

# Lab2 - Computer Networks



Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

**Turma 6:**

Duarte Miguel de Novo Faria - 201607176

Maria Teresa Queiroz Machado Urbano Ferreira - 201603811

Maria João Senra Viana - 201604751

Faculdade de Engenharia da Universidade do Porto

23 de Dezembro de 2018

# Conteúdo

<b>1</b>	<b>Sumário</b>	<b>3</b>
<b>2</b>	<b>Introdução</b>	<b>3</b>
<b>3</b>	<b>Download Application</b>	<b>3</b>
3.1	Arquitetura . . . . .	3
3.1.1	Parser . . . . .	3
3.1.2	Client . . . . .	4
3.2	Resultados . . . . .	4
<b>4</b>	<b>Configuração e estudo da rede</b>	<b>5</b>
4.1	Configurar uma rede IP . . . . .	5
4.2	Implementar duas LAN's virtuais no switch . . . . .	5
4.2.1	Como se configura a VLANy0? . . . . .	5
4.3	Configurar um router em Linux . . . . .	6
4.4	DNS . . . . .	7
4.5	Ligações TCP . . . . .	7
<b>5</b>	<b>Conclusão</b>	<b>8</b>
<b>A</b>	<b>Código fonte</b>	<b>9</b>
A.1	parser.h . . . . .	9
A.2	parser.c . . . . .	10
A.3	client.h . . . . .	12
A.4	client.c . . . . .	13
A.5	main.c . . . . .	16
A.6	makefile . . . . .	17
<b>B</b>	<b>Anexos:</b>	<b>18</b>
B.1	Arquitetura das experiências: . . . . .	18
B.2	Logs das Experiências . . . . .	19
B.2.1	Exp. 1 . . . . .	19
B.2.2	Exp. 2 . . . . .	19
B.2.3	Exp. 3 . . . . .	20
B.2.4	Exp. 4 . . . . .	20
B.2.5	Exp. 5 . . . . .	20
B.2.6	Exp. 6 . . . . .	20

# 1 Sumário

Este projeto teve como objetivo o desenvolvimento de uma aplicação de download FTP e a configuração e estudo de uma rede. Assim, foram realizadas diversos experimentos na rede de acordo com o abordado nas aulas teóricas (Mac Sublayer, Network Layer e Transport Layer) de forma a concretizar o objetivo esperado.

A realização deste projeto/relatório permitiu-nos aplicar os conhecimentos teóricos previamente adquiridos numa vertente mais prática, e assim possibilitou uma melhor aprendizagem por parte do grupo nestes conteúdos.

## 2 Introdução

Este projeto encontra-se dividido em duas grandes partes. A primeira é implementar um cliente FTP (File Transfer Protocol) para fazer a transferência de um ficheiro, sendo que nesse sentido foi desenvolvida uma aplicação, Download Application. Numa primeira parte do relatório vamos explicar como é que essa aplicação foi desenvolvida assim como a sua arquitetura. De seguida, o relatório vai-se debruçar sobre a configuração de uma rede de computadores. Ao longo das aulas práticas foram realizadas um conjunto de experiências no sentido de configurar essa mesma rede, nesta secção vamos explicar o que é suposto ser alcançado em cada experiência assim como os resultados que conseguimos obter. As experiências são as seguintes:

1. Configuração de um IP de rede;
2. Configuração de duas Redes LAN virtuais num switch;
3. Configuração de um router em Linux;
4. Configuração de um router comercial implementando NAT;
5. DNS;
6. Conexões TCP;

## 3 Download Application

A primeira parte deste trabalho consiste numa aplicação responsável pela transferência de um ficheiro utilizando o protocolo FTP. Esta aplicação permite que o download seja feito tanto em modo anónimo com através de um username e respectiva password.

### 3.1 Arquitetura

A aplicação desenvolvida encontra-se devida em duas partes fundamentais, a primeira engloba o parser do url fornecido como argumento, que pode ser encontrada nos ficheiros parser.c e parser.h. A segunda parte, contida nos ficheiros client.c e client.h, é responsável por estabelecer e gerir as ligações TCP's necessárias para controlo e transferência de dados.

#### 3.1.1 Parser

Por questões de organização e simplificação foi criada a estrutura url responsável por guardar informação fundamental para as ligação que serão criadas, informações estas que são retiradas do input do utilizador.

A primeira função a ser invocada será a `parseUrl()` que recebe como argumento o input do utilizador e uma instanciação da estrutura inserida no parser.h onde será armazenada a informação contida no primeiro argumento que será, antes disso, devidamente interpretada e validada. Esta função recorre a outras duas, nomeadamente `parseUsernamePassword()` e `getIP()`.

### 3.1.2 Client

Depois de interpretada e devidamente validada a informação introduzida pelo utilizador é chamada a função `initConnection()` responsável pela inicialização de uma socket de controlo. A ligação será por fim estabelecida através da função `login()` que usará a socket inicializada posteriormente permitindo desta forma validar as credencias fornecidas pelo utilizador.

Seguidamente será enviado o comando PASV, que funciona como um pedido ao servidor FTP para transferir dados em modo passivo, ficando o cliente responsável pela abertura da ligação TCP para os dados. Será então criada uma nova conexão recorrendo diretamente à função `connectSocket()`, a fim de permitir a receção do ficheiro. A função `retrieve()` enviará o comando RETR necessário para inicializar a transferência, a receção dos dados e a respetiva escrita dos mesmos em disco será realizada através da função `download()`.

Terminada a receção de dados, a função `endConnection()` terá o papel de enviar o comando QUIT após o qual irá a fechar as sockets abertas anteriormente e libertar a memória alocada ao longo da execução a fim de terminar o programa com sucesso.

## 3.2 Resultados

A aplicação desenvolvida foi testada com diversos ficheiros nos dois modos, nunca sendo detetada uma anomalia na versão final. Em caso de erro, para além de se proceder à terminação do processo será originada uma mensagem de erro contendo, dentro do possível, informação acerca do sucedido.

## 4 Configuração e estudo da rede

### 4.1 Configurar uma rede IP

Para esta experiência temos como objetivo configurar os endereços de IP dos computadores tuxy1 e tuxy4 (em que y representa o número da bancada que o utilizador está a utilizar), para que estes se consigam comunicar entre si.

