@tux12:~# traceroute -n 172.16.10.1
traceroute to 172.16.10.1 (172.16.10.1), 30 hops max, 60 byte packets
 1 172.16.11.254 0.200 ms 0.191 ms 0.200 ms
 2 172.16.11.253 0.305 ms 0.290 ms 0.300 ms
 3 172.16.10.1 0.443 ms 0.429 ms 0.434 ms
root@tux12:~# sysctl net.ipv4.conf.all.accept_redirects=0
net.ipv4.conf.all.accept_redirects = 0
root@tux12:~# sysctl net.ipv4.conf.eth1.accept_redirects=0
net.ipv4.conf.eth1.accept_redirects = 0
root@tux12:~# traceroute -n 172.16.10.1
traceroute to 172.16.10.1 (172.16.10.1), 30 hops max, 60 byte packets
 1 172.16.11.254 0.169 ms 0.147 ms 0.155 ms
 2 172.16.11.253 0.248 ms 0.227 ms 0.211 ms
 3 172.16.10.1 0.302 ms 0.285 ms 0.297 ms
root@tux12:~# route add -net 172.16.10.0/24 gw 172.16.11.253
root@tux12:~# traceroute -n 172.16.10.1
traceroute to 172.16.10.1 (172.16.10.1), 30 hops max, 60 byte packets
 1 172.16.11.253 0.162 ms 0.139 ms 0.122 ms
 2 172.16.10.1 0.254 ms 0.239 ms 0.254 ms
root@tux12:~# nano /etc/resolv.conf
root@tux12:~# ping google.com
PING google.com (142.250.185.14) 56(84) bytes of data:
64 bytes from mad41s11-in-f14.1e100.net (142.250.185.14): icmp_seq=1
ttl=112 time=17.5 ms
64 bytes from mad41s11-in-f14.1e100.net (142.250.185.14): icmp_seq=2
ttl=112 time=17.3 ms
64 bytes from mad41s11-in-f14.1e100.net (142.250.185.14): icmp_seq=3
ttl=112 time=16.6 ms
64 bytes from mad41s11-in-f14.1e100.net (142.250.185.14): icmp_seq=4
ttl=112 time=16.9 ms
^C
--- google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 7ms
rtt min/avg/max/mdev = 16.594/17.087/17.472/0.344 ms
root@tux12:~#
```

**Capturing from eth1 Window:**

The packet capture window shows a list of captured packets. The selected packet (No. 178) is an ICMP Echo (ping) packet from 172.16.11.254 to 172.16.11.1.

No.	Time	Source	Destination	Protocol	Length	Info
166	282.265845627	Routerbo_1c:8c:9a	Spanning-tree-(for...	STP	60	RS
167	284.267999581	Routerbo_1c:8c:9a	Spanning-tree-(for...	STP	60	RS
168	284.450843509	172.16.11.1	172.16.10.1	ICMP	98	Ecl
169	284.451269671	172.16.10.1	172.16.11.1	ICMP	98	Ecl
170	285.026549228	0.0.0.0	255.255.255.255	MNDP	150	56
171	285.026579539	Routerbo_1c:8c:9a	CDP/VTP/DTP/PAGP/U...	CDP	93	Det
172	285.026631431	Routerbo_1c:8c:9a	LLDP_Multicast	LLDP	110	TT
173	285.456249437	172.16.11.1	172.16.10.1	ICMP	98	Ecl
174	285.456412935	Routerbo_ea:ae:33	Broadcast	ARP	60	Wh
175	285.456421665	Netronix_50:35:0c	Routerbo_ea:ae:33	ARP	42	17
176	285.456516649	172.16.11.254	172.16.11.1	ICMP	120	Re
177	285.456649697	172.16.10.1	172.16.11.1	ICMP	98	Ecl
178	286.278153371	Routerbo_1c:8c:9a	Spanning-tree-(for...	STP	60	RS
179	286.480250252	172.16.11.1	172.16.10.1	ICMP	98	Ecl
180	286.480400830	172.16.11.254	172.16.11.1	ICMP	120	Re
181	286.480597014	172.16.10.1	172.16.11.1	ICMP	98	Ecl
182	287.504502211	172.16.11.1	172.16.10.1	ICMP	98	Ecl
183	287.504686173	172.16.11.254	172.16.11.1	ICMP	120	Re
184	287.504874325	172.16.10.1	172.16.11.1	ICMP	98	Ecl
185	287.522337008	Routerbo_1c:8c:9a	Spanning-tree-(for...	STP	60	RS
