

Configuração de uma Rede e Desenvolvimento de uma Aplicação FTP

Relatório Final



Mestrado Integrado em Engenharia Informática e
Computação

Redes de Computadores

Maria Eduarda Santos Cunha - up201506524
João Francisco Veríssimo Dias Esteves - up201505145
João Miguel Matos Monteiro – up201506130

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

31 de Dezembro de 2017

Conteúdo

1	Sumário	3
2	Introdução	3
3	Aplicação FTP	4
3.1	Protocolo	4
3.2	Arquitetura	4
3.2.1	Url	4
3.2.2	FtpData	4
3.2.3	main.c	4
3.3	Na prática	5
4	Configuração de Rede e Análise	6
4.1	Experiência 1: Configuração de uma Rede IP	6
4.2	Experiência 2: Configuração de 2 Redes LAN Virtuais dentro de um Switch	6
4.3	Experiência 3: Configuração de um Router em Linux	6
4.4	Experiência 4: Configuração de um Router Comercial com NAT	7
4.5	Experiência 5: DNS	8
4.6	Experiência 6: Conexões TCP	8
4.7	Experiência 7 - Implementação de NAT em Linux	9
5	Conclusão	9
A	Código da aplicação FTP	10
A.1	Imagens	10
A.2	Código	10
A.2.1	main.c	10
A.2.2	Url.h	10
A.2.3	Url.c	11
A.2.4	FtpData.h	14
A.2.5	FtpData.c	15
B	Logs Wireshark	20
B.1	Experiências	20
B.1.1	Experiência 1	20
B.1.2	Experiência 3	21
B.1.3	Experiência 4	22
B.1.4	Experiência 5	23
B.1.5	Experiência 6	24
B.1.6	Experiência 7	28
C	Comandos de configuração	29
C.1	Repôr configurações originais	29
C.2	Experiência 1	29
C.3	Experiência 2	29
C.4	Experiência 3	30
C.5	Experiência 4	30
C.6	Experiência 5	31
C.7	Experiência 6	31
C.8	Experiência 7	31

1 Sumário

O presente relatório tem como objetivo a consolidação do segundo trabalho efetuado na cadeira de Redes de Computadores, que consiste na configuração de uma rede e o desenvolvimento de uma aplicação de download, através da explicação da metodologia aplicada e da análise de resultados. Os problemas propostos foram implementados com sucesso.

2 Introdução

O projeto aborda duas componentes: o desenvolvimento em C de uma aplicação que efetua o download de um ficheiro através do *File Transfer Protocol* (FTP) e a configuração de uma rede, sendo sugeridas seis experiências mais uma opcional.

As experiências são a configuração de um IP de rede, duas redes LAN virtuais dentro de um switch, um router em Linux, um router comercial implementando NAT, DNS, conexões TCP e, por fim, a implementação de NAT em Linux. Neste relatório é descrita a arquitetura da aplicação de download e como esta foi desenvolvida e, de seguida, as conclusões e resultados derivados das várias experiências acima referidas.

3 Aplicação FTP

3.1 Protocolo

O FTP, *File Transfer Protocol*, permite transferir ficheiros de forma segura. Este tem dois modos de operação, ativo e passivo, sendo o último o abordado neste projeto.

Neste protocolo, em particular no modo passivo, uma série de comandos e respostas são trocadas entre uma porta do cliente e a porta 21 do servidor FTP, sendo a transferência em si efetuada entre outra porta do cliente e uma porta do servidor escolhida por este.

A sequência de interações cliente-servidor é a seguinte: é aberta a ligação com a porta 21, são enviados para esta o username e a password, e é ativo o modo passivo; a seguir, é aberta uma ligação com uma outra porta do servidor, o ficheiro é transferido por esta, e finalmente fecham-se ambas as ligações.

3.2 Arquitetura

A aplicação consiste nos 2 módulos *Url* e *FtpData*, cada um com um ficheiro .c e um .h. O ponto de entrada é o ficheiro *main.c*.

3.2.1 Url

A função *parseUrl()* obtém para cada campo de uma *struct Url* a informação contida no link: username, password, host e caminho do ficheiro. Caso o username e a password não sejam dados, estes ficam como "anonymous".

3.2.2 FtpData

Neste módulo devem ser chamadas ordenadamente as funções *initFtpData()*, *setupConnection()*, *downloadFile()* e *closeConnection()*.

A primeira obtém o endereço IP do servidor através do seu nome.

A função *setupConnection()* abre a ligação à porta 21, inicia sessão através dos comandos *USER 'username'* e *PASS 'password'*, ativa o modo passivo com o comando *PASV*, obtém da resposta deste último a porta do servidor que irá enviar o ficheiro, e abre uma nova ligação com essa porta.

A função *downloadFile()* envia para a porta 21 o comando *RETR 'filename com caminho'* para assinalar o começo da transferência do ficheiro, obtendo como resposta o tamanho do ficheiro, e vai lendo o ficheiro da ligação de dados até que acabem os dados na ligação, mostrando o progresso da transferência ao utilizador.

Por fim, a função *closeConnection()* fecha ambas as ligações ao servidor.

3.2.3 main.c

Usa o módulo *Url* para obter a informação do link passado como argumento e, de seguida, usa o módulo *FtpData* para transferir o ficheiro pedido.

3.3 Na prática

O programa é iniciado com um *link* como primeiro argumento. O link segue a forma *ftp://['user': 'password'@]'host'/'url-path'*. O campo com o username e a password é opcional, ficando estes dois com o valor "anonymous" caso o campo seja omitido.

O programa prossegue para a transferência do ficheiro, mostrando a percentagem atual dos bytes transferidos pelos totais (ver figura 1).

No fim, o ficheiro é transferido com sucesso, ficando no mesmo diretório onde se encontra o executável do cliente FTP e com o mesmo nome que tem no servidor.

4 Configuração de Rede e Análise

4.1 Experiência 1: Configuração de uma Rede IP

Tem como objetivo a comunicação direta entre 2 computadores ligados a um switch, percebendo o funcionamento do *Address Resolution Protocol* (ARP), tendo-se capturado no Tux1 o ping deste para o Tux4. Os comandos usados são os presentes em C.2.

Os pacotes ARP servem para descobrir o endereço MAC associado a um endereço IP, necessário para a comunicação entre dois IP's.

Cada pacote ARP tem os endereços IP e MAC do transmissor e do recetor pretendido, ou seja, no primeiro pacote enviado o endereço MAC do recetor é nulo pois este é o que se pretende descobrir (fig. 2).

O comando ping gera pacotes ICMP, dos tipos *request* a partir do que inicia o ping e *reply* desde o alvo do ping, cada um contendo os endereços IP e MAC dos respetivos transmissores e recetores.

Os pacotes ARP e IP são diferenciados através do campo "Type" de 2 bytes no header Ethernet (fig. 2), e o ICMP, sendo um sub-tipo dos IP, pelo campo "Protocol" de 1 byte no header IP (fig. 3).

No caso dos *frames* de pacotes ARP, o tamanho é fixo a 60 bytes. Para os pacotes IP, o tamanho é a soma do header de tamanho fixo de 14 bytes com o campo de 2 bytes representando o tamanho do pacote IP, presente no header IP. Na figura 4 está o exemplo de um *frame* de um pacote IP com tamanho total de 98 bytes, que é a soma dos 14 bytes com os 84 bytes do pacote IP.

O loopback serve para poder enviar informação para o próprio dispositivo, podendo assim haver comunicação entre programas a correr no mesmo.

4.2 Experiência 2: Configuração de 2 Redes LAN Virtuais dentro de um Switch

Nesta experiência, o Tux1 e o Tux4 estão numa VLAN e o Tux2 está noutra. Os comandos usados são os presentes em C.3 e anteriores.

Para criar uma VLANy0 enviam-se os comandos em C.2, para o caso de y=1 e y=0, para o Switch através da porta série ligada a um dos Tux.

Há 2 broadcast domains, pois verifica-se nos logs que ao fazer *ping broadcast* no Tux1 apenas o Tux4 lhe responde, e ao fazer *ping broadcast* no Tux2 ninguém lhe responde.

4.3 Experiência 3: Configuração de um Router em Linux

Nesta experiência há uma VLAN com o Tux1 e uma das portas do Tux4, e outra VLAN com o Tux2 e uma outra porta do Tux4. Os comandos usados são os presentes em C.4 e anteriores.

O Tux1 e o Tux2 têm, cada um, uma rota para aceder o outro usando o Tux4 como *gateway*. O Tux4 contém uma rota para o domínio do Tux1 pela porta Ethernet 0 e outra para o domínio do Tux2 pela porta Ethernet 1. Uma entrada na tabela de endereçamentos contém o IP e máscara de destino, o IP da *gateway* e a interface, de forma a que um pacote será reencaminhado para a *gateway* pela tal interface da entrada na tabela com a maior máscara cujo IP de destino corresponda ao IP de destino no pacote com a máscara aplicada.

Apagando as tabelas ARP nos 3 Tuxes, eles têm de enviar pacotes ARP entre si para as preencher novamente de forma a poderem comunicar entre eles.