Para configurar o tuxy1 temos de fazer os seguintes comandos no terminal:

---

```
1 $ ifconfig eth0 up
2 $ ifconfig eth0 172.16.y0.1/24
3 $ route add default gw 172.16.20.254
```

---

Para configurar o tuxy4 temos de fazer os seguintes comandos no terminal:

---

```
1 $ ifconfig eth0 up
2 $ ifconfig eth0 172.16.y0.254/24
```

---

- O ARP (Address Resolution Protocol) é um protocolo de comunicação utilizado para obter o endereço MAC associado a um endereço IP fornecido.
- Num pacote ARP o endereço MAC corresponde ao endereço físico do emissor do pacote original para que possa depois receber a resposta da mensagem, enquanto o endereço IP corresponde ao endereço IP da máquina cujo endereço MAC se pretende determinar.
- O comando “ping” gera pacotes ICMP com o objetivo de captar respostas (eco) de um destinatário, assim determinando se ele se encontra conectado e ativo.
- Para conseguir distinguir os pacotes ARP dos IP e dos ICMP temos de analisar os 2 bytes do cabeçalho da trama Ethernet. Se esses 2 bytes tiverem o valor 0x0800 então sabemos que se trata de um IP, se tiver o valor 0x0806 sabemos que se trata de uma trama do tipo ARP.  
Como a trama ICMP é um sub-protocolo do protocolo IP para distinguir um protocolo ICMP dos restantes temos de analisar o valor do IP header. Se este for 1 então trata-se de uma trama ICMP.
- Para ver o tamanho dos pacotes basta analisar no wireshark o campo frame length.
- A interface loopback é uma interface virtual que permite que o computador receba respostas de si mesmo. É fundamental pois testa se a carta da rede está corretamente configurada.

### 4.2 Implementar duas LAN's virtuais no switch

O objetivo desta experiência é criar duas LAN's virtuais no switch, em que a primeira contenha o tuxy1 e o tuxy4 (VLANy0) e outra que contenha apenas o tuxy2 (VLANy1), e analisar que se consegue manter a comunicação entre o tuxy1 e o tuxy4 mas nenhum destes consegue-se comunicar com o tuxy2 pois não existe ligação entre as LAN's virtuais.

#### 4.2.1 Como se configura a VLANy0?

Criação da VLAN

---

```
1 enable
2 configure terminal
3 vlan y0
4 end
```

---

Adicionar as portas do tuxy1 e tuxy2

---

```
1 configure terminal
2 interface fastethernet 0/#numero da porta
3 switchport mode access
4 switchmode access vlan y0
5 end
```

---

Existem 2 domínios de transmissão, podemos ver isso pois o tuxy1 recebe resposta do tuxy4 quando se faz *ping -b*, mas não recebe qualquer resposta quando o mesmo é feito para o tuxy2. O tuxy2 não recebe nenhuma resposta de ninguém ao se fazer *ping -b*, podendo assim concluir que existem 2 domínios de transmissão, um que contem o tuxy1 e o tuxy4 e outro que apenas contem o tuxy2.

### 4.3 Configurar um router em Linux

O objetivo desta experiência é tornar o tuxy4 num router de modo que as VLAN's anteriormente criadas (VLANy0 e VLANy1) consigam comunicar entre si.

Para isto foi preciso configurar a porta eth1 do tuxy4 da seguinte maneira:

---

```
1 $ ifconfig eth1 up
2 $ ifconfig eth1 172.16.y1.253/24
```

---

Depois disto foi necessario adicionar a porta eth1 do tuxy4 à VLANy1, através do terminal do switch, da seguinte maneira:

---

```
1 configure terminal
2 interface fastethernet 0/#numero da porta
3 switchport mode access
4 switchmode access vlan y1
5 end
```

---

Por fim foi necessario acrescentar rotas no tuxy1 e tuxy2 para que estes se consigam comunicar entre si:

#### 1. No tuxy1

---

```
1 $ route add -net 172.16.y1.0/24 gw 172.16.y0.254
```

---

#### 2. No tuxy2

---

```
1 $ route add -net 172.16.y0.0/24 gw 172.16.y1.253
```

---

- Cada route tem uma *destination* e um *gateway*, em que o primeiro identifica a gama de endereços a que se vai adicionar a rota e o segundo é o IP para qual o *packet* vai ser reencaminhado.
- A informação mais importante para a experiência presente nas tabelas de *forwarding* é: Destination (destino da rota); Gateway (ip para onde o packet esta a ser reencaminhado); Netmask (juntamente com a *destination* formam o ID da rede) e Interface (placa de rede responsavel pela *gateway* (eth0 ou eth1))
- As mensagens de ARP mostram que o tux que recebeu o ping nao reconheceu o MAC address do emissor e por isso pergunta a quem pertence o MAC address do tux que tem aquele IP.
- São observados pacotes ICMP de request e reply devido a se ter adicionado rotas a todos os tuxs para que todos consigam comunicar entre si.
- Consultando os logs do Wireshark podemos verificar que ao fazermos ping do tuxy1 para o tuxy2 os endereços IP e MAC associados aos pacotes ICMP contêm a informação do tuxy4, porque este é o que esta a fazer de "ponte" entre os emissor e o recetor.
- Esta experiência tinha como objetivo configurar um router comercial, primeiramente sem NAT e posteriormente com NAT, permitindo a possibilidade de comunicar com a Internet. Sendo assim o router foi configurado para que fizesse parte da VLANy1.

- NAT (*Network Address Translation*) é uma técnica que tem como principal função reescrever os endereços IP utilizando uma tabela de *hash*, e assim com apenas um endereço público pode-se servir vários endereços privados fazendo com que seja possível poupar espaço de endereçamento público.

## 4.4 DNS

Nesta experiência temos como objetivo configurar um DNS (Domain Name System) este tem como principal função traduzir os *hostnames* para os seus respetivos IPs e vice-versa.<sup>3</sup>

- Para configurar o DNS para que este use o servidor netlab.fe.up.pt temos de alterar o ficheiro `resolv.conf` localizado na pasta `/etc` da seguinte forma:

---

```
1      $vi/etc/resolv.conf
2      search netlab.fe.up.pt
3      nameserver 172.16.1.1
```

---

- O *host* manda para o servidor DNS o *hostname* que pretende aceder, pedindo o respetivo IP. O DNS responde com um pacote contendo varias informações incluindo o IP necessário.

## 4.5 Ligações TCP

Nesta experiência tínhamos como objetivo analisar o protocolo TCP quando se faz o download de um ficheiro a partir da aplicação desenvolvida para a primeira parte do trabalho.

