Computer Networks

First Laboratory Work

Gabriel Carvalho (up202208939@up.pt)
Vasco Melo (up202207564@up.pt)

Summary:

This project, carried out as part of the Computer Networks course unit, focuses on implementing a download program that follows the file transfer protocol, as well as establishing, configuring and utilizing a computer network.

Through this project, we were able to apply the knowledge gained in the theoretical lessons and consolidate it through the implementation of the program with the specified protocol and the network configuration.

Introduction:

The goal of this project was to develop and test a download program using FTP, as well as to configure a computer network, in accordance with the specifications outlined in the guidelines, with the final objective of transferring a file from the internet using the configured network.

This report is structured into three sections, each focusing on a specific aspect of the project:

1. **Downloader:** This section describes the architecture of the download application and the results we obtained from its implementation
2. **Network Configuration and Analysis:** This section includes the documentation, and log analysis for all the experiments made throughout the project
3. **Conclusions:** The final section synthesizes the information presented in the previous one, some reflections and conclusions as well as potential improvements.

1. Downloader:

Application Architecture

The objective of the first part of the project was to develop an application capable of performing file downloads using the FTP protocol. To accomplish this, the FTP protocol specification and pre-existing FTP applications were used to gain an understanding of socket programming and its implementation. All references consulted are documented in the reference section.

When the application is initialized, a URL is provided by the user. This URL follows a fixed structure, which makes the parsing process universal for any URL. Our URL parsing

begins by validating the URL format, then proceeds to identify the location of the URL separators (e.g., the "@" symbol separates the username and authentication password). The relevant fields required for file transfer are extracted in the following order: user and password for server authentication, which are set to default values (user: ftp, password: eu@) if not specified in the input; hostname, the server name where the communication will be established; port, the FTP server port, which defaults to 21 if not specified in the URL; filepath, the path to the desired file; and file, the name and extension of the file to be transferred. Finally, the server's IP address is obtained using the provided hostname and port, enabling the establishment of a socket connection to the server by the usage of *getaddrinfo* of the library *netdb.h*.

The application is based on socket programming, where it uses a stream socket as the communication medium between the application and the FTP server. The application sends commands (e.g., "USER") and the server responds with three-digit codes (e.g., 230 for successful login). To send commands to the FTP server, the *send* function is used, and to read the response, the *recv* function is used, both from the *sys/socket.h* library.

The communication between the application and the FTP server begins with the handshake process, during which the user's credentials are transmitted. This is accomplished using the "USER <username>" command to send the username and the "PASS <password>" command to transmit the password.

In the next phase, information is sent regarding how the download will be performed, including the file and its size. Authentication is then carried out, followed by sending the "PASV" command to put the server into Passive Mode. In response, the server sends the IP address and port number where it is listening, allowing the client to establish a connection. To do this, the client will create a second socket using the IP and port provided by the FTP server. Then, the "TYPE I" command is sent to inform the server that the file to be transferred is in binary format. Next, the "SIZE <filename>" command is sent to check the size of the desired file. Finally, the "RETR <file path>/<file name>" command is sent to the server to retrieve the <file name> file.

During the download phase, the second socket is used, created with the IP address and port provided by the FTP server after sending the "PASV" command. The FTP server will transmit the requested file content to the second socket, where the application will be listening to read the data in order to perform the download. The *write* function from the *sys/socket.h* library is used for this purpose. Once all bytes have been sent, the application reads the response sent by the FTP server through the first socket, which provides the download status (226 if the operation was successful).

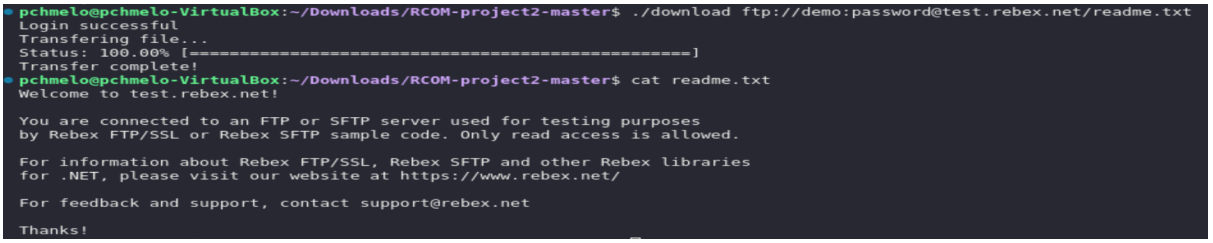
Finally, the application sends the "QUIT" command to terminate the connection between the application and the FTP server. Both sockets are then closed, marking the completion of the file download process.

Tests and Results

To test the FTP application, various file transfer tests were conducted with different sizes and types of files. Additionally, connection attempts were made to both public servers

provided by Moodle and the FEUP FTP server, with the latter yielding a positive result. The tests covered both anonymous and authenticated modes. Finally, the FTP application was successfully used to carry out Experiment 6 of the second part of the project.

Below is an example of a successful transfer (the Wireshark capture from Experiment 6 of the second part of the project, and an image showing all commands and responses exchanged between the client and the server of the example shown below has been included in the appendices).



Note: This image shows the download being made with the debug mode turned off. An image showing the process with the debug mode on is shown here.

2. Network Configuration and Analysis:

Experiment 1 - Configure an IP Network

This experiment's goal was to configure two IP addresses of two different computers, tux13 and tux14, both connected to a switch. We were also tasked with analyzing how the ARP protocol worked, as well as its behaviour when configured IP addresses are deleted from an ARP table.

In order to achieve this, we started by connecting the E1 of each tux to the switch and used the command ifconfig to configure their respective IPs. After which, we used the ifconfig command again, but this time to visualize the IP and MAC addresses of each respective computer, which were respectively 172.16.10.1/24 and 00:50:fc:ed:bb:37 for tux13, and 172.16.10.254/24 and 00:c0:df:04:20:99 for tux14.

So as to verify connectivity between the two computers we used the ping command which generates and sends via broadcast ARP packets, a type of network protocol used to map an IP address to a MAC address in a local network, effectively connecting the data link layer(MAC) to the network layer(IP), until obtaining the destination MAC address, then sends ICMP Echo Request packets to initiate the communication. If the target device is reachable, it responds with ICMP Echo Reply packets. These ICMP packets allow the ping command to test connectivity between the two devices, measure round-trip time, and check if the target device responds correctly.

Another great way of troubleshooting, and testing network connectivity locally within a system is the loopback interface, which is a virtual network interface in a computer system

that allows a device to communicate with itself through its assigned IP address (usually 127.0.0.1 in IPv4).

During the ping command, when sending ARP packets, after a connection is established, it gets recorded the ARP table of both computers in order to make communication faster and more efficient by allowing devices to quickly resolve IP addresses to MAC addresses without requiring new ARP exchanges for every packet sent. When these entries are deleted, an exchange needs to be made again to restore the associations.

Additional information from this experiment, such as the size or type of a received ethernet frame can be seen in the Wireshark capture here, and the commands used here.

Experiment 2 - Implement two bridges in a switch

This experiment's goal was the creation of two local networks (LAN) through the use of bridges in the switch, one containing tux13 and tux14, and the other containing just tux12.

To do this, we began from where we stopped in experiment 1, connected the E1 of tux12 to the switch and used the ifconfig command to configure tux12 with the IP 172.16.11.1/24. In order to create the bridges, we must configure the switch by connecting its console with any tux's serial port. After which, we use the command '/interface bridge add' to create bridge0 and bridge1, use '/interface bridge port remove' to remove the ports each tux's interface was connected to by default, and finally used '/interface bridge port add' to add the tuxes 13 and 14 to bridge0 and tux12 to bridge1.

In this experiment, there were two broadcast domains, each corresponding to one of the two local networks we had implemented. When broadcasting from tux13, tux14 could be reached but tux12 couldn't. This happened because tux13 and 14 were on the same LAN, whereas tux12 resides in an isolated LAN. Doing the same in tux12 showed that both other tuxes were unreachable, due to the same reason stated previously.

