Faculdade de Engenharia da Universidade do Porto

Rede de computadores

2º Trabalho Laboratorial - Redes de Computadores

Tomás Eiras Silva Martins – up202108776

Wallen Marcos Ribeiro – up202109260

Índice

Sumário

Este trabalho laboratorial foi realizado no âmbito da cadeira de Redes de Computadores e tem como objetivo desenvolver um programa de Download usando pacotes FTP e, configurar uma rede de computadores

Com este trabalho fomos capazes de colocar em prática os conhecimentos relacionados com esta unidade curricular, conseguindo implementar, com êxito, o programa de Download e a configuração da rede.

Introdução

O objetivo principal deste projeto era o desenvolvimento de uma aplicação de *download* utilizando FTP e a configuração de uma rede seguindo as instruções que se encontram no guião fornecido.

O foco principal deste relatório é a análise dos resultados do programa desenvolvido, tal como das *logs* capturadas durante a configuração da rede.

Este relatório é constituído pelas seguintes secções:

- **Introdução –** Descrição breve do projeto e do relatório;
- Aplicação de transferência de ficheiros:
    1. Arquitetura;
    2. Testes e Resultados;
- Rede configurada e análise de resultados:
    1. Configurar uma rede IP;
    2. Implementar duas bridges num Switch;
    3. Configurar um router em Linux;
    4. Configurar um router comercial e implementar NAT;
    5. DNS;
    6. Conexões TCP;
- Conclusão;

Arquitetura da aplicação de Download

A aplicação criada tem como propósito realizar o download de ficheiros seguindo o protocolo FTP. A função **convertToURL** analisa o URL FTP fornecido e processa o para um objeto da *structure* URL. Este retém detalhes essenciais como o *host*, *user*, *password* e URL *path*. Além disso, a função mencionada garante que o URL possui o formato desejado, iniciando-se com "ftp://".

De seguida, função **connectSocket** estabelece conexão com o servidor FTP, utilizando *sockets* TCP. Essa ligação serve como canal para transmitir comandos e receber respostas do servidor. De forma a prevenir possíveis falhas de conexão, implementou-se um robusto tratamento de erros, tornando a comunicação mais confiável e estável.

O tratamento da resposta do servidor está incorporado dentro da função **getReply**, utilizando uma máquina de estados para interpretar as respostas do servidor. As respostas são processadas até que uma resposta com um formato específico é encontrada, garantindo uma interpretação precisa do feedback passado pelo servidor.

A autenticação do utilizador é tratada pela função **login**, que envia o *user* e a *password* para o servidor que, em caso de falha na autenticação, encerra a conexão para evitar acesso não autorizado. Além disso, caso os dados do *user* e *password* não sejam fornecidos, estabelece se uma conexão anónima.

A função **pasv** é responsável pela entrada no modo passivo, através do envio do comando

"pasv" para o servidor.

A obtenção do ficheiro desejado é controlada pela função **retrieveResource**, que envia o comando "retr" para solicitar um recurso específico (ficheiro) do servidor. Esta abre um ficheiro para escrita e inicia a transferência de dados através de uma segunda conexão de dados.

Por fim, função **main** serve como ponto inicial do programa, coordenando a execução geral. Esta 'controla' que funções são utilizadas, valida argumentos da linha de comando, analisa o URL, processa o endereço IP do *host*, estabelece a conexão de controle, realiza o login do utilizador, entra no modo passivo, obtém o ficheiro solicitado e conclui, fechando tanto as conexões de controle quanto de dados.

Testes e resultados

De forma a verificar o bom funcionamento da aplicação produzida, testou-se o *download* de diversos tipos de ficheiros, com tamanhos e extensões diferentes. Além disso, testou-se a robustez do código, através da colocação de inputs errados e URLs com erros.

Podemos então verificar o output dado por um desses testes:

```
wallenribeiro@pop-os:~/Documents/feup/rcom/feup-rcom-proj2$ ./download ftp://rcom:rcom@netlab1.fe.up.pt/pipe.txt
Host: netlab1.fe.up.pt

URL path: pipe.txt

Host name  : netlab1.fe.up.pt

IP Address : 192.168.109.136

Reply: 220 Welcome to netlab-FTP server


Reply: 331 Please specify the password.


Reply: 230 Login successful.


Reply: 227 Entering Passive Mode (192,168,109,136,161,186).


port[0] is 161

port[1] is 186

port: 41402

Passive mode established

Reply: 150 Opening BINARY mode data connection for pipe.txt (1863 bytes).


Filename: pipe.txt

Resource downloaded

wallenribeiro@pop-os:~/Documents/feup/rcom/feup-rcom-proj2$
```

```
Filename: pipe.txt

Resource downloaded

wallenribeiro@pop-os:~/Documents/feup/rcom/feup-rcom-proj2$ cat pipe.txt
..... ...  .          .  . .
........ . .....    .  . ........   .     .  .
........ . ... .   . ......  .  ...    .
....   . . ..,*//////*..  .. .
 ...  . %#/,*******,**#&&&%                                (&&&&&&%&%%&%&&&%*
 ......,#(*,,((%%%%%&&&&@@@&                          /&%(,,,/#&@@@@&@@(,..../(.
.......%#(,,*(#%%%%%&&&@@@&*                    &%(,,.,#&@@@@@#..
..    .%#(,,*(#%%%%%&&@@@@&(.              (%/*../&@@@@@(.
   ....%#(,,*(#%%%&%&&@@@@@/            ##/***(#%#&@@&
...  . #%(*,*(#%%%%%&@@@@@@*         ,##//**(#%&&&&%.
.  .   .%#/,*(#%%&%&&@@@@@@@,     *%#/**,*(#%&@@&&%
      %#(//(#%&%%&&@@@@@@&#((/**,*/(#%&@@@&&%
..  .   .%#(((#%%%&&@@@@@@@%#(((#%%%&@@@@@&%
. .      .%%%%%%%&@@@@@@@@@@@@@@@@@@@@@@@&&.
..  ..     /&&&&&@@@@@@@@@@@@@@@@@@@@&&%
..            .&&&&@@@@@@@@@@@@@&&%.
..  . .          ..  .
..  ....
......     .            .  .
......      ..% ..   .. .. .    ..  %   .      /    . ,*  *   %# * %* ##
.. . . ....#*..%.%.(...  %/ ././#%%. ../(.#/,#  %(,, /%.(   *.(,.* %.%#./
.................. ..    .    .   . .               .       .... .   ..
.....................  ...   ... .                  .                ......
wallenribeiro@pop-os:~/Documents/feup/rcom/feup-rcom-proj2$
```

Exp1 – Configurar uma rede IP

Nesta experiência, foi estabelecida a conexão entre os Tux43 e Tux44, que foram conectados ao switch e tiveram os seus respetivos endereços IP configurados. A configuração dos endereços IP foi realizada utilizando o comando *ifconfig*, e posteriormente a conexão entre ambos foi testada utilizando o comando *ping*. Os endereços IP e MAC de ambos os computadores foram identificados ao serem registados nas tabelas ARP, após a execução do comando *ping*. Dessa forma, confirmou-se que o Tux43 possui o IP 172.16.40.1/24 com o endereço MAC 00:21:5a:61:2f:d4, enquanto o Tux44 possui o IP 172.16.40.254/24 com o endereço MAC 00:21:5a:5a:7b:ea.

Durante a execução do comando ping, é observada a troca de pacotes ARP no início da captura, o que possibilita a conexão entre os computadores. Esses pacotes têm a função de mapear um endereço IP para um endereço MAC, incluindo tanto o endereço IP e MAC do remetente quanto do destinatário. Enquanto o endereço IP identifica um dispositivo numa rede, o endereço MAC identifica a interface de rede física do dispositivo. Após os pacotes ARP iniciais, verifica-se que o comando *ping* gera pacotes ICMP. Estes pacotes são usados para testar a conexão com outro endereço IP, possuindo, neste caso, mensagens de reply e request. É também possível analisar que tipo de pacote são gerados, tal como o tamanho dos mesmos, ao analisar a captura no *Wireshark*, como é possível observar em Exp1b.

Além da comunicação entre computadores, é relevante considerar a presença de uma interface de *loopback* em cada máquina. A referida interface, de natureza virtual e privada, permite que o computador comunique consigo mesmo, tornando esta interface numa ferramenta essencial para diagnosticar eventuais erros de rede.

As logs referentes às capturas realizadas encontram-se em Exp1.

Exp2 – Implementar duas bridges num switch

Para esta experiência, foi solicitada a configuração do Tux42 com um endereço IP de 172.16.41.1/24 e um endereço MAC de 00:1f:29:d7:45:c4, conforme realizado na primeira experiência. De seguida, foram criadas duas bridges, bridge40 e bridge41, utilizando o comando **'/interface bridge add'**, e as portas de cada Tux foram removidas da bridge *default* por meio do comando **'/interface bridge port remove'**. Posteriormente, as portas de cada Tux foram adicionadas às respetivas bridges com o comando **'/interface bridge port add'**, conectando o Tux43 e o Tux44 à bridge40, e o Tux42 à bridge41, conforme a estrutura apresentada Exp2a.