- São feitas 2 ligações TCP, a primeira que envia os comandos do cliente e recebe a respetiva resposta por parte do servidor, enquanto que a segunda envia os dados do servidor e o cliente recebe a respetiva resposta.
- A informação de controlo é transportada pela ligação responsável pelo transporte dos comandos.
- A ligação ftp é constituída por 3 fases. Primeiramente temos a fase de conexão, em que é estabelecida a conexão entre o servidor e o cliente. Depois disto temos a fase da transferência do ficheiro pretendido. Por fim temos o fecho da ligação entre o servidor e o cliente.
- O mecanismo ARQ TCP é um método de controlo de erros de transmissão que recebe 2 tipos de respostas, do tipo *timeout* e do tipo *acknowledge* (ACK). Se o emissor não receber ACK antes de acontecer o *timeout* o pacote é transmitido outra vez até que se receba uma resposta ACK antes do *timeout*.

O *header* do TCP tem como principais informações

1. as portas de envio e destino do envio
2. um *sequence number* de 32 bits que identifica o primeiro *byte* da data a ser transferida
3. um *ArkNumber* de 32 bits que indica o próximo *byte* que o recetor deverá ler
4. *Window size* usado para o *flow control*(ARQ) e *congestion control*
5. *Cheksum* usado para detetar erros recebidos

Podemos ver a informação dos logs nas figuras 13, 14 e 15.

- O mecanismo de controlo de congestão TCP mantém uma janela de congestão que consiste numa estimativa do número de octetos que a rede consegue encaminhar, não enviando mais octetos do que o mínimo da janela definida pelo recetor e pela janela de congestão.

Ao longo do tempo o fluxo de dados transferidos vai aumentando até que a chega a um pico onde os pacotes já não chegam corretamente dentro do período de *timeout* e por isso o número de pacotes começa a diminuir, voltando outra vez a aumentar até atingir um novo pico, acontecendo isto até que chegue ao fim da transferência, e por isso se encontra de acordo com o mecanismo de controlo de congestão TCP.

- Após o aparecimento da segunda conexão TCP a taxa de transmissão sofre uma queda de velocidade pois o canal fica congestionado e quando o protocolo TCP verifica que existe congestão na rede a taxa de transferência é diminuída.

Inicialmente a taxa de transferência aumenta normalmente até que a segunda conexão é feita. Quando esta acontece a taxa de transferência diminui e ao fim de algum tempo começa a estabilizar.

## 5 Conclusão

Com este trabalho foi possível compreender com maior pormenor como funciona uma rede de computadores e como são feitas as comunicações na mesma, através da realização da sua configuração. Através do desenvolvimento da aplicação de download houve também a oportunidade de passar a conhecer em maior detalhe um dos protocolos mais importantes e mais usados na transferência de ficheiros entre um cliente e um servidor, o FTP. No final, todas as experiências planeadas foram executadas com sucesso e o protocolo foi de igual forma implementado, pelo que o grupo considera que o resultado final é positivo.



## A Código fonte

### A.1 parser.h

---

```
1  #include <stdio.h>
2  #include <sys/types.h>
3  #include <sys/socket.h>
4  #include <netinet/in.h>
5  #include <arpa/inet.h>
6  #include <stdlib.h>
7  #include <unistd.h>
8  #include <signal.h>
9  #include <netdb.h>
10 #include <string.h>
11 #include <strings.h>
12 #include <termios.h>
13 #include <fcntl.h>
14
15 #define MAX_SIZE 256
16
17 typedef struct{
18     char    user[MAX_SIZE];
19     char    password[MAX_SIZE];
20     char    host[MAX_SIZE];
21     char    filePath[MAX_SIZE];
22     char    fileName[MAX_SIZE];
23     char    ip[MAX_SIZE];
24 } urlInfo;
25
26 int parseUsernamePassword(urlInfo * infoStruct, char * completeUrl);
27 int parseUrl(char completeUrl[], urlInfo * infoStruct);
28 int getIp(urlInfo * infoStruct);
```

---

parser.h

## A.2 parser.c

---

```
1  #include "parser.h"
2
3  int parseUsernamePassword(urlInfo * infoStruct, char * completeUrl){
4      //ftp://[<user>:<password>@]<host>/<url-path>
5
6      char * at = strrchr(completeUrl, '@');
7      char* firstSlash = strchr(completeUrl, '/');
8      firstSlash += 2; // pointing to the beginning of the username
9
10     char* password = strchr(firstSlash, ':'); //password pointing to :
11     if(password == NULL){
12         fprintf(stderr, "Your link must contain a ':' separating the username and password!\n");
13         return 1;
14     }
15
16     memcpy(infoStruct->user, firstSlash, password - firstSlash); //password - slash it's the
17     //size of username in bytes
18     infoStruct->user[password-firstSlash]=0;
19     password++; // now pointing to the first character of the password
20
21     memcpy(infoStruct->password, password, at - password);
22     infoStruct->password[at-password] = 0; //string end character
23     return 0;
24 }
25
26 int parseUrl(char completeUrl[], urlInfo * infoStruct){
27     //see if it begins with ftp://
28     if(strncmp(completeUrl, "ftp://", strlen("ftp://")) != 0){
29         fprintf(stderr, "The link does not begin with 'ftp://'\n");
30         return 1;
31     }
32
33     char* slashAfterHost;
34
35     if(!strchr(completeUrl, '@')){ //if it doesnt find @ it's password and username are
36         //anonymous
37         memcpy(infoStruct->user, "anonymous", strlen("anonymous") + 1);
38         memcpy(infoStruct->password, "anonymous", strlen("anonymous") + 1);
39
40         char * s1 = strchr(completeUrl, '/');
41         s1++;
42         slashAfterHost = strchr(s1, '/');
43         memcpy(infoStruct->host, s1, slashAfterHost-s1);
44         infoStruct->host[slashAfterHost-s1] = 0;
45     }
46
47     else{
48         if(parseUsernamePassword(infoStruct,completeUrl)!=0)
49             return 1;
50
51         char * at = strrchr(completeUrl, '@');
52         at++;
53
54         slashAfterHost = strchr(at, '/');
55         memcpy(infoStruct->host, at, slashAfterHost-at);
56         infoStruct->host[slashAfterHost-at] = 0;
57     }
58
59     char* lastSlash = strrchr(completeUrl, '/');
60     lastSlash++; //to point to the element after the slash
61     memcpy(infoStruct->filePath, slashAfterHost, lastSlash-slashAfterHost);
```

```

62     infoStruct->filePath[lastSlash-slashAfterHost] = 0;
63
64     memcpy(infoStruct->fileName, lastSlash, strlen(lastSlash) + 1);
65
66     getIp(infoStruct);
67
68     return 0;
69 }
70
71 int getIp(urlInfo * infoStruct) {
72     struct hostent* h;
73
74     if ((h = gethostbyname(infoStruct->host)) == NULL) {
75         perror("gethostbyname");
76         return 1;
77     }
78
79     // printf("Host name   : %s\n", h->h_name);
80     // printf("IP Address : %s\n", inet_ntoa(*((struct in_addr *)h->h_addr)));
81
82     char* ip = inet_ntoa(*((struct in_addr *) h->h_addr));
83     strcpy(infoStruct->ip, ip);
84
85     return 0;
86 }