The conclusions we reached in the last paragraph were based on the analyses of the Wireshark captures. We observed that in tux13's broadcast, both ICMP requests and replies were present. However, in tux12's broadcast, only ICMP requests were seen, with no replies.

Additional information from this experiment can be seen in the Wireshark captures here, and the commands used here.

Experiment 3 - Configure a Router in Linux

This experiment's goal was to transform tux14 into a router, so as to tux13 and tux12 will be able to communicate through tux14, despite being on different LANs.

In order to complete our goal, we began from where we last stopped on experiment 2 and connected tux4's E2 to the switch, making sure to connect it to bridge1 using '/interface bridge port add'. After which, we used ifconfig to make its IP address 172.16.11.253/24, activated IP forwarding, and disabled ICMP echo ignore broadcasts. Lastly, using the 'route add' command we created new routes in tux12 and 13, using tux14 as a gateway, accessible by them both through 172.16.11.253/24 and 172.16.10.254/24 respectively.

Now, in tux13, after utilizing the ping command to tux12, all the ARP and ICMP requests would return a corresponding reply. The ARP and ICMP packets processed by Tux54 contained the destination machine's IP address but the MAC address of Tux54 itself, since it, like a router, is responsible for redirecting information between the two networks it has created. The routing tables generated through the creation of routes ensure that for every destination IP, there is another IP address (gateway) to which the source machine should forward the information.

Additional information from this experiment can be seen in the Wireshark captures here, and the commands used here.

Experiment 4 - Configure a Commercial Router and Implement NAT

This experiment's goal was to configure a commercial router with implemented NAT, and add it to bridge1 with the objective of obtaining access to the internet.

To do this, we started from where we last stopped in experiment 3 and connected ether1 and ether2 of the commercial router to PY.12 and the switch respectively. Later, we added the interface to bridge1 in the same way as stated in previous experiments, and removed the cable connected to the switch's console, connecting it to routerMT, so as to configure the router. On the router's console, we configured the IPs corresponding to each interface with the command '/ip address add'. Lastly, we created default routes in each tux connecting them to the router, and another route that would allow the router access to both LANs through tux14.

After removing the route connecting Tux12 to Tux14 and disabling ICMP redirects, when pinging tux13 from 12, the data packets were forwarded through the router by the default route until reaching the destination IP. In a second test, after enabling redirects once again, the packets didn't pass through the router due to a more direct route, this being to simply use the tux14, connecting the two LANs. The tests allowed us to conclude that ICMP packets follow the path of least resistance, performing redirects only when necessary.

We were also asked in this experience to ping an FTP server with and without NAT enabled, using the command '/ip firewall nat enable/disable 0', and observe the results. NAT (Network Address Translation) is a method that modifies the source or destination IP while they are in transit across a routing device, such as a router or firewall. It is used to enable multiple devices on a private network to access the internet, by translating their private IP addresses, used only in the local network, into a public IP address that is accessible over the internet. This allows devices to address the issues stemming from the limited number of addresses allowed by the ipv4 protocol.

With NAT enabled we could verify that a connection to the internet was established. Whereas, when we disabled it, a connection couldn't be established, due to the router not being able to translate the destination address received.

Additional information from this experiment can be seen in the Wireshark captures here, and the commands used here.

Experiment 5 - DNS

The goal of this experiment was to configure the DNS (Domain Name System), so as to, using the network we configured throughout these experiments, be able to access the internet and ping a website.

In order to achieve this, all we needed to do was change the content of the file '/etc/resolv.conf', in each tux, to include the following line: 'nameserver 10.227.20.3'. Now we were able to reach the internet and successfully pinged google.

In the Wireshark capture logs we observed that the DNS packets were the first ones being exchanged and happened before any other protocol. The reason for this is that all the protocols down the line will need the domain's IP address in order to work, and to obtain that address, the DNS is needed.

Additional information from this experiment can be seen in the Wireshark captures here, and the commands used here.

Experiment 6 - TCP connections

The goal of this experiment is to combine both the downloader, made in the first part of the project, and the configured network, established in the second, in order to analyse the network traffic captured during FTP file transfer so as to evaluate packet transmission, internal packet structure, and TCP's flow control and congestion control mechanisms.

By examining the Wireshark capture, we observed multiple TCP connections established for FTP operations. The initial connection setup showed the TCP handshake with SYN-ACK packets establishing the control channel on port 21. The capture revealed that the FTP application establishes two distinct TCP connections: a control connection for sending commands and a separate data connection for file transfer.

The FTP control connection carries commands and responses between client and server, as evidenced by messages such as "ProFTPD Server (Debian)" responses and user authentication exchanges. These control messages operate over a dedicated TCP connection, maintaining separation from the data transfer channel.

All observed TCP connections implement ARQ (Automatic Repeat Request) mechanisms, which ensure reliable data delivery by requiring receivers to acknowledge received packets and senders to retransmit lost packets. The capture shows this through sequence numbers (Seq), acknowledgment numbers (Ack), and window size fields (Win). For example, we observed consistent acknowledgment patterns with sequence numbers incrementing properly and corresponding acknowledgments confirming received data. The TCP connections maintained window sizes around 64256 bytes, indicating stable flow control conditions.

The flow control mechanism, which prevents a fast sender from overwhelming a slow receiver by allowing the receiver to specify how much data it can handle, is visible through the Window field in TCP headers. Throughout the capture, window size values remained

relatively stable, indicating that receiver buffer capacity was not a limiting factor during this transfer.

Regarding congestion control, which prevents network congestion by regulating the rate at which packets are sent, the capture shows TCP connections operating with Maximum Segment Size (MSS) values of 1460 bytes, as seen in the initial SYN packets. The relatively stable window sizes and consistent packet transmission patterns suggest that the network remained in a steady state during the capture period, without entering congestion scenarios that would trigger congestion avoidance mechanisms.

The transfer patterns visible in the capture align with TCP's standard congestion control mechanisms, though we don't see dramatic congestion window adjustments in this particular sample. This suggests the network conditions were favorable during the capture period, maintaining stable throughput without triggering congestion responses.

When examining the interaction between multiple TCP connections, the capture shows concurrent control and data connections operating without significant interference. The separation of control and data channels allows for efficient management of the FTP session while maintaining optimal data transfer rates. The consistent window sizes and acknowledgment patterns across both connections indicate that the network adequately handled the concurrent TCP streams without notable performance degradation.

Additional information from this experiment can be seen in the Wireshark captures here, and the commands used here.

3. Conclusions:

Through this project, we gained comprehensive insight into the intricate workings of network protocols and data transfer mechanisms in computer networks. The practical experiments enhanced our understanding of how data packets propagate through the network layer, including routing decisions and addressing schemes, and their subsequent transition to the link layer for physical transmission. This hands-on experience with protocol analysis and network traffic observation helped bridge the gap between theoretical networking concepts and their real-world implementation, providing valuable practical knowledge of modern network communications.

The project particularly illuminated the complex interactions between different networking layers, demonstrating how higher-level protocols like FTP rely on the robust mechanisms like TCP and IP, and how these protocols work together to ensure reliable and efficient data transfer across network infrastructure.