Após adicionar os três Tux às suas respetivas bridges, realizou se um teste de conexão para verificar a comunicação entre eles. Esse teste foi realizado utilizando o comando **ping -b** para verificar a existência de domínios Broadcast referentes a cada uma das bridges. Os resultados indicaram a existência de dois domínios Broadcast, um para cada uma das bridges. Foi observado que o Tux43 consegue alcançar apenas o Tux44, indicando que ambos estão na mesma bridge. Por outro lado, verificou-se que a comunicação a partir do Tux42 para os outros dois Tux não foi bem-sucedida, tal como mostram as logs em Exp2.

Exp3 – Configurar um router em Linux

Para permitir a comunicação entre os Tux em diferentes bridges, o Tux44 foi transformado em um *router*. Isso envolveu a configuração do Tux44.eth1 com o endereço IP 172.16.41.253/24, a remoção de sua porta da bridge *default* e a adição da mesma à bridge41 usando comandos mencionados tanto na experiência anterior, como nos anexos referentes a esta experiência, encontrando-se os Tux com a seguinte estrutura na rede Exp3a. Além disso, o *IP forwarding* foi ativado usando o comando **'sysctl net.ipv4.ip_forward=1'** e, de forma a completar a configuração do *router* no Tux44, o *ICMP echo ignore broadcast* foi desativado com o comando **'sysctl net.ipv4.icmp_echo_ignore_broadcasts=0'**. Para que então seja possível que o Tux43

alcance o Tux42 foi necessário criar uma rota para cada um, que os ligasse ao Tux44 já que este Tux é o único comum às duas bridges. Isso foi possível através do comando route add -net e utilizando os IPs 172.16.40.254/24 e 172.16.41.253/24, definindo o Tux44 como gateaway em ambos os Tux42 e Tux43.

De forma a testar se eventualmente cada Tux alcança os restantes, utilizamos o comando ping. Com isto, observou se que todos os pacotes gerados chegavam ao destino desejado e que de facto a configuração foi bem feita, tal como mostra o anexo Exp3b.

Durante este processo, constatou-se que durante a conexão entre o Tux43 e Tux42, os pacotes ARP e ICMP que foram capturados no Tux44 continham o endereço IP da máquina de destino desejada, porém o endereço MAC do Tux44. Essa ocorrência explica se pelo facto de que o Tux44 atua como um *router*, redirecionando as informações até o destino, sem necessariamente estabelecer uma conexão direta do Tux42 para o Tux43. Ao receber um pacote, o Tux44 consulta a sua *forwarding table* para determinar o próximo passo a ser tomado. Este procura uma correspondência para o endereço IP de destino, de forma a reencaminhar as informações de forma apropriada.

As restantes logs da experiência podem ser consultadas no anexo Exp3.

Exp4 – Configurar um router comercial e implementar NAT

Na configuração de rede anterior, existiam duas bridges interligadas por um *router*. Agora, a intenção é adicionar um *router* comercial com NAT à bridge 41.

Para isso tivemos que primeiramente conectar uma das portas do router comercial a uma das entradas do *switch* e adicionar o mesmo à bridge41 como foi realizado em experiências anteriores. Após isso foi necessário aceder à consola do router e configurar o seu endereço IP utilizando o comando 'ip address add'. Com isso criamos *default routes* em cada Tux de forma que se consigam ligar ao router e uma *route* capaz de ligar o router aos outros Tux, resultante na seguinte estrutura Exp4a.

Após executar as instruções anteriores, usando o comando *traceroute*, pudemos observar o caminho percorrido pelos pacotes do Tux42 para o Tux43. Ao remover a rota para 172.16.40.0 que passava pelo Tux44 e desativar os *redirects*, os pacotes seguiram do Tux42 para o Rc (router comercial) e depois para o Tux43. Ao reativar os *redirects* e verificar o caminho dos pacotes, notamos que estes foram diretamente do Tux42 para o Tux43, sem passar pelo Rc. Isso indica nos que os *redirects* são capazes de aproveitar rotas existentes para escolher uma rota mais eficiente.

Inicialmente, utilizando o comando *ping* de forma a verificar a comunicação com o router do lab, concluiu-se que estes eram capazes de estabelecer uma conexão e que o mesmo era possível devido ao NAT estar ativo por *default.* O NAT (*Network Address Translation*) é uma funcionalidade que permite que vários dispositivos numa rede compartilhem um único endereço IP público para aceder à internet. Desta forma, o NAT ajuda a economizar endereços IP públicos e protege os dispositivos internos ao ocultar os seus endereços IP privados da internet. Resumindo, o NAT num *router* funciona como um "tradutor", permitindo que vários dispositivos compartilhem a mesma "identidade" (endereço IP público) para se comunicar com a internet. Após desativar o NAT **(/ip firewall nat disable 0**) e testar a conexão à internet, foi verificado que não foi possível estabelecer comunicação. Isso ocorreu porque ao desativar o NAT, a capacidade de tradução de endereços públicos para endereços privados deixou de existir, impedindo a comunicação com a internet.

As logs das capturas podem ser encontradas no anexo Exp4.

Exp5 – DNS

Nesta experiência, o objetivo era adicionar um DNS (Domain Name System), um sistema de

gestão de nomes que pode associar um nome a um endereço IP, semelhante a uma tabela de entradas, onde uma entrada relaciona um hostname a um endereço IP. Para realizar isso, foi necessário editar o arquivo /etc/resolv.conf com o seguinte conteúdo: 'nameserver 172.16.1.1'. Para testar o correto funcionamento, utilizamos o comando ping para um domínio, como por exemplo, www.google.pt, e analisamos os pacotes trocados e o tipo de informação transportada. Foi então possível verificar a receção do endereço IP de destino aquando da 'tradução' do hostname para o respetivo IP.

Para além disso, através das logs foi possível observar que os pacotes transportados são do tipo DNS, para que seja possível a tradução e identificação do ip de destino. Estas logs encontram-se disponíveis no anexo Exp5.

Exp6 – Conexões TCP

Para esta experiência, utilizou-se a rede configurada até este passo de modo a verificar o bom funcionamento da aplicação de download desenvolvida.

Para isso compilou-se o programa no Tux43 e testou-se com os ficheiros de teste fornecidos. Após isso testamos se era possível executar o programa simultaneamente em dois Tux diferentes, verificando-se que o download ocorria sem problemas em ambos os Tux.

De forma que o download seja possível, são abertas duas ligações TCP (*Transmission Control Protocol*). A primeira liga-se ao servidor para enviar comandos, enquanto a segunda liga-se para receber os dados do servidor. Este método de duas ligações distintas permite uma separação clara entre o controlo e a transferência de dados, facilitando a gestão eficiente da comunicação entre o cliente FTP e o servidor.

A informação de controlo do FTP é transportada na primeira ligação, a qual é responsável por enviar comandos para o servidor. Isso é necessário para que o servidor tenha as informações necessárias para enviar uma resposta.

Para estabelecer comunicação entre o cliente e o servidor, é necessário passar por três fases distintas da conexão TCP:

1. **Estabelecimento de ligação –** Durante esta fase, o cliente e o servidor estabelecem uma conexão utilizando o método three-way handshake. O cliente envia um pacote SYN para o servidor, o servidor responde com um pacote SYN- ACK e, por fim, o cliente envia um pacote ACK para confirmar a resposta do servidor.
2. **Transferência de dados –** Após a conexão estar estabelecida, a transferência de dados pode ocorrer entre o cliente e o servidor.
3. **Fecho de ligação –** Quando a transferência de dados termina, a conexão é encerrada através do envio de pacotes FIN e ACK tanto por parte do cliente como por parte do servidor.

O mecanismo ARQ é um protocolo projetado para garantir a entrega confiável de dados. Este utiliza o método *Selective Repeat*, que permite o envio de vários pacotes simultaneamente sem esperar pelos respetivos ACK. Quando um remetente transmite dados, o recetor confirma se a receção foi bem-sucedida. Caso o remetente não receba um *acknowledgment* dentro de um tempo especificado, os dados são retransmitidos para assegurar a integridade e confiabilidade da transmissão.

De forma a poder analisar os pacotes TCP precisamos de perceber o que se encontra no cabeçalho destes:

- **Source port -** porta de origem;
- **Destination port –** porta de destino;

- **Sequence number –** identifica o primeiro byte;
- **ACKnumber –** identifica o próximo sequence number que o recetor irá receber;
- **Window Size –** quanta informação pode ser recebida;
- **Checksum –** identificador de erros;
- **Urgent Pointer –** permite marcar um segmento da informação como ‘urgente’;

Pode se encontrar no anexo Exp6a. um exemplo das informações no cabeçalho de um pacote TCP.