```

---

parser.c

### A.3 client.h

---

```
1  #include <stdio.h>
2  #include <arpa/inet.h>
3  #include <stdlib.h>
4  #include <unistd.h>
5  #include <netdb.h>
6  #include <string.h>
7
8  #include "parser.h"
9
10 typedef struct {
11     int controlSocketFd; // file descriptor to control socket
12     int dataSocketFd; // file descriptor to data socket
13 } FTPInfo;
14
15 int initConnection(FTPInfo * ftp, char * ip, int port);
16 int connectSocket(char * ip, int port);
17 void login(FTPInfo ftp, urlInfo url);
18 void passiveMode(FTPInfo ftp, char * ip, int * port);
19 void retrieve(FTPInfo ftp, urlInfo url);
20 int readMessage(int socketfd, char * repply);
21 int sendMessage(int socketfd, char * cmd);
22 int download(FTPInfo ftp, urlInfo url);
23 int endConnection(FTPInfo ftp);
```

---

client.h

## A.4 client.c

---

```
1  #include "client.h"
2
3  int initConnection(FTPInfo* ftp, char* ip, int port){
4      int sockfd;
5
6      if ((sockfd = connectSocket(ip, port)) < 0) {
7          printf("Error connecting socket.\n");
8          return 1;
9      }
10
11     ftp->controlSocketFd = sockfd;
12     ftp->dataSocketFd = 0;
13
14     return 0;
15 }
16
17
18 int connectSocket(char * ip, int port){
19     int sockfd;
20     struct sockaddr_in server_addr;
21
22     /*server address handling*/
23     bzero((char*)&server_addr, sizeof(server_addr));
24     server_addr.sin_family = AF_INET;
25     server_addr.sin_addr.s_addr = inet_addr(ip); /*32 bit Internet address network byte ordered
26     */
27     server_addr.sin_port = htons(port); /*server TCP port must be network byte ordered */
28     /*open an TCP socket*/
29     if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
30         perror("socket()");
31         return -1;
32     }
33     /*connect to the server*/
34     if (connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0){
35         perror("connect()");
36         return -1;
37     }
38     return sockfd;
39 }
40
41 void login(FTPInfo ftp, urlInfo url){
42     char user[MAX_SIZE];
43     char password[MAX_SIZE];
44
45     readMessage(ftp.controlSocketFd, NULL);
46
47     sprintf(user, "USER %s\r\n", url.user);
48     printf(">%s", user);
49
50     sendMessage(ftp.controlSocketFd, user);
51     readMessage(ftp.controlSocketFd, NULL);
52
53     sprintf(password, "PASS %s\r\n", url.password);
54     printf(">%s", password);
55
56     sendMessage(ftp.controlSocketFd, password);
57     if (readMessage(ftp.controlSocketFd, NULL) != 0){
58         fprintf(stderr, "Wrong credentials. Exiting...\n");
59         exit(1);
60     }
61 }
62 }
```

```

63 void passiveMode(FTPInfo ftp, char * ip_address, int * port){
64     char reply[MAX_SIZE];
65
66     sendMessage(ftp.controlSocketFd, "PASV\r\n");
67     if(readMessage(ftp.controlSocketFd,reply) !=0){
68         fprintf(stderr, "Error entering passive mode. Exiting...\n");
69         exit(1);
70     }
71
72     int values[6];
73     char* data = strchr(reply, '(');
74     sscanf(data, "(%d, %d, %d, %d, %d, %d)", &values[0],&values[1],&values[2],&values[3],&values
75         [4],&values[5]);
76     sprintf(ip_address, "%d.%d.%d.%d", values[0],values[1],values[2],values[3]);
77     *port = values[4]*256+values[5];
78 }
79 void retrieve(FTPInfo ftp, urlInfo url){
80     char cmd[MAX_SIZE];
81
82     sprintf(cmd, "RETR %s%s\r\n", url.filePath, url.fileName);
83     printf(">%s",cmd);
84     sendMessage(ftp.controlSocketFd, cmd);
85
86     if(readMessage(ftp.controlSocketFd,NULL) != 0){
87         fprintf(stderr, "Error retrieving file. Exiting...\n");
88         exit(1);
89     }
90 }
91
92 int readMessage(int sockfd, char * reply){
93     FILE* fd = fdopen(sockfd, "r");
94     int allocated = 0;
95
96     if(reply == NULL){
97         reply = (char*) malloc(sizeof(char) * MAX_SIZE);
98         allocated = 1;
99     }
100
101     do {
102         memset(reply, 0, MAX_SIZE);
103         reply = fgets(reply, MAX_SIZE, fd);
104
105         if(reply == NULL){
106             printf("ERROR: Cannot read the message.\n");
107             return -1;
108         }
109
110         printf("<%s", reply);
111     } while (!(('1' <= reply[0] && reply[0] <= '5') || reply[3] != ' '));
112
113     char r0 = reply[0];
114
115     if(allocated)
116         free(reply);
117
118     return (r0>'4');
119 }
120
121 int sendMessage(int sockfd, char * cmd){
122     int ret = write(sockfd, cmd, strlen(cmd));
123     return ret;
124 }
125
126 int download(FTPInfo ftp, urlInfo url){

```

```

127 FILE* dest_file;
128 if(!(dest_file = fopen(url.fileName, "w"))) {
129     printf("Error opening file %s.\n",url.fileName);
130     return 1;
131 }
132
133 char buf[1024];
134 int bytes;
135 while ((bytes = read(ftp.dataSocketFd, buf, sizeof(buf)))) {
136     if (bytes < 0) {
137         fprintf(stderr, "Error, nothing was received from data socket fd.\n");
138         return 1;
139     }
140
141     if ((bytes = fwrite(buf, bytes, 1, dest_file)) < 0) {
142         fprintf(stderr, "Error, cannot write data in file.\n");
143         return 1;
144     }
145 }
146
147 fclose(dest_file);
148
149 printf("Finished downloading file\n");
150
151 return 0;
152 }
153
154
155 int endConnection(FTPInfo ftp){
156     printf("Closing connection\n");
157     sendMessage(ftp.controlSocketFd,"QUITTING\r\n");
158
159     if(readMessage(ftp.controlSocketFd,NULL) != 0){
160         fprintf(stderr, "Error closing connection. Closing...\n");
161         close(ftp.dataSocketFd);
162         close(ftp.controlSocketFd);
163         exit(1);
164     }
165
166     close(ftp.dataSocketFd);
167     close(ftp.controlSocketFd);
168
169     printf("Connection ended\n");
170
171     return 0;
172 }