Annexes

References:

- https://datatracker.ietf.org/doc/html/rfc959
- https://github.com/thevickypedia/ftp_server_client
- https://github.com/literallysofia/feup-rcom/tree/master
- https://beej.us/guide/bgnet/pdf/bgnet_a4_c_2.pdf

Download Application Code:

download.h

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

#define DEBUG 0
#define RESP_BUFFER_SIZE 1024
#define FILE_BUFFER_SIZE 1024

typedef struct connectionInfo{
	char user[128];
	char password[128];
	char hostname[128];
	char port[8];
	char filepath[256];
	char filename[128];

}connectionInfo;

int parseUrl(char * url, connectionInfo* connInfo);
void ftpClient(connectionInfo cInfo);
void dataConnection(int sd2, char * filename, int filesize);
```

```
void loadingBar(float file_size_processed, float file_total_size);
```

download.c

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

/*
Protoclo FTP
1. Parse URL
2. Procura pelas resolved addresses com as informações do hostname e port
3. Cria uma socket com as características dos resolved addresses
3. Conecta a socket ao servidor
4. Recebe a resposta do servidor e espera pelo código 220
5. Envia o comando USER com o username e espera pelo código 331
6. Envia o comando PASS com a password e espera pelo código 230
7. Envia o comando PASV e espera pelo código 227
8. Com o IP e porta do PASV, cria uma nova socket e conecta ao servidor
9. Envia o comando TYPE I e espera pelo código 200
10. Envio o comando SIZE com o nome do ficheiro e espera pelo código 213, e
guarda o tamanho do ficheiro
11. Envia o comando RETR com o nome do ficheiro e espera pelo código 150v ou
125
12. Recebe o ficheiro pela segunda socket
13. Fecha a segunda socket
14. Recebe a resposta do servidor e espera pelo código 226, para confirmar a
transferência
15. Fecha a primeira socket
*/

int main(int argc, char** argv)
{
	if (argc!=2)
	{
		printf("Wrong number of arguments\n");
		return -1;
	}

	connectionInfo cInfo;

	if( parseUrl(argv[1], &cInfo) != 0 )
		printf("Wrong url input.\n");

	if (DEBUG){
		printf("\n<PARSE URL>\n");
```

```
                printf("user = %s\n", cInfo.user);
                printf("password = %s\n", cInfo.password);
                printf("hostname = %s\n", cInfo.hostname);
                printf("port = %s\n", cInfo.port);
                printf("filepath = %s\n", cInfo.filepath);
                printf("filename = %s\n\n", cInfo.filename);
        }

        ftpClient(cInfo);

        return 0;
}

int sendComand(int sd, char * userCmd, char * response)
{
        if(send(sd, userCmd, strlen(userCmd), 0) == -1){
                printf("Error sending USER cmd\n");
                exit(-1);
        }

        sleep(1);

        int received = recv(sd, response, RESP_BUFFER_SIZE, 0);
        if(received == -1){
                perror("Error receiving response from USER");
                exit(1);
        }

        response[received] = 0;


        return received;
}

void ftpClient(connectionInfo cInfo)
{
    //--- GET IP ---

        struct addrinfo hints;
        struct addrinfo *result;

        memset(&hints, 0, sizeof(struct addrinfo));
        hints.ai_family = AF_INET;                                      //IPv4
        hints.ai_socktype = SOCK_STREAM;                        //TCP
        hints.ai_flags = 0;
//no flags
        hints.ai_protocol = 0;
//Any protocol can be used
```

```
	//getaddringo returns the resolved addresses
	int r = getaddrinfo(cInfo.hostname, cInfo.port, &hints, &result);

	if (r != 0) {
		fprintf(stderr, "getaddrinfo failed: %s\n", gai_strerror(r));
		exit(-1);
	}

	//--- SOCKET ---

	//cria uma socket para o servidor ftp onde podem ser enviados comandos
	int sd = socket(result->ai_family, result->ai_socktype,
result->ai_protocol);
	if(sd == -1){
		perror("Error opening socket");
		exit(-1);
	}

	//--- CONNECT ---

	//make the socket connect to the server using the resolved address
	if(connect(sd, result->ai_addr, result->ai_addrlen) == -1){
		perror("Error connecting to server");
		exit(-1);
	}

	char response[RESP_BUFFER_SIZE];

	int received = recv(sd, response, RESP_BUFFER_SIZE, 0);
	if(received == -1){
		perror("Error receiving response from connect");
		exit(1);
	}
	response[received] = 0;
	if (DEBUG) printf("<CONNECT RESPONSE>\n%s\n", response);

	if(strncmp("220", response, 3) != 0){
		printf("Connection failed\n");
		exit(-1);
	}

	freeaddrinfo(result);

	//--- LOGIN ---

	char userCmd[128];
	char passCmd[128];
```

```c
sprintf(userCmd, "USER %s\r\n", cInfo.user);
if (DEBUG) printf("<USER CMD>\n%s\n", userCmd);

received = sendComand(sd, &userCmd, &response);
if (DEBUG) printf("<USER RESPONSE>\n%s\n", response);

if(strncmp("331", response, 3) != 0){
	printf("Error in USER\n");
	exit(-1);
}

sprintf(passCmd, "PASS %s\r\n", cInfo.password);
if (DEBUG) printf("<PASS CMD>\n%s\n", passCmd);

received = sendComand(sd, &passCmd, &response);
if (DEBUG) printf("<PASS RESPONSE>\n%s\n", response);

if(strncmp("530", response, 3) == 0){
	printf("Login incorrect\n");
	exit(-1);
}
else if(strncmp("230", response, 3) != 0)	{
	printf("Login failed\n");
	exit(-1);
}

if (!DEBUG) printf("Login successful\n");

//--- PASV ---

if (DEBUG) printf("<PASV CMD>\nPASV\n\n");

received = sendComand(sd, "PASV\r\n", response);
if (DEBUG) printf("<PASV RESPONSE>\n%s\n", response);

if(strncmp(response, "227", 3) != 0){
	printf("PASV request not accepted\n");
	exit(-1);
}

//--- 2ND SOCKET ---

int i = 0;
char *token; //Pointer to hold each token parsed from the response.
int parsedIP[6];
```

```
	token = strtok(response, "(,)"); //parse the response by the
delimiters '(', ')' and ','

	while( (token = strtok(NULL, "(,)")) != NULL && i < 6)
	{
		parsedIP[i] = atoi(token);
		//printf("Parsed value %d : %d\n", i, parsedIP[i]);
		i++;
	}
	if(i < 6){
		printf("Error parsing Pasv IP address.\n");
		exit(-1);
	}

	char pasvIpAddr[20];
	sprintf(pasvIpAddr, "%d.%d.%d.%d", parsedIP[0], parsedIP[1],
parsedIP[2], parsedIP[3]);

	struct sockaddr_in addr;
	addr.sin_family = AF_INET;

	addr.sin_port = htons( (parsedIP[4] << 8) + parsedIP[5] ); //cria a
porta para a socket multiplicando o byte 4 por 256 e somando o byte 5
	inet_aton(pasvIpAddr, (struct in_addr*) &addr.sin_addr.s_addr);
//converte o IP para uma struct in_addr

	if (DEBUG) printf("Parsed Pasv IP address - %s:%d\n\n", pasvIpAddr,
addr.sin_port);

	int sd2 = socket(AF_INET, SOCK_STREAM, 0);
	if( (connect(sd2, (const struct sockaddr*) &addr, sizeof(addr)) ) ==
-1){
		printf("Data connection not established.\n");
		exit(-1);
	}

	//--- TYPE ---

	if (DEBUG) printf("<TYPE CMD>\nTYPE I\n\n");

	received = sendComand(sd, "TYPE I\r\n", response);
	if (DEBUG) printf("<TYPE RESPONSE>\n%s\n", response);

	if(strncmp(response, "200", 3) != 0){
		printf("TYPE request not accepted\n");
		exit(-1);
	}
```

```
	//--- SIZE ---

	char sizeCmd[512];
	if(strlen(cInfo.filepath) > 0)
		sprintf(sizeCmd, "SIZE %s/%s\r\n", cInfo.filepath,
cInfo.filename);
	else
		sprintf(sizeCmd, "SIZE %s\r\n", cInfo.filename);

	if (DEBUG) printf("<SIZE CMD>\nSIZE\n\n");

	received = sendComand(sd, &sizeCmd, &response);
	if (DEBUG) printf("<SIZE RESPONSE>\n%s\n", response);

	if(strncmp("550", response, 3) == 0){
		printf("No such file or directory.\n");
		exit(-1);
	}
	else if(strncmp("213", response, 3) != 0){
		printf("No such file or directory.\n");
		exit(-1);
	}

	int filesize = atoi(response + 4);

	//--- RETR ---

	char retrCmd[512];
	if(strlen(cInfo.filepath) > 0)
		sprintf(retrCmd, "RETR %s/%s\r\n", cInfo.filepath,
cInfo.filename);
	else
		sprintf(retrCmd, "RETR %s\r\n", cInfo.filename);

	if (DEBUG) printf("<RETR CMD>\n%s\n", retrCmd);

	received = sendComand(sd, &retrCmd, &response);;
	if (DEBUG) printf("<RETR RESPONSE>\n%s\n", response);

	if(strncmp("150", response, 3) != 0 && strncmp("125", response, 3) !=
0){
		printf("File is unavailable/not found.\n");
		exit(-1);
	}

	if (DEBUG) printf("Transfering file (bytes received)\n");
	else printf("Transfering file...\n");
```

```
	//--- Transfer file  ---

	dataConnection(sd2, cInfo.filename, filesize);

	close(sd2);

	//--- END ---

	received = recv(sd, response, RESP_BUFFER_SIZE, 0);
	response[received] = 0;
	if (DEBUG) printf("\n<RETR FINAL RESPONSE>\n%s\n", response);

	if(strncmp(response, "226", 3) != 0){
		printf("Transfer failed.\n");
		exit(-1);
	}
	else{
		printf("Transfer complete!\n");
	}

	close(sd);
}

void dataConnection(int sd2, char * filename, int filesize)
{
	char buffer[FILE_BUFFER_SIZE];
	int bytesReceived;
	int totalBytes = 0;

	int fd = open(filename, O_CREAT|O_TRUNC|O_WRONLY, 0777);

	while (1) {

		bytesReceived = recv(sd2, buffer, FILE_BUFFER_SIZE, 0);

		if (bytesReceived == 0) {
			break;
		} else if (bytesReceived == -1) {
			perror("Error receiving file data");
			break;
		} else {
			int bytesWritten = 0;
			while (bytesWritten < bytesReceived) {
				int result = write(fd, buffer + bytesWritten,
bytesReceived - bytesWritten);
				if (result == -1) {
					perror("Error writing file data");
					break;
```

```
                        }
                        bytesWritten += result;
                }
                totalBytes += bytesReceived;
        }

        if (DEBUG) {
                printf("(%d)", bytesReceived);
        } else {
                loadingBar(totalBytes, filesize);
        }

        if (totalBytes == filesize) {
                break;
        }
    }
    printf("\n");

    close(fd);

    if (totalBytes != filesize){
          printf("File size mismatch: %d (received) / %d (expected)\n",
totalBytes, filesize);
          //TODO delete file
          exit(1);
    }
}

/*
ftp://user:pass@hostname:2121/path/to/file.txt, the function will extract:

user: "user"
password: "pass"
hostname: "hostname"
port: "2121"
filepath: "path/to"
filename: "file.txt"
*/

int parseUrl(char* url, connectionInfo *cInfo)
{
    int pos_arroba = -1;
    int first_bar = -1;
    int second_bar = -1;
    int first_colon = -1;
    int second_colon = -1;
    int third_bar = -1;
    int colon_count = 0;
```

```
    int anonymous = 1;
    unsigned int i;

    // First cycle to determine if url has a valid syntax;
    for (i = 0; i < strlen(url); i++) {
        if (url[i] == '@') {
            anonymous = 0;
            pos_arroba = i;
        }
        if (url[i] == '/') {
            if (first_bar == -1)
                first_bar = i;
            else if (second_bar == -1) {
                if (first_bar == i - 1)
                    second_bar = i;
                else
                    return -1;
            } else if (third_bar == -1)
                third_bar = i;
        }
        if (url[i] == ':' && first_bar != -1 && second_bar != -1) {
            if (colon_count > 1)
                return -3;
            else
                colon_count++;
        }
    }

    for (i = 0; i < strlen(url); i++) {
        if (url[i] == ':' && i > second_bar) {
            if (first_colon == -1)
                first_colon = i;
            else
                second_colon = i;
        }
    }

    int hostname_begin = 0;
    if (anonymous == 0) {
        memcpy(cInfo->user, url + second_bar + 1, first_colon - second_bar -
1);
        cInfo->user[first_colon - second_bar - 1] = '\0';

        memcpy(cInfo->password, url + first_colon + 1, pos_arroba -
first_colon - 1);
        cInfo->password[pos_arroba - first_colon - 1] = '\0';

        hostname_begin = pos_arroba;
```

```
    } else {
        strcpy(cInfo->user, "ftp");
        strcpy(cInfo->password, "eu@");

        hostname_begin = second_bar;
    }

    if (second_colon != -1) {
        memcpy(cInfo->hostname, url + hostname_begin + 1, second_colon -
hostname_begin - 1);
        cInfo->hostname[second_colon - hostname_begin - 1] = '\0';

        memcpy(cInfo->port, url + second_colon + 1, third_bar - second_colon
- 1);
        cInfo->port[third_bar - second_colon - 1] = '\0';
    } else {
        memcpy(cInfo->hostname, url + hostname_begin + 1, third_bar -
hostname_begin - 1);
        cInfo->hostname[third_bar - hostname_begin - 1] = '\0';

        strcpy(cInfo->port, "21");
    }

    char *last_barPtr = strrchr(url, '/');
    int last_bar = last_barPtr - url;
    if (last_bar > third_bar) {
        memcpy(cInfo->filepath, url + third_bar + 1, last_bar - third_bar -
1);
        cInfo->filepath[last_bar - third_bar - 1] = '\0';
    } else {
        cInfo->filepath[0] = '\0';
    }

    memcpy(cInfo->filename, last_barPtr + 1, strlen(url) - last_bar - 1);
    cInfo->filename[strlen(url) - last_bar - 1] = '\0';

    return 0;
}

void loadingBar(float file_size_processed, float file_total_size)
{
       float percentage = 100.0 * file_size_processed / file_total_size;
       printf("\rStatus: %6.2f%% [", percentage);     //display progress from
0 to 100%

       int i, len = 50;
       int pos = percentage * len / 100.0;
```

```
        //displays visual progress
        for (i = 0; i < len; i++){
              if(i <= pos)
                    printf("=");
              else
                    printf(" ");
              }

        printf("]");

        fflush(stdout);
}
```

