2° Projeto

Redes de Computadores

Lara Santos Bastos up202108740@up.pt
Lia Margarida Mota Sobral up202108741@up.pt

19 de Dezembro de 2022

Sumário

O segundo projeto da unidade curricular de Rede de Computadores tinha como principal objetivo a consolidação e aplicação prática dos conhecimentos lecionados nas aulas teóricas acerca de cliente FTP e configuração de uma rede.

Este relatório visa detalhar a implementação do projeto bem como responder a perguntas propostas sobre as várias fases da implementação. Consideramos que o projeto foi concluído com sucesso, uma vez que realizamos todas as tarefas atempadamente e o código realiza descarregar os ficheiros sem qualquer erro.

Índice

1. Introdução

Este projeto foi desenvolvido no âmbito da unidade curricular de Redes de Computadores e o seu objetivo principal passava por configurar uma rede de computadores para posterior utilização da aplicação de transferência de ficheiros usando de FTP (File Transfer Protocol) desenvolvido.

De modo a que ambas estas componentes sejam analisadas neste relatório, dividimo-lo da seguinte forma:

- **Parte 1 -** Aplicação de download de ficheiros
    - **Arquitetura da aplicação de download** - Descrição da arquitetura e estrutura da aplicação
    - **Resultados -** Resultados de um download bem sucedido
- **Parte 2 -** Configuração e análise da rede
    - **Configuração de um IP da rede**
    - **Implementação de duas *bridges* num *switch***
    - **Configuração de um router em Linux**
    - **Configuração de um router comercial e implementação da NAT**
    - **DNS**
    - **Ligações TCP**

Na secção Anexos, encontram-se todos os ficheiros necessários ao suporte do relatório, nomeadamente código da aplicação e logs de cada experiência.

2. Aplicação de *download*

2.1. Arquitetura da aplicação de download

Na primeira fase do projeto, começamos por desenvolver uma aplicação de download que utiliza o protocolo de transferências FTP. Esta é constituída por 3 ficheiros: "download.c", "parse.c" e "downloadApplication.c" .

No ficheiro principal "download.c" (anexo A.1) é chamado inicialmente a função parse (anexo A.3). Esta é responsável por ler e interpretar o URL fornecido como argumento pelo utilizador, com o objetivo de o decompor nos seus vários campos essenciais. Para este efeito, começamos por verificar se foram passados argumentos para login, ou seja, username e password. Em caso de ausência, são enviados os valores de username e password default. Todos os argumentos (username, password, path, host e IP) são armazenados na variável url da struct URL para poderem ser utilizados quando necessário. Para obter o IP, utilizamos a função "gethostbyname". Por fim, é verificado se a estrutura do URL e os valores obtidos a partir deste são válidos.

De seguida, é chamada a função "download" (anexo A.5), responsável por efetivamente descarregar o ficheiro. A função "createSocketConnection", é responsável por criar um socket através da porta 21, enquanto a função "login" realiza a autenticação com username e password fornecidas através do envio dos comandos "USER" e "PASS". Posteriormente, colocamos o servidor em modo passivo, com o comando "PASS" e utilizamos o IP obtido para criar um segundo socket, cuja responsabilidade é descarregar o ficheiro em questão. De forma a descarregar o ficheiro, é então enviado o comando "RETR" para que seja efetuada a obtenção dos dados.

Todos os comandos são enviados através da função "sendCommand" e a sua resposta é lida através de "readResponse". Após receber qualquer código de retorno, é sempre efetuada uma verificação para assegurar o sucesso da operação. Por fim são fechadas as conexões de ambos os sockets, sendo também o programa terminado.

O código está disponível no anexo A.

2.2. Resultados

De forma a garantir a viabilidade da aplicação, esta foi testada com vários ficheiros com diversas características desde diferentes tamanhos e extensões, bem como em modo anónimo e não anónimo.

É importante notar que foram também testados urls com erros de forma a garantir a integridade do programa.

```
(base) larabastos@Laras-MacBook-Air-4 proj2 % ./download ftp://rcom:rcom@netlab1.fe.up.pt/files/pic1.jpg

User: rcom
Password: rcom
Host: netlab1.fe.up.pt
Path: files/pic1.jpg
File name: pic1.jpg
IP Address : 192.168.109.136

220 Welcome to netlab-FTP server
331 Please specify the password.
230 Login successful.
227 Entering Passive Mode (192,168,109,136,184,8).
150 Opening BINARY mode data connection for files/pic1.jpg (340603 bytes).
226 Transfer complete.
```

Figura 1 - Funcionamento do da aplicação

Figura 2 - Ficheiro descarregado com a aplicação

3. Configuração e análise da rede

3.1. Configuração de um IP da rede

O objetivo da primeira experiência era a configuração do IP do computador tuxY3 e tuxY4 e criar uma ligação entre estes através de um switch. Após estas configurações, foi testada a conexão do tuxY3 para o tuxY4 e vice versa através do comando ping.

Foi configurado o IP 172.16.Y0.1/24 para o tuxY3 e o IP 172.16.Y0.254/24 no tuxY4.

3.1.1. Quais são os comandos necessários para configurar esta experiência?

Ver anexo B.1.

3.1.2. O que são pacotes ARP e para que são usados?

O ARP (Address Resolution Protocol) é um protocolo responsável pela resolução de endereços da camada de rede, os endereços IP, em endereços da camada de ligação de dados, endereços MAC. Aquando do envio de uma trama de um computador, o emissor verifica a sua tabela ARP à procura do endereço físico do recetor. Caso esta informação não exista, é enviado uma mensagem de “broadcast” a perguntar a quem pertence o IP de destino, e o destinatário envia uma pacote do mesmo protocolo com o seu endereço MAC. Esta associação é guardada na tabela ARP do emissor.

3.1.3. Quais são os endereços MAC e IP dos pacotes ARP e porque?

Os endereços IP e MAC do recetor. (Ver anexo na figura C.1)

3.1.4. Que pacotes o comando ping gera?

Após obter o endereço MAC, gera pacotes ICMP.

3.1.5. Quais são os endereços MAC e IP dos pacotes de ping?

O IP de origem do pacote de ping é o do emissor enquanto que o de chegada é o do recetor. O endereço MAC são os obtidos pelo protocolo ARP.

3.1.6. Como determinar se uma trama Ethernet recebida é ARP, IP ou ICMP?

Para distinguir os diferentes tipos de trama, é necessário avaliar o payload. O cabeçalho da trama contém um campo de 2 bytes que indica esta informação:

- **ARP**: 0x0806;
- **IP**: 0x0800;
- **ICMP:** o campo de protocolo no cabeçalho IP é 1.

No WireShark é ainda possível ver o tipo da trama uma vez que este está detalhado e são distinguidos por cores.