Fazendo ping do Tuxy1 (172.16.50.1) ao Tuxy2 (172.16.51.1), foram medidas ambas as interfaces do Tuxy4. Na porta ligada ao Tuxy1, figura 5, o pacote ARP pergunta quem tem o MAC do Tuxy4 Eth0 (172.16.50.254) e para responder ao Tuxy1. Na porta ligada ao Tuxy2, figura 6, pergunta quem tem o MAC do Tuxy2 e para responder ao Tuxy4 Eth1 (172.16.51.253). Isto significa que os pacotes ARP funcionam em cadeia, ou seja, há uma troca de pacotes ARP em cada troço para cada nó intermédio descobrir os endereços MAC do nó seguinte ao longo da rota.

A seguir aos pacotes ARP, em ambas interfaces, são então trocados os pacotes ICMP *request* e *reply* necessários para a execução do ping.

Os endereços IP dos pacotes ICMP são os dos nós inicial e final das respetivas rotas, mas os endereços MAC são os dos nós adjacentes de cada troço. Dado que os endereços MAC pertencem a uma camada de mais baixo nível, verifica-se então que estes servem para a comunicação direta entre 2 dispositivos adjacentes.

4.4 Experiência 4: Configuração de um Router Comercial com NAT

Esta experiência tem uma disposição idêntica à anterior, adicionando-se o Router da bancada à VLAN que contém os Tux2 e Tux4 e ligando-o ao Router da sala para acesso exterior com NAT implementado. Os comandos usados são os presentes em C.5 e anteriores.

Para adicionar uma rota estática ao Router da bancada, usa-se o comando *ip route IPdest MASKdest IPgw*, em que *IPdest* e *MASKdest* são 0.0.0.0 para a rota *default*. Por exemplo, a rota para o Tux1 através do Tux4 será *ip route 172.16.10.0 255.255.255.0 172.16.11.253*.

Com e sem o *ICMP redirects* ativado, sem a rota do Tuxy2 para o Tuxy1 através do Tuxy4, os pacotes do Tuxy2 para o Tuxy1 vão primeiro para o Router da bancada, pois apenas há a rota do Tuxy2 para este, deste para o Tuxy4 devido à rota do Router da bancada para este, e finalmente para o Tuxy1. Com o *ICMP redirects* desativado e sem a rota do Tuxy2 para o Tuxy1 através do Tuxy4, verifica-se na fig. 7 que assim que os pacotes chegam ao Router da bancada o Router envia um pacote ICMP para que o Tuxy2 passe a enviar os pacotes para o Tuxy1 diretamente pelo Tuxy4, mas com o *ICMP redirects* desativado estes são ignorados. Quando o *ICMP redirects* é ativado, o Router envia o pacote apenas na primeira vez, passando de seguida o Tuxy2 a enviar diretamente pelo Tuxy4.

Para configurar NAT no Router da bancada, são usados alguns dos comandos no anexo C.5, nomeadamente os que começam com *ip nat* e *access-list*. É necessário definir a porta para a rede privada com *ip nat inside* e a porta para o exterior com *ip nat outside*, e com *access-list* permitir os IP's privados que são supostos usar NAT.

A implementação de NAT, *Network Address Translation*, garante o acesso aos computadores da rede interna criada por nós às redes externas. Este processo efetua-se através da tradução dos endereços IP privados, que, caso contrário, não seriam reconhecidos fora da rede. São atribuídos portos únicos para os dispositivos da rede privada e, então, no *router* onde é implementado o NAT, as comunicações para o exterior tomam o IP deste com o porto do dispositivo fonte associado, sendo assim possível associar vários computadores a um único IP. Quando se pretender o inverso, ou seja, que o computador da rede interna receba a resposta, o *router* traduz o porto de destino para o respetivo IP privado.

4.5 Experiência 5: DNS

Esta experiência tem uma disposição idêntica à anterior, com DNS ativado nos Tuxy's. O DNS permite traduzir os nomes de hosts em IP's, sendo necessário para um uso cómodo da internet. Os comandos usados são os presentes em C.6 e anteriores.

Para configurar o DNS num dos Tuxy, edita-se o ficheiro */etc/resolv.conf*. Neste caso, é usado um DNS da FEUP, sendo o seu nome e IP escritos no ficheiro.

O fluxo de pacotes é demonstrado na figura 8, para o caso do host *google.com*. É primeiro requisitado ao DNS configurado o IP do nome do host, respondendo este com o IP. Finalmente, é feito o ping ao IP obtido.

4.6 Experiência 6: Conexões TCP

Esta experiência tem uma disposição idêntica à anterior, sendo o objetivo o de testar a aplicação desenvolvida para transferência FTP e analisar o respetivo tráfego na rede ao transferir um ficheiro fora desta.

A aplicação FTP abre 2 ligações TCP: uma na porta 21 do servidor para o envio de comandos (fig. 9), e outra noutra porta do servidor para a transferência de ficheiros (fig. 10).

Há 3 fases numa ligação TCP:

- o estabelecimento de ligação através dos pacotes SYN (pelo cliente), SYN-ACK (pelo servidor) e ACK (pelo cliente), como comprovado pelas figuras 9 e 10;
- a transferência de dados;
- o término da ligação através dos pacotes FIN-ACK e ACK (fig. 12).

Para evitar perdas de pacotes, o mecanismo ARQ do TCP usa pacotes ACK para o recetor informar o transmissor que recebeu os seus pacotes corretamente. Por exemplo, na figura 11 observa-se que o cliente constantemente envia ACK's ao servidor à medida que o servidor lhe envia o ficheiro por FTP.

TODO:How does the TCP congestion control mechanism work? What are the relevant fields. How did the throughput of the data connection evolve along the time? Is it according the TCP congestion control mechanism?

O número máximo de bytes que se podem transmitir até receber um ACK, a *Congestion Window*, começa baixa mas rapidamente é aumentada por meio do algoritmo *Slow Start* que a duplica por cada ACK, passando para o estado *Congestion Avoidance* assim que chegar ao máximo indicado por um erro, passando a ser mais conservador nos aumentos, ou seja, vai incrementando a *Congestion Window* e divide-a por 2 assim que há um erro.

É possível observar a *Congestion Window* de cada pacote num campo de 2 bytes do TCP header. Assim, nas figuras 13 e 14, medidas respetivamente nos Tuxy1 e Tuxy2, que são as transferências FTP concorrentes para o Tuxy1, apresentam uma subida rápido da *Congestion Window* no início, estabilizando durante o resto da transferência.

Uma transferência em simultâneo na mesma rede tornam ambas mais lentas. Na figura 15 está a transferência para o Tuxy1 e na figura 16 está a transferência para o Tuxy2. Por volta dos 3 segundos no Tuxy1, quando o débito começa a diminuir, é iniciada a transferência para o Tuxy2. Aos 8 segundos no Tuxy2, quando se dá a subida mais alta de débito, acaba a transferência no Tuxy1.

Verifica-se por estes grafos que quando o débito de um aumenta o outro diminui, ou seja, os dois Tuxy competem pelo débito total que a rede lhes fornece.

4.7 Experiência 7 - Implementação de NAT em Linux

Esta experiência tem uma disposição idêntica à anterior, sendo que o Tuxy4, funcionando como router entre o Tuxy1 e o resto, passa a ter NAT implementado, ou seja, o Tuxy1 passa a estar dentro de uma sub-rede privada. Os comandos usados são os presentes em C.8 e anteriores.

O Tuxy4 é agora a interface do Tuxy1 com o resto, estando as suas portas Eth0 ligada ao Tuxy1 e Eth1 ligada ao resto. Observando os pacotes de tráfego UDP, TCP e ICMP na porta Eth0, verifica-se que os pacotes com o Tuxy1 como fonte ou destino têm o seu IP privado. Porém, na porta Eth1, os pacotes que seguem de ou para o Tuxy1 têm o IP do Tuxy4 Eth1. Isto significa que o Tuxy4, agora com NAT, é responsável por traduzir os IP's privados da sua sub-rede para o seu próprio IP nos pacotes que redireccione para fora, e por traduzir o próprio IP para os privados nos pacotes que na realidade se destinam a esses.

5 Conclusão

Este último projeto proposto na unidade curricular de Redes de Computadores exigiu bastante autonomia por parte do grupo, dado que tinha que haver um estudo pessoal do protocolo FTP para o desenvolvimento da aplicação e de como configurar o equipamento de laboratório.

O desenvolvimento da aplicação e a configuração de rede permitiram-nos perceber, por exemplo, como se processam as transferências por FTP e como computadores comunicam entre si pela Internet.

Porém, cremos que o guião deste trabalho poderia ser mais objetivo em relação a como fazer as várias experiências, podendo por exemplo descrever os vários comandos usados para configurar o Switch e o Router da bancada. Contudo, acreditamos que realizámos com sucesso aquilo a que nos propusemos.