```

---

client.c

## A.5 main.c

---

```
1  #include <stdio.h>
2  #include <arpa/inet.h>
3  #include <stdlib.h>
4  #include <unistd.h>
5  #include <netdb.h>
6  #include <string.h>
7  #include "client.h"
8
9  #define PORT 21
10
11 int main(int argc, char** argv){
12
13     if(argc != 2){
14         fprintf(stderr, "Usage: %s <address>\n", argv[0]);
15         exit(1);
16     }
17
18     urlInfo url;
19     FTPInfo ftp;
20
21     if(parseUrl(argv[1], &url) != 0){
22         fprintf(stderr, "Invalid URL\n");
23         exit(1);
24     }
25
26     printf("\nusername:%s\n", url.user);
27     printf("password:%s\n", url.password);
28     printf("ip:%s\n", url.ip);
29     printf("path:%s\n", url.filePath);
30     printf("file name:%s\n", url.fileName);
31     printf("host:%s\n", url.host);
32     printf("\n\n");
33
34     if(initConnection(&ftp, url.ip, PORT) != 0){
35         fprintf(stderr, "Error opening control connection\n");
36         exit(1);
37     }
38
39     login(ftp, url);
40
41     char ip[MAX_SIZE];
42     int port;
43
44     passiveMode(ftp, ip, &port);
45
46     if ((ftp.dataSocketFd = connectSocket(ip, port)) < 0){
47         fprintf(stderr, "Error opening data connection\n");
48         exit(1);
49     }
50
51     retrieve(ftp, url);
52     download(ftp, url);
53     endConnection(ftp);
54
55     return 0;
56 }
```

---

main.c



## A.6 makefile

---

```
1 gs: client.c parser.c main.c
2     gcc -Wall -o t2 client.c parser.c main.c
3
4 clean:
5     rm -f t2
```