3.1.7. Como determinar o comprimento de uma trama recebida?

Esta informação está presente num campo no cabeçalho da trama.

3.1.8. O que é a interface de loopback e porque é importante?

A interface de loopback é um canal de comunicação que permite que qualquer mensagem transmitida seja recebida pelo canal que a enviou, ou seja, para sim mesmo. É essencial para confirmar que a ligação e a rede estão corretamente configuradas.

3.2. Implementação de duas bridges num switch

O objetivo da segunda experiência passava por criar duas LANs: uma com o tuxY3 e tuxY4 e outra com o tuxY2. A comunicação entre tuxY3 e tuxY4 manteve-se, no entanto, nenhum destes conseguiu comunicar com o tuxY2, uma vez que se encontravam em subredes diferentes.

3.2.1.Como configurar a bridge Y0?

Começamos por configurar o IP do tuxY2. De seguida criamos duas bridges a partir da consola do switch (Y0 e Y1) adicionar os tux às bridges pretendidas.

Os comandos utilizados encontram-se no anexo B.2.

3.2.2.Quantos domínios de transmissão existem? Como se pode concluí-lo através dos logs?

Conseguimos perceber que existem 2 domínios de broadcast, uma vez que o tuxY3 obteve resposta do tuxY4, enquanto o tuxY2 não obteve qualquer resposta. Concluimos que estes domínios correspondem às duas VLANs criadas.

Ver anexo C.2.

3.3. Configurar um router em Linux

Esta experiência tem como objetivo estabelecer a comunicação entre o tuxY3 e o tuxY2, que se encontram em LANs diferentes, utilizando o tuxY4 como um router dedicado para essa finalidade.

Aproveitando a infraestrutura de rede resultante da experiência 2, o procedimento inicia-se ligando o ether1 do tuxY4 ao switch e configurando o IP como 172.16.Y1.253/24. Em seguida, removemos a interface do switch previamente ligada à porta padrão do tuxY4, transferindo-a para a bridgeY1. Posteriormente, desativamos a opção ICMP echo-ignore-broadcast e ativamos o IP forwarding.

3.3.1. Quais são os comandos necessários para configurar esta experiência?

Ver comandos no anexo B.3.

3.3.2. Quais são as rotas que se encontram no tuxes? Qual o seu significado?

As rotas nos tuxes são adicionadas usando o comando route add. Nesta experiência, no tuxY2, é configurada uma rota para a rede 172.16.Y0.0/24, utilizando o tuxY4 como gateway, com o IP 172.16.Y1.253. Já no tuxY3, é configurada uma rota para a rede 172.16.Y1.0/24, também utilizando o tuxY4 como gateway, mas com o IP 172.16.Y0.254. Essas rotas são essenciais para direcionar o tráfego para as respetivas LANs usando o tuxY4 como router.

3.3.3. Qual é a informação que a tabela de encaminhamento contém?

A tabela de encaminhamento, pode ser acedida através do comando route -n, e exibe detalhes sobre as rotas configuradas. Cada entrada inclui informações como o destino da rota, a máscara de rede, o gateway (neste caso, o tuxY4), e a interface de saída.

3.3.4. Que mensagens ARP, e respetivos endereços MAC, são observadas e porquê?

Durante a comunicação entre tuxY3 e tuxY2 através do tuxY4, são observadas mensagens ARP para obter os endereços MAC correspondentes. O tuxY3 procura o endereço MAC da interface eth0 do tuxY4, enquanto o tuxY2 procura o endereço MAC da interface eth1

do tuxY4. Essas trocas de mensagens ARP são essenciais para construir as tabelas de ARP nos tuxes, permitindo a correta entrega dos pacotes.

3.3.5. Que pacotes ICMP são observados e porquê?

Pacotes ICMP de pedido e resposta são observados durante a execução do comando ping entre o tuxY3 e o tuxY2. Esses pacotes são cruciais para testar a conectividade entre os tuxes e para a resolução de endereços, garantindo que os tuxes conhecem e alcançam corretamente os outros dispositivos na rede.

3.3.6. Quais são os endereços IP e MAC associados com os pacotes ICMP e porquê?

Os pacotes ICMP mantêm os IPs e MACs de origem como os dos respetivos tuxes de origem. No entanto, o endereço MAC de destino é o do tuxY4, pois os pacotes passam por este para realizar o roteamento e redirecionamento de informações entre as duas redes.

3.4. Configurar um router comercial e implementar NAT

O objetivo desta experiência foi adicionar um router comercial com NAT implementado para permitir o acesso à internet.

3.4.1. Como configurar um router estático num router comercial?

Aproveitando a infraestrutura de redes da experiência 3, iniciamos conectando uma das entradas do router comercial à rede dos laboratórios e a outra ao switch. Adicionalmente, incorporamos esta interface na bridgeY1, da mesma maneira que nas experiências anteriores.

Para configurar o router, alteramos o cabo conectado com o S0 do tuxY2, da consola do switch para a consola do router comercial. Em seguida, configuramos os IPs correspondentes a cada interface usando o comando /ip address add: 172.16.S.Y9/24 para a interface ligada à rede do laboratório e 172.16.Y1.254/24 para a interface ligada à bridgeY1. Adicionamos rotas default nos vários tuxes para conectá-los ao router e uma rota no router para se ligar a cada uma das LANs através do tuxY4.

Todos os comandos podem ser observados no anexo B.4.

3.4.2. Quais são os caminhos que os pacotes seguem nos testes realizados e porquê?

Nos experimentos realizados, os pacotes seguem diferentes caminhos de acordo com as configurações. Ao remover a rota do tuxY2 para a rede 172.16.Y0.0/24 via tuxY4 e desativar os redirecionamentos do ICMP, os dados são encaminhados pelo router através da rota default até chegarem ao destino. Ao readicionar a rota e reativar os redirecionamentos, a ligação mais direta entre o tuxY2 e o tuxY3, utilizando apenas o tuxY4, exclui o router da rota dos pacotes de dados.

Além de observar o caminho dos dados através das capturas dos pacotes ARP e ICMP, podemos utilizar o comando traceroute -n 172.16.Y0.1 no tuxY2 para visualizar o caminho dos dados do tuxY2 para o tuxY3, anexo C.4. Com isso, concluímos que os pacotes ICMP são cruciais para otimizar as ligações na rede, implementando redirecionamentos quando benéficos.

3.4.3. Como configurar o NAT no router comercial?

Observar comandos no anexo B.4.

3.4.4. O que faz o NAT?

O NAT, ou Network Address Translation, tem como função principal traduzir os endereços IP privados utilizados numa rede local para um único endereço IP público visível na Internet. Dessa forma, o NAT possibilita a comunicação entre dispositivos internos e externos, preservando a limitada disponibilidade de endereços IPv4 públicos.