A Código da aplicação FTP

A.1 Imagens

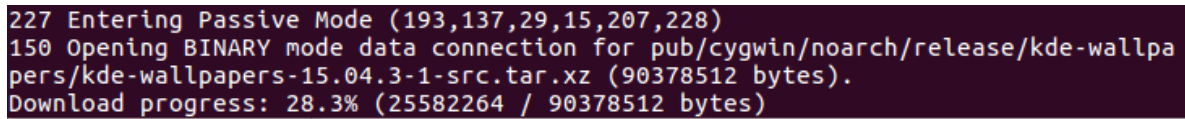


Figura 1: Transferência de um ficheiro por FTP em progresso

A.2 Código

A.2.1 main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #include "Url.h"
5  #include "FtpData.h"
6
7  void printUsage(char progName[]) {
8      printf("USAGE: %s ftp://[<user>:<password>@]<host>/<url-path>",
9             progName);
10 }
11
12 int main(int argc, char *argv[]) {
13     if (argc != 2) {
14         printUsage(argv[0]);
15         exit(1);
16     }
17
18     struct Url url;
19     memset(&url, 0, sizeof(url));
20     parseUrl(&url, argv[1]);
21
22     struct FtpData ftp;
23     initFtpData(&ftp, url.host);
24     setupConnection(&ftp, &url);
25
26     downloadFile(&ftp, url.path);
27
28     closeConnection(&ftp);
29
30     freeUrl(&url);
31
32     return 0;
33 }
```

A.2.2 Url.h

```
1  #ifndef URL_H
```

```

2  #define URL_H
3
4  #include <stdio.h>
5  #include <string.h>
6
7  void logError(char *msg);
8
9  struct Url {
10     char *username;
11     char *password;
12     char *host;
13     char *path;
14 };
15
16 struct LinkIndexes {
17     int colonInd;
18     int atInd;
19     int firstSlashInd;
20 };
21
22 void initLinkInds(struct LinkIndexes *linkInds);
23
24 int getSeparatorInds(struct LinkIndexes *linkInds, const char *link);
25
26
27 int parseUsername(struct Url *url, const char *link, const struct
    LinkIndexes *linkInds);
28
29 int parsePassword(struct Url *url, const char *link, const struct
    LinkIndexes *linkInds);
30
31 int parseLogin(struct Url *url, const char *link, const struct
    LinkIndexes *linkInds);
32
33 int parseHost(struct Url *url, const char *link, const struct
    LinkIndexes *linkInds);
34
35 int parsePath(struct Url *url, const char *link, const struct
    LinkIndexes *linkInds);
36
37 int parseUrl(struct Url *url, char *str);
38
39 void freeUrl(struct Url *url);
40
41 #endif

```

A.2.3 Url.c

```

1  #include "Url.h"
2
3  #include <stdlib.h>
4
5  void logError(char *msg) {
6     printf("ERROR: %s\n", msg);
7 }
8

```

```

9 void initLinkInds(struct LinkIndexes *linkInds) {
10     linkInds->colonInd = -1;
11     linkInds->atInd = -1;
12     linkInds->firstSlashInd = -1;
13 }
14
15 int getSeparatorInds(struct LinkIndexes *linkInds, const char *link) {
16     char *colonAddr = strchr(link, ':');
17     char *atAddr = strchr(link, '@');
18     char *firstSlashAddr = strchr(link, '/');
19
20     if (firstSlashAddr == NULL) {
21         logError("No '/' found between host and path.");
22         return -1;
23     }
24
25     if (atAddr != NULL && colonAddr == NULL) {
26         logError("There must be a ':' if there is a '@'.");
27         return -1;
28     }
29
30     linkInds->colonInd = (colonAddr != NULL) ? (colonAddr - link) : -1;
31     linkInds->atInd = (atAddr != NULL) ? (atAddr - link) : -1;
32     linkInds->firstSlashInd = (firstSlashAddr != NULL) ? (firstSlashAddr
        - link) : -1;
33
34     return 0;
35 }
36
37 int parseUsername(struct Url *url, const char *link, const struct
    LinkIndexes *linkInds) {
38     if (linkInds->atInd == -1) {
39         url->username = malloc(strlen("anonymous") + 1);
40         url->username[0] = 0;
41         strcat(url->username, "anonymous");
42     } else {
43         const int usernameLength = linkInds->colonInd;
44         url->username = malloc(usernameLength + 1);
45         strncpy(url->username, link, usernameLength);
46         url->username[usernameLength] = 0;
47     }
48
49     return 0;
50 }
51
52 int parsePassword(struct Url *url, const char *link, const struct
    LinkIndexes *linkInds) {
53     if (linkInds->atInd == -1) {
54         url->password = malloc(strlen("anonymous") + 1);
55         url->password[0] = 0;
56         strcat(url->password, "anonymous");
57     } else {
58         int passwordLength = linkInds->atInd - linkInds->colonInd - 1;
59         url->password = malloc(passwordLength + 1);
60         strncpy(url->password, link + linkInds->colonInd + 1,
            passwordLength);

```

```

61     url->password[passwordLength] = 0;
62 }
63
64     return 0;
65 }
66
67 int parseLogin(struct Url *url, const char *link, const struct
    LinkIndexes *linkInds) {
68     parseUsername(url, link, linkInds);
69     parsePassword(url, link, linkInds);
70
71     return 0;
72 }
73
74 int parseHost(struct Url *url, const char *link, const struct
    LinkIndexes *linkInds) {
75     int hostLength = -1;
76     if (linkInds->atInd == -1) {
77         hostLength = linkInds->firstSlashInd;
78     } else {
79         hostLength = linkInds->firstSlashInd - linkInds->atInd - 1;
80     }
81     url->host = malloc(hostLength + 1);
82     strncpy(url->host, link + linkInds->atInd + 1, hostLength);
83     url->host[hostLength] = 0;
84
85     return 0;
86 }
87
88 int parsePath(struct Url *url, const char *link, const struct
    LinkIndexes *linkInds) {
89     const int pathLength = strlen(link) - linkInds->firstSlashInd - 1;
90     url->path = malloc(pathLength + 1);
91     strncpy(url->path, link + linkInds->firstSlashInd + 1, pathLength);
92     url->path[pathLength] = 0;
93
94     return 0;
95 }
96
97 int parseUrl(struct Url *url, char *str) {
98     if (strncmp(str, "ftp://", strlen("ftp://")) != 0) {
99         logError("Link must start with 'ftp://'");
100         exit(1);
101     }
102     str += strlen("ftp://");
103
104     struct LinkIndexes linkInds;
105     initLinkInds(&linkInds);
106     getSeparatorInds(&linkInds, str);
107
108     if (parseLogin(url, str, &linkInds) == -1) {
109         return -1;
110     }
111
112     #ifdef DEBUG_PRINTS
113     printf("username is - %s\n", url->username);

```

```

114     printf("password is - %s\n", url->password);
115     #endif
116
117     if (parseHost(url, str, &linkInds) == -1) {
118         return -1;
119     }
120     if (parsePath(url, str, &linkInds) == -1) {
121         return -1;
122     }
123
124     #ifdef DEBUG_PRINTS
125     printf("Host is - %s\n", url->host);
126     printf("Path is - %s\n\n", url->path);
127     #endif
128
129     return 0;
130
131 }
132
133 void freeUrl(struct Url *url) {
134     free(url->username);
135     free(url->password);
136     free(url->host);
137     free(url->path);
138 }

```

A.2.4 FtpData.h

```

1  #ifndef FTP_DATA_H
2  #define FTP_DATA_H
3
4  #include "Url.h"
5
6  #define FTP_CMD_PORT 21
7
8  struct FtpData {
9      char *ipAddress;
10     int dataPort;
11     int cmdSocketFd;
12     int dataSocketFd;
13 };
14
15 /**
16  * Gets IP address from host name.
17  */
18 int initFtpData(struct FtpData *ftpData, const char *hostName);
19
20 /**
21  * Logs in, sets passive mode, opens server data socket.
22  */
23 int setupConnection(struct FtpData *ftpData, const struct Url *url);
24
25 /**
26  * Downloads file.
27  */
28 int downloadFile(struct FtpData *ftpData, const char *filePath);

```

```

29
30 /**
31  * Closes sockets.
32  */
33 void closeConnection(struct FtpData *ftpData);
34
35 #endif // FTP_DATA_H

```

A.2.5 FtpData.c

```

1  #include "FtpData.h"
2  #include <stdlib.h>
3  #include <stdio.h>
4  #include <string.h>
5  #include <unistd.h>
6  #include <netdb.h>
7  #include <sys/socket.h>
8  #include <netinet/in.h>
9  #include <arpa/inet.h>
10
11 #define DiscardUptoOpenParen "%*[^("
12 #define DiscardUptoCloseParen "%*[^)]"
13
14 void logFtpError(char *msg) {
15     printf("ERROR: %s\n", msg);
16 }
17
18
19
20 int msgCode(const char *msg) {
21     int code = -1;
22     char trash[1024];
23     sscanf(msg, "%d%s", &code, trash);
24     return code;
25 }
26
27 int getFileSize(const char *msg) {
28     int fileSize = -1;
29     sscanf(msg, DiscardUptoOpenParen "(%d bytes).", &fileSize);
30     return fileSize;
31 }
32
33 void readFtp(int socketFd, char *buf, const int bufLength, char
    **readData, int *readDataLength) {
34     int bytesRead = -1;
35     memset(buf, 0, bufLength);
36     while (buf[3] != ' ') {
37         memset(buf, 0, bufLength);
38         bytesRead = recv(socketFd, buf, bufLength, MSG_DONTWAIT);
39     }
40     *readData = malloc(bytesRead);
41     memcpy(*readData, buf, bytesRead);
42     *readDataLength = bytesRead;
43
44     //#ifdef DEBUG_PRINTS
45     printf("%s", *readData);

```