O mecanismo de controlo de congestão visa regular o ritmo de transmissão de dados de uma rede para evitar o congestionamento da mesma. Através do monitoramento das respostas ACK recebidas por unidade de tempo, é possível determinar a possibilidade de congestionamento. Para evitar que a rede fique congestionada, o TCP reduz o ritmo de transmissão de dados para tentar aliviar a congestão (*congestion avoidance*). Se ocorrer perda de pacotes, é assumido que houve congestionamento, resultando na retransmissão do pacote perdido sem esperar por um *timeout*, a fim de retomar a transmissão normal (*fast retransmit e fast recovery*). Com isso, podemos concluir que o ritmo de transferência de pacotes aumenta até atingir um pico, indicando congestionamento da rede, o que reduz a quantidade de pacotes transferidos por segundo, até alcançar novamente um pico (Exp6b.).

Ao analisar a situação em que a aplicação de download foi utilizada em dois Tux simultaneamente, observou-se que, embora ambos os downloads tenham sido concluídos com êxito, a transferência levou mais tempo. Esse fenômeno ocorre devido ao uso do mesmo canal por duas conexões TCP, causando congestionamento na rede. Portanto, como mencionado anteriormente, a congestão na rede resulta numa redução na taxa de transferência de pacotes, consequentemente prolongando o tempo de download.

Os logs das capturas encontram-se em Exp6.

Conclusão

Após a conclusão do projeto, podemos afirmar que este foi um sucesso, uma vez que conseguimos configurar com êxito a rede de computadores e implementar a aplicação de transferência de arquivos. Durante o projeto, pudemos aplicar os conhecimentos teóricos adquiridos nas aulas de Redes de Computadores, ao mesmo tempo em que adquirimos novos conhecimentos. Este projeto não apenas nos permitiu aplicar a teoria na prática, mas também possibilitou expandir os nossos conhecimentos e aprofundar a nossa compreensão sobre o funcionamento e a implementação de redes de computadores e sistemas de transferência de ficheiros.

Anexos

A. Aplicação de Download

A1. download.h

```c
#ifndef DOWNLOAD_H
#define DOWNLOAD_H

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

#include <netdb.h>

#define MAX_SIZE 1024

#define SERVER 21

#define h_addr h_addr_list[0]

/**
 * @brief Structure representing a URL.
 *
 * This structure holds the components of a URL, including the user, password, host, and URL path.
 * Each component is represented by a character array with a maximum length of 1024 characters.
 */
typedef struct {
    char user[1024];
    char password[1024];
    char host[1024];
    char url_path[1024];
} url;

/**
 * @brief Converts a string representing a URL to a URL object.
 *
 * This function takes a string `urlStr` as input and converts it to a URL object.
 * The URL object represents the URL in a structured format, allowing easy manipulation and retrieval of its components.
 *
 * @param urlStr The string representing the URL.
 * @return The URL object representing the converted URL.
 */
url convertToURL(const char* urlStr);

/**
 * @brief Establishes a connection to a server using the specified IP address and port number.
 *
 * This function creates a socket and connects it to the server specified by the given IP address and port number.
 *
 * @param ip_addr The IP address of the server to connect to.
 * @param port The port number of the server to connect to.
 * @return The file descriptor of the connected socket, or -1 if an error occurred.
 */
int connectSocket(char *ip_addr, int port);
```

```
 56.
 57. /**
 58.  * @brief Sets the FTP server to passive mode.
 59.  *
 60.  * This function sends the PASV command to the FTP server, which instructs the server to enter
passive mode.
 61.  * In passive mode, the server opens a random port and waits for the client to establish a
connection.
 62.  *
 63.  * @param sockfd The socket file descriptor connected to the FTP server.
 64.  * @return Returns 0 on success, or -1 if an error occurred.
 65.  */
 66. int pasv(int sockfd);
 67.
 68. /**
 69.  * @brief Performs the login operation for the given URL object and socket file descriptor.
 70.  *
 71.  * This function is responsible for performing the login operation using the provided URL object
 72.  * and socket file descriptor. It establishes a connection with the server and sends the
necessary
 73.  * login credentials. The function returns an integer value indicating the success or failure of
 74.  * the login operation.
 75.  *
 76.  * @param urlObj A pointer to the URL object containing the necessary login information.
 77.  * @param sockfd The socket file descriptor used for communication with the server.
 78.  * @return An integer value indicating the success or failure of the login operation.
 79.  */
 80. int login(url* urlObj, int sockfd);
 81.
 82. /**
 83.  * @brief Receives a reply from the server through the specified socket.
 84.  *
 85.  * This function reads the server's response from the socket and returns it as a null-terminated
string.
 86.  *
 87.  * @param sockfd The socket descriptor.
 88.  * @return A pointer to the received reply as a null-terminated string. The caller is responsible
for freeing the memory allocated for the string.
 89.  */
 90. char * getReply(int sockfd);
 91.
 92. /**
 93.  * @brief Retrieves a resource from a given URL path using two socket descriptors.
 94.  *
 95.  * This function establishes a connection with the server using the provided socket descriptors
 96.  * and retrieves the resource specified by the URL path. The retrieved resource is then stored
 97.  * in the local file system.
 98.  *
 99.  * @param sockfd The socket descriptor for the control connection.
100.  * @param sockfd2 The socket descriptor for the data connection.
101.  * @param url_path The URL path of the resource to be retrieved.
102.  * @return Returns 0 on success, or a negative value on failure.
103.  */
104. int retrieveResource(int sockfd, int sockfd2, const char* url_path);
105.
106. #endif // DOWNLOAD_H
107.
```