186	288.528259966	172.16.11.1	172.16.10.1	ICMP	98	Ecl
187	288.528414106	172.16.11.254	172.16.11.1	ICMP	120	Re
188	288.528633198	172.16.10.1	172.16.11.1	ICMP	98	Ecl
189	289.477544520	Tp-LinkT_02:03:02	Netronix_50:35:0c	ARP	60	Wh
190	289.477565822	Netronix_50:35:0c	Tp-LinkT_02:03:02	ARP	42	17
191	289.488210957	Netronix_50:35:0c	Routerbo_ea:ae:33	ARP	42	Wh
192	289.488320746	Routerbo_ea:ae:33	Netronix_50:35:0c	ARP	60	17
193	289.552250210	172.16.11.1	172.16.10.1	ICMP	98	Ecl
194	289.552392604	172.16.11.254	172.16.11.1	ICMP	120	Re
195	289.552574559	172.16.10.1	172.16.11.1	ICMP	98	Ecl
196	289.274474216	Routerbo_1c:8c:9a	Spanning-tree-(for...	STP	60	RS

Frame 178: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0  
• IEEE 802.3 Ethernet  
• Logical-Link Control  
• Spanning Tree Protocol

eth1: <live capture in progress> Packets: 1011 · Displayed: 1011 (100.0%) Profile: Default



## 2.5.3 - Tux12 pinging google.com logs

The screenshot shows a Wireshark capture of a network traffic. The top bar indicates the capture is on interface \*eth0. The packet list pane shows a series of ICMP Echo (ping) requests and responses. The packet details pane shows the structure of the ICMP Echo request, including the type, code, identifier, and sequence number. The packet bytes pane shows the raw data of the packet.

No.	Time	Source	Destination	Protocol	Length	Info
946	88.998441033	Cisco_a1:3a:8c	Spanning-tree (for_	STP	60	Conf. Root = 32768/0/4c:00:82:2e:0a:00 Cost = 23 Port = 0x800c
946	89.996027944	10.227.20.12	10.227.20.3	DNS	86	Standard query 0x218e PTR 93.243.107.34.in-addr.arpa
947	89.9960579399	10.227.20.3	10.227.20.12	DNS	138	Standard query response 0x218e PTR 93.243.107.34.in-addr.arpa PTR 93.243.107.34.bc.google
950	89.382435126	bc:24:11:e7:5e:5b	HewlettP_61:2e:c3	ARP	60	Who has 10.227.20.12? Tell 10.227.20.3
950	89.382469269	HewlettP_61:2e:c3	bc:24:11:e7:5e:5b	ARP	42	10.227.20.12 is at 00:21:5a:61:2e:c3
968	90.011758835	Cisco_a1:3a:8c	Spanning-tree (for_	STP	60	Conf. Root = 32768/0/4c:00:82:2e:0a:00 Cost = 23 Port = 0x800c
973	91.990111802	10.227.20.12	10.227.20.3	DNS	70	Standard query 0x175a A google.com
974	91.990126888	10.227.20.12	10.227.20.3	DNS	70	Standard query 0x3b63 AAAA google.com
975	91.990648880	10.227.20.3	10.227.20.12	DNS	86	Standard query response 0x175a A google.com A 142.250.200.142
976	91.990668854	10.227.20.3	10.227.20.12	DNS	98	Standard query response 0x3b63 AAAA google.com AAAA 2a00:1450:4003:80f::200e
977	91.990974199	10.227.20.12	142.250.200.142	ICMP	98	Echo (ping) request id=0x4a2d, seq=1/256, ttl=64 (reply in 978)
978	92.000417200	142.250.200.142	10.227.20.12	ICMP	98	Echo (ping) reply id=0x4a2d, seq=1/256, ttl=112 (request in 977)
979	92.000601659	10.227.20.12	10.227.20.3	DNS	98	Standard query 0xe017 PTR 142.200.250.142.in-addr.arpa
980	92.000957582	10.227.20.3	10.227.20.12	DNS	127	Standard query response 0xe017 PTR 142.200.250.142.in-addr.arpa PTR mad41s14-in-f14.1e100
988	92.024872213	Cisco_a1:3a:8c	Spanning-tree (for_	STP	60	Conf. Root = 32768/0/4c:00:82:2e:0a:00 Cost = 23 Port = 0x800c