```

46     //endif
47 }
48
49 int initFtpData(struct FtpData *ftpData, const char *hostName) {
50     struct hostent *h;
51
52     if ((h = gethostbyname(hostName)) == NULL) {
53         perror("gethostbyname");
54         exit(1);
55     }
56
57     ftpData->ipAddress = inet_ntoa(*(struct in_addr *) h->h_addr));
58     ftpData->dataPort = -1;
59     ftpData->cmdSocketFd = -1;
60     ftpData->dataSocketFd = -1;
61
62     return 0;
63 }
64
65 int openCmdSocket(const struct FtpData *ftpData) {
66     int cmdSocketFd;
67     struct sockaddr_in server_addr;
68
69     bzero((char *) &server_addr, sizeof(server_addr));
70     server_addr.sin_family = AF_INET;
71     server_addr.sin_addr.s_addr = inet_addr(ftpData->ipAddress); //32 bit
72     //Internet address network byte ordered
73     server_addr.sin_port = htons(FTP_CMD_PORT);
74
75     if ((cmdSocketFd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
76         perror("socket()");
77         exit(1);
78     }
79
80     if (connect(cmdSocketFd, (struct sockaddr *) &server_addr,
81               sizeof(server_addr)) < 0) {
82         perror("connect()");
83         exit(1);
84     }
85
86     return cmdSocketFd;
87 }
88
89 int openDataSocket(struct FtpData *ftpData) {
90     int dataSocketFd;
91     struct sockaddr_in server_addr;
92
93     bzero((char *) &server_addr, sizeof(server_addr));
94     server_addr.sin_family = AF_INET;
95     server_addr.sin_addr.s_addr = inet_addr(ftpData->ipAddress); //32 bit
96     //Internet address network byte ordered
97     server_addr.sin_port = htons(ftpData->dataPort);
98
99     if ((dataSocketFd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
100         perror("socket()");
101         closeConnection(ftpData);

```



```

99     exit(1);
100 }
101
102 if(connect(dataSocketFd, (struct sockaddr *) &server_addr,
103     sizeof(server_addr)) < 0){
104     perror("connect()");
105     closeConnection(ftpData);
106     exit(1);
107 }
108 return dataSocketFd;
109 }
110
111 void sendLogin(const struct FtpData *ftpData, const struct Url *url) {
112     char buf[1024];
113
114     const int userMsgLength = strlen("USER ") + strlen(url->username) +
115         strlen("\n");
116     char *userMsg = malloc(userMsgLength + 1);
117     userMsg[0] = 0;
118     strcat(userMsg, "USER ");
119     strcat(userMsg, url->username);
120     strcat(userMsg, "\n");
121
122     send(ftpData->cmdSocketFd, userMsg, strlen(userMsg), 0);
123     char *response = NULL;
124     int responseLength = -1;
125     readFtp(ftpData->cmdSocketFd, buf, 1024, &response, &responseLength);
126     //printf("%s\n", buf);
127     // clear "password required" message
128     if (msgCode(response) == 220) {
129         readFtp(ftpData->cmdSocketFd, buf, 1024, &response, &responseLength);
130     }
131
132     const int passwordLength = strlen("PASS ") + strlen(url->password) +
133         strlen("\n");
134     char *passwordMsg = malloc(passwordLength + 1);
135     passwordMsg[0] = 0;
136     strcat(passwordMsg, "PASS ");
137     strcat(passwordMsg, url->password);
138     strcat(passwordMsg, "\n");
139
140     send(ftpData->cmdSocketFd, passwordMsg, strlen(passwordMsg), 0);
141     response = NULL;
142     responseLength = -1;
143     readFtp(ftpData->cmdSocketFd, buf, 1024, &response, &responseLength);
144     //printf("%s\n", buf);
145 }
146
147 void setPassive(const struct FtpData *ftpData, int *dataPort) {
148     char buf[1024];
149
150     send(ftpData->cmdSocketFd, "PASV\n", strlen("PASV\n"), 0);
151     char *response = NULL;
152     int responseLength = -1;
153     readFtp(ftpData->cmdSocketFd, buf, 1024, &response, &responseLength);

```

```

152     //printf("%s\n", buf);
153
154     int ipPart1, ipPart2, ipPart3, ipPart4, dataPortPart1, dataPortPart2;
155     sscanf(buf, DiscardUptoOpenParen "(%d,%d,%d,%d,%d,%d)",
156           &ipPart1, &ipPart2, &ipPart3, &ipPart4, &dataPortPart1,
157           &dataPortPart2);
158     *dataPort = 256 * dataPortPart1 + dataPortPart2;
159 }
160
161 int setupConnection(struct FtpData *ftpData, const struct Url *url) {
162     ftpData->cmdSocketFd = openCmdSocket(ftpData);
163     sendLogin(ftpData, url);
164
165     setPassive(ftpData, &ftpData->dataPort);
166     ftpData->dataSocketFd = openDataSocket(ftpData);
167
168     return 0;
169 }
170
171 void sendRetr(const struct FtpData *ftpData, const char *filePath, int
172             *fileSize) {
173     char buf[1024];
174
175     const int retrMsgLength = strlen("RETR ") + strlen(filePath) +
176                             strlen("\n");
177     char *retrMsg = malloc(retrMsgLength + 1);
178     retrMsg[0] = 0;
179     strcat(retrMsg, "RETR ");
180     strcat(retrMsg, filePath);
181     strcat(retrMsg, "\n");
182
183     send(ftpData->cmdSocketFd, retrMsg, strlen(retrMsg), 0);
184     char *response = NULL;
185     int responseLength = -1;
186     readFtp(ftpData->cmdSocketFd, buf, 1024, &response, &responseLength);
187     //printf("%s\n", buf);
188     *fileSize = getFileSize(response);
189 }
190
191 char * getFilenameFromPath(const char *filePath) {
192     const char *lastSlashAddr = strrchr(filePath, '/');
193     const int lastSlashInd = lastSlashAddr - filePath;
194     const int filenameLength = strlen(filePath) - lastSlashInd - 1;
195     char *filename = malloc(filenameLength + 1);
196     filename[0] = 0;
197     strcat(filename, lastSlashAddr + 1);
198
199     return filename;
200 }
201
202 void receiveFile(struct FtpData *ftpData, const char *filePath, const
203                int fileSize) {
204     char *filename = getFilenameFromPath(filePath);
205     FILE *fp;
206     if ((fp = fopen(filename, "wb")) == NULL) {
207         logFtpError("Cannot open file '%s' for writing.");

```