3.5. DNS

Aproveitando a infraestrutura de redes configurada ao longo das experiências anteriores, nas quais conectamos os vários tuxes à Internet, o objetivo desta experiência é configurar o DNS de forma a permitir o acesso a websites através dos seus nomes de domínio dentro da rede criada.

3.5.1. Como configurar o serviço DNS no host?

Para configurar o serviço DNS, apenas é necessário alterar o conteúdo do ficheiro /etc/resolv.conf em cada um dos tuxes, adicionando a linha 'nameserver 172.16.S.1'. Este endereço IP representa o router presente no laboratório que tem acesso à rede externa.

3.5.2. Que pacotes são trocados pelo DNS e qual a informação que estes contêm?

Os pacotes DNS contêm a informação do IP associado ao nome de domínio a ser traduzido. Sendo assim, estes pacotes são os primeiros a serem transmitidos, uma vez que o IP fornecido pelo DNS é necessário para todos os outros protocolos.

3.6. Ligações TCP

Nesta experiência, aproveitamos a infraestrutura de redes configurada nas etapas anteriores e utilizamos a aplicação de download desenvolvida na primeira parte do projeto para analisar o envio e receção de pacotes, a sua constituição interna e os mecanismos de controlo de fluxo e congestionamento das ligações TCP. Iniciamos a experiência compilando o downloader no tuxY3 e realizando o download de um ficheiro.

3.6.1. Quantas ligações TCP são abertas pela aplicação FTP?

A aplicação FTP estabelece duas ligações TCP: uma para informação de controlo (FTP control connection) e outra para a transferência de dados (FTP data connection).

3.6.2. Em que ligação é transportada a informação de controlo FTP?

A informação de controlo FTP é transportada na ligação TCP designada como FTP control connection.

3.6.3. Quais são as fases da ligação TCP?

A ligação TCP passa por três fases diferentes:

- Estabelecimento da ligação (handshake): Utiliza pacotes SYN-ACK
- Transferência de dados: Utiliza pacotes de dados, como os pacotes FTP Data com números de sequência incrementais.
- Encerramento da ligação: Envolvendo pacotes FIN-ACK.

3.6.4. Como é que funciona o mecanismo ARQ TCP? Quais são os campos TCP relevantes? Que informação relevante pode ser observada nos logs?

O mecanismo ARQ (Automatic Repeat reQuest) no TCP é usado para o controlo de erros. Mensagens ACK (acknowledge) indicam a correta receção do pacote, e timeouts são empregues para averiguar o tempo de receção do mesmo. Se um timeout é ultrapassado, o pacote é considerado perdido e é retransmitido pela rede. Nos logs, podemos observar números sequenciais, ACKs e o fluxo de mensagens ACK, que são indicadores do funcionamento do ARQ.

3.6.5. Como é que funciona o mecanismo TCP de controlo de congestionamento? Quais são os campos relevantes? Como é que evoluiu a capacidade de transmissão ao longo do tempo? Vai de acordo com o mecanismo TCP de controlo de congestionamento?

O controlo de congestionamento, executado pelos emissores de dados, baseia-se no método Selective Repeat, que envia pacotes pela rede sem aguardar pelos respetivos ACK. Uma rede é considerada congestionada quando ocorrem perdas de pacotes. Para determinar o limiar da rede durante a transferência, o emissor transmite vários pacotes de uma vez, seguindo uma de duas abordagens: Additive Increase (na transferência seguinte, envia um pacote adicional em relação à anterior) ou Slow Start (semelhante à abordagem anterior, mas com incrementos exponenciais na base dois). A perda de um pacote pode ser detetada através de timeout ou de três ACK consecutivos, resultando numa drástica redução no número de pacotes transferidos na transmissão subsequente, aproximando-se da metade do valor que desencadeou a perda. Na análise gráfica da velocidade dos pacotes em função do tempo, observamos a influência do controlo de congestionamento. Após períodos de aumento de pacotes (Additive Increase ou Slow Start), há uma redução para cerca de metade do valor, refletindo o ajuste do controlo de congestionamento.

3.6.5. A capacidade de transmissão da TCP data connection é alterada pelo aparecimento de uma segunda ligação TCP? Porquê?

Sim, a capacidade de transmissão da TCP data connection é afetada pelo aparecimento de uma segunda ligação TCP. Para testarmos este caso, iniciamos o download de um ficheiro no tuxY3 e, logo de seguida, outro no tuxY2, observando que quando a segunda ligação é iniciada, ocorre uma redução para cerca de metade da capacidade de transmissão, evidenciando a atuação do controlo de congestionamento durante a transferência mútua.

4. Conclusão

Após o término do projeto, consideramos que este foi concluído com sucesso. Tanto a implementação da aplicação de download, como a configuração de rede de foram concluídas atempada e corretamente e permitiram aplicar de forma prática os conhecimentos acerca do protocolo FTP, programação de socket e configuração de switches e routers.

Por fim, a análise dos logs e posterior realização deste relatório permitiu a consolidação dos conhecimentos acerca de redes de computadores lecionadas nas aulas teóricas.

5. Anexos

A. Código da aplicação de download

A.1. download.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include "downloadApplication.h"

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

if (argc != 2){
fprintf(stderr, "usage: download ftp::///[<user>:<password>@]<host>/<url-path>\n");
exit(1);
}

URL url;

if (parse(argv[1], &url) < 0){
fprintf(stderr, "Parsing url error\n");
exit(1);
}

if(download(&url) < 0){
fprintf(stderr, "Downloading file error\n");
exit(1);
}

return 0;
}
```

A.2. parse.h

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PASSWORD_DEFAULT "password"
#define ANONYMOUS_USER "anonymous"

typedef struct{
char user[256];
char password[256];
char path[1024];
char filename[512];
char ip[20];
char host[512];
} URL;

int parse(char * url, URL * parsed);
```