A2. Download.c

```
 1. #include "../include/download.h"
 2.
 3. url convertToURL(const char* urlStr) {
 4.     url urlObj;
 5.     memset(&urlObj, 0, sizeof(urlObj));
```

```
 6.
 7.      // Check if the URL starts with "ftp://"
 8.      const char* ftpProtocol = "ftp://";
 9.      if (strncmp(urlStr, ftpProtocol, strlen(ftpProtocol)) != 0) {
10.          fprintf(stderr, "Invalid URL format. The URL should start with 'ftp://'.\n\n");
11.          return urlObj;
12.      }
13.
14.      // Skip the "ftp://" part
15.      urlStr += strlen(ftpProtocol);
16.
17.      // Parse the user info if present
18.      const char* userInfoEnd = strchr(urlStr, '@');
19.      if (userInfoEnd != NULL) {
20.          size_t userInfoLen = userInfoEnd - urlStr;
21.
22.          // Check for the presence of ":" to separate username and password
23.          const char* passwordSeparator = strchr(urlStr, ':');
24.          if (passwordSeparator != NULL && passwordSeparator < userInfoEnd) {
25.              size_t usernameLen = passwordSeparator - urlStr;
26.              size_t passwordLen = userInfoEnd - passwordSeparator - 1;
27.
28.              strncpy(urlObj.user, urlStr, usernameLen);
29.              urlObj.user[usernameLen] = '\0';
30.
31.              strncpy(urlObj.password, passwordSeparator + 1, passwordLen);
32.              urlObj.password[passwordLen] = '\0';
33.          } else {
34.              // If no ":", the entire user info is the username
35.              strncpy(urlObj.user, urlStr, userInfoLen);
36.              urlObj.user[userInfoLen] = '\0';
37.          }
38.
39.          // Move the pointer past the "@" symbol
40.          urlStr = userInfoEnd + 1;
41.      }
42.
43.      // Parse the host
44.      const char* hostEnd = strchr(urlStr, '/');
45.      if (hostEnd != NULL && hostEnd > urlStr) {
46.          size_t hostLen = hostEnd - urlStr;
47.          strncpy(urlObj.host, urlStr, hostLen);
48.          urlObj.host[hostLen] = '\0';
49.
50.          urlStr = hostEnd + 1;
51.      } else {
52.          // If there's no "/", the entire remaining string is the host
53.          strcpy(urlObj.host, urlStr);
54.          urlStr += strlen(urlStr);
55.      }
56.
57.      // The remaining part is the URL path
58.      strncpy(urlObj.url_path, urlStr, sizeof(urlObj.url_path) - 1);
59.      urlObj.url_path[sizeof(urlObj.url_path) - 1] = '\0';
60.
61.      return urlObj;
62.  }
63.
64.  int connectSocket(char *ip_addr, int port) {
65.      int sockfd;
66.      struct sockaddr_in server_addr;
67.
68.      /*server address handling*/
69.      bzero((char *) &server_addr, sizeof(server_addr));
70.      server_addr.sin_family = AF_INET;
71.      server_addr.sin_addr.s_addr = inet_addr(ip_addr);    /*32 bit Internet address network byte
ordered*/
```

```
72.     server_addr.sin_port = htons(port);        /*server TCP port must be network byte ordered */
73.
74.     /*open a TCP socket*/
75.     if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
76.         perror("socket()");
77.         exit(-1);
78.     }
79.     /*connect to the server*/
80.     if (connect(sockfd,
81.                 (struct sockaddr *) &server_addr,
82.                 sizeof(server_addr)) < 0) {
83.         perror("connect()");
84.         exit(-1);
85.     }
86.
87.     return sockfd;
88. }
89.
90. char * getReply(int sockfd){
91.     char *reply = malloc(MAX_SIZE);
92.
93.      size_t n = 0;
94.      ssize_t read;
95.
96.      FILE* fp = fdopen(sockfd, "r");
97.      while((read = getline(&reply, &n, fp)) != -1) {
98.                if(reply[3] == ' ') break;
99.      }
100.
101.     reply[1023] = '\0';
102.     printf("Reply: %s\n\n", reply);
103.
104.     return reply;
105.
106. }
107.
108. int login(url* urlObj, int sockfd) {
109.     char login[MAX_SIZE];
110.
111.     sprintf(login, "user %s\n\n", urlObj->user);
112.     write(sockfd, login, strlen(login));
113.
114.     char* reply = getReply(sockfd);
115.
116.     if (strncmp(reply, "4", 1) == 0 || strncmp(reply, "5", 1) == 0) {
117.         close(sockfd);
118.         return -1;
119.     }
120.
121.     sprintf(login, "pass %s\n\n", urlObj->password);
122.     write(sockfd, login, strlen(login));
123.
124.     reply = getReply(sockfd);
125.
126.     if (strncmp(reply, "4", 1) == 0 || strncmp(reply, "5", 1) == 0) {
127.         close(sockfd);
128.         return -1;
129.     }
130.
131.     return 0;
132. }
133.
134. int pasv(int sockfd) {
135.     char *pasv = malloc(MAX_SIZE);
136.     sprintf(pasv, "pasv\n\n");
137.     write(sockfd, pasv, strlen(pasv));
138.
```

```
    char *reply = getReply(sockfd);

    if (strncmp(reply, "4", 1) == 0 || strncmp(reply, "5", 1) == 0) {
        close(sockfd);
        return -1;
    }

    int port[2];

    sscanf(reply, "227 Entering Passive Mode (%*[^,],%*d,%*d,%*d,%d,%d)", &port[0], &port[1]);

    int port_pasv = port[0] * 256 + port[1];

    printf("port[0] is %d\n\n", port[0]);
    printf("port[1] is %d\n\n", port[1]);

    printf("port: %d\n\n", port_pasv);

    return port_pasv;
}

int retrieveResource(int sockfd, int sockfd2, const char* url_path) {
    char request[MAX_SIZE];
    sprintf(request, "retr %s\n\n", url_path);

    write(sockfd, request, strlen(request));

    char* reply = getReply(sockfd);

    if (strncmp(reply, "4", 1) == 0 || strncmp(reply, "5", 1) == 0) {
        close(sockfd);
        return -1;
    }

    const char* lastSlash = strrchr(url_path, '/');
    const char* filename = (lastSlash != NULL) ? lastSlash + 1 : url_path;

    printf("Filename: %s\n\n", filename);

     FILE *f = fopen(filename, "wb");
    if (f == NULL) {
        fprintf(stderr, "Error opening file for writing\n\n");
        close(sockfd);
        return -1;
    }

    char buf[MAX_SIZE];

    int bytes;

    while ((bytes = recv(sockfd2, buf, MAX_SIZE, 0)) > 0) {
        if (f == NULL) {
            fprintf(stderr, "Error: File pointer is NULL.\n\n");
            break;
        }

        size_t bytes_written = fwrite(buf, 1, bytes, f);

        if (bytes_written != bytes) {
            fprintf(stderr, "Error writing to the file.\n\n");
            break;
        }
    }

    fclose(f);

    return 0;
```

```
206. }
207.
208. int main(int argc, char *argv[]) {
209.     if (argc != 2) {
210.         fprintf(stderr, "Invalid number of arguments. The format should be: download
ftp://[<user>:<password>@]<host>/<url-path>\n\n");
211.         exit(-1);
212.     }
213.
214.     // check if url is valid and convert to url object
215.     url urlObj = convertToURL(argv[1]);
216.     if (strlen(urlObj.host) == 0) {
217.         fprintf(stderr, "Invalid URL\n\n");
218.         exit(-1);
219.     }
220.
221.     if (strlen(urlObj.user) == 0) {
222.         strcpy(urlObj.user, "anonymous");
223.     }
224.     if (strlen(urlObj.password) == 0) {
225.         strcpy(urlObj.password, "anonymous");
226.     }
227.
228.
229.     printf("Host: %s\n\n", urlObj.host);
230.     printf("URL path: %s\n\n", urlObj.url_path);
231.
232.     // get ip address from host
233.     struct hostent *h;
234.
235.     if ((h = gethostbyname(urlObj.host)) == NULL) {
236.         herror("gethostbyname()");
237.         exit(-1);
238.     }
239.
240.     printf("Host name  : %s\n\n", h->h_name);
241.     printf("IP Address : %s\n\n", inet_ntoa(*((struct in_addr *) h->h_addr)));
242.
243.     char *ip_addr = inet_ntoa(*((struct in_addr *) h->h_addr));
244.
245.     int sockfd = connectSocket(ip_addr, SERVER);
246.
247.     // connect to server A
248.     if (sockfd < 0) {
249.         fprintf(stderr, "Error connecting to server\n\n");
250.         exit(-1);
251.     }
252.
253.     char * reply = getReply(sockfd);
254.
255.     if (strncmp(reply, "4", 1) == 0 || strncmp(reply, "5", 1) == 0) {
256.         close(sockfd);
257.         return -1;
258.     }
259.
260.     // login
261.     if (login(&urlObj, sockfd) < 0) {
262.         fprintf(stderr, "Error logging in\n\n");
263.         exit(-1);
264.     }
265.
266.     // pasv and connect to server B
267.     int passive_port = pasv(sockfd);
268.
269.     if (passive_port < 0) {
270.         fprintf(stderr, "Error entering passive mode\n\n");
271.         exit(-1);
```

```
272.     }
273.
274.     printf("Passive mode established\n\n");
275.
276.
277.     // request resource and download it
278.
279.     int sockfd2 = connectSocket(ip_addr, passive_port);
280.
281.
282.
283.     int command = retrieveResource(sockfd, sockfd2, urlObj.url_path);
284.
285.     if (command < 0){
286.         fprintf(stderr, "Error requesting resource\n\n");
287.         exit(-1);
288.     }
289.
290.
291.     printf("Resource downloaded\n\n");
292.
293.     write(sockfd, "quit\n\n", strlen("quit\n\n"));
294.
295.     if(close(sockfd)!=0 || close(sockfd2)!=0) {
296.         printf("Error: Couldn't close sockets.\n\n");
297.         exit(-1);
298.     }
299.
300.     return 0;
301. }
302.
```

B. Comandos utilizado nas experiências

B1. Exp1

B1a. Tux43

```
1. ifconfig eth0 down
2. ifconfig eth0 up
3. ifconfig eth0 172.16.40.1/24
```

B1b. Tux44

```
1. ifconfig eth0 down
2. ifconfig eth0 up
3. ifconfig eth0 172.16.40.254/24
4. ping 172.16.40.1
```

B1c. Tux43

```
1. ping 172.16.40.254
2. route -n
3. arp -a
```

B1d. Tux44

```
1. ping 172.16.40.1
2. route -n
3. arp -a
```

B1e. Tux43

```
1. arp -d 172.16.40.254/24
2. ping 172.16.40.254
```

B2. Exp2

B2a. Tux42

```
1. ifconfig eth0 down
2. ifconfig eth0 up
3. ifconfig eth0 172.16.41.1/24
```

B2b. Consola do Switch

```
 1. /system reset-configuration
 2. /interface bridge add name=bridge40
 3. /interface bridge add name=bridge41
 4. /interface bridge port remove [find interface=ether22] //ether22 =
porta Tux42
 5. /interface bridge port remove [find interface=ether23] //ether2 = porta
Tux43
 6. /interface bridge port remove [find interface=ether24] //ether3 = porta
Tux44
 7. /interface bridge port add bridge=bridge40 interface=ether23
 8. /interface bridge port add bridge=bridge40 interface=ether24
 9. /interface bridge port add bridge=bridge41 interface=ether22
```

B2c. Tux43

```
1. ping 172.16.40.254 //Tux44
2. ping 172.16.41.1 //Tux42
3. ping -b 172.16.40.255
```

B2d. Tux42

```
   1. ping -b 172.16.41.255
```

B3. Exp3

B3a. Tux44

```
1. ifconfig eth1 down
2. ifconfig eth1 up
3. ifconfig eth1 172.16.41.253/24
```

B3b. Consola do Switch

```
1. /interface bridge port remove [find interface=ether21] //ether21
   = porta Tux44eth1
2. /interface bridge port add bridge=bridge41 interface=ether21
```