```

204     closeConnection(ftpData);
205     exit(1);
206 }
207
208 char buf[1024];
209 int bytesRead = -1;
210 int totalBytesRead = 0;
211 printf("\e[?25l"); // hide cursor
212 while ((bytesRead = recv(ftpData->dataSocketFd, buf, 1024,
213     MSG_DONTWAIT)) != 0) {
214     if (bytesRead == -1) {
215         // perror("recv");
216         continue;
217     }
218     if (fwrite(buf, bytesRead, 1, fp) == 0) {
219         logFtpError("Local file writing failure.");
220     }
221     totalBytesRead += bytesRead;
222     float transferProgress = (float) totalBytesRead / fileSize * 100.0;
223     printf("\rDownload progress: %.1f%% (%d / %d bytes)",
224         transferProgress, totalBytesRead, fileSize);
225 }
226 printf("\e[?25h"); // display cursor
227
228 fclose(fp);
229 free(filename);
230 }
231
232 int downloadFile(struct FtpData *ftpData, const char *filePath) {
233     int fileSize = -1;
234     sendRetr(ftpData, filePath, &fileSize);
235     receiveFile(ftpData, filePath, fileSize);
236
237     return 0;
238 }
239
240 void closeConnection(struct FtpData *ftpData) {
241     if (ftpData->cmdSocketFd != -1) {
242         close(ftpData->cmdSocketFd);
243     }
244     if (ftpData->dataSocketFd != -1) {
245         close(ftpData->dataSocketFd);
246     }
247 }

```

B Logs Wireshark

B.1 Experiências

B.1.1 Experiência 1

No.	Time	Source	Destination	Protocol	Length	Info
4	4.512635	HewlettP_5a:75:bb	Broadcast	ARP	42	Who has 172.16.10.254? Tell 172.16.10.1
5	4.512775	HewlettP_19:09:5c	HewlettP_5a:75:bb	ARP	60	172.16.10.254 is at 00:22:64:19:09:5c
6	4.512786	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x1614, seq=1/256, ttl=64 (reply in 7)
7	4.512941	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x1614, seq=1/256, ttl=64 (request in 6)
8	5.511640	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x1614, seq=2/512, ttl=64 (reply in 9)
9	5.511783	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x1614, seq=2/512, ttl=64 (request in 8)

[Coloring Rule String: arp]

▼ Ethernet II, Src: HewlettP_5a:75:bb (00:21:5a:5a:75:bb), Dst: Broadcast (ff:ff:ff:ff:ff:ff)

- ▼ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
Address: Broadcast (ff:ff:ff:ff:ff:ff)
.... 1. = LG bit: Locally administered address (this is NOT the factory default)
.... 1. = IG bit: Group address (multicast/broadcast)
- ▼ Source: HewlettP_5a:75:bb (00:21:5a:5a:75:bb)
Address: HewlettP_5a:75:bb (00:21:5a:5a:75:bb)
.... 0. = LG bit: Globally unique address (factory default)
.... 0. = IG bit: Individual address (unicast)

Type: ARP (0x0806)

▼ Address Resolution Protocol (request)

Hardware type: Ethernet (1)
Protocol type: IPv4 (0x0800)
Hardware size: 6

0000	ff ff ff ff ff ff 00 21 5a 5a 75 bb 08 06 00 01! ZZu.
0010	08 00 06 04 00 01 00 21 5a 5a 75 bb ac 10 0a 01! ZZu.
0020	00 00 00 00 00 00 ac 10 0a fe

Figura 2: Pacote ARP com endereço MAC desconhecido e tipo selecionado

No.	Time	Source	Destination	Protocol	Length	Info
4	4.512635	HewlettP_5a:75:bb	Broadcast	ARP	42	Who has 172.16.10.254? Tell 172.16.10.1
5	4.512775	HewlettP_19:09:5c	HewlettP_5a:75:bb	ARP	60	172.16.10.254 is at 00:22:64:19:09:5c
→ 6	4.512786	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x1614, seq=1/256, ttl=64 (reply in 7)
← 7	4.512941	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x1614, seq=1/256, ttl=64 (request in 6)
8	5.511640	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x1614, seq=2/512, ttl=64 (reply in 9)
9	5.511783	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x1614, seq=2/512, ttl=64 (request in 8)

.... 0. = IG bit: Individual address (unicast)

Type: IPv4 (0x0800)

▼ Internet Protocol Version 4, Src: 172.16.10.1, Dst: 172.16.10.254

0100 = Version: 4
.... 0101 = Header Length: 20 bytes (5)

▼ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)

0000 00.. = Differentiated Services Codepoint: Default (0)
.... 00.. = Explicit Congestion Notification: Not ECN-Capable Transport (0)

Total Length: 84
Identification: 0x6b88 (27528)

► Flags: 0x02 (Don't Fragment)
Fragment offset: 0
Time to live: 64

Protocol: ICMP (1)

Header checksum: 0x6201 [validation disabled]

0010	00 54 6b 88 40 00 01 62 01 ac 10 0a 01 ac 10	.Tk.@. b.
0020	0a fe 08 00 c2 10 16 14 00 01 25 98 32 5a 00 00 %.2Z..
0030	00 00 fb 14 0e 00 00 00 00 00 10 11 12 13 14 15
0040	16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 !"#%\$
0050	26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35	&'()*+,-./012345

Figura 3: Pacote IP com protocolo ICMP selecionado

→	6	4.512786	172.16.10.1	172.16.10.254	ICMP	98 Echo (ping) request	id=0x1614, seq=1/256, ttl=64 (reply in 7)
←	7	4.512941	172.16.10.254	172.16.10.1	ICMP	98 Echo (ping) reply	id=0x1614, seq=1/256, ttl=64 (request in 6)
	8	5.511640	172.16.10.1	172.16.10.254	ICMP	98 Echo (ping) request	id=0x1614, seq=2/512, ttl=64 (reply in 9)
	9	5.511783	172.16.10.254	172.16.10.1	ICMP	98 Echo (ping) reply	id=0x1614, seq=2/512, ttl=64 (request in 8)