A.3. parse.c

```c
#include "parse.h"

int parse(char * url, URL * parsed){

char * ftp = strtok(url, "/");
char * loginargs = strtok(NULL, "/");
char * path = strtok(NULL, "");

if (ftp == NULL || loginargs == NULL || path == NULL){
fprintf(stderr, "Invalid URL\n");
return -1;
}


if (strchr(loginargs, '@') != NULL){

char * login = strtok(loginargs, "@");
char * host = strtok(NULL, "@");
char * user = strtok(login, ":");
char * password = strtok(NULL, ":");

strcpy(parsed->user, user);

if(password){

strcpy(parsed->user, user);
strcpy(parsed->password, password);
strcpy(parsed->host, host);

}
else{

strcpy(parsed->user, user);
strcpy(parsed->password, PASSWORD_DEFAULT);
strcpy(parsed->host, host);
}

}

else{
```

```
strcpy(parsed->user, ANONYMOUS_USER);
strcpy(parsed->password, PASSWORD_DEFAULT);
strcpy(parsed->host, loginargs);

}

strcpy(parsed->path, path);
char *lastSlash = strrchr(parsed->path, '/');

if (lastSlash != NULL) {

strcpy(parsed->filename, lastSlash + 1);
}
else {

strcpy(parsed->filename, parsed->path);
}

if (strlen(parsed->host) == 0 || strlen(parsed->path) == 0){
fprintf(stderr, "Invalid URL\n");
return -1;
}

//dns
struct hostent *host;
if ((host = gethostbyname(parsed->host)) == NULL){
fprintf(stderr, "Invalid hostname\n");
return -1;
}
//h_addr: points to list of network addresses.
//in_addr: represent an IPv4 address.
//IPv4 address binary to a string
strcpy(parsed->ip, inet_ntoa(*((struct in_addr *) host->h_addr)));
printf("\nUser: %s\n", parsed->user);
printf("Password: %s\n", parsed->password);
printf("Host: %s\n", parsed->host);
printf("Path: %s\n", parsed->path);
printf("File name: %s\n", parsed->filename);
printf("IP Address : %s\n\n", parsed->ip);
return 0;
}
```

A.4. downloadApplication.h

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <fcntl.h>

#include "parse.h"

int download(URL * urlArgs);
int createSocketConnection(char * ip, int port);
int readResponse(int socketfd, char * response, int * code);
int sendCommand(int flag, char * command, int socketfd, char * arguments);
int login(int socketfd, URL * urlArgs);
int passive(int socketfd, char * response_ip, int * response_port);
```

A.5. downloadApplication.h

```
#include "downloadApplication.h"

int download(URL * url){
int socketfd=-1;
char answer[1024];
int code = 0;
int port;
char ip[20];
// established to the FTP server
socketfd = createSocketConnection(url->ip, 21);
if ( socketfd < 0){
fprintf(stderr, "Initializing socket error\n");
return -1;
}

if (readResponse(socketfd, answer, &code) != 0) return -1;

if (code != 220) {
fprintf(stderr, "Response code error\n");
return -1;
}

if (login(socketfd, url) < 0) {
fprintf(stderr, "Login error\n");
return -1;
}

//created using the IP and port obtained in passive mode
if (passive(socketfd, ip, &port) < 0) {
fprintf(stderr, "Setting passive mode error\n");
return -1;
}

int fd_1 = createSocketConnection(ip, port);
if( fd_1 < 0 ){
fprintf(stderr, "Data connection error\n");
return -1;
}

if (sendCommand(1, "RETR", socketfd, url->path) < 0){
```

```
fprintf(stderr, "Sending RETR command error\n");
return -1;
}

code = 0;
memset(answer, 0, sizeof(answer));

if (readResponse(socketfd, answer, &code) != 0) return -1;
if (code != 150) {
fprintf(stderr, "Response code error\n");
return -1;
}


int fd_2 = open(url->filename,O_WRONLY | O_CREAT, 0777);
if (fd_2 < 0) {
fprintf(stderr, "Opening file error\n");
return -1;
}

char buf[1024];
int bytesRead;

do{
bytesRead = read(fd_1, buf, 1024);
if (write(fd_2,buf,bytesRead)<0) return -1;
}while(bytesRead);

if (close(fd_2) < 0) {
fprintf(stderr, "Error closing file!\n");
return -1;
}

code = 0;
memset(answer, 0, sizeof(answer));

if (readResponse(socketfd, answer, &code) != 0) return -1;
if (code != 226) {
fprintf(stderr, "Response code error\n");
return -1;
}
```

```
if ((close(socketfd) || close(fd_1))!=0){
fprintf(stderr,"Closing socket error\n");
return -1;
}

return 0;
}

int createSocketConnection(char * ip, int port){

int sockfd;
struct sockaddr_in address;

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

if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
fprintf(stderr, "Opening socket error\n");
return -1;
}


if(connect(sockfd, (struct sockaddr *) &address, sizeof(address)) < 0){
fprintf(stderr, "Server connection error\n");
return -1;
}

return sockfd;
}

int readResponse(int socketfd, char * answer, int * code){

FILE * socket = fdopen(socketfd, "r");
char * buf=NULL;

size_t readBytes = 0;


do {
if (getline(&buf, &readBytes, socket) <= 0) {
```

```
break;
}

strncat(answer, buf, readBytes - 1);

printf("%s", buf);

if (buf[3] == ' ') {
sscanf(buf, "%d", code);
return 0;
}
} while (1);

return 1;
}

int sendCommand(int flag, char * command, int socketfd, char * arguments){
char cmd[256];
strcpy(cmd, command);

if (flag){
strcat(cmd, " ");
strcat(cmd, arguments);
}

strcat(cmd, "\r\n");

if ((write(socketfd, cmd, strlen(cmd)) != strlen(cmd))){
fprintf(stderr, "Writing message error\n");
return -1;
}

return 0;
}

int login(int socketfd, URL * url){
if (sendCommand(1, "USER", socketfd, url->user) < 0){
fprintf(stderr, "Sending USER command error\n");
return -1;
}

char answer[1024];
```

```
int code = 0;

memset(answer, 0, sizeof(answer));
code = 0;

if (readResponse(socketfd, answer, &code) != 0) return -1;
if (code != 331 && code != 230) {
fprintf(stderr, "Response code error\n");
return -1;
}


if (sendCommand(1, "PASS", socketfd, url->password) < 0){
fprintf(stderr, "Sending PASS command error\n");
return -1;
}

memset(answer, 0, sizeof(answer));
code = 0;

if (readResponse(socketfd, answer, &code) != 0) return -1;
if (code != 230) {
fprintf(stderr, "Response code error\n");
return -1;
}

return 0;
}

int passive(int socketfd, char * response_ip, int * response_port){

char answer[1024];
int code = 0;

if (sendCommand(0, "PASV", socketfd, NULL) < 0){
fprintf(stderr, "Sending PASV command error\n");
return -1;
}

if (readResponse(socketfd, answer, &code) != 0) return -1;

if (code != 227) {
```

```
fprintf(stderr, "Response code error\n");
return -1;
}

int ip[4], port[2];

strtok(answer, "(");
char * arguments = strtok(NULL, ")");
sscanf(arguments, "%d, %d, %d, %d, %d, %d", &ip[0], &ip[1], &ip[2], &ip[3], &port[0],
&port[1]);
sprintf(response_ip, "%d.%d.%d.%d", ip[0], ip[1], ip[2], ip[3]);
*response_port = port[0] * 256 + port[1];