B3c. Tux44

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

B3d. Tux42

```
   1. route add -net 172.16.40.0/24 gw 172.16.41.253
```

B3e. Tux43

```
1. route add -net 172.16.41.0/24 gw 172.16.40.254
2. route -n
3. ping 172.16.40.254
4. ping 172.16.41.253
5. ping 172.16.41.1
6. arp -d 172.16.40.254
```

B3f. Tux42

```
   1. arp -d 172.16.41.253
```

B3g. Tux44

```
1. arp -d 172.16.40.1
2. arp -d 172.16.41.1
```

B4. Exp4

B4a. Consola do switch

```
1. /interface bridge port remove [find interface=ether2] //ether2 = porta Rc
2. /interface bridge port add bridge=bridge41 interface=ether2
```

B4b. Consola do router

```
1. /system reset-configuration
2. /ip address add address=172.16.1.49/24 interface=ether1
3. /ip address add address=172.16.41.254/24 interface=ether2
```

B4c. Tux43

```
1. route add default gw 172.16.40.254
```

B4d. Tux44

```
1. route add default gw 172.16.41.254
```

B4e. Tux42

```
1. route add default gw 172.16.41.254
```

B4f. Consola do router

```
1. /ip route add dst-address=172.16.40.0/24 gateway=172.16.41.253
2. /ip route add dst-address=0.0.0.0/0 gateway=172.16.1.254
```

B4g. Tux43

```
1. ping 172.16.40.254
2. ping 172.16.41.253
3. ping 172.16.41.1
4. ping 172.16.41.254
```

B4h. Tux42

```
1. sysctl net.ipv4.conf.eth0.accept_redirects=0
2. sysctl net.ipv4.conf.all.accept_redirects=0
3. route del -net 172.16.40.0 gw 172.16.41.253
```

```
4. ping 172.1
5. traceroute 172.16.40.1
6. route add -net 172.16.40.0/24 gw 172.16.41.253
7. traceroute 172.16.40.1
8. sysctl net.ipv4.conf.eth0.accept_redirects=1
9. sysctl net.ipv4.conf.all.accept_redirects=1
```

B4i. Tux43

```
1. ping 172.16.1.254
```

B4j. Consola router

```
1. /ip firewall nat disable 0
```

B4k. Tux43

```
1. ping 172.16.1.254
```

B4l. Consola router

```
1. /ip firewall nat enable 0
```

B5. Exp5

B5a. Tux43, Tux44, Tux42

```
1. nano /etc/resolv.conf
2. alterar conteúdo no ficheiro para ‘nameserver 172.16.1.1’
3. ping google.com
```

B6. Exp6

B6a. Tux43

```
1. alterar DNS
2. make
3. ./download ftp://rcom:rcom@netlab1.fe.up.pt/pipe.txt
4. ./download ftp://rcom:rcom@netlab1.fe.up.pt/files/crab.mp4
5. ./download ftp://rcom:rcom@netlab1.fe.up.pt/files/crab.mp4
```

B6b. Tux42

```
1. ./download ftp://rcom:rcom@netlab1.fe.up.pt/files/crab.mp4
```

C. Logs e estrutura da reda das experiências

C1. Exp1

C1a. Tux43 Ping

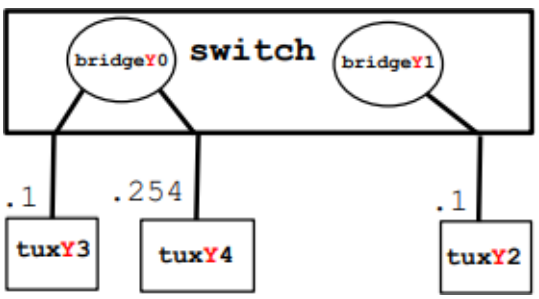
No.	Time	Source	Destination	Protocol	Length	Info
22	37.379197024	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x3304, seq=1/256, ttl=64 (reply in 23)
23	37.379335796	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x3304, seq=1/256, ttl=64 (request in 22)
24	38.041259925	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8002
25	38.395839060	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x3304, seq=2/512, ttl=64 (reply in 26)
26	38.396019108	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x3304, seq=2/512, ttl=64 (request in 25)
27	39.419823997	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x3304, seq=3/768, ttl=64 (reply in 28)
28	39.419974292	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x3304, seq=3/768, ttl=64 (request in 27)
29	40.043471934	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8002
30	40.443823390	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x3304, seq=4/1024, ttl=64 (reply in 31)
31	40.443971241	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x3304, seq=4/1024, ttl=64 (request in 30)
32	41.467831793	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x3304, seq=5/1280, ttl=64 (reply in 33)
33	41.467975104	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x3304, seq=5/1280, ttl=64 (request in 32)

C1b. Atributos dos pacotes – wireshark

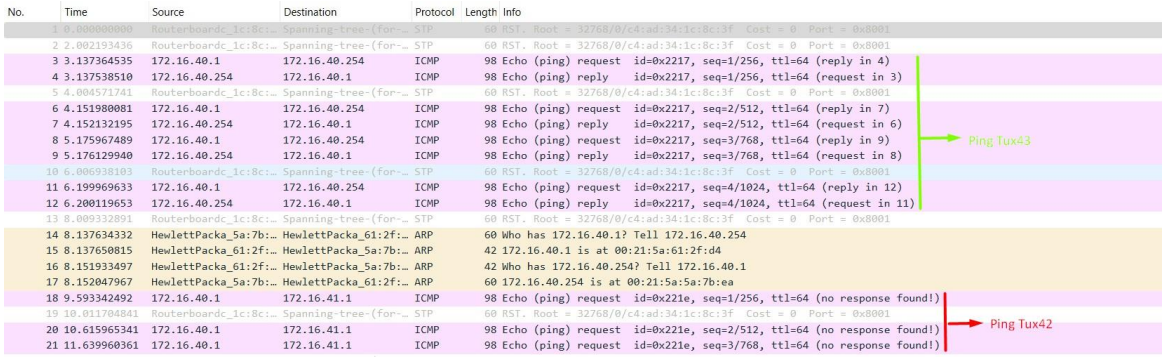
No.	Time	Source	Destination	Protocol	Length	Info
22	37.379197024	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x3304, seq=1/256, ttl=64 (reply in 23)
23	37.379335796	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x3304, seq=1/256, ttl=64 (request in 22)
24	38.041259925	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8002
25	38.395839060	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x3304, seq=2/512, ttl=64 (reply in 26)
26	38.396019108	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x3304, seq=2/512, ttl=64 (request in 25)
27	39.419823997	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x3304, seq=3/768, ttl=64 (reply in 28)
28	39.419974292	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x3304, seq=3/768, ttl=64 (request in 27)
29	40.043471934	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8002
30	40.443823390	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x3304, seq=4/1024, ttl=64 (reply in 31)
31	40.443971241	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x3304, seq=4/1024, ttl=64 (request in 30)
32	41.467831793	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x3304, seq=5/1280, ttl=64 (reply in 33)
33	41.467975104	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x3304, seq=5/1280, ttl=64 (request in 32)

C2. Exp2

C2a. Estrutura dos Tuxs na rede



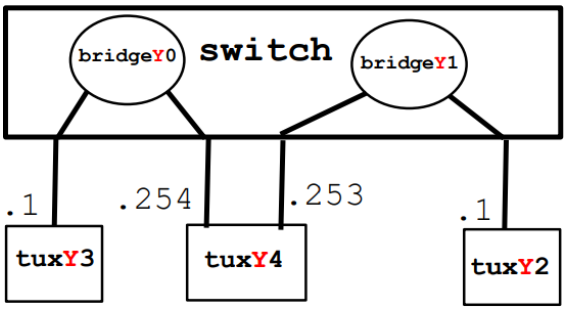
C2b. Tux43 – ping Tux44 (172.16.40.254) – ping Tux42 (172.16.41.1)



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
2	2.002193436	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
3	3.137364535	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x2217, seq=1/256, ttl=64 (reply in 4)
4	3.137538510	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x2217, seq=1/256, ttl=64 (request in 3)
5	4.004571741	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
6	4.151980081	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x2217, seq=2/512, ttl=64 (reply in 7)
7	4.152132195	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x2217, seq=2/512, ttl=64 (request in 6)
8	5.175967489	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x2217, seq=3/768, ttl=64 (reply in 9)
9	5.176129940	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x2217, seq=3/768, ttl=64 (request in 8)
10	6.006938103	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
11	6.199969633	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x2217, seq=4/1024, ttl=64 (reply in 12)
12	6.200119653	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x2217, seq=4/1024, ttl=64 (request in 11)
13	8.009332891	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
14	8.137634332	HewlettPacka_5a:7b:…	HewlettPacka_61:2f:…	ARP	60	Who has 172.16.40.1? Tell 172.16.40.254
15	8.137650815	HewlettPacka_61:2f:…	HewlettPacka_5a:7b:…	ARP	42	172.16.40.1 is at 00:21:5a:61:2f:d4
16	8.151933497	HewlettPacka_61:2f:…	HewlettPacka_5a:7b:…	ARP	42	Who has 172.16.40.254? Tell 172.16.40.1
17	8.152047967	HewlettPacka_5a:7b:…	HewlettPacka_61:2f:…	ARP	60	172.16.40.254 is at 00:21:5a:5a:7b:ea
18	9.593342492	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x221e, seq=1/256, ttl=64 (no response found!)
19	10.011704841	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
20	10.615965341	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x221e, seq=2/512, ttl=64 (no response found!)
21	11.639960361	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x221e, seq=3/768, ttl=64 (no response found!)