> Frame 6: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0 > Ethernet II, Src: HewlettP_5a:75:bb (00:21:5a:5a:75:bb), Dst: HewlettP_19:09:5c (00:22:64:19:09:5c) > Destination: HewlettP_19:09:5c (00:22:64:19:09:5c) > Source: HewlettP_5a:75:bb (00:21:5a:5a:75:bb) Type: IPv4 (0x0800) > Internet Protocol Version 4, Src: 172.16.10.1, Dst: 172.16.10.254 0100 = Version: 4 0101 = Header Length: 20 bytes (5) > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT) Total Length: 84 Identification: 0x0000 (0x0000) Flags: 0x0000 (0x0000) Window: 0x0000 (0x0000) Checksum: 0x0000 (0x0000) TTL: 64 Protocol: 1 Length: 84 Options: 0x0000 (0x0000)							
0000	00	22	64	19	09	5c	00 21 5a 5a 75 bb 08 00 45 00 . "d.. \. ! ZZu...E.
0010	00	5d	6b	88	40	00	01 62 01 ac 10 0a 01 ac 10 k.@.@. b.....

Figura 4: Pacote IP com o seu tamanho visível

B.1.2 Experiência 3

No.	Time	Source	Destination	Protocol	Length	Info
122	196.261819	G-ProCom_8b:e4:a7	Broadcast	ARP	60	Who has 172.16.50.254? Tell 172.16.50.1
123	196.261840	HewlettP_c3:78:70	G-ProCom_8b:e4:a7	ARP	42	172.16.50.254 is at 00:21:5a:c3:78:70
124	196.261964	172.16.50.1	172.16.51.1	ICMP	98	Echo (ping) request id=0x269e, seq=1/256, ttl=64 (reply in 125)
125	196.262263	172.16.51.1	172.16.50.1	ICMP	98	Echo (ping) reply id=0x269e, seq=1/256, ttl=63 (request in 124)
126	196.494322	Cisco_3a:f6:06	Spanning-tree-(for-...	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00 Cost = 0 Port = 0x8006
127	197.260815	172.16.50.1	172.16.51.1	ICMP	98	Echo (ping) request id=0x269e, seq=2/512, ttl=64 (reply in 128)
128	197.260988	172.16.51.1	172.16.50.1	ICMP	98	Echo (ping) reply id=0x269e, seq=2/512, ttl=63 (request in 127)

Figura 5: Ping do Tuxy1 ao Tuxy2 medido em Tuxy4 Eth0 com tabelas ARP vazias

No.	Time	Source	Destination	Protocol	Length	Info
129	205.246636	Kye_08:d5:b0	Broadcast	ARP	42	Who has 172.16.51.1? Tell 172.16.51.253
130	205.246752	HewlettP_61:2f:d6	Kye_08:d5:b0	ARP	60	172.16.51.1 is at 00:21:5a:61:2f:d6
131	205.246762	172.16.50.1	172.16.51.1	ICMP	98	Echo (ping) request id=0x269e, seq=1/256, ttl=63 (reply in 132)
132	205.246883	172.16.51.1	172.16.50.1	ICMP	98	Echo (ping) reply id=0x269e, seq=1/256, ttl=64 (request in 131)
133	206.245475	172.16.50.1	172.16.51.1	ICMP	98	Echo (ping) request id=0x269e, seq=2/512, ttl=63 (reply in 134)
134	206.245590	172.16.51.1	172.16.50.1	ICMP	98	Echo (ping) reply id=0x269e, seq=2/512, ttl=64 (request in 133)

Figura 6: Ping do Tuxy1 ao Tuxy2 medido em Tuxy4 Eth1 com tabelas ARP vazias

B.1.3 Experiência 4

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	Cisco_7b:d5:03	Spanning-tree-(for-...	STP	60	Conf. Root = 32768/51/00:1e:14:7b:d5:00 Cc
2	1.918454	172.16.51.1	172.16.50.1	ICMP	98	Echo (ping) request id=0x1925, seq=1/256,
3	1.919121	172.16.51.254	172.16.51.1	ICMP	70	Redirect (Redirect for host)
4	1.919483	172.16.50.1	172.16.51.1	ICMP	98	Echo (ping) reply id=0x1925, seq=1/256,
5	2.000712	Cisco_7b:d5:03	Spanning-tree-(for-...	STP	60	Conf. Root = 32768/51/00:1e:14:7b:d5:00 Cc
6	2.919565	172.16.51.1	172.16.50.1	ICMP	98	Echo (ping) request id=0x1925, seq=2/512,
7	2.920244	172.16.51.254	172.16.51.1	ICMP	70	Redirect (Redirect for host)
8	2.920566	172.16.50.1	172.16.51.1	ICMP	98	Echo (ping) reply id=0x1925, seq=2/512,
9	3.229924	Cisco_7b:d5:03	CDP/VTP/DTP/PagP/UD...	CDP	604	Device ID: gnu-sw5 Port ID: FastEthernet0/
10	3.920630	172.16.51.1	172.16.50.1	ICMP	98	Echo (ping) request id=0x1925, seq=3/768,
11	3.921304	172.16.51.254	172.16.51.1	ICMP	70	Redirect (Redirect for host)
12	3.921622	172.16.50.1	172.16.51.1	ICMP	98	Echo (ping) reply id=0x1925, seq=3/768,

> Frame 3: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface 0

> Ethernet II, Src: Cisco_96:ea:d2 (00:1e:7a:96:ea:d2), Dst: HewlettP_5a:7c:e7 (00:21:5a:5a:7c:e7)

> Internet Protocol Version 4, Src: 172.16.51.254, Dst: 172.16.51.1

Internet Control Message Protocol

Type: 5 (Redirect)

Code: 1 (Redirect for host)

Checksum: 0x8881 [correct]

[Checksum Status: Good]

Gateway address: 172.16.51.253

> Internet Protocol Version 4, Src: 172.16.51.1, Dst: 172.16.50.1

Internet Control Message Protocol

0000	00 21 5a 5a 7c e7 00 1e 7a 96 ea d2 08 00 45 00	.!ZZ ... z.....E.
0010	00 38 00 74 00 00 ff 01 fc 30 ac 10 33 fe ac 10	.8.t.... .0..3...
0020	33 01 05 01 88 81 ac 10 33 fd 45 00 00 54 29 eb	3..... 3.E..T).
0030	40 00 3f 01 54 9b ac 10 33 01 ac 10 32 01 08 00	@.?.T... 3...2...
0040	71 49 19 25 00 01	qI.%..

Figura 7:

B.1.4 Experiência 5

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	Cisco_7b:d5:01	Spanning-tree-(for-...	STP	60	Conf. Root = 32768/1/00:1e:14:7b:d5:00 Cost = 0 Port = 0x8001
2	0.000833	Cisco_7b:d5:01	Spanning-tree-(for-...	STP	60	Conf. Root = 32768/1/00:1e:14:7b:d5:00 Cost = 0 Port = 0x8001
3	2.545484	172.16.50.1	172.16.2.1	DNS	70	Standard query 0x5a3c A google.com
4	2.547775	172.16.2.1	172.16.50.1	DNS	222	Standard query response 0x5a3c A google.com A 216.58.211.238 NS ns3.google.com NS ns4.google.co...