return 0;
}
```

B. Comandos para configuração

B.1. Configuração de um IP da rede

- tuxY3:
    - ifconfig eth0 up
    - ifconfig eth0 172.16.Y0.1/24
- tuxY4:
    - ifconfig eth0 up
    - ifconfig eth0 172.16.Y0.254/24
- tuxY3:
    - ping 172.16.Y0.254
- tuxY4
    - ping 172.16.Y0.1
- tuxY3
    - arp -d 172.16.Y0.254/24

B.2. Implementação de duas bridges num switch

- Switch Console:
    - /system reset-configuration
- tuxY2:
    - ifconfig eth0 up
    - ifconfig eth0 172.16.Y1.1/24
- Switch Console:
    - /interface bridge add name=bridgeY0
    - /interface bridge add name=bridgeY1
    - /interface bridge port remove [find interface=ether2]
    - /interface bridge port remove [find interface=ether3]
    - /interface bridge port remove [find interface=ether4]
    - /interface bridge port add bridge=bridgeY0 interface=ether2
    - /interface bridge port add bridge=bridgeY0 interface=ether3
    - /interface bridge port add bridge=bridgeY1 interface=ether4
- tuxY3:
    - ping 172.16.Y0.254
    - ping 172.16.Y1.1
    - ping -b 172.16.Y0.255
- tuxY2:

    - ping 172.16.Y0.1
- tuxY3
    - ping -b 172.16.Y1.255

B.3. Configurar um router em Linux

- tuxY4:
    - ifconfig eth1 up
    - ifconfig eth1 172.16.Y1.253/24
- Switch console:
    - /interface bridge port remove [find interface=ether5]
    - /interface bridge port add bridge=bridgeY1 interface=ether5
- tuxY4:
    - sysctl -w net.ipv4.ip_forward=1
    - sysctl -w net.ipv4.icmp_echo_ignore_broadcasts=0
- tuxY2:
    - route add -net 172.16.Y0.0/24 gw 172.16.Y1.253
- tuxY3:
    - route add -net 172.16.Y1.0/24 gw 172.16.Y0.254
    - ping 172.16.Y0.254
    - ping 172.16.Y1.253
    - ping 172.16.Y1.1
- tuxY2:
    - arp -d 172.16.Y1.253
- tuxY3:
    - arp -d 172.16.Y0.254
- tuxY4:
    - arp -d 172.16.Y0.1
    - arp -d 172.16.Y1.1

B.4. Configurar um router comercial e implementar NAT

- Switch console:
    - /interface bridge port remove [find interface=ether6]
    - /interface bridge port add bridge=bridgeY1 interface=ether6
- Router console:
    - /system reset-configuration

        - /ip address add address=172.16.S.Y9/24 interface=ether1
        - /ip address add address=172.16.Y1.254/24 interface=ether2
        - /ip route add dst-address=172.16.Y0.0/24 gateway=172.16.Y1.253
        - /ip route add dst-address=0.0.0.0/0 gateway=172.16.S.254
- tuxY2:
    - route add default gw 172.16.Y1.254
- tuxY3:
    - route add default gw 172.16.Y0.254
- tuxY4:
    - route add default gw 172.16.Y1.254
- tuxY3:
    - ping 172.16.Y0.254
    - ping 172.16.Y1.1
    - ping 172.16.Y1.254
- tuxY2:
    - sysctl -w net.ipv4.conf.eth0.accept_redirects=0
    - sysctl -w net.ipv4.conf.all.accept_redirects=0
    - route del -net 172.16.Y0.0 gw 172.16.Y1.253 netmask 255.255.255.0
    - ping 172.16.Y0.1
    - traceroute -n 172.16.Y0.1
    - route add -net 172.16.Y0.0/24 gw 172.16.Y1.253
    - traceroute -n 172.16.Y0.1
    - sysctl -w net.ipv4.conf.eth0.accept_redirects=1
    - sysctl -w net.ipv4.conf.all.accept_redirects=1
- tuxY3:
    - ping 172.16.S.254
- Router console:
- /ip firewall nat disable 0
- tuxY3:
    - ping 172.16.S.254 (deixa de responder)
- Router console:
    - /ip firewall nat enable 0

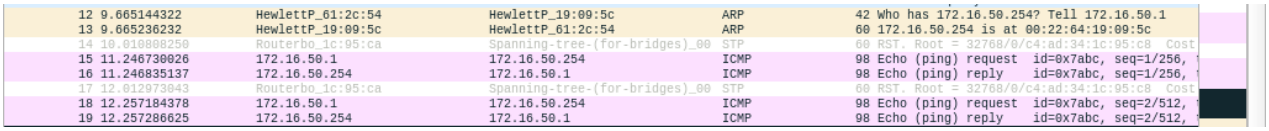
B.5. DNS

- tuxY2, tuxY3, tuxY4:
    - sudo nano /etc/resolv.conf
    - nameserver 172.16.S.1
    - ping google.com

C. Logs capturados

C.1. Configuração de um IP da rede

Ping do tuxY3 para tux Y4:



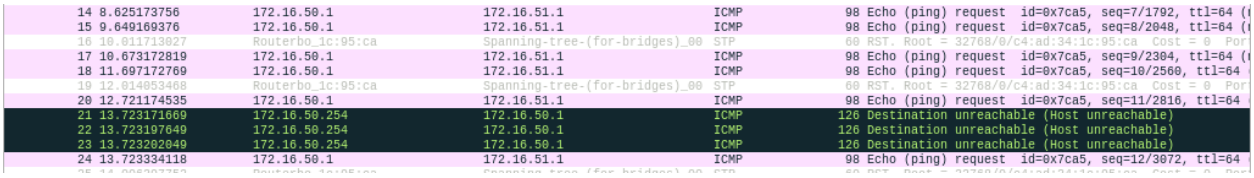
12	9.665144322	HewlettP_61:2c:54	HewlettP_19:09:5c	ARP	42 Who has 172.16.50.254? Tell 172.16.50.1