989	92.991436104	10.227.20.12	142.250.200.142	ICMP	98	Echo (ping) request id=0x4a2d, seq=2/512, ttl=64 (reply in 990)
990	93.008347833	142.250.200.142	10.227.20.12	ICMP	98	Echo (ping) reply id=0x4a2d, seq=2/512, ttl=112 (request in 989)
992	93.350918542	Cisco_a1:3a:8c	Cisco_a1:3a:8c	LOOP	0	Reply
999	93.992450941	10.227.20.12	142.250.200.142	ICMP	98	Echo (ping) request id=0x4a2d, seq=3/768, ttl=64 (reply in 1000)
1000	94.009358998	142.250.200.142	10.227.20.12	ICMP	98	Echo (ping) reply id=0x4a2d, seq=3/768, ttl=112 (request in 999)
1011	94.012212455	Cisco_a1:3a:8c	Spanning-tree (for_	STP	60	Conf. Root = 32768/0/4c:00:82:2e:0a:00 Cost = 23 Port = 0x800c
1012	94.004442085	10.227.20.12	142.250.200.142	ICMP	98	Echo (ping) request id=0x4a2d, seq=4/1024, ttl=64 (reply in 1015)
1013	95.000542071	10.227.20.12	10.227.20.3	DNS	86	Standard query 0x074e PTR 93.243.107.34.in-addr.arpa
1014	95.001118819	10.227.20.3	10.227.20.12	DNS	138	Standard query response 0x074e PTR 93.243.107.34.in-addr.arpa PTR 93.243.107.34.bc.google
1015	95.011363452	142.250.200.142	10.227.20.12	ICMP	98	Echo (ping) reply id=0x4a2d, seq=4/1024, ttl=112 (request in 1012)
1035	96.504667755	193.136.152.72	10.227.20.220	NTP	90	NTP Version 4, server
1036	96.057649705	Cisco_a1:3a:8c	Spanning-tree (for_	STP	60	Conf. Root = 32768/0/4c:00:82:2e:0a:00 Cost = 23 Port = 0x800c
1037	96.999373582	Routerbo_20:25:c8	HewlettP_61:2e:c3	ARP	60	Who has 10.227.20.12? Tell 10.227.20.254
1038	96.999498261	HewlettP_61:2e:c3	Routerbo_20:25:c8	ARP	42	10.227.20.12 is at 00:21:5a:61:2e:c3
1050	97.012212455	Cisco_a1:3a:8c	Spanning-tree (for_	STP	60	Conf. Root = 32768/0/4c:00:82:2e:0a:00 Cost = 23 Port = 0x800c
1069	100.003905981	10.227.20.12	10.227.20.3	DNS	86	Standard query 0x220d PTR 93.243.107.34.in-addr.arpa
1070	100.004455910	10.227.20.3	10.227.20.12	DNS	138	Standard query response 0x220d PTR 93.243.107.34.in-addr.arpa PTR 93.243.107.34.bc.google
1076	100.984344178	Cisco_a1:3a:8c	Spanning-tree (for_	STP	60	Conf. Root = 32768/0/4c:00:82:2e:0a:00 Cost = 23 Port = 0x800c
1077	101.191260085	fe80::221:5aff:fe6_	ff02::fb	MDNS	180	Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR _nfs._tcp.local, "QM" questio
1078	101.191323975	10.227.20.43	224.0.0.251	MDNS	160	Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR _nfs._tcp.local, "QM" questio

Frame 1035: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface 0  
Ethernet II, Src: Routerbo\_20:25:c8 (ea:8d:8c:20:25:c8), Dst: Dellc\_0c:b2:b4 (84:2b:2b:0c:02:b4)  
Internet Protocol Version 4, Src: 193.136.152.72, Dst: 10.227.20.220

0000 84 2b 2b 0c 02 b4 e4 8d 8c 20 25 c8 08 00 45 00 +-+...%...E-  
0010 00 4c 00 b4 00 00 11 24 5d c1 08 00 48 0a e3 .L @ 6 4]...H...  
0020 14 dc 00 7b 85 ae 00 38 40 ec 24 c2 0a 00 00 .[- B F \$...-  
0030 01 21 00 00 06 e5 e4 08 91 d2 eb 07 e5 8a 0e 77 !:.....-W  
0040 bf c6 16 8e 37 cb 09 5a 24 2e eb 07 e8 7b 2d 3d ....7 iz \$...."=  
0050 56 49 eb 07 e8 7b 2d 48 4f 82 VI...{"H 0.