---

makefile

## B Anexos:

### B.1 Arquitetura das experiências:

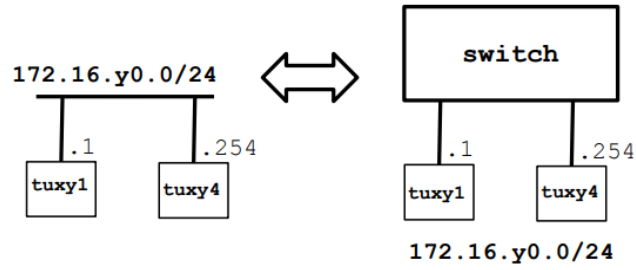


Figura 1: Arquitetura da experiência 1

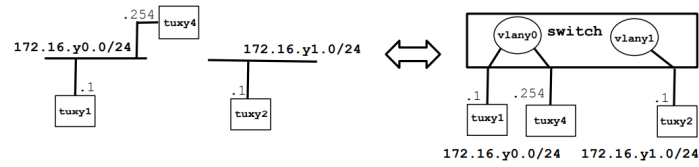


Figura 2: Arquitetura da experiência 2

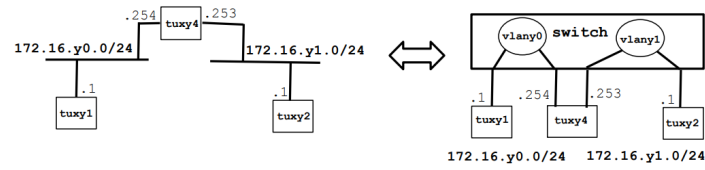


Figura 3: Arquitetura da experiência 3

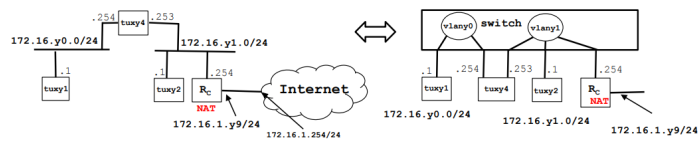


Figura 4: Arquitetura da experiência 4

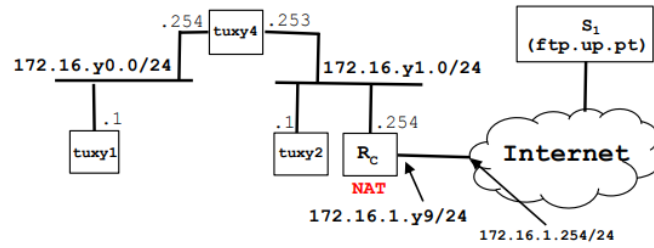


Figura 5: Arquitetura da experiência 5

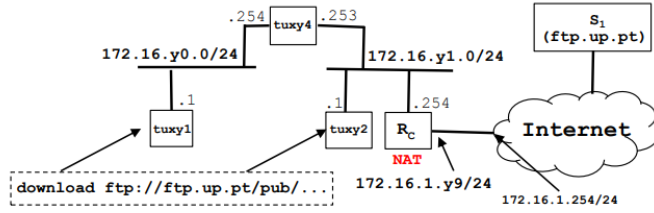


Figura 6: Arquitetura da experiência 6

## B.2 Logs das Experiências

### B.2.1 Exp. 1

No	Time	Source	Destination	Protocol	Length	Info
1	0.00000000	Clisco:Sci44:83	Clisco:Sci44:83	ICMP	60	Sequence ID: 0x00000000, Port ID: 0x00000000
2	0.001870000	172.16.20.1	172.16.20.1	ICMP	98	Echo (ping) request id=0x0000, seq=001/11521, ttl=64 (reply in 3)
3	0.001870000	172.16.20.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0000, seq=001/11521, ttl=64 (request in 2)
4	0.003740000	Clisco:Sci44:83	Clisco:Sci44:83	ICMP	60	Sequence ID: 0x00000000, Port ID: 0x00000000
5	1.001870000	172.16.20.1	172.16.20.1	ICMP	98	Echo (ping) request id=0x0000, seq=002/11777, ttl=64 (reply in 6)
6	1.001870000	172.16.20.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0000, seq=002/11777, ttl=64 (request in 5)
7	2.001870000	172.16.20.1	172.16.20.1	ICMP	98	Echo (ping) request id=0x0000, seq=003/12033, ttl=64 (reply in 8)
8	2.001870000	172.16.20.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0000, seq=003/12033, ttl=64 (request in 7)
9	3.001870000	Clisco:Sci44:83	Clisco:Sci44:83	ICMP	60	Sequence ID: 0x00000000, Port ID: 0x00000000
10	3.001870000	172.16.20.1	172.16.20.1	ICMP	98	Echo (ping) request id=0x0000, seq=004/12289, ttl=64 (reply in 11)
11	3.001870000	172.16.20.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0000, seq=004/12289, ttl=64 (request in 10)
12	4.001870000	172.16.20.1	172.16.20.1	ICMP	98	Echo (ping) request id=0x0000, seq=005/12545, ttl=64 (reply in 13)
13	4.001870000	172.16.20.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0000, seq=005/12545, ttl=64 (request in 12)
14	5.001870000	Clisco:Sci44:83	Clisco:Sci44:83	ICMP	60	Sequence ID: 0x00000000, Port ID: 0x00000000
15	5.001870000	172.16.20.1	172.16.20.1	ICMP	98	Echo (ping) request id=0x0000, seq=006/12801, ttl=64 (reply in 16)
16	5.001870000	172.16.20.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0000, seq=006/12801, ttl=64 (request in 15)
17	5.001870000	Clisco:Sci44:83	Clisco:Sci44:83	ICMP	60	Sequence ID: 0x00000000, Port ID: 0x00000000
18	6.001870000	172.16.20.1	172.16.20.1	ICMP	98	Echo (ping) request id=0x0000, seq=007/13057, ttl=64 (reply in 19)
19	6.001870000	172.16.20.1	172.16.20.1	ICMP	98	Echo (ping) reply id=0x0000, seq=007/13057, ttl=64 (request in 18)

Figura 7: Ping do tuxy1 para o tuxy4

### B.2.2 Exp. 2

1	0.00000000	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1f3c, seq=12/3072, ttl=64 (no response found)
2	0.000020000	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1f3c, seq=12/3072, ttl=64
4	0.000000000	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1f3c, seq=13/3028, ttl=64 (no response found)
5	1.000040000	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1f3c, seq=13/3028, ttl=64
7	2.001200000	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1f3c, seq=14/3084, ttl=64 (no response found)
8	2.001500000	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1f3c, seq=14/3084, ttl=64
10	3.000020000	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1f3c, seq=15/3040, ttl=64 (no response found)
11	3.000070000	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1f3c, seq=15/3040, ttl=64
12	4.000000000	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1f3c, seq=16/3096, ttl=64 (no response found)
13	4.000040000	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1f3c, seq=16/3096, ttl=64
15	5.000000000	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1f3c, seq=17/3052, ttl=64 (no response found)
16	5.000040000	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1f3c, seq=17/3052, ttl=64
17	5.000000000	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1f3c, seq=18/3008, ttl=64 (no response found)
18	6.000020000	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1f3c, seq=18/3008, ttl=64
20	6.999980000	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1f3c, seq=19/3064, ttl=64 (no response found)
21	7.000000000	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1f3c, seq=19/3064, ttl=64

Figura 8: Ping do tuxy1 para o broadcast 172.16.40.255

### B.2.3 Exp. 3

172.16.40.1				172.16.40.1			
Time	Source	Destination	Protocol	Length	Info	Time	Source
3	0.40247000	172.16.41.253	ICMP	98	Echo (ping) reply id=0x2147, seq=61536, ttl=64 (request in 2)		
4	1.40212700	172.16.41.253	ICMP	98	Echo (ping) request id=0x2147, seq=71792, ttl=64 (reply in 5)		
5	1.40208300	172.16.41.253	ICMP	98	Echo (ping) reply id=0x2147, seq=71792, ttl=64 (request in 4)		
7	2.40211300	172.16.41.253	ICMP	98	Echo (ping) request id=0x2147, seq=82048, ttl=64 (reply in 8)		
8	2.40248600	172.16.41.253	ICMP	98	Echo (ping) reply id=0x2147, seq=82048, ttl=64 (request in 7)		
9	3.40213800	172.16.41.253	ICMP	98	Echo (ping) request id=0x2147, seq=92304, ttl=64 (reply in 10)		
10	3.40259800	172.16.41.253	ICMP	98	Echo (ping) reply id=0x2147, seq=92304, ttl=64 (request in 9)		
12	4.40213000	172.16.41.253	ICMP	98	Echo (ping) request id=0x2147, seq=102560, ttl=64 (reply in 13)		
13	4.40244000	172.16.41.253	ICMP	98	Echo (ping) reply id=0x2147, seq=102560, ttl=64 (request in 12)		
14	5.40213200	172.16.41.253	ICMP	98	Echo (ping) request id=0x2147, seq=112816, ttl=64 (reply in 15)		
15	5.40234400	172.16.41.253	ICMP	98	Echo (ping) reply id=0x2147, seq=112816, ttl=64 (request in 14)		
17	6.40213900	172.16.41.253	ICMP	98	Echo (ping) request id=0x2147, seq=123072, ttl=64 (reply in 18)		
18	6.40240400	172.16.41.253	ICMP	98	Echo (ping) reply id=0x2147, seq=123072, ttl=64 (request in 17)		
19	7.40212700	172.16.41.253	ICMP	98	Echo (ping) request id=0x2147, seq=133328, ttl=64 (reply in 20)		
20	7.40236100	172.16.41.253	ICMP	98	Echo (ping) reply id=0x2147, seq=133328, ttl=64 (request in 19)		
22	8.40213100	172.16.41.253	ICMP	98	Echo (ping) request id=0x2147, seq=143584, ttl=64 (reply in 23)		
23	8.40248500	172.16.41.253	ICMP	98	Echo (ping) reply id=0x2147, seq=143584, ttl=64 (request in 22)		
25	9.40210100	172.16.41.253	ICMP	98	Echo (ping) request id=0x2147, seq=153840, ttl=64 (reply in 26)		