13	9.665236232	HewlettP_19:09:5c	HewlettP_61:2c:54	ARP	60 172.16.50.254 is at 00:22:64:19:09:5c
14	10.010808250	Routerbo_1c:95:ca	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:95:c8 Cost
15	11.246730026	172.16.50.1	172.16.50.254	ICMP	98 Echo (ping) request id=0x7abc, seq=1/256,
16	11.246835137	172.16.50.254	172.16.50.1	ICMP	98 Echo (ping) reply id=0x7abc, seq=1/256,
17	12.012973043	Routerbo_1c:95:ca	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:95:c8 Cost
18	12.257184378	172.16.50.1	172.16.50.254	ICMP	98 Echo (ping) request id=0x7abc, seq=2/512,
19	12.257286625	172.16.50.254	172.16.50.1	ICMP	98 Echo (ping) reply id=0x7abc, seq=2/512,

C.2. Implementação de duas bridges num switch

Ping do tuxY3 para tux Y4:

6	4.354073114	172.16.50.1	172.16.50.254	ICMP	98 Echo (ping) request id=0x7c79, seq=2/512, ttl=64 (r
7	4.354202738	172.16.50.254	172.16.50.1	ICMP	98 Echo (ping) reply id=0x7c79, seq=2/512, ttl=64 (r
8	5.378073692	172.16.50.1	172.16.50.254	ICMP	98 Echo (ping) request id=0x7c79, seq=3/768, ttl=64 (r
9	5.378205831	172.16.50.254	172.16.50.1	ICMP	98 Echo (ping) reply id=0x7c79, seq=3/768, ttl=64 (r

Ping do tuxY3 para tuxY2:

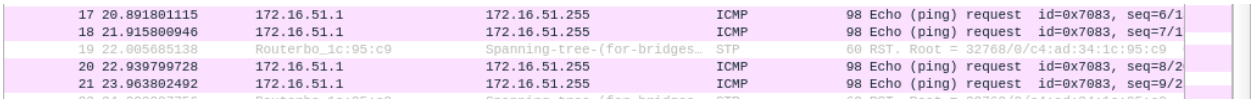


14	8.625173756	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) request id=0x7ca5, seq=7/1792, ttl=64 (
15	9.649169376	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) request id=0x7ca5, seq=8/2048, ttl=64 (
16	10.011713027	Routerbo_1c:95:ca	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Por
17	10.673172819	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) request id=0x7ca5, seq=9/2304, ttl=64 (
18	11.697172769	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) request id=0x7ca5, seq=10/2560, ttl=64
19	12.014053468	Routerbo_1c:95:ca	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Por
20	12.721174535	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) request id=0x7ca5, seq=11/2816, ttl=64
21	13.723171669	172.16.50.254	172.16.50.1	ICMP	126 Destination unreachable (Host unreachable)
22	13.723197649	172.16.50.254	172.16.50.1	ICMP	126 Destination unreachable (Host unreachable)
23	13.723202049	172.16.50.254	172.16.50.1	ICMP	126 Destination unreachable (Host unreachable)
24	13.723334118	172.16.50.1	172.16.51.1	ICMP	98 Echo (ping) request id=0x7ca5, seq=12/3072, ttl=64

Ping broadcast tuxY3:

11	8.009334733	Routerbo_1c:95:ca	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Por
12	8.668238426	172.16.50.1	172.16.50.255	ICMP	98 Echo (ping) request id=0x7cfc, seq=4/1024, ttl=64 (
13	8.668379155	172.16.50.254	172.16.50.1	ICMP	98 Echo (ping) reply id=0x7cfc, seq=4/1024, ttl=64
14	9.692264496	172.16.50.1	172.16.50.255	ICMP	98 Echo (ping) request id=0x7cfc, seq=5/1280, ttl=64 (
15	9.692420800	172.16.50.254	172.16.50.1	ICMP	98 Echo (ping) reply id=0x7cfc, seq=5/1280, ttl=64

Ping broadcast tuxY2:



17	20.891801115	172.16.51.1	172.16.51.255	ICMP	98 Echo (ping) request id=0x7083, seq=6/1
18	21.915800946	172.16.51.1	172.16.51.255	ICMP	98 Echo (ping) request id=0x7083, seq=7/1
19	22.005685138	Routerbo_1c:95:c9	Spanning-tree-(for-bridges…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:95:c9
20	22.939799728	172.16.51.1	172.16.51.255	ICMP	98 Echo (ping) request id=0x7083, seq=8/2
21	23.963802492	172.16.51.1	172.16.51.255	ICMP	98 Echo (ping) request id=0x7083, seq=9/2

C.3. Configuração de um router em Linux

Ping do tuxY3 para tuxY2:

15	7.999263147	Routerbo_1c:95:ca	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Port = 0x80
16	8.795100451	HewlettP_61:2c:54	HewlettP_19:09:5c	ARP	42	Who has 172.16.50.254? Tell 172.16.50.1