Configuration Commands:

Experiment 1 Commands

tux13:
ifconfig eth1 172.16.10.1/24
tux14:
ifconfig eth1 172.16.10.254/24
tux13:
ping 172.16.10.254
tux14:
ping 172.16.10.1
tux13:
arp -a
arp -d 172.16.10.254/24
arp -a

Experiment 2 Commands

Switch console:
/system reset-configuration
tux12:
ifconfig eth1 172.16.11.1/24
Switch console:
/interface bridge add name=bridge50
/interface bridge add name=bridge51
/interface bridge port remove [find interface=ether4]
/interface bridge port remove [find interface=ether7]
/interface bridge port remove [find interface=ether12]
/interface bridge port add bridge=bridge0 interface=ether4
/interface bridge port add bridge=bridge0 interface=ether7
/interface bridge port add bridge=bridge1 interface=ether12

tux13:

ping 172.16.10.254
ping 172.16.11.1
ping -b 172.16.10.255

tux12:

ping -b 172.16.11.255

Experiment 3 Commands

tux14:

ifconfig eth2 up
ifconfig eth2 172.16.11.253/24

Switch console:

/interface bridge port remove [find interface=ether15]
/interface bridge port add bridge=bridge1 interface=ether15

tux14:

sysctl net.ipv4.ip_forward=1
sysctl net.ipv4.icmp_echo_ignore_broadcasts=0

tux12:

route add -net 172.16.10.0/24 gw 172.16.11.253

tux13:

route add -net 172.16.11.0/24 gw 172.16.10.254
ping 172.16.10.254
ping 172.16.11.253
ping 172.16.11.1

tux12:

arp -d 172.16.11.253

tux13:

arp -d 172.16.10.254

tux14:

arp -d 172.16.10.1
arp -d 172.16.11.1

Experiment 4 Commands

Switch console:

/interface bridge port remove [find interface=ether24]
/interface bridge port add bridge=bridge1 interface=ether24

Router console: /system reset-configuration

/ip address add address=172.16.1.11/24 interface=ether1
/ip address add address=172.16.11.254/24 interface=ether2
/ip route add dst-address=172.16.10.0/24 gateway=172.16.11.253
/ip route add address=0.0.0.0/0 gateway=172.16.1.254

tux12:

route add default gw 172.16.11.254

tux13:

route add default gw 172.16.10.254

tux14:

route add default gw 172.16.11.254

tux13:

ping 172.16.10.254
ping 172.16.11.1
ping 172.16.11.254

tux12:

sysctl net.ipv4.conf.eth1.accept_redirects=0
sysctl net.ipv4.conf.all.accept_redirects=0
route del -net 172.16.10.0 gw 172.16.11.253/24
ping 172.16.10.1
traceroute -n 172.16.10.1
route add -net 172.16.10.0/24 gw 172.16.11.253
traceroute -n 172.16.10.1
sysctl net.ipv4.conf.eth1.accept_redirects=1
sysctl net.ipv4.conf.all.accept_redirects=1

tux13:

ping 172.16.1.10

Router console:

/ip firewall nat disable 0

tux13:

ping 172.16.1.10

Router console:

/ip firewall nat enable 0

Experiment 5 Commands

add the DNS server (10.227.20.3) in case it isn't seated as default.
Tux12, Tux13, Tux14: ping google.com

Experiment 6 Commands

run the ftp application in the tux13

Wireshark Captures:

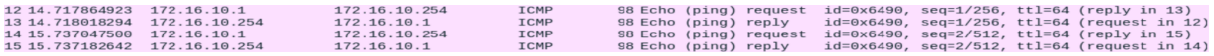
Experiment 1 Captures

1.1 - Ping from tux13 to tux14

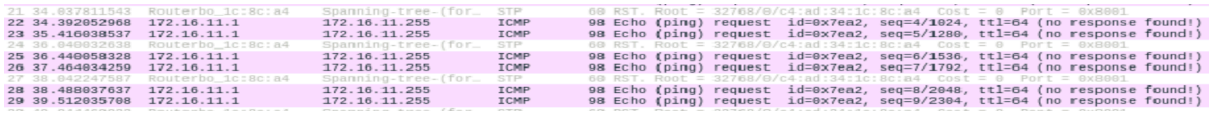
5	2.509886309	EdimaxTe_ed:bb:37	Broadcast	ARP	42	Who has 172.16.10.254? Tell 172.16.10.1
6	2.509961738	Kye_04:20:99	EdimaxTe_ed:bb:37	ARP	60	172.16.10.254 is at 00:c0:df:04:20:99
7	2.509969490	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x1644, seq=1/256, ttl=64 (reply in 8)
8	2.510062728	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x1644, seq=1/256, ttl=64 (request in 7)
9	3.527047075	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x1644, seq=2/512, ttl=64 (reply in 10)
10	3.527151487	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x1644, seq=2/512, ttl=64 (request in 9)
11	4.080473359	Routerbo_1c:8c:9f	Spanning-tree-(for-...	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:99 Cost = 0 Port = 0x8007
12	4.551045348	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x1644, seq=3/768, ttl=64 (reply in 13)
13	4.551131392	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x1644, seq=3/768, ttl=64 (request in 12)
14	5.575051582	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x1644, seq=4/1024, ttl=64 (reply in 15)
15	5.575138464	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x1644, seq=4/1024, ttl=64 (request in 14)

Experiment 2 Captures

2.1 - Ping from tux13 to tux14

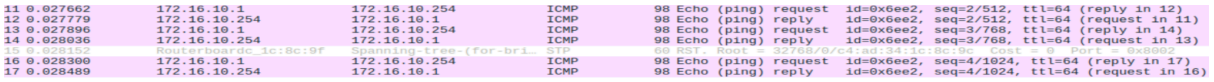


2.2 - Ping from tux13 to tux12

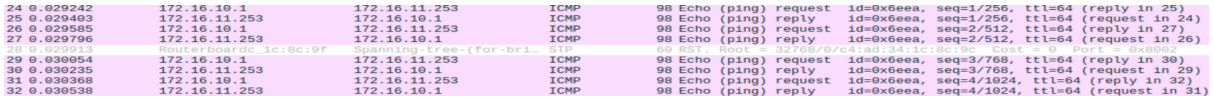


Experiment 3 Captures

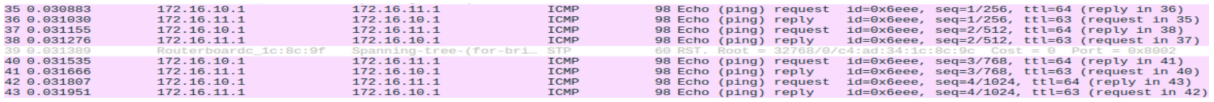
3.1.1 - ICMP packets, captured in tux13 during the ping from tux13 to the eth1 interface of tux14



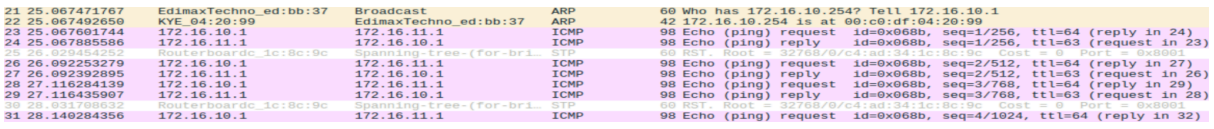
3.1.2 - ICMP packets, captured in tux13 during the ping from tux13 to the eth2 interface of tux14



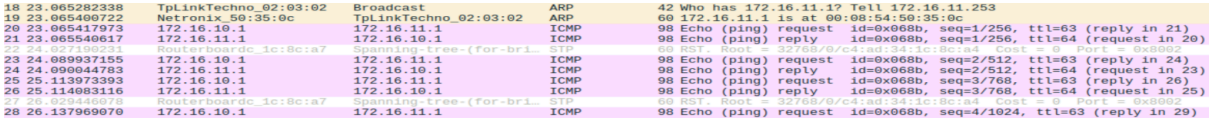
3.1.3 - ICMP packets, captured in tux13 during the ping from tux13 to tux12



3.2.1 - ARP and ICMP packets, captured in tux14's eth1 during the establishment of the connection between tux13 and tux12

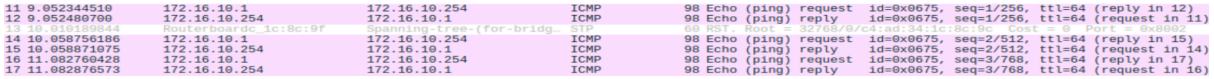


3.2.2 - ARP e ICMP packets, captured in tux14's eth2 during the establishment of the connection between tux13 and tux12



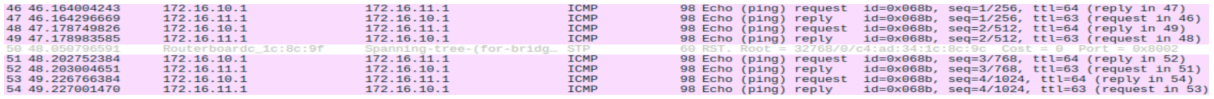
Experiment 4 Captures

4.1.1 - ICMP packets, captured in tux13 during the ping from tux13 to the eth1 interface of tux14