5	2.547899	172.16.50.1	216.58.211.238	ICMP	98	Echo (ping) request id=0x569a, seq=1/256, ttl=64 (reply in 6)
6	2.564155	216.58.211.238	172.16.50.1	ICMP	98	Echo (ping) reply id=0x569a, seq=1/256, ttl=51 (request in 5)
7	2.564299	172.16.50.1	172.16.2.1	DNS	87	Standard query 0xc69c PTR 238.211.58.216.in-addr.arpa
8	2.566404	172.16.2.1	172.16.50.1	DNS	303	Standard query response 0xc69c PTR 238.211.58.216.in-addr.arpa PTR mad01s24-in-f14.1e100.net PT...
9	3.549494	172.16.50.1	216.58.211.238	ICMP	98	Echo (ping) request id=0x569a, seq=2/512, ttl=64 (reply in 10)
10	3.565202	216.58.211.238	172.16.50.1	ICMP	98	Echo (ping) reply id=0x569a, seq=2/512, ttl=51 (request in 9)
11	4.005682	Cisco_7b:d5:01	Spanning-tree-(for-...	STP	60	Conf. Root = 32768/1/00:1e:14:7b:d5:00 Cost = 0 Port = 0x8001
12	4.551276	172.16.50.1	216.58.211.238	ICMP	98	Echo (ping) request id=0x569a, seq=3/768, ttl=64 (reply in 13)
13	4.566938	216.58.211.238	172.16.50.1	ICMP	98	Echo (ping) reply id=0x569a, seq=3/768, ttl=51 (request in 12)
14	5.553013	172.16.50.1	216.58.211.238	ICMP	98	Echo (ping) request id=0x569a, seq=4/1024, ttl=64 (reply in 15)
15	5.568760	216.58.211.238	172.16.50.1	ICMP	98	Echo (ping) reply id=0x569a, seq=4/1024, ttl=51 (request in 14)

Figura 8: Ping a google.com com obtenção do respectivo IP através do DNS

B.1.5 Experiência 6

9 5.547356	172.16.30.1	193.137.29.15	TCP	74 53835 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=44438236 TSecr=0 WS=128
10 5.551492	193.137.29.15	172.16.30.1	TCP	74 21 → 53835 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1380 SACK_PERM=1 TSval=628647711 TSecr=4...
11 5.551519	172.16.30.1	193.137.29.15	TCP	66 53835 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=44438237 TSecr=628647711
12 5.551536	172.16.30.1	193.137.29.15	FTP	81 Request: USER anonymous
13 5.554616	193.137.29.15	172.16.30.1	TCP	66 21 → 53835 [ACK] Seq=1 Ack=16 Win=29056 Len=0 TSval=628647712 TSecr=44438237
14 5.557795	193.137.29.15	172.16.30.1	FTP	139 Response: 220-Welcome to the University of Porto's mirror archive (mirrors.up.pt)

Figura 9: Estabelecimento de ligação TCP à porta 21 de um servidor pela aplicação FTP

39 5.666779	172.16.30.1	193.137.29.15	FTP	71 Request: PASV
40 5.669546	193.137.29.15	172.16.30.1	TCP	66 21 → 53835 [ACK] Seq=450 Ack=36 Win=29056 Len=0 TSval=628647741 TSecr=44438266
41 5.669933	193.137.29.15	172.16.30.1	FTP	118 Response: 227 Entering Passive Mode (193,137,29,15,214,254).
42 5.669998	172.16.30.1	193.137.29.15	TCP	74 46260 → 55038 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=44438267 TSecr=0 WS=128
43 5.672075	193.137.29.15	172.16.30.1	TCP	74 55038 → 46260 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1380 SACK_PERM=1 TSval=628647741 TSecr=...
44 5.672101	172.16.30.1	193.137.29.15	TCP	66 46260 → 55038 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=44438268 TSecr=628647741
45 5.672124	172.16.30.1	193.137.29.15	FTP	148 Request: RETR pub/cygwin/noarch/release/kde-wallpapers/kde-wallpapers-15.04.3-1-src.tar.xz
46 5.682127	193.137.29.15	172.16.30.1	FTP-DA..	1434 FTP Data: 1368 bytes
47 5.682147	172.16.30.1	193.137.29.15	TCP	66 46260 → 55038 [ACK] Seq=1 Ack=1369 Win=32128 Len=0 TSval=44438270 TSecr=628647744

Figura 10: Estabelecimento de ligação TCP a uma porta de dados de um servidor pela aplicação FTP

No.	Time	Source	Destination	Protocol	Length	Info
112	5.687462	172.16.30.1	193.137.29.15	TCP	66	46260 → 55038 [ACK] Seq=1 Ack=45145 Win=124800 Len=0 TSval=44438271 TSecr=628647745
113	5.687570	193.137.29.15	172.16.30.1	FTP-DA..	1434	FTP Data: 1368 bytes
114	5.687578	172.16.30.1	193.137.29.15	TCP	66	46260 → 55038 [ACK] Seq=1 Ack=46513 Win=127744 Len=0 TSval=44438271 TSecr=628647745
115	5.687687	193.137.29.15	172.16.30.1	FTP-DA..	1434	FTP Data: 1368 bytes
116	5.687695	172.16.30.1	193.137.29.15	TCP	66	46260 → 55038 [ACK] Seq=1 Ack=47881 Win=130560 Len=0 TSval=44438272 TSecr=628647745
117	5.687804	193.137.29.15	172.16.30.1	FTP-DA..	1434	FTP Data: 1368 bytes
118	5.687814	172.16.30.1	193.137.29.15	TCP	66	46260 → 55038 [ACK] Seq=1 Ack=49249 Win=133504 Len=0 TSval=44438272 TSecr=628647745
119	5.687903	193.137.29.15	172.16.30.1	FTP-DA..	1434	FTP Data: 1368 bytes
120	5.687914	172.16.30.1	193.137.29.15	TCP	66	46260 → 55038 [ACK] Seq=1 Ack=50617 Win=136448 Len=0 TSval=44438272 TSecr=628647745
121	5.688037	193.137.29.15	172.16.30.1	FTP-DA..	1434	FTP Data: 1368 bytes
122	5.688048	172.16.30.1	193.137.29.15	TCP	66	46260 → 55038 [ACK] Seq=1 Ack=51985 Win=139264 Len=0 TSval=44438272 TSecr=628647745
123	5.688495	193.137.29.15	172.16.30.1	FTP-DA..	1434	FTP Data: 1368 bytes
124	5.688505	172.16.30.1	193.137.29.15	TCP	66	46260 → 55038 [ACK] Seq=1 Ack=53353 Win=142208 Len=0 TSval=44438272 TSecr=628647745
125	5.688621	193.137.29.15	172.16.30.1	FTP-DA..	1434	FTP Data: 1368 bytes
126	5.688630	172.16.30.1	193.137.29.15	TCP	66	46260 → 55038 [ACK] Seq=1 Ack=54721 Win=145152 Len=0 TSval=44438272 TSecr=628647745

Figura 11: Envio contínuo de ACK's para o servidor a assinalar a boa receção de pacotes deste