17	8.795226793	HewlettP_19:09:5c	HewlettP_61:2c:54	ARP	60	172.16.50.254 is at 00:22:64:19:09:5c
18	8.859125494	172.16.50.1	172.16.51.1	ICMP	98	Echo (ping) request id=0x7ed8, seq=6/1536, ttl=64 (reply in
19	8.859353943	172.16.51.1	172.16.50.1	ICMP	98	Echo (ping) reply id=0x7ed8, seq=6/1536, ttl=63 (request

Ping do tuxY3 para o IP 172.16.Y1.253:

17	11.993872164	Routerbo_1c:95:ca	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Por
18	12.400148091	HewlettP_61:2c:54	HewlettP_19:09:5c	ARP	42	Who has 172.16.50.254? Tell 172.16.50.1
19	12.400284210	HewlettP_19:09:5c	HewlettP_61:2c:54	ARP	60	172.16.50.254 is at 00:22:64:19:09:5c
20	12.460169168	172.16.50.1	172.16.51.253	ICMP	98	Echo (ping) request id=0x7eb1, seq=6/1536, ttl=64 (
21	12.460305986	172.16.51.253	172.16.50.1	ICMP	98	Echo (ping) reply id=0x7eb1, seq=6/1536, ttl=64 (

Ping do tuxY3 para o IP 172.16.Y1.253:

22	13.082058838	HewlettP_19:09:5c	HewlettP_61:2c:54	ARP	60	Who has 172.16.50.1? Tell 172.16.50.254
23	13.082078393	HewlettP_61:2c:54	HewlettP_19:09:5c	ARP	42	172.16.50.1 is at 00:21:5a:61:2c:54
24	14.016172370	Routerbo_1c:95:ca	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 32768/0/c4:ad:34:1c:95:ca Cost = 0 Por
25	14.064220521	172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) request id=0x7e85, seq=7/1792, ttl=64 (
26	14.064351961	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x7e85, seq=7/1792, ttl=64 (

C.4. Configuração de um router comercial e implementação da NAT

Ping do tuxY3 para o tuxY4 :

9	10.709324444	172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) request id=0x0228, seq=2/512, ttl=64 (r
10	10.709457351	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x0228, seq=2/512, ttl=64 (r
11	11.733325512	172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) request id=0x0228, seq=3/768, ttl=64 (r
12	11.733460304	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x0228, seq=3/768, ttl=64 (r

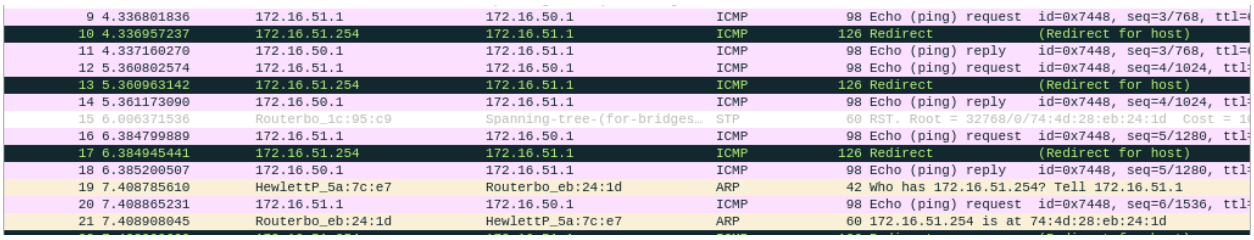
Ping do tuxY3 para o tuxY2 :

14	8.337479404	172.16.50.1	172.16.51.1	ICMP	98	Echo (ping) request id=0x0253, seq=5/1280, ttl=64 (
15	8.337721891	172.16.51.1	172.16.50.1	ICMP	98	Echo (ping) reply id=0x0253, seq=5/1280, ttl=63 (
16	9.361492973	172.16.50.1	172.16.51.1	ICMP	98	Echo (ping) request id=0x0253, seq=6/1536, ttl=64 (
17	9.361782323	172.16.51.1	172.16.50.1	ICMP	98	Echo (ping) reply id=0x0253, seq=6/1536, ttl=63 (

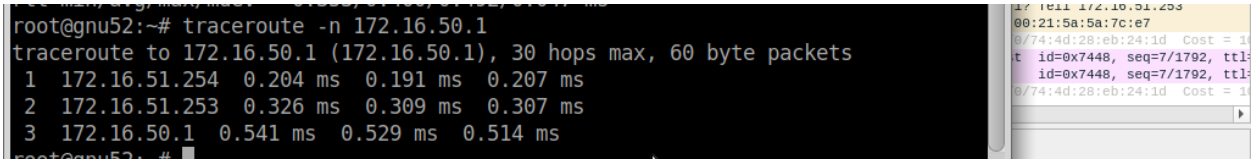
Ping do tuxY3 para o router :

10	8.985993038	172.16.50.1	172.16.51.254	ICMP	98	Echo (ping) request id=0x0299, seq=3/768, ttl=64 (r
11	8.986273867	172.16.51.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x0299, seq=3/768, ttl=63 (r
12	10.009995292	172.16.50.1	172.16.51.254	ICMP	98	Echo (ping) request id=0x0299, seq=4/1024, ttl=64 (
13	10.010256357	172.16.51.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x0299, seq=4/1024, ttl=63 (

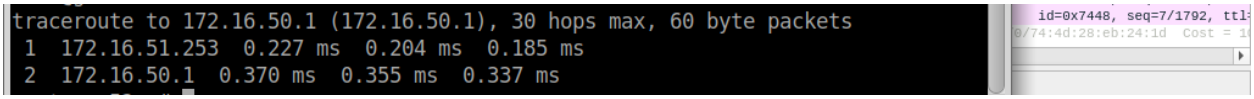
Ping do tuxY2 para tuxY3 com redirecionamento :



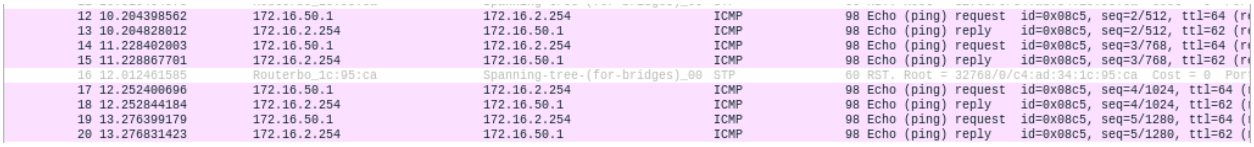
Traceroute do tuxY2 até a 172.16.Y0.1 antes de adicionar a rota :



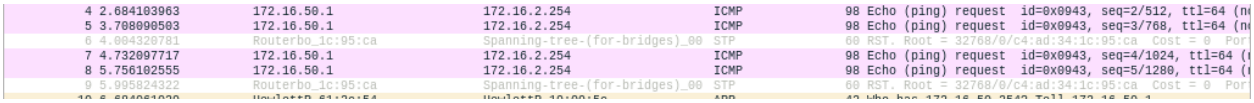
Traceroute do tuxY2 até a 172.16.Y0.1 depois de adicionar a rota :



Ping do tuxY3 para a Internet :

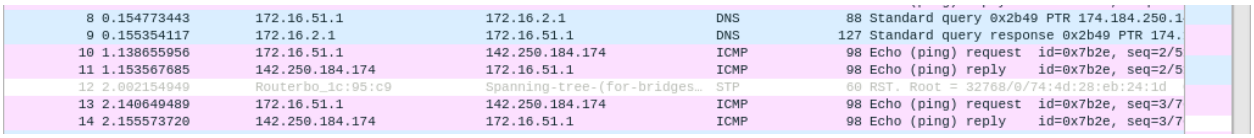


Ping do tuxY3 para a Internet sem NAT:

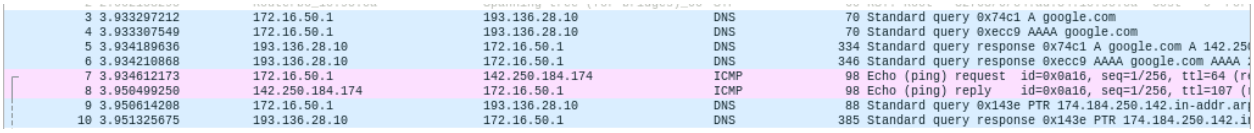


C.5. DNS

Ping do tuxY2 para a Google (8.8.8.8):



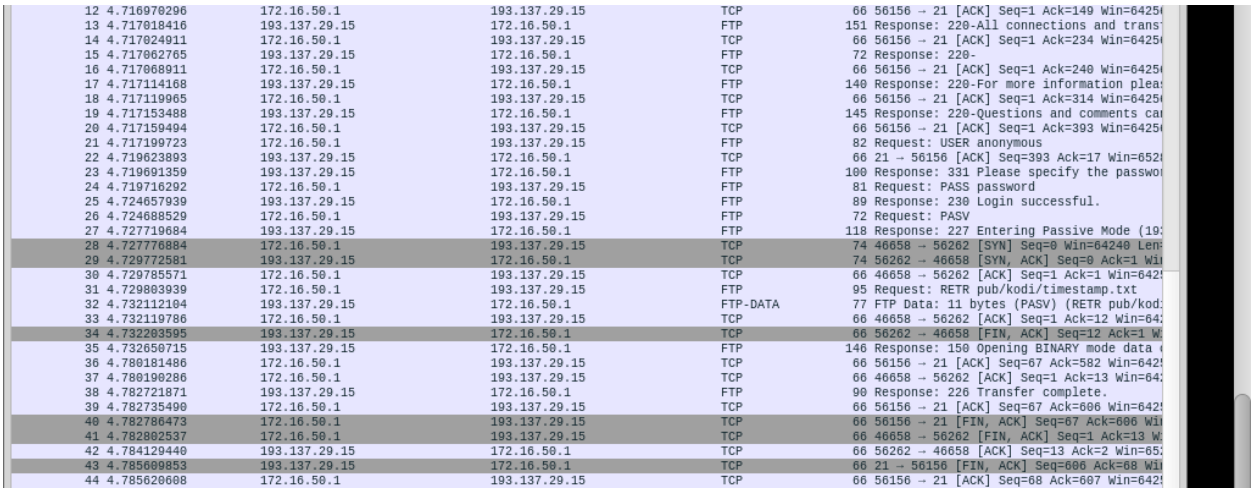
Ping do tuxY3 para a Google (8.8.8.8):



3	3.933297212	172.16.50.1	193.136.28.10	DNS	70 Standard query 0x74c1 A google.com