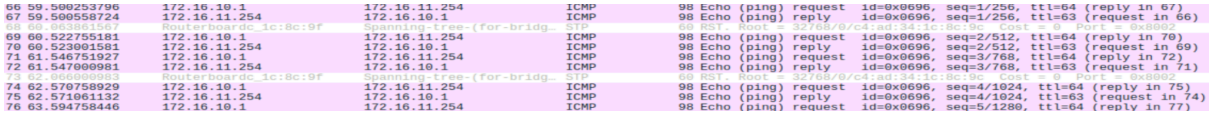
11	9.052344510	172.16.10.1	172.16.10.254	ICMP	98 Echo (ping) request id=0x0675, seq=1/256, ttl=64 (reply in 12)
12	9.052480700	172.16.10.254	172.16.10.1	ICMP	98 Echo (ping) reply id=0x0675, seq=1/256, ttl=64 (request in 11)
13	10.010810044	Routerboardc_1c:8c:9f	Spanning-tree-(for-bridg…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:9c Cost = 0 Port = 0x8002
14	10.058756186	172.16.10.1	172.16.10.254	ICMP	98 Echo (ping) request id=0x0675, seq=2/512, ttl=64 (reply in 15)
15	10.058871075	172.16.10.254	172.16.10.1	ICMP	98 Echo (ping) reply id=0x0675, seq=2/512, ttl=64 (request in 14)
16	11.082760428	172.16.10.1	172.16.10.254	ICMP	98 Echo (ping) request id=0x0675, seq=3/768, ttl=64 (reply in 17)
17	11.082876573	172.16.10.254	172.16.10.1	ICMP	98 Echo (ping) reply id=0x0675, seq=3/768, ttl=64 (request in 16)

4.1.2 - ICMP packets, captured in tux13 during the ping from0 tux13 to tux12



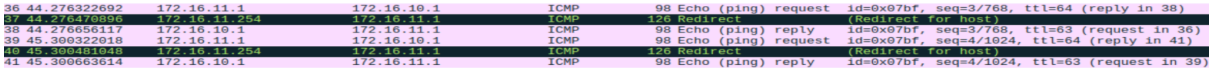
46	46.164004243	172.16.10.1	172.16.11.1	ICMP	98 Echo (ping) request id=0x068b, seq=1/256, ttl=64 (reply in 47)
47	46.164296669	172.16.11.1	172.16.10.1	ICMP	98 Echo (ping) reply id=0x068b, seq=1/256, ttl=63 (request in 46)
48	47.178749826	172.16.10.1	172.16.11.1	ICMP	98 Echo (ping) request id=0x068b, seq=2/512, ttl=64 (reply in 49)
49	47.178983585	172.16.11.1	172.16.10.1	ICMP	98 Echo (ping) reply id=0x068b, seq=2/512, ttl=63 (request in 48)
50	48.058796593	Routerboardc_1c:8c:9f	Spanning-tree-(for-bridg…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:9c Cost = 0 Port = 0x8002
51	48.202752384	172.16.10.1	172.16.11.1	ICMP	98 Echo (ping) request id=0x068b, seq=3/768, ttl=64 (reply in 52)
52	48.203004651	172.16.11.1	172.16.10.1	ICMP	98 Echo (ping) reply id=0x068b, seq=3/768, ttl=63 (request in 51)
53	49.226766384	172.16.10.1	172.16.11.1	ICMP	98 Echo (ping) request id=0x068b, seq=4/1024, ttl=64 (reply in 54)
54	49.227001470	172.16.11.1	172.16.10.1	ICMP	98 Echo (ping) reply id=0x068b, seq=4/1024, ttl=63 (request in 53)

4.1.3 - ICMP packets, captured in tux13 during the ping from tux13 to the router



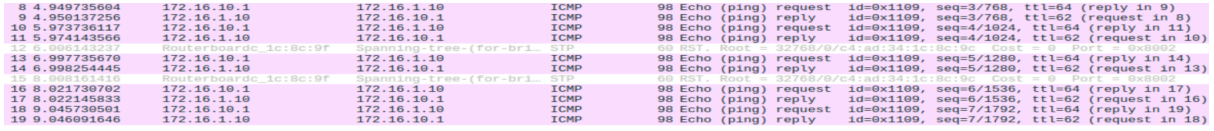
66	59.500253796	172.16.10.1	172.16.11.254	ICMP	98 Echo (ping) request id=0x0696, seq=1/256, ttl=64 (reply in 67)
67	59.500558724	172.16.11.254	172.16.10.1	ICMP	98 Echo (ping) reply id=0x0696, seq=1/256, ttl=63 (request in 66)
68	60.003961587	Routerboardc_1c:8c:9f	Spanning-tree-(for-bridg…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:9c Cost = 0 Port = 0x8002
69	60.522755181	172.16.10.1	172.16.11.254	ICMP	98 Echo (ping) request id=0x0696, seq=2/512, ttl=64 (reply in 70)
70	60.523001581	172.16.11.254	172.16.10.1	ICMP	98 Echo (ping) reply id=0x0696, seq=2/512, ttl=63 (request in 69)
71	61.546751927	172.16.10.1	172.16.11.254	ICMP	98 Echo (ping) request id=0x0696, seq=3/768, ttl=64 (reply in 72)
72	61.547000981	172.16.11.254	172.16.10.1	ICMP	98 Echo (ping) reply id=0x0696, seq=3/768, ttl=63 (request in 71)
73	62.028796593	Routerboardc_1c:8c:9f	Spanning-tree-(for-bridg…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:9c Cost = 0 Port = 0x8002
74	62.570758929	172.16.10.1	172.16.11.254	ICMP	98 Echo (ping) request id=0x0696, seq=4/1024, ttl=64 (reply in 75)
75	62.571061132	172.16.11.254	172.16.10.1	ICMP	98 Echo (ping) reply id=0x0696, seq=4/1024, ttl=63 (request in 74)
76	63.594758446	172.16.10.1	172.16.11.254	ICMP	98 Echo (ping) request id=0x0696, seq=5/1280, ttl=64 (reply in 77)

4.2 - ICMP packets, captured in tux12 during the ping from tux12 to tux13 after removing the route between tux12 and tux14



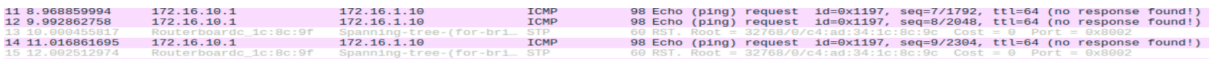
36	44.276322692	172.16.11.1	172.16.10.1	ICMP	98 Echo (ping) request id=0x07bf, seq=3/768, ttl=64 (reply in 38)
37	44.276470896	172.16.11.254	172.16.11.1	ICMP	126 Redirect (Redirect for host)
38	44.276656117	172.16.10.1	172.16.11.1	ICMP	98 Echo (ping) reply id=0x07bf, seq=3/768, ttl=63 (request in 36)
39	45.300322018	172.16.11.1	172.16.10.1	ICMP	98 Echo (ping) request id=0x07bf, seq=4/1024, ttl=64 (reply in 41)
40	45.300481048	172.16.11.254	172.16.11.1	ICMP	126 Redirect (Redirect for host)
41	45.300663614	172.16.10.1	172.16.11.1	ICMP	98 Echo (ping) reply id=0x07bf, seq=4/1024, ttl=63 (request in 39)

4.3 - ICMP packets, captured in tux13 during the ping from tux13 to the server FTP(172.16.1.10) with NAT enabled