C3. Exp3

C3a. Estrutura dos Tuxs na rede



C3b. Tux43

No.	Time	Source	Destination	Protocol	Length	Info
7	11.148164477	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x1865, seq=1/256, ttl=64 (reply in 8)
8	11.148328463	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1865, seq=1/256, ttl=64 (request in 7)
9	12.013368168	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
10	12.154468289	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x1865, seq=2/512, ttl=64 (reply in 11)
11	12.154620263	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1865, seq=2/512, ttl=64 (request in 10)
12	13.178488130	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x1865, seq=3/768, ttl=64 (reply in 13)
13	13.178621176	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1865, seq=3/768, ttl=64 (request in 12)
14	14.015844452	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
15	14.202486250	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x1865, seq=4/1024, ttl=64 (reply in 16)
16	14.202612522	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1865, seq=4/1024, ttl=64 (request in 15)
17	15.226491004	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x1865, seq=5/1280, ttl=64 (reply in 18)
18	15.226622724	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1865, seq=5/1280, ttl=64 (request in 17)
19	16.018109283	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
20	16.227599010	HewlettPacka_5a:7b:…	HewlettPacka_61:2f:…	ARP	60	Who has 172.16.40.1? Tell 172.16.40.254
21	16.227617309	HewlettPacka_61:2f:…	HewlettPacka_5a:7b:…	ARP	42	172.16.40.1 is at 00:21:5a:61:2f:d4
22	16.250448407	HewlettPacka_61:2f:…	HewlettPacka_5a:7b:…	ARP	42	Who has 172.16.40.254? Tell 172.16.40.1
23	16.250557918	HewlettPacka_5a:7b:…	HewlettPacka_61:2f:…	ARP	60	172.16.40.254 is at 00:21:5a:5a:7b:ea
24	18.020385358	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
25	20.022651656	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
26	22.024903846	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
27	24.027176359	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
28	24.828039395	172.16.40.1	172.16.41.253	ICMP	98	Echo (ping) request id=0x186c, seq=1/256, ttl=64 (reply in 29)
29	24.828205267	172.16.41.253	172.16.40.1	ICMP	98	Echo (ping) reply id=0x186c, seq=1/256, ttl=64 (request in 28)
30	25.850487748	172.16.40.1	172.16.41.253	ICMP	98	Echo (ping) request id=0x186c, seq=2/512, ttl=64 (reply in 31)
31	25.850622052	172.16.41.253	172.16.40.1	ICMP	98	Echo (ping) reply id=0x186c, seq=2/512, ttl=64 (request in 30)
32	26.029427991	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
33	26.874489221	172.16.40.1	172.16.41.253	ICMP	98	Echo (ping) request id=0x186c, seq=3/768, ttl=64 (reply in 34)
34	26.874648598	172.16.41.253	172.16.40.1	ICMP	98	Echo (ping) reply id=0x186c, seq=3/768, ttl=64 (request in 33)
35	27.898486294	172.16.40.1	172.16.41.253	ICMP	98	Echo (ping) request id=0x186c, seq=4/1024, ttl=64 (reply in 36)
36	27.898623042	172.16.41.253	172.16.40.1	ICMP	98	Echo (ping) reply id=0x186c, seq=4/1024, ttl=64 (request in 35)
37	28.031201901	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
38	28.922487348	172.16.40.1	172.16.41.253	ICMP	98	Echo (ping) request id=0x186c, seq=5/1280, ttl=64 (reply in 39)
39	28.922627448	172.16.41.253	172.16.40.1	ICMP	98	Echo (ping) reply id=0x186c, seq=5/1280, ttl=64 (request in 38)
40	30.033964149	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
41	31.851969802	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1873, seq=1/256, ttl=64 (reply in 42)
42	31.852418809	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1873, seq=1/256, ttl=63 (request in 41)

Ping Tux44

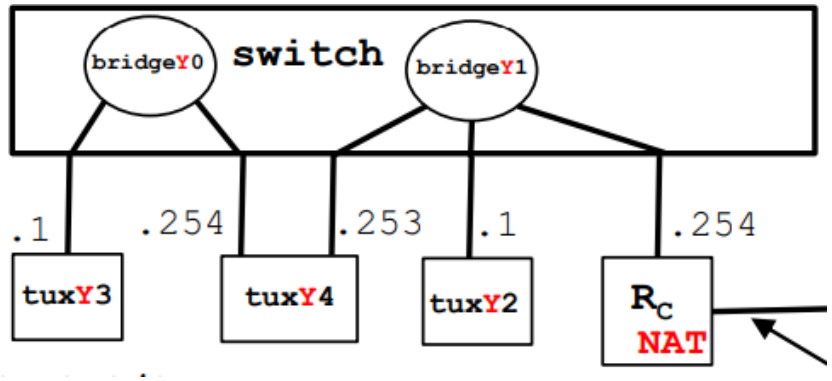
Ping Tux42

C3c. Tux44

49	89.858119503	KYE_25:1a:f4	Broadcast	ARP	42	Who has 172.16.41.1? Tell 172.16.41.253
50	89.858267077	HewlettPacka_d7:45:…	KYE_25:1a:f4	ARP	60	172.16.41.1 is at 00:1f:29:d7:45:c4
51	89.858273712	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x18e4, seq=1/256, ttl=63 (reply in 52)
52	89.858408365	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x18e4, seq=1/256, ttl=64 (request in 51)
53	90.098026462	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3e Cost = 0 Port = 0x8002
54	90.865445888	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x18e4, seq=2/512, ttl=63 (reply in 55)
55	90.865577189	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x18e4, seq=2/512, ttl=64 (request in 54)
56	91.889424722	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x18e4, seq=3/768, ttl=63 (reply in 57)
57	91.889556163	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x18e4, seq=3/768, ttl=64 (request in 56)
58	92.100268642	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3e Cost = 0 Port = 0x8002
59	92.913413962	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x18e4, seq=4/1024, ttl=63 (reply in 60)
60	92.913546171	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x18e4, seq=4/1024, ttl=64 (request in 59)
61	93.937419964	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x18e4, seq=5/1280, ttl=63 (reply in 62)
62	93.937575220	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x18e4, seq=5/1280, ttl=64 (request in 61)
63	94.102502720	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3e Cost = 0 Port = 0x8002
64	95.088987530	HewlettPacka_d7:45:…	KYE_25:1a:f4	ARP	60	Who has 172.16.41.253? Tell 172.16.41.1
65	95.088996749	KYE_25:1a:f4	HewlettPacka_d7:45:…	ARP	42	172.16.41.253 is at 00:c0:df:25:1a:f4

C4. Exp4

C4a. Estrutura dos Tuxs na rede



C4b. Tux43

9	14.298210561	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bf3, seq=1/256, ttl=64 (reply in 10)
10	14.298567029	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bf3, seq=1/256, ttl=63 (request in 9)
11	15.321735456	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bf3, seq=2/512, ttl=64 (reply in 12)
12	15.322018661	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bf3, seq=2/512, ttl=63 (request in 11)
13	16.017166546	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
14	16.345732126	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bf3, seq=3/768, ttl=64 (reply in 15)
15	16.346006670	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bf3, seq=3/768, ttl=63 (request in 14)
16	17.369732915	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bf3, seq=4/1024, ttl=64 (reply in 17)
17	17.370007390	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bf3, seq=4/1024, ttl=63 (request in 16)
18	18.018866884	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
19	18.393732658	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1bf3, seq=5/1280, ttl=64 (reply in 20)
20	18.394025361	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bf3, seq=5/1280, ttl=63 (request in 19)
21	19.449701051	HewlettPacka_61:2f:…	HewlettPacka_5a:7b:…	ARP	42	Who has 172.16.40.254? Tell 172.16.40.1
22	19.449820619	HewlettPacka_5a:7b:…	HewlettPacka_61:2f:…	ARP	60	172.16.40.254 is at 00:21:5a:5a:7b:ea
23	20.021542969	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
24	22.023622962	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
25	24.025077182	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
26	26.027402386	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
27	26.953876498	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x1bfa, seq=1/256, ttl=64 (reply in 28)
28	26.954042439	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bfa, seq=1/256, ttl=64 (request in 27)
29	27.961737916	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x1bfa, seq=2/512, ttl=64 (reply in 30)
30	27.961875642	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bfa, seq=2/512, ttl=64 (request in 29)
31	28.029037703	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
32	28.985733747	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x1bfa, seq=3/768, ttl=64 (reply in 33)
33	28.985875105	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bfa, seq=3/768, ttl=64 (request in 32)
34	30.009737121	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x1bfa, seq=4/1024, ttl=64 (reply in 35)
35	30.009898313	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bfa, seq=4/1024, ttl=64 (request in 34)
36	30.032097493	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
37	31.033731067	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x1bfa, seq=5/1280, ttl=64 (reply in 38)
38	31.033864253	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1bfa, seq=5/1280, ttl=64 (request in 37)
39	31.989995285	HewlettPacka_5a:7b:…	HewlettPacka_61:2f:…	ARP	60	Who has 172.16.40.1? Tell 172.16.40.254
40	31.990017564	HewlettPacka_61:2f:…	HewlettPacka_5a:7b:…	ARP	42	172.16.40.1 is at 00:21:5a:61:2f:d4