## 2.6.1 - Exp 6 Wireshark logs

The screenshot shows a Wireshark capture of a network traffic. The top bar indicates the capture is on interface \*eth1. The packet list pane shows a series of TCP and UDP packets. The packet details pane shows the structure of the TCP packet, including the source and destination ports. The packet bytes pane shows the raw data of the packet.

No.	Time	Source	Destination	Protocol	Length	Info
3	2.380151812	172.16.1.10	172.16.1.10	TCP	74	50258 → 21 [SYN] Seq=0 Win=64288 Len=0 MSS=1460 SACK_PERM TSval=2559450868 TSecr=0 WS=128
4	2.380729138	172.16.1.10	172.16.1.10	TCP	74	21 → 50258 [SYN, ACK] Seq=0 Ack=1 Win=65536 Len=0 MSS=1460 SACK_PERM TSval=3354191740 TSecr=2559450868 WS=128
5	2.380769436	172.16.1.10	172.16.1.10	TCP	66	50258 → 21 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2559450868 TSecr=3354191740
6	2.380809732	172.16.1.10	172.16.1.10	FTP	116	Response: 220 ProFTPD Server (Debian) [1:ffff:172.16.1.10]
7	2.380917909	172.16.1.10	172.16.1.10	TCP	66	50258 → 21 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2559450868 TSecr=3354191753
8	2.380970699	172.16.1.10	172.16.1.10	FTP	77	Request: USER rcom
9	2.380980977	172.16.1.10	172.16.1.10	TCP	66	21 → 50258 [ACK] Seq=1 Ack=12 Win=65280 Len=0 TSval=3354191754 TSecr=2559450868
10	2.380989717	172.16.1.10	172.16.1.10	FTP	98	Response: 331 Password required for rcom
11	2.380743108	172.16.1.10	172.16.1.10	FTP	77	Request: PASS rcom
12	2.347792301	172.16.1.10	172.16.1.10	TCP	66	21 → 50258 [ACK] Seq=3 Ack=23 Win=65280 Len=0 TSval=3354191786 TSecr=2559450868
13	2.403713841	172.16.1.10	172.16.1.10	FTP	112	Response: 230 Welcome, archive user rcom[172.16.1.10]
14	2.403737671	172.16.1.10	172.16.1.10	FTP	69	Response:
15	2.403744825	172.16.1.10	172.16.1.10	FTP	112	Response: The local time is: Sat Dec 14 11:25:08 2024
16	2.403770837	172.16.1.10	172.16.1.10	FTP	69	Response:
17	2.403818266	172.16.1.10	172.16.1.10	FTP	139	Response: This is an experimental FTP server. If you have any unusual problems,
18	2.403840805	172.16.1.10	172.16.1.10	FTP	129	Response: please report them via e-mail to <root@ftp.netlab.fe.up.pt>.
19	2.403927916	172.16.1.10	172.16.1.10	FTP	94	Response:
20	2.40410131	172.16.1.10	172.16.1.10	TCP	66	50258 → 21 [ACK] Seq=23 Ack=345 Win=64128 Len=0 TSval=2559450868 TSecr=3354191931
21	2.404148474	172.16.1.10	172.16.1.10	FTP	72	Request: PASV
22	2.404419387	172.16.1.10	172.16.1.10	TCP	66	21 → 50258 [ACK] Seq=345 Ack=29 Win=65280 Len=0 TSval=3354191932 TSecr=2559450868
23	2.404547189	172.16.1.10	172.16.1.10	FTP	116	Response: 227 Entering Passive Mode (172,16,1,10,159)
24	2.405136794	172.16.1.10	172.16.1.10	TCP	74	50258 → 38559 [SYN] Seq=0 Win=64288 Len=0 MSS=1460 SACK_PERM TSval=2559450869 TSecr=0 WS=128
25	2.405448486	172.16.1.10	172.16.1.10	TCP	74	38559 → 50258 [SYN, ACK] Seq=0 Ack=1 Win=65536 Len=0 MSS=1460 SACK_PERM TSval=3354191933 TSecr=2559450869 WS=128
26	2.405449262	172.16.1.10	172.16.1.10	TCP	66	50258 → 38559 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2559450869 TSecr=3354191933