8	4.949735604	172.16.10.1	172.16.1.10	ICMP	98 Echo (ping) request id=0x1109, seq=3/768, ttl=64 (reply in 9)
9	4.950137256	172.16.1.10	172.16.10.1	ICMP	98 Echo (ping) reply id=0x1109, seq=3/768, ttl=62 (request in 8)
10	5.973736117	172.16.10.1	172.16.1.10	ICMP	98 Echo (ping) request id=0x1109, seq=4/1024, ttl=64 (reply in 11)
11	5.974143566	172.16.1.10	172.16.10.1	ICMP	98 Echo (ping) reply id=0x1109, seq=4/1024, ttl=62 (request in 10)
12	6.000143237	Routerboardc_1c:8c:9f	Spanning-tree-(for-bri…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:9c Cost = 0 Port = 0x8002
13	6.997735670	172.16.10.1	172.16.1.10	ICMP	98 Echo (ping) request id=0x1109, seq=5/1280, ttl=64 (reply in 14)
14	6.998254445	172.16.1.10	172.16.10.1	ICMP	98 Echo (ping) reply id=0x1109, seq=5/1280, ttl=62 (request in 13)
15	8.008161416	Routerboardc_1c:8c:9f	Spanning-tree-(for-bri…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:9c Cost = 0 Port = 0x8002
16	8.021730702	172.16.10.1	172.16.1.10	ICMP	98 Echo (ping) request id=0x1109, seq=6/1536, ttl=64 (reply in 17)
17	8.022145833	172.16.1.10	172.16.10.1	ICMP	98 Echo (ping) reply id=0x1109, seq=6/1536, ttl=62 (request in 16)
18	9.045730501	172.16.10.1	172.16.1.10	ICMP	98 Echo (ping) request id=0x1109, seq=7/1792, ttl=64 (reply in 19)
19	9.046091646	172.16.1.10	172.16.10.1	ICMP	98 Echo (ping) reply id=0x1109, seq=7/1792, ttl=62 (request in 18)

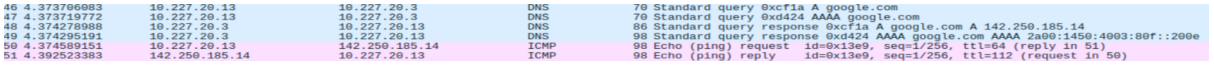
4.4 - ICMP packets, captured in tux13 during the ping from tux13 to the server FTP(172.16.1.10) with NAT disabled



11	8.968859994	172.16.10.1	172.16.1.10	ICMP	98 Echo (ping) request id=0x1197, seq=7/1792, ttl=64 (no response found!)
12	9.992862758	172.16.10.1	172.16.1.10	ICMP	98 Echo (ping) request id=0x1197, seq=8/2048, ttl=64 (no response found!)
13	10.000465617	Routerboardc_1c:8c:9f	Spanning-tree-(for-bri…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:9c Cost = 0 Port = 0x8002
14	11.016861695	172.16.10.1	172.16.1.10	ICMP	98 Echo (ping) request id=0x1197, seq=9/2304, ttl=64 (no response found!)
15	12.002512974	Routerboardc_1c:8c:9f	Spanning-tree-(for-bri…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:9c Cost = 0 Port = 0x8002
16	12.040866932	172.16.10.1	172.16.1.10	ICMP	98 Echo (ping) request id=0x1197, seq=10/2560, ttl=64 (no response found!)
17	14.004572491	Routerboardc_1c:8c:9f	Spanning-tree-(for-bri…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:9c Cost = 0 Port = 0x8002

Experiment 5 Captures

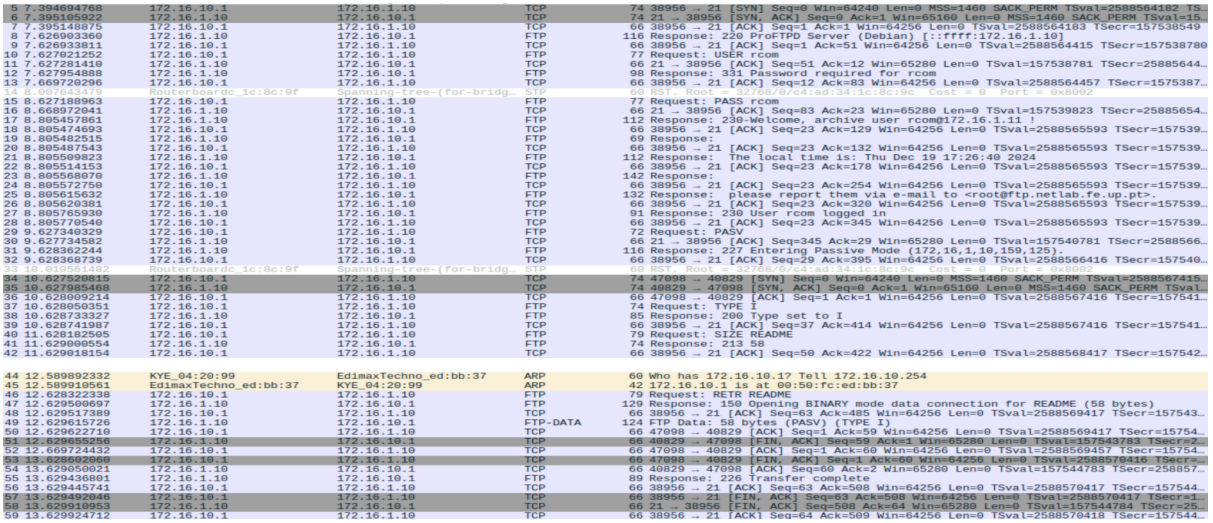
5.1 - DNS and ICMP packets, captured in tux13 during the ping from tux13 to google.com



46	4.373706083	10.227.20.13	10.227.20.3	DNS	70 Standard query 0xcf1a A google.com
47	4.373719772	10.227.20.13	10.227.20.3	DNS	70 Standard query 0xd424 AAAA google.com
48	4.374278988	10.227.20.3	10.227.20.13	DNS	86 Standard query response 0xcf1a A google.com A 142.250.185.14
49	4.374295191	10.227.20.3	10.227.20.13	DNS	98 Standard query response 0xd424 AAAA google.com AAAA 2a00:1450:4003:80f::200e
50	4.374589151	10.227.20.13	142.250.185.14	ICMP	98 Echo (ping) request id=0x13e9, seq=1/256, ttl=64 (reply in 51)
51	4.392523383	142.250.185.14	10.227.20.13	ICMP	98 Echo (ping) reply id=0x13e9, seq=1/256, ttl=112 (request in 50)

Experiment 6 Captures

6.1 - TCP and FTP packets, captured in tux13 during the download of a file in the FTP server