C4c. Tux42

No.	Time	Source	Destination	Protocol	Length	Info
6	10.010740022	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/74:4d:28:eb:18:db Cost = 10 Port = 0x8001
7	12.012472457	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/74:4d:28:eb:18:db Cost = 10 Port = 0x8001
8	14.014654320	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/74:4d:28:eb:18:db Cost = 10 Port = 0x8001
9	14.732935332	0.0.0.0	255.255.255.255	MNDP	159	5678 → 5678 Len=117
10	14.732957612	Routerboardc_1c:8c:…	CDP/VTP/DTP/PAgP/UD…	CDP	93	Device ID: MikroTik Port ID: bridge41
11	14.733012228	Routerboardc_1c:8c:…	LLDP_Multicast	LLDP	110	MA/c4:ad:34:1c:8c:3e IN/bridge41 120 SysN=MikroTik SysD=MikroTik Rout
12	16.006826657	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/74:4d:28:eb:18:db Cost = 10 Port = 0x8001
13	18.008965638	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/74:4d:28:eb:18:db Cost = 10 Port = 0x8001
14	20.011136048	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/74:4d:28:eb:18:db Cost = 10 Port = 0x8001
15	22.013296330	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/74:4d:28:eb:18:db Cost = 10 Port = 0x8001
16	24.014925679	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/74:4d:28:eb:18:db Cost = 10 Port = 0x8001
17	26.007085165	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/74:4d:28:eb:18:db Cost = 10 Port = 0x8001
18	28.009224076	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/74:4d:28:eb:18:db Cost = 10 Port = 0x8001
19	28.049309605	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) request id=0x2f61, seq=1/256, ttl=64 (reply in 20)
20	28.049736755	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) reply id=0x2f61, seq=1/256, ttl=63 (request in 19)
21	29.081558907	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) request id=0x2f61, seq=2/512, ttl=64 (reply in 23)
22	29.081743148	172.16.41.254	172.16.41.1	ICMP	126	Redirect (Redirect for host)
23	29.081967968	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) reply id=0x2f61, seq=2/512, ttl=63 (request in 21)
24	30.011391622	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/74:4d:28:eb:18:db Cost = 10 Port = 0x8001
25	30.105550303	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) request id=0x2f61, seq=3/768, ttl=64 (reply in 27)
26	30.105727141	172.16.41.254	172.16.41.1	ICMP	126	Redirect (Redirect for host)
27	30.105935618	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) reply id=0x2f61, seq=3/768, ttl=63 (request in 25)
28	31.133551991	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) request id=0x2f61, seq=4/1024, ttl=64 (reply in 30)
29	31.133732182	172.16.41.254	172.16.41.1	ICMP	126	Redirect (Redirect for host)
30	31.133920265	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) reply id=0x2f61, seq=4/1024, ttl=63 (request in 28)
31	32.012846646	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/74:4d:28:eb:18:db Cost = 10 Port = 0x8001
32	32.153557330	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) request id=0x2f61, seq=5/1280, ttl=64 (reply in 34)
33	32.153753166	172.16.41.254	172.16.41.1	ICMP	126	Redirect (Redirect for host)
34	32.153939433	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) reply id=0x2f61, seq=5/1280, ttl=63 (request in 32)
35	33.081514176	HewlettPacka_d7:45:…	Routerboardc_eb:18:…	ARP	42	Who has 172.16.41.254? Tell 172.16.41.1
36	33.081637865	Routerboardc_eb:18:…	HewlettPacka_d7:45:…	ARP	60	172.16.41.254 is at 74:4d:28:eb:18:db
37	33.094883772	KYE_25:1a:f4	HewlettPacka_d7:45:…	ARP	60	Who has 172.16.41.1? Tell 172.16.41.253
38	33.094896134	HewlettPacka_d7:45:…	KYE_25:1a:f4	ARP	42	172.16.41.1 is at 00:1f:29:d7:45:c4
39	33.177555222	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) request id=0x2f61, seq=6/1536, ttl=64 (reply in 41)
40	33.177735413	172.16.41.254	172.16.41.1	ICMP	126	Redirect (Redirect for host)
41	33.177907851	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) reply id=0x2f61, seq=6/1536, ttl=63 (request in 39)

C5. Exp5

C5a. Tux43

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
2	0.572241198	172.16.40.1	172.16.1.1	DNS	70	Standard query 0x0bf3 A google.com
3	0.572252233	172.16.40.1	172.16.1.1	DNS	70	Standard query 0x63fd AAAA google.com
4	0.573009727	172.16.1.1	172.16.40.1	DNS	86	Standard query response 0x0bf3 A google.com A 142.250.184.174
5	0.573030958	172.16.1.1	172.16.40.1	DNS	98	Standard query response 0x63fd AAAA google.com AAAA 2a00:1450:4003:80c::200e
6	0.573423184	172.16.40.1	142.250.184.174	ICMP	98	Echo (ping) request id=0x20cc, seq=1/256, ttl=64 (reply in 7)
7	0.589323631	142.250.184.174	172.16.40.1	ICMP	98	Echo (ping) reply id=0x20cc, seq=1/256, ttl=107 (request in 6)
8	0.589448157	172.16.40.1	172.16.1.1	DNS	88	Standard query 0xce58 PTR 174.184.250.142.in-addr.arpa
9	0.589995779	172.16.1.1	172.16.40.1	DNS	127	Standard query response 0xce58 PTR 174.184.250.142.in-addr.arpa PTR mad07s23-in-f14.1e100.net
10	1.575087070	172.16.40.1	142.250.184.174	ICMP	98	Echo (ping) request id=0x20cc, seq=2/512, ttl=64 (reply in 11)
11	1.590336879	142.250.184.174	172.16.40.1	ICMP	98	Echo (ping) reply id=0x20cc, seq=2/512, ttl=107 (request in 10)
12	2.002165062	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
13	2.576416488	172.16.40.1	142.250.184.174	ICMP	98	Echo (ping) request id=0x20cc, seq=3/768, ttl=64 (reply in 14)
14	2.591396782	142.250.184.174	172.16.40.1	ICMP	98	Echo (ping) reply id=0x20cc, seq=3/768, ttl=107 (request in 13)
15	3.576694033	172.16.40.1	142.250.184.174	ICMP	98	Echo (ping) request id=0x20cc, seq=4/1024, ttl=64 (reply in 16)
16	3.592355485	142.250.184.174	172.16.40.1	ICMP	98	Echo (ping) reply id=0x20cc, seq=4/1024, ttl=107 (request in 15)