4	3.933307549	172.16.50.1	193.136.28.10	DNS	70 Standard query 0xecc9 AAAA google.com
5	3.934189636	193.136.28.10	172.16.50.1	DNS	334 Standard query response 0x74c1 A google.com A 142.25
6	3.934210868	193.136.28.10	172.16.50.1	DNS	346 Standard query response 0xecc9 AAAA google.com AAAA
7	3.934612173	172.16.50.1	142.250.184.174	ICMP	98 Echo (ping) request id=0x0a16, seq=1/256, ttl=64 (r
8	3.950499250	142.250.184.174	172.16.50.1	ICMP	98 Echo (ping) reply id=0x0a16, seq=1/256, ttl=107 (
9	3.950614208	172.16.50.1	193.136.28.10	DNS	88 Standard query 0x143e PTR 174.184.250.142.in-addr.ar
10	3.951325675	193.136.28.10	172.16.50.1	DNS	385 Standard query response 0x143e PTR 174.184.250.142.i

C.6. Ligações TCP

Download de um ficheiro usando a aplicação desenvolvida



12	4.716970296	172.16.50.1	193.137.29.15	TCP	66 56156 → 21 [ACK] Seq=1 Ack=149 Win=6425
13	4.717018416	193.137.29.15	172.16.50.1	FTP	151 Response: 220-All connections and trans
14	4.717024911	172.16.50.1	193.137.29.15	TCP	66 56156 → 21 [ACK] Seq=1 Ack=234 Win=6425
15	4.717062765	193.137.29.15	172.16.50.1	FTP	72 Response: 220-
16	4.717068911	172.16.50.1	193.137.29.15	TCP	66 56156 → 21 [ACK] Seq=1 Ack=240 Win=6425
17	4.717114168	193.137.29.15	172.16.50.1	FTP	140 Response: 220-For more information plea
18	4.717119965	172.16.50.1	193.137.29.15	TCP	66 56156 → 21 [ACK] Seq=1 Ack=314 Win=6425
19	4.717153488	193.137.29.15	172.16.50.1	FTP	145 Response: 220-Questions and comments ca
20	4.717159494	172.16.50.1	193.137.29.15	TCP	66 56156 → 21 [ACK] Seq=1 Ack=393 Win=6425
21	4.717199723	172.16.50.1	193.137.29.15	FTP	82 Request: USER anonymous
22	4.719623893	193.137.29.15	172.16.50.1	TCP	66 21 → 56156 [ACK] Seq=393 Ack=17 Win=652
23	4.719691359	193.137.29.15	172.16.50.1	FTP	100 Response: 331 Please specify the passwo
24	4.719716292	172.16.50.1	193.137.29.15	FTP	81 Request: PASS password
25	4.724657939	193.137.29.15	172.16.50.1	FTP	89 Response: 230 Login successful.
26	4.724688529	172.16.50.1	193.137.29.15	FTP	72 Request: PASV
27	4.727719684	193.137.29.15	172.16.50.1	FTP	118 Response: 227 Entering Passive Mode (19
28	4.727776884	172.16.50.1	193.137.29.15	TCP	74 46658 → 56262 [SYN] Seq=0 Win=64240 Len
29	4.729772581	193.137.29.15	172.16.50.1	TCP	74 56262 → 46658 [SYN, ACK] Seq=0 Ack=1 Wi
30	4.729785571	172.16.50.1	193.137.29.15	TCP	66 46658 → 56262 [ACK] Seq=1 Ack=1 Win=642
31	4.729803939	172.16.50.1	193.137.29.15	FTP	95 Request: RETR pub/kodi/timestamp.txt
32	4.732112104	193.137.29.15	172.16.50.1	FTP-DATA	77 FTP Data: 11 bytes (PASV) (RETR pub/kod
33	4.732119786	172.16.50.1	193.137.29.15	TCP	66 46658 → 56262 [ACK] Seq=1 Ack=12 Win=64
34	4.732203595	193.137.29.15	172.16.50.1	TCP	66 56262 → 46658 [FIN, ACK] Seq=12 Ack=1 W
35	4.732650715	193.137.29.15	172.16.50.1	FTP	146 Response: 150 Opening BINARY mode data
36	4.780181486	172.16.50.1	193.137.29.15	TCP	66 56156 → 21 [ACK] Seq=67 Ack=582 Win=642
37	4.780190286	172.16.50.1	193.137.29.15	TCP	66 46658 → 56262 [ACK] Seq=1 Ack=13 Win=64
38	4.782721871	193.137.29.15	172.16.50.1	FTP	90 Response: 226 Transfer complete.
39	4.782735490	172.16.50.1	193.137.29.15	TCP	66 56156 → 21 [ACK] Seq=67 Ack=606 Win=642
40	4.782786473	172.16.50.1	193.137.29.15	TCP	66 56156 → 21 [FIN, ACK] Seq=67 Ack=606 Wi
41	4.782802537	172.16.50.1	193.137.29.15	TCP	66 46658 → 56262 [FIN, ACK] Seq=1 Ack=13 W
42	4.784129440	193.137.29.15	172.16.50.1	TCP	66 56262 → 46658 [ACK] Seq=13 Ack=2 Win=65
43	4.785609853	193.137.29.15	172.16.50.1	TCP	66 21 → 56156 [FIN, ACK] Seq=606 Ack=68 Wi
44	4.785620608	172.16.50.1	193.137.29.15	TCP	66 56156 → 21 [ACK] Seq=68 Ack=607 Win=642