5	7.394694768	172.16.10.1	172.16.1.10	TCP	74 38956 → 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2588564182 TS…
6	7.395148922	172.16.1.10	172.16.10.1	TCP	74 21 → 38956 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=15…
7	7.395148875	172.16.10.1	172.16.1.10	TCP	66 38956 → 21 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2588564183 TSecr=157538549
8	7.626903360	172.16.1.10	172.16.10.1	FTP	116 Response: 220 ProFTPD Server (Debian) [::ffff:172.16.1.10]
9	7.626933811	172.16.10.1	172.16.1.10	TCP	66 38956 → 21 [ACK] Seq=1 Ack=51 Win=64256 Len=0 TSval=2588564415 TSecr=157538780
10	7.627021252	172.16.10.1	172.16.1.10	FTP	77 Request: USER rcom
11	7.627281410	172.16.1.10	172.16.10.1	TCP	66 21 → 38956 [ACK] Seq=51 Ack=12 Win=65280 Len=0 TSval=157538781 TSecr=25885644…
12	7.627954888	172.16.1.10	172.16.10.1	FTP	98 Response: 331 Password required for rcom
13	7.669720256	172.16.10.1	172.16.1.10	TCP	66 38956 → 21 [ACK] Seq=12 Ack=83 Win=64256 Len=0 TSval=2588564457 TSecr=1575387…
14	8.897643479	Routerboardc_1c:8c:9f	Spanning-tree-(for-bridg…	STP	99 RST. Root = 32768/0/c4:ad:34:1c:8c:9c Cost = 0 Port = 0x8002
15	8.627188963	172.16.10.1	172.16.1.10	FTP	77 Request: PASS rcom
16	8.668972041	172.16.1.10	172.16.10.1	TCP	66 21 → 38956 [ACK] Seq=83 Ack=23 Win=65280 Len=0 TSval=157539823 TSecr=25885654…
17	8.805457861	172.16.1.10	172.16.10.1	FTP	112 Response: 230-Welcome, archive user rcom@172.16.1.11 !
18	8.805474693	172.16.10.1	172.16.1.10	TCP	66 38956 → 21 [ACK] Seq=23 Ack=129 Win=64256 Len=0 TSval=2588565593 TSecr=157539…
19	8.805482515	172.16.1.10	172.16.10.1	FTP	69 Response:
20	8.805487543	172.16.10.1	172.16.1.10	TCP	66 38956 → 21 [ACK] Seq=23 Ack=132 Win=64256 Len=0 TSval=2588565593 TSecr=157539…
21	8.805509823	172.16.1.10	172.16.10.1	FTP	112 Response: The local time is: Thu Dec 19 17:26:40 2024
22	8.805514153	172.16.10.1	172.16.1.10	TCP	66 38956 → 21 [ACK] Seq=23 Ack=178 Win=64256 Len=0 TSval=2588565593 TSecr=157539…
23	8.805568070	172.16.1.10	172.16.10.1	FTP	142 Response:
24	8.805572750	172.16.10.1	172.16.1.10	TCP	66 38956 → 21 [ACK] Seq=23 Ack=254 Win=64256 Len=0 TSval=2588565593 TSecr=157539…
25	8.805615632	172.16.1.10	172.16.10.1	FTP	132 Response: please report them via e-mail to <root@ftp.netlab.fe.up.pt>.
26	8.805620381	172.16.10.1	172.16.1.10	TCP	66 38956 → 21 [ACK] Seq=23 Ack=320 Win=64256 Len=0 TSval=2588565593 TSecr=157539…
27	8.805765930	172.16.1.10	172.16.10.1	FTP	91 Response: 230 User rcom logged in
28	8.805779540	172.16.10.1	172.16.1.10	TCP	66 38956 → 21 [ACK] Seq=23 Ack=345 Win=64256 Len=0 TSval=2588565593 TSecr=157539…
29	9.627340329	172.16.10.1	172.16.1.10	FTP	72 Request: PASV
30	9.627734582	172.16.1.10	172.16.10.1	TCP	66 21 → 38956 [ACK] Seq=345 Ack=29 Win=65280 Len=0 TSval=157540781 TSecr=2588566…
31	9.628362244	172.16.1.10	172.16.10.1	FTP	116 Response: 227 Entering Passive Mode (172,16,1,10,159,125).
32	9.628368739	172.16.10.1	172.16.1.10	TCP	66 38956 → 21 [ACK] Seq=29 Ack=395 Win=64256 Len=0 TSval=2588566416 TSecr=157540…
33	10.819561482	Routerboardc_1c:8c:9f	Spanning-tree-(for-bridg…	STP	99 RST. Root = 32768/0/c4:ad:34:1c:8c:9c Cost = 0 Port = 0x8002
34	10.627520815	172.16.10.1	172.16.1.10	TCP	74 47098 → 40829 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2588567415…
35	10.627985468	172.16.1.10	172.16.10.1	TCP	74 40829 → 47098 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval…
36	10.628009214	172.16.10.1	172.16.1.10	TCP	66 47098 → 40829 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2588567416 TSecr=157541…
37	10.628050351	172.16.10.1	172.16.1.10	FTP	74 Request: TYPE I
38	10.628733327	172.16.1.10	172.16.10.1	FTP	85 Response: 200 Type set to I
39	10.628741987	172.16.10.1	172.16.1.10	TCP	66 38956 → 21 [ACK] Seq=37 Ack=414 Win=64256 Len=0 TSval=2588567416 TSecr=157541…
40	11.628182565	172.16.10.1	172.16.1.10	FTP	79 Request: SIZE README
41	11.629000554	172.16.1.10	172.16.10.1	FTP	74 Response: 213 58
42	11.629018154	172.16.10.1	172.16.1.10	TCP	66 38956 → 21 [ACK] Seq=50 Ack=422 Win=64256 Len=0 TSval=2588568417 TSecr=157542…
44	12.589892332	KYE_04:20:99	EdimaxTechno_ed:bb:37	ARP	60 Who has 172.16.10.1? Tell 172.16.10.254
45	12.589910561	EdimaxTechno_ed:bb:37	KYE_04:20:99	ARP	42 172.16.10.1 is at 00:50:fc:ed:bb:37
46	12.628322338	172.16.10.1	172.16.1.10	FTP	79 Request: RETR README
47	12.629500697	172.16.1.10	172.16.10.1	FTP	129 Response: 150 Opening BINARY mode data connection for README (58 bytes)
48	12.629517389	172.16.10.1	172.16.1.10	TCP	66 38956 → 21 [ACK] Seq=63 Ack=485 Win=64256 Len=0 TSval=2588569417 TSecr=157543…
49	12.629615726	172.16.1.10	172.16.10.1	FTP-DATA	124 FTP Data: 58 bytes (PASV) (TYPE I)
50	12.629622710	172.16.10.1	172.16.1.10	TCP	66 47098 → 40829 [ACK] Seq=1 Ack=59 Win=64256 Len=0 TSval=2588569417 TSecr=15754…
51	12.629655256	172.16.1.10	172.16.10.1	TCP	66 40829 → 47098 [FIN, ACK] Seq=59 Ack=1 Win=65280 Len=0 TSval=157543783 TSecr=2…
52	12.630272432	172.16.10.1	172.16.1.10	TCP	66 47098 → 40829 [ACK] Seq=1 Ack=60 Win=64256 Len=0 TSval=2588569457 TSecr=15754…
53	13.628602060	172.16.10.1	172.16.1.10	TCP	66 47098 → 40829 [FIN, ACK] Seq=1 Ack=60 Win=64256 Len=0 TSval=2588570416 TSecr=…
54	13.629650021	172.16.1.10	172.16.10.1	TCP	66 40829 → 47098 [ACK] Seq=60 Ack=2 Win=65280 Len=0 TSval=157544783 TSecr=258857…
55	13.629436801	172.16.1.10	172.16.10.1	FTP	89 Response: 226 Transfer complete
56	13.629445741	172.16.10.1	172.16.1.10	TCP	66 38956 → 21 [ACK] Seq=63 Ack=508 Win=64256 Len=0 TSval=2588570417 TSecr=157544…
57	13.629492046	172.16.10.1	172.16.1.10	TCP	66 38956 → 21 [FIN, ACK] Seq=63 Ack=508 Win=64256 Len=0 TSval=2588570417 TSecr=1…
58	13.629910953	172.16.1.10	172.16.10.1	TCP	66 21 → 38956 [FIN, ACK] Seq=508 Ack=64 Win=65280 Len=0 TSval=157544784 TSecr=25…
59	13.629924712	172.16.10.1	172.16.1.10	TCP	66 38956 → 21 [ACK] Seq=64 Ack=509 Win=64256 Len=0 TSval=2588570418 TSecr=157544…

Extra Annexes:

Download Application with the debug mode on

```
pchmelo@pchmelo-VirtualBox:~/Downloads/RCOM-project2-master$ ./download ftp://demo:password@test.rebex.net/readme.txt

<PARSE URL>
user = demo
password = password
hostname = test.rebex.net
port = 21
filepath =
filename = readme.txt

<CONNECT RESPONSE>
220-Welcome to test.rebex.net!
    See https://test.rebex.net/ for more information and terms of use.
220 If you don't have an account, log in as 'anonymous' or 'ftp'.

<USER CMD>
USER demo

<USER RESPONSE>
331 Anonymous login OK, send your complete email address as your password.

<PASS CMD>
PASS password

<PASS RESPONSE>
230 User 'demo' logged in.

<PASV CMD>
PASV

<PASV RESPONSE>
227 Entering Passive Mode (194,108,117,16,4,20)

Parsed Pasv IP address - 194.108.117.16:5124

<TYPE CMD>
TYPE I

<TYPE RESPONSE>
200 TYPE set to I.

<SIZE CMD>
SIZE

<SIZE RESPONSE>
213 379

<RETR CMD>
RETR readme.txt

<RETR RESPONSE>
125 Data connection already open; starting 'BINARY' transfer.

Transfering file (bytes received)
(379)

<RETR FINAL RESPONSE>
226 Transfer complete.

Transfer complete!
```

Graph - Speed of packets forwarded by the network per second, in a single transfer of Tux53