Figura 9: Ping do tuxy1 para o tuxy4 na interface eth1

172.16.40.1				172.16.40.1			
Time	Source	Destination	Protocol	Length	Info	Time	Source
2	0.00252000	172.16.41.1	ICMP	98	Echo (ping) reply id=0x1a3e, seq=3011203, ttl=63 (request in 1)		
4	0.99997100	172.16.41.1	ICMP	98	Echo (ping) request id=0x1a3e, seq=30412289, ttl=64 (reply in 5)		
5	1.00048000	172.16.41.1	ICMP	98	Echo (ping) reply id=0x1a3e, seq=30412289, ttl=63 (request in 4)		
6	1.99997700	172.16.41.1	ICMP	98	Echo (ping) request id=0x1a3e, seq=30512545, ttl=64 (reply in 7)		
7	2.00048600	172.16.41.1	ICMP	98	Echo (ping) reply id=0x1a3e, seq=30512545, ttl=63 (request in 6)		
9	2.99997200	172.16.41.1	ICMP	98	Echo (ping) request id=0x1a3e, seq=30612801, ttl=64 (reply in 10)		
10	3.00047400	172.16.41.1	ICMP	98	Echo (ping) reply id=0x1a3e, seq=30612801, ttl=63 (request in 9)		
11	3.99998800	172.16.41.1	ICMP	98	Echo (ping) request id=0x1a3e, seq=30713057, ttl=64 (reply in 12)		
12	4.00047800	172.16.41.1	ICMP	98	Echo (ping) reply id=0x1a3e, seq=30713057, ttl=63 (request in 11)		
15	4.99996700	172.16.41.1	ICMP	98	Echo (ping) request id=0x1a3e, seq=30813313, ttl=64 (reply in 16)		
16	5.00047200	172.16.41.1	ICMP	98	Echo (ping) reply id=0x1a3e, seq=30813313, ttl=63 (request in 15)		
17	5.99996200	172.16.41.1	ICMP	98	Echo (ping) request id=0x1a3e, seq=30913569, ttl=64 (reply in 18)		
18	6.00046100	172.16.41.1	ICMP	98	Echo (ping) reply id=0x1a3e, seq=30913569, ttl=63 (request in 17)		
20	6.99996800	172.16.41.1	ICMP	98	Echo (ping) request id=0x1a3e, seq=31013825, ttl=64 (reply in 21)		
21	7.00047100	172.16.41.1	ICMP	98	Echo (ping) reply id=0x1a3e, seq=31013825, ttl=63 (request in 20)		
22	7.99996500	172.16.41.1	ICMP	98	Echo (ping) request id=0x1a3e, seq=31114081, ttl=64 (reply in 23)		
23	8.00047400	172.16.41.1	ICMP	98	Echo (ping) reply id=0x1a3e, seq=31114081, ttl=63 (request in 22)		
25	8.99996500	172.16.41.1	ICMP	98	Echo (ping) request id=0x1a3e, seq=31214337, ttl=64 (reply in 26)		

Figura 10: Ping do tuxy1 para o tuxy2

### B.2.4 Exp. 4

96	13.178711000	172.16.1.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x190a, seq=20/5120, ttl=64
98	14.177962000	172.16.40.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x190a, seq=21/5376, ttl=64
99	14.178739000	172.16.1.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x190a, seq=21/5376, ttl=64
40	15.177964000	172.16.40.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x190a, seq=22/5632, ttl=64
41	15.178745000	172.16.1.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x190a, seq=22/5632, ttl=64
43	16.177965000	172.16.40.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x190a, seq=23/5888, ttl=64
44	16.178760000	172.16.1.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x190a, seq=23/5888, ttl=64
46	18.026752000	172.16.40.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x1918, seq=1/256, ttl=64
47	18.027610000	172.16.1.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1918, seq=1/256, ttl=62
48	19.025652000	172.16.40.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x1918, seq=2/512, ttl=64
49	19.026724000	172.16.1.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1918, seq=2/512, ttl=62
52	20.025948000	172.16.40.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x1918, seq=3/768, ttl=64
53	20.026682000	172.16.1.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1918, seq=3/768, ttl=62
54	21.025950000	172.16.40.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x1918, seq=4/1024, ttl=64
55	21.026697000	172.16.1.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1918, seq=4/1024, ttl=62

Figura 11: Ping do tuxy1 para o tuxy4 na interface eth1

### B.2.5 Exp. 5

5	4.253108000	172.16.40.1	172.16.1.1	DNS	73	Standard query 0x2986 A www.google.pt
6	4.256320000	172.16.1.1	172.16.40.1	DNS	548	Standard query response 0x2986 A 172.217.16.227
7	4.256821000	172.16.40.1	172.217.16.227	ICMP	98	Echo (ping) request id=0x05f8, seq=1/256, ttl=64 (reply in 8)
8	4.273171000	172.217.16.227	172.16.40.1	ICMP	98	Echo (ping) reply id=0x05f8, seq=1/256, ttl=60 (request in 7)

Figura 12: Resposta do comando DNS

### B.2.6 Exp. 6

12	2.813037000	172.16.40.1	172.16.1.1	DNS	69	Standard query 0xf077 A ftp.up.pt
13	2.814890000	172.16.1.1	172.16.40.1	DNS	395	Standard query response 0xf077 CNAME mirrors.up.pt A 193.137.29.15
14	2.815933000	172.16.40.1	193.137.29.15	TCP	74	30130-21 SYN Seq=0 Win=0 Len=0 MSG=1460 SACK_FLAGS=TSval=13790588 TSsrc=0 WS=128
15	2.821170000	193.137.29.15	172.16.40.1	TCP	74	21-30130-21 SYN Seq=1 Win=0 Len=0 MSG=1380 SACK_FLAGS=TSval=13790588 TSsrc=13790588
16	2.821192000	172.16.40.1	193.137.29.15	TCP	66	30130-21 ACK Seq=1 Ack=1 Win=20312 Len=0 TSval=13790588 TSsrc=236065677
17	2.831030000	193.137.29.15	172.16.40.1	FTP	139	Response: 220-Welcome to the University of Porto's mirror archive (mirrors.up.pt)
18	2.831079000	193.137.29.15	172.16.40.1	FTP	135	Response: 220.....
19	2.831348000	172.16.40.1	193.137.29.15	TCP	66	30130-21 ACK Seq=1 Ack=74 Win=20312 Len=0 TSval=13790588 TSsrc=236065679
20	2.831381000	172.16.40.1	193.137.29.15	TCP	66	30130-21 ACK Seq=1 Ack=143 Win=20312 Len=0 TSval=13790588 TSsrc=236065679
21	2.831388000	193.137.29.15	172.16.40.1	FTP	72	Response: 220
22	2.831403000	172.16.40.1	193.137.29.15	TCP	66	30130-21 ACK Seq=1 Ack=149 Win=20312 Len=0 TSval=13790588 TSsrc=236065679
23	2.831409000	193.137.29.15	172.16.40.1	FTP	151	Response: 220-All connections and transfers are logged. The max number of connections is 200.