C5b. Tux42

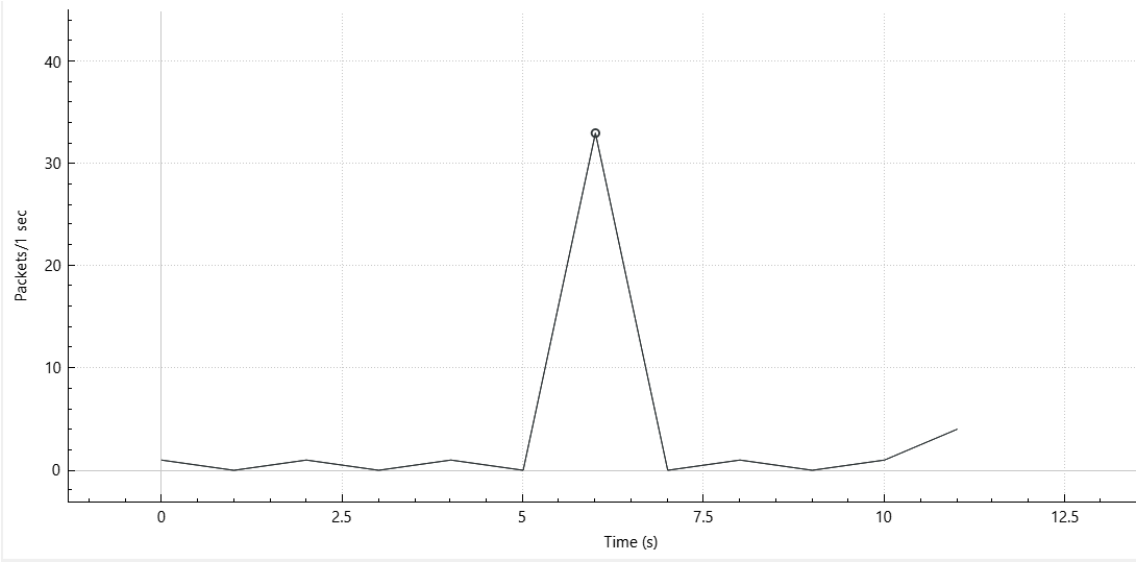
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/74:4d:28:eb:18:db Cost = 10 Port = 0x8001
2	0.806762846	172.16.41.1	172.16.1.1	DNS	70	Standard query 0x09a9 A google.com
3	0.806772624	172.16.41.1	172.16.1.1	DNS	70	Standard query 0xe2b1 AAAA google.com
4	0.807369139	172.16.1.1	172.16.41.1	DNS	86	Standard query response 0x09a9 A google.com A 142.250.184.174
5	0.807385063	172.16.1.1	172.16.41.1	DNS	98	Standard query response 0xe2b1 AAAA google.com AAAA 2a00:1450:4003:80c::200e
6	0.807678466	172.16.41.1	142.250.184.174	ICMP	98	Echo (ping) request id=0x3362, seq=1/256, ttl=64 (reply in 7)
7	0.823839901	142.250.184.174	172.16.41.1	ICMP	98	Echo (ping) reply id=0x3362, seq=1/256, ttl=108 (request in 6)
8	0.823939774	172.16.41.1	172.16.1.1	DNS	88	Standard query 0x0d12 PTR 174.184.250.142.in-addr.arpa
9	0.824497737	172.16.1.1	172.16.41.1	DNS	127	Standard query response 0x0d12 PTR 174.184.250.142.in-addr.arpa PTR mad07s23-in-f14.1e100.net
10	1.809570444	172.16.41.1	142.250.184.174	ICMP	98	Echo (ping) request id=0x3362, seq=2/512, ttl=64 (reply in 11)
11	1.824844125	142.250.184.174	172.16.41.1	ICMP	98	Echo (ping) reply id=0x3362, seq=2/512, ttl=108 (request in 10)
12	2.002154555	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/74:4d:28:eb:18:db Cost = 10 Port = 0x8001
13	2.810905855	172.16.41.1	142.250.184.174	ICMP	98	Echo (ping) request id=0x3362, seq=3/768, ttl=64 (reply in 14)
14	2.825837803	142.250.184.174	172.16.41.1	ICMP	98	Echo (ping) reply id=0x3362, seq=3/768, ttl=108 (request in 13)
15	3.811921952	172.16.41.1	142.250.184.174	ICMP	98	Echo (ping) request id=0x3362, seq=4/1024, ttl=64 (reply in 16)
16	3.826852922	142.250.184.174	172.16.41.1	ICMP	98	Echo (ping) reply id=0x3362, seq=4/1024, ttl=108 (request in 15)

C6. Exp6

C6a. TCP Header

```
▾ Transmission Control Protocol, Src Port: 37198, Dst Port: 21, Seq: 1, Ack: 1, Len: 0
    Source Port: 37198
    Destination Port: 21
    [Stream index: 0]
  › [Conversation completeness: Complete, WITH_DATA (63)]
    [TCP Segment Len: 0]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 407172498
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 3762196234
    1000 .... = Header Length: 32 bytes (8)
  › Flags: 0x010 (ACK)
    Window: 502
    [Calculated window size: 64256]
    [Window size scaling factor: 128]
    Checksum: 0x0269 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  › Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  › [Timestamps]
  › [SEQ/ACK analysis]
```

C6b. Packets/sec



C6c. Log da captura

No.	Time	Source	Destination	Protocol	Length	Info
4	6.006444270	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
5	6.764016147	172.16.40.1	193.136.28.10	DNS	76	Standard query 0xb438 A netlab1.fe.up.pt
6	6.764790542	193.136.28.10	172.16.40.1	DNS	286	Standard query response 0xb438 A netlab1.fe.up.pt A 192.168.109.136 NS cns1.fe.up.pt NS ns2.fe.up.pt NS cns2.fe.…
7	6.764892649	172.16.40.1	192.168.109.136	TCP	74	37198 → 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2162437872 TSecr=0 WS=128
8	6.765794084	192.168.109.136	172.16.40.1	TCP	74	21 → 37198 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=2429736645 TSecr=2162437872 WS=128
9	6.765812941	172.16.40.1	192.168.109.136	TCP	66	37198 → 21 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2162437873 TSecr=2429736645
10	6.767720084	192.168.109.136	172.16.40.1	FTP	100	Response: 220 Welcome to netlab-FTP server
11	6.767730141	172.16.40.1	192.168.109.136	TCP	66	37198 → 21 [ACK] Seq=1 Ack=35 Win=64256 Len=0 TSval=2162437875 TSecr=2429736646
12	6.767759684	172.16.40.1	192.168.109.136	FTP	76	Request: user rcom
13	6.768331610	192.168.109.136	172.16.40.1	TCP	66	21 → 37198 [ACK] Seq=35 Ack=11 Win=65280 Len=0 TSval=2429736647 TSecr=2162437875
14	6.768378264	192.168.109.136	172.16.40.1	FTP	100	Response: 331 Please specify the password.
15	6.768403477	172.16.40.1	192.168.109.136	FTP	76	Request: pass rcom
16	6.769025199	192.168.109.136	172.16.40.1	TCP	66	21 → 37198 [ACK] Seq=69 Ack=21 Win=65280 Len=0 TSval=2429736648 TSecr=2162437875
17	6.777899731	192.168.109.136	172.16.40.1	FTP	89	Response: 230 Login successful.
18	6.777979070	172.16.40.1	192.168.109.136	FTP	71	Request: pasv
19	6.778561892	192.168.109.136	172.16.40.1	TCP	66	21 → 37198 [ACK] Seq=92 Ack=26 Win=65280 Len=0 TSval=2429736657 TSecr=2162437885
20	6.778724271	192.168.109.136	172.16.40.1	FTP	120	Response: 227 Entering Passive Mode (192,168,109,136,164,231).
21	6.778816950	172.16.40.1	192.168.109.136	TCP	74	42464 → 42215 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2162437886 TSecr=0 WS=128
22	6.779476178	192.168.109.136	172.16.40.1	TCP	74	42215 → 42464 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=2429736658 TSecr=2162437886 WS=128
23	6.779498317	172.16.40.1	192.168.109.136	TCP	66	42464 → 42215 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2162437887 TSecr=2429736658
24	6.779517663	172.16.40.1	192.168.109.136	FTP	80	Request: retr pipe.txt
25	6.779964784	192.168.109.136	172.16.40.1	TCP	66	21 → 37198 [ACK] Seq=146 Ack=40 Win=65280 Len=0 TSval=2429736659 TSecr=2162437887
26	6.780098040	192.168.109.136	172.16.40.1	FTP	134	Response: 150 Opening BINARY mode data connection for pipe.txt (1863 bytes).
27	6.780441377	192.168.109.136	172.16.40.1	FTP-DA…	1929	FTP Data: 1863 bytes (PASV) (retr pipe.txt)
28	6.780452273	172.16.40.1	192.168.109.136	TCP	66	42464 → 42215 [ACK] Seq=1 Ack=1864 Win=63488 Len=0 TSval=2162437888 TSecr=2429736659
29	6.780455974	192.168.109.136	172.16.40.1	TCP	66	42215 → 42464 [FIN, ACK] Seq=1864 Ack=1 Win=65280 Len=0 TSval=2429736659 TSecr=2162437887
30	6.780601103	172.16.40.1	192.168.109.136	TCP	66	37198 → 21 [FIN, ACK] Seq=40 Ack=214 Win=64256 Len=0 TSval=2162437888 TSecr=2429736659
31	6.780613535	172.16.40.1	192.168.109.136	TCP	66	42464 → 42215 [FIN, ACK] Seq=1 Ack=1865 Win=64128 Len=0 TSval=2162437888 TSecr=2429736659
32	6.781127633	192.168.109.136	172.16.40.1	TCP	66	42215 → 42464 [ACK] Seq=1865 Ack=2 Win=65280 Len=0 TSval=2429736660 TSecr=2162437888
33	6.781179874	192.168.109.136	172.16.40.1	FTP	90	Response: 226 Transfer complete.
34	6.781195169	172.16.40.1	192.168.109.136	TCP	54	37198 → 21 [RST] Seq=41 Win=0 Len=0
35	6.781209347	192.168.109.136	172.16.40.1	TCP	66	21 → 37198 [FIN, ACK] Seq=238 Ack=41 Win=65280 Len=0 TSval=2429736660 TSecr=2162437888
36	6.781216261	172.16.40.1	192.168.109.136	TCP	54	37198 → 21 [RST] Seq=41 Win=0 Len=0