27	2.405483624	172.16.1.10	172.16.1.10	FTP	74	Request: TYPE I
28	2.405592818	172.16.1.10	172.16.1.10	FTP	85	Response: 200 Type set to I
29	2.405993463	172.16.1.10	172.16.1.10	FTP	81	Request: RETR pipe.txt
30	2.405978705	172.16.1.10	172.16.1.10	FTP	132	Response: 150 Opening ASCII mode data connection for pipe.txt (418 bytes)
31	2.407164084	172.16.1.10	172.16.1.10	FTP-DATA	484	FTP Data: 418 bytes (PASV) (TYPE I)
32	2.407056463	172.16.1.10	172.16.1.10	TCP	66	50258 → 38559 [ACK] Seq=1 Ack=419 Win=64128 Len=0 TSval=2559450871 TSecr=3354191935
33	2.407091488	172.16.1.10	172.16.1.10	TCP	66	38559 → 50258 [ACK] Seq=1 Ack=1 Win=65280 Len=0 TSval=3354191935 TSecr=2559450869
34	2.527238586	172.16.1.10	172.16.1.10	TCP	66	50258 → 38559 [ACK] Seq=1 Ack=420 Win=64128 Len=0 TSval=2559450871 TSecr=3354191935
35	2.527237718	172.16.1.10	172.16.1.10	TCP	66	50258 → 21 [ACK] Seq=2 Ack=409 Win=64128 Len=0 TSval=2559450861 TSecr=3354191935
38	7.372831445	KVE_04:20:99	EdinaTechno.edibb_	ARP	60	Who has 172.16.1.1? Tell 172.16.1.254
39	7.37284766	EdinaTechno.edibb_	KVE_04:20:99	ARP	42	172.16.1.1 is at 00:50:fced:bb:37
42	11.002420688	172.16.1.10	172.16.1.10	TCP	66	38559 → 4021 [FIN, ACK] Seq=1 Ack=1 Win=64128 Len=0 TSval=2559450921 TSecr=3354136132
44	13.288229883	172.16.1.10	172.16.1.10	FTP	89	Response: 226 Transfer complete
45	13.288232780	172.16.1.10	172.16.1.10	TCP	66	50258 → 21 [ACK] Seq=2 Ack=407 Win=64128 Len=0 TSval=2559461372 TSecr=3354202737
46	13.288341277	172.16.1.10	172.16.1.10	TCP	66	50258 → 38559 [FIN, ACK] Seq=1 Ack=420 Win=64128 Len=0 TSval=2559461372 TSecr=3354191935
47	13.288379122	172.16.1.10	172.16.1.10	FTP	72	Request: QUIT
48	13.290920201	172.16.1.10	172.16.1.10	FTP	68	Response: 221 Goodbye.
49	13.289083224	172.16.1.10	172.16.1.10	TCP	66	50258 → 21 [FIN, ACK] Seq=58 Ack=517 Win=64128 Len=0 TSval=2559461373 TSecr=3354202737
50	13.289228664	172.16.1.10	172.16.1.10	TCP	66	21 → 50258 [FIN, ACK] Seq=57 Ack=58 Win=65280 Len=0 TSval=3354202738 TSecr=2559461372
51	13.289237617	172.16.1.10	172.16.1.10	TCP	66	50258 → 21 [ACK] Seq=58 Ack=518 Win=64128 Len=0 TSval=2559461373 TSecr=3354202738
52	13.289238707	172.16.1.10	172.16.1.10	TCP	66	21 → 50258 [ACK] Seq=518 Ack=59 Win=65280 Len=0 TSval=3354202739 TSecr=2559461373
53	14.002212455	172.16.1.10	172.16.1.10	TCP	66	50258 → 38559 [ACK] Seq=1 Ack=420 Win=64128 Len=0 TSval=2559461375 TSecr=3354191935
54	13.095227960	172.16.1.10	172.16.1.10	TCP	66	[TCP Retransmission] 50258 → 38559 [FIN, ACK] Seq=1 Ack=420 Win=64128 Len=0 TSval=2559461779 TSecr=3354191935
56	14.193241110	172.16.1.10	172.16.1.10	TCP	66	[TCP Retransmission] 50258 → 38559 [FIN, ACK] Seq=1 Ack=420 Win=64128 Len=0 TSval=2559462203 TSecr=3354191935
57	14.051242772	172.16.1.10	172.16.1.10	TCP	66	[TCP Retransmission] 50258 → 38559 [FIN, ACK] Seq=1 Ack=420 Win=64128 Len=0 TSval=2559464305 TSecr=3354191935
58	14.051242772	172.16.1.10	172.16.1.10	TCP	66	[TCP Retransmission] 50258 → 38559 [FIN, ACK] Seq=1 Ack=420 Win=64128 Len=0 TSval=2559464305 TSecr=3354191935