Figura 13: Estabelecimento de conexão através do protocolo TCP

58	2.980043000	172.16.40.1	193.137.29.15	TCP	66	34879-58032 [ACK] Seq=1 Ack=10945 Win=51200 Len=0 TSval=13790626 TSecr=236009616
59	2.980269000	193.137.29.15	172.16.40.1	FTP-DATA	2802	FTP Data: 2798 bytes
60	2.980269000	172.16.40.1	193.137.29.15	TCP	66	34879-58032 [ACK] Seq=1 Ack=13681 Win=50376 Len=0 TSval=13790626 TSecr=236009616
61	2.980499000	193.137.29.15	172.16.40.1	FTP-DATA	2802	FTP Data: 2798 bytes
62	2.980499000	172.16.40.1	193.137.29.15	TCP	66	34879-58032 [ACK] Seq=1 Ack=16417 Win=52080 Len=0 TSval=13790626 TSecr=236009617
63	2.980626000	193.137.29.15	172.16.40.1	FTP-DATA	2802	FTP Data: 2798 bytes
64	2.980626000	172.16.40.1	193.137.29.15	TCP	66	34879-58032 [ACK] Seq=1 Ack=19153 Win=57584 Len=0 TSval=13790626 TSecr=236009617
65	2.980911000	193.137.29.15	172.16.40.1	FTP-DATA	2802	FTP Data: 2798 bytes
66	2.980920000	172.16.40.1	193.137.29.15	TCP	66	34879-58032 [ACK] Seq=1 Ack=21889 Win=73088 Len=0 TSval=13790626 TSecr=236009617
67	2.981164000	193.137.29.15	172.16.40.1	FTP-DATA	2802	FTP Data: 2798 bytes
68	2.981164000	172.16.40.1	193.137.29.15	TCP	66	34879-58032 [ACK] Seq=1 Ack=24625 Win=78464 Len=0 TSval=13790626 TSecr=236009617
69	2.981419000	193.137.29.15	172.16.40.1	FTP-DATA	2802	FTP Data: 2798 bytes
70	2.981440000	172.16.40.1	193.137.29.15	TCP	66	34879-58032 [ACK] Seq=1 Ack=27361 Win=83968 Len=0 TSval=13790627 TSecr=236009617
71	2.981661000	193.137.29.15	172.16.40.1	FTP-DATA	2802	FTP Data: 2798 bytes
72	2.981661000	172.16.40.1	193.137.29.15	TCP	66	34879-58032 [ACK] Seq=1 Ack=30097 Win=89472 Len=0 TSval=13790627 TSecr=236009617
73	2.981911000	193.137.29.15	172.16.40.1	FTP-DATA	2802	FTP Data: 2798 bytes
74	2.981920000	193.137.29.15	172.16.40.1	FTP-DATA	1434	FTP Data: 1368 bytes
75	2.981930000	172.16.40.1	193.137.29.15	TCP	66	34879-58032 [ACK] Seq=1 Ack=32833 Win=94076 Len=0 TSval=13790627 TSecr=236009617
76	2.981945000	172.16.40.1	193.137.29.15	TCP	66	34879-58032 [ACK] Seq=1 Ack=34201 Win=97792 Len=0 TSval=13790627 TSecr=236009617

Figura 14: Transferência de dados através do protocolo TCP

736	3.068052000	172.16.40.1	193.137.29.15	TCP	66	34879-58032 [ACK] Seq=1 Ack=962225 Win=230528 Len=0 TSval=13790648 TSecr=236009637
737	3.068081000	193.137.29.15	172.16.40.1	FTP-DATA	2802	FTP Data: 2798 bytes
738	3.068899000	172.16.40.1	193.137.29.15	TCP	66	34879-58032 [ACK] Seq=1 Ack=984961 Win=230528 Len=0 TSval=13790648 TSecr=236009637
739	3.068918000	193.137.29.15	172.16.40.1	FTP-DATA	2802	FTP Data: 2798 bytes
740	3.069152000	172.16.40.1	193.137.29.15	TCP	66	34879-58032 [ACK] Seq=1 Ack=987998 Win=229632 Len=0 TSval=13790648 TSecr=236009637
741	3.069820000	172.16.40.1	193.137.29.15	FTP	76	Request: QUITTING
742	3.071407000	193.137.29.15	172.16.40.1	FTP	80	Response: 226 Transfer complete.
743	3.071502000	172.16.40.1	193.137.29.15	TCP	66	39130->21 [ACK] Seq=480 Ack=610 Win=29312 Len=0 TSval=13790649 TSecr=236009640
744	3.071539000	172.16.40.1	193.137.29.15	TCP	66	34879-58032 [FIN, ACK] Seq=1 Ack=987998 Win=0 Len=0 TSval=13790649 TSecr=236009637
745	3.071552000	172.16.40.1	193.137.29.15	TCP	66	39130->21 [FIN, ACK] Seq=480 Ack=610 Win=0 Len=0 TSval=13790649 TSecr=236009640
746	3.072427000	193.137.29.15	172.16.40.1	FTP	88	Response: 500 Unknown command.
747	3.072455000	172.16.40.1	193.137.29.15	TCP	54	39130->21 [RST] Seq=1 Win=0 Len=0
748	3.073117000	193.137.29.15	172.16.40.1	TCP	66	58032->34879 [ACK] Seq=987998 Ack=2 Win=29056 Len=0 TSval=236009640 TSecr=13790649
749	3.073129000	172.16.40.1	193.137.29.15	TCP	66	34879-58032 [ACK] Seq=1 Ack=29056 Win=0 Len=0 TSval=236009640 TSecr=13790649
750	3.073991000	172.16.40.1	193.137.29.15	TCP	54	39130->21 [RST] Seq=1 Win=0 Len=0
751	3.641833000	157.240.1.18	172.16.40.1	TLSv1.2	221	Application Data
752	3.641805000	172.16.40.1	157.240.1.18	TCP	66	4048->449 [ACK] Seq=1 Ack=156 Win=3006 Len=0 TSval=13790791 TSecr=2300528569
753	3.669754000	172.16.40.1	172.16.1.1	DNS	85	Standard query 0x5050 A 2-edge-chat.messenger.com

Figura 15: Terminação da conexão através do protocolo TCP

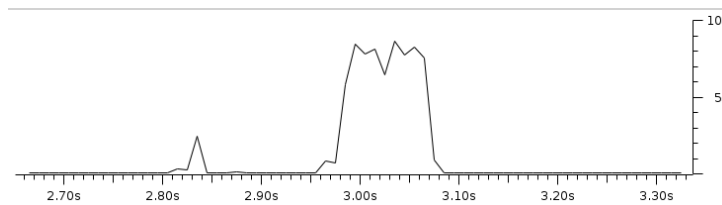


Figura 16: Gráfico de transferência dos pacotes por unidade de tempo