91139	16.237104	193.137.29.15	172.16.30.1	FTP-DA..	290	FTP Data: 224 bytes
91140	16.237117	172.16.30.1	193.137.29.15	TCP	66	46260 → 55038 [ACK] Seq=1 Ack=90378514 Win=481152 Len=0
91141	16.237179	172.16.30.1	193.137.29.15	TCP	66	53835 → 21 [FIN, ACK] Seq=118 Ack=642 Win=30336 Len=0
91142	16.237201	172.16.30.1	193.137.29.15	TCP	66	46260 → 55038 [FIN, ACK] Seq=1 Ack=90378514 Win=482176
91143	16.238701	193.137.29.15	172.16.30.1	TCP	66	55038 → 46260 [ACK] Seq=90378514 Ack=2 Win=29056 Len=0
91144	16.239298	193.137.29.15	172.16.30.1	FTP	90	Response: 226 Transfer complete.
91145	16.239323	172.16.30.1	193.137.29.15	TCP	54	53835 → 21 [RST] Seq=118 Win=0 Len=0
91146	16.240678	193.137.29.15	172.16.30.1	TCP	66	21 → 53835 [FIN, ACK] Seq=666 Ack=119 Win=29056 Len=0
91147	16.240693	172.16.30.1	193.137.29.15	TCP	54	53835 → 21 [RST] Seq=119 Win=0 Len=0

Figura 12: Término de ambas ligações TCP da aplicação FTP

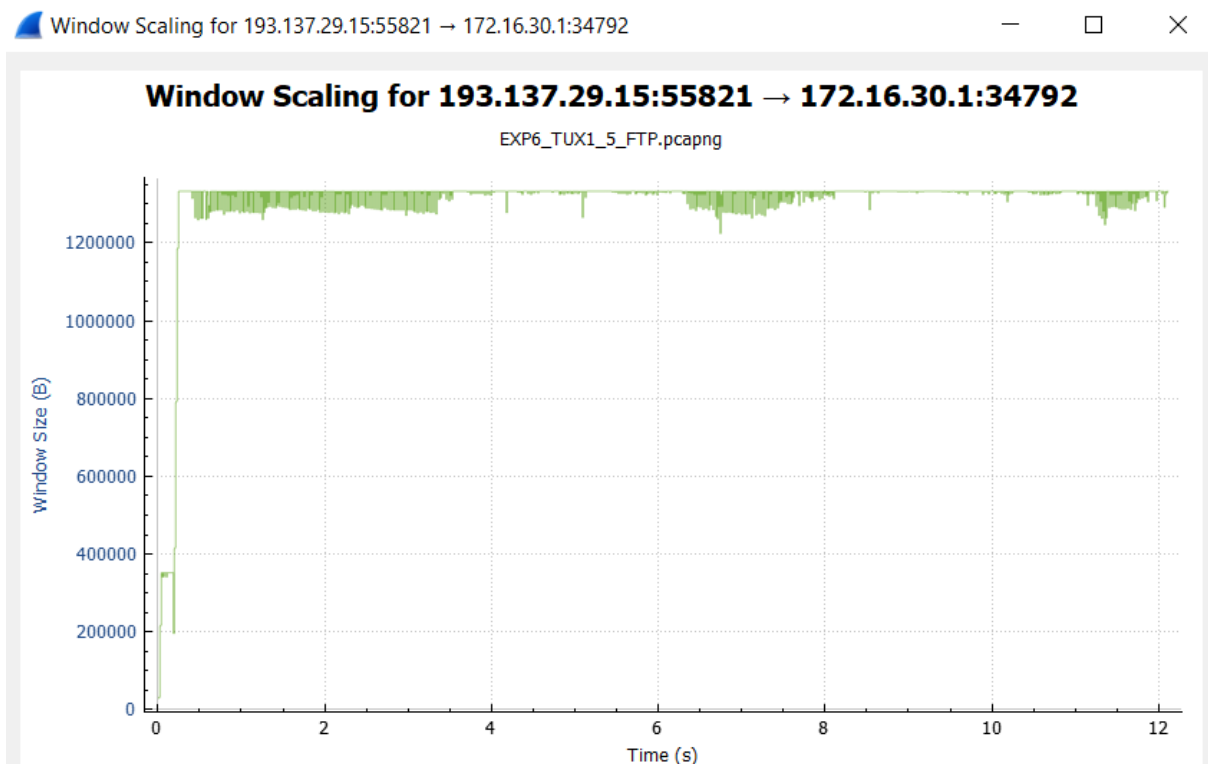


Figura 13: *Congestion window* no Tuxy1 durante uma transferência FTP concorrente com o Tuxy2

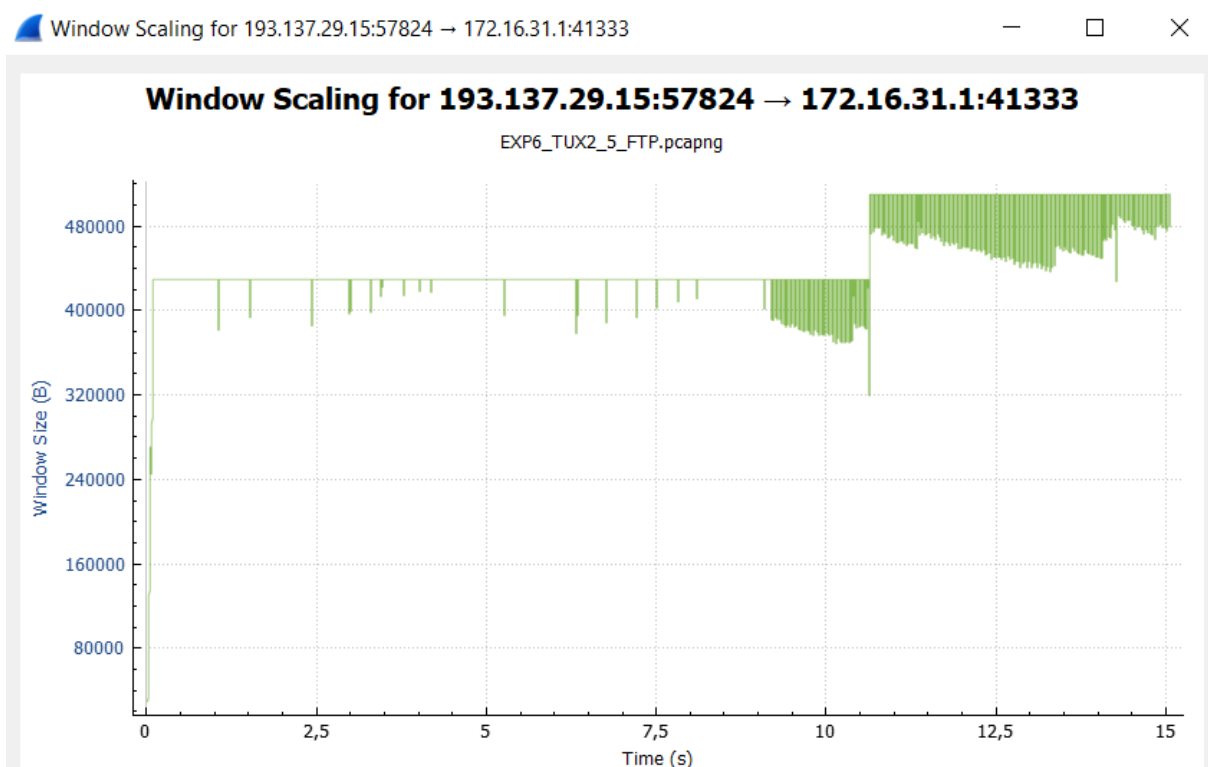


Figura 14: *Congestion window* no Tuxy2 durante uma transferência FTP concorrente com o Tuxy1

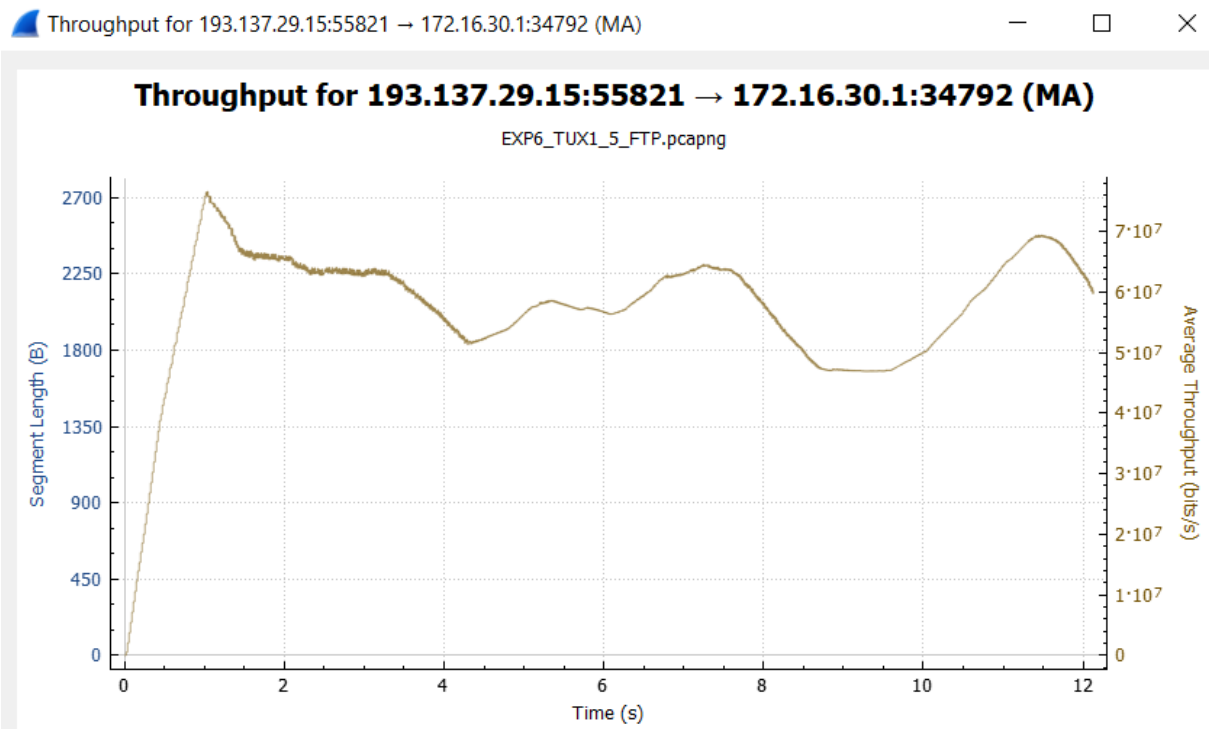


Figura 15: Transferência FTP concorrente de um ficheiro para o Tuxy1 (visível) e para o Tuxy2

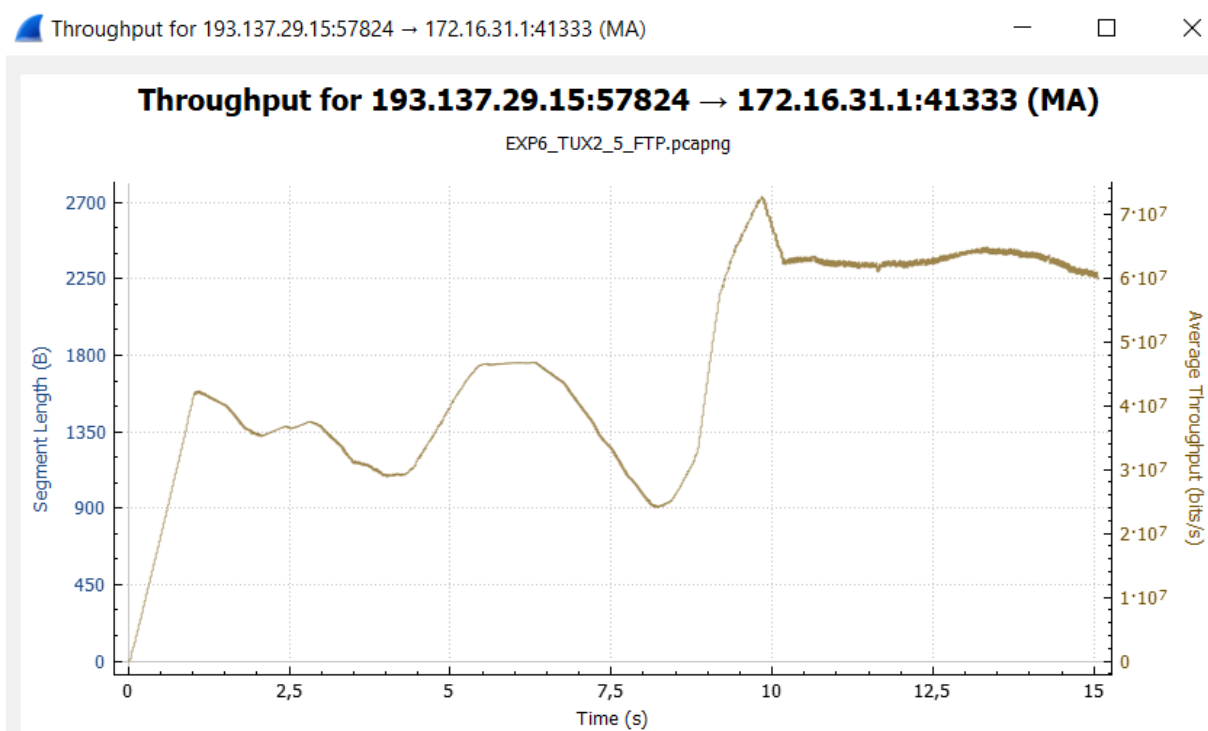


Figura 16: Transferência FTP concorrente de um ficheiro para o Tuxy1 e para o Tuxy2 (visível)

B.1.6 Experiência 7

No.	Time	Source	Destination	Protocol	Length	Info
99690	46.975362	Cisco_b6:8c:02	Spanning-tree-(for-...	STP	60	Conf. Root = 32768/1
99691	47.453568	10.10.0.1	172.16.31.1	ICMP	98	Echo (ping) request
99692	47.453785	172.16.31.1	10.10.0.1	ICMP	98	Echo (ping) reply
99693	47.462365	Cisco_b6:8c:02	CDP/VTP/DTP/PAgP/UD...	CDP	604	Device ID: gnu-sw3
99694	48.453804	10.10.0.1	172.16.31.1	ICMP	98	Echo (ping) request
99695	48.454011	172.16.31.1	10.10.0.1	ICMP	98	Echo (ping) reply

Figura 17: Tráfego ICMP no Tuxy4 Eth0, ping do Tuxy1 ao Tuxy2

No.	Time	Source	Destination	Protocol	Length	Info
99691	50.121019	Cisco_b6:8c:04	Spanning-tree-(for-...	STP	60	Conf. Root = 32768/11
→ 99692	50.464741	172.16.31.253	172.16.31.1	ICMP	98	Echo (ping) request
← 99693	50.464852	172.16.31.1	172.16.31.253	ICMP	98	Echo (ping) reply
99694	51.031918	Cisco_b6:8c:04	Cisco_b6:8c:04	LOOP	60	Reply
99695	51.464699	172.16.31.253	172.16.31.1	ICMP	98	Echo (ping) request
99696	51.464837	172.16.31.1	172.16.31.253	ICMP	98	Echo (ping) reply

Figura 18: Tráfego ICMP no Tuxy4 Eth1, ping do Tuxy1 ao Tuxy2

No.	Time	Source	Destination	Protocol	Length	Info
99648	32.941412	Cisco_b6:8c:02	Spanning-tree-(for-...	STP	60	Conf. Root = 32768/10/
99649	33.901570	10.10.0.1	172.16.31.1	UDP	74	38291 → 33434 Len=32
99650	33.901626	10.10.0.254	10.10.0.1	ICMP	102	Time-to-live exceeded
99651	33.901632	10.10.0.1	172.16.31.1	UDP	74	56220 → 33435 Len=32
99652	33.901641	10.10.0.254	10.10.0.1	ICMP	102	Time-to-live exceeded
99653	33.901644	10.10.0.1	172.16.31.1	UDP	74	35670 → 33436 Len=32
99654	33.901657	10.10.0.254	10.10.0.1	ICMP	102	Time-to-live exceeded
99655	33.901660	10.10.0.1	172.16.31.1	UDP	74	33059 → 33437 Len=32
99656	33.901681	10.10.0.1	172.16.31.1	UDP	74	48417 → 33438 Len=32

Figura 19: Tráfego UDP no Tuxy4 Eth0, traceroute do Tuxy1 ao Tuxy2

No.	Time	Source	Destination	Protocol	Length	Info
99655	34.912743	172.16.31.253	172.16.31.1	UDP	74	47732 → 33442 Len=32
99656	34.912756	172.16.31.253	172.16.31.1	UDP	74	41484 → 33443 Len=32
99657	34.912759	172.16.31.1	172.16.31.253	ICMP	102	Destination unreachable (Port unreachable)
99658	34.912773	172.16.31.253	172.16.31.1	UDP	74	47835 → 33444 Len=32
99659	34.912782	172.16.31.1	172.16.31.253	ICMP	102	Destination unreachable (Port unreachable)
99660	34.912811	172.16.31.253	172.16.31.1	UDP	74	58722 → 33445 Len=32

Figura 20: Tráfego UDP no Tuxy4 Eth1, traceroute do Tuxy1 ao Tuxy2

No.	Time	Source	Destination	Protocol	Length	Info
99600	15.425263	193.137.29.15	10.10.0.1	FTP-DA...	1434	FTP Data: 1368 bytes
99601	15.425381	193.137.29.15	10.10.0.1	FTP-DA...	1434	FTP Data: 1368 bytes
99602	15.425414	10.10.0.1	193.137.29.15	TCP	66	37941 → 53463 [ACK] Seq=1
99603	15.425499	193.137.29.15	10.10.0.1	FTP-DA...	1434	FTP Data: 1368 bytes
99604	15.425615	193.137.29.15	10.10.0.1	FTP-DA...	1434	FTP Data: 1368 bytes
99605	15.425648	10.10.0.1	193.137.29.15	TCP	66	37941 → 53463 [ACK] Seq=1

Figura 21: Tráfego TCP no Tuxy4 Eth0, transferência FTP do exterior para o Tuxy1

99601	16.436182	193.137.29.15	172.16.31.253	FTP-DA...	1434	FTP Data: 1368 bytes
99602	16.436299	193.137.29.15	172.16.31.253	FTP-DA...	1434	FTP Data: 1368 bytes
99603	16.436349	172.16.31.253	193.137.29.15	TCP	66	37941 → 53463 [ACK] Seq=1
99604	16.436416	193.137.29.15	172.16.31.253	FTP-DA...	1434	FTP Data: 1368 bytes
99605	16.436532	193.137.29.15	172.16.31.253	FTP-DA...	1434	FTP Data: 1368 bytes
99606	16.436579	172.16.31.253	193.137.29.15	TCP	66	37941 → 53463 [ACK] Seq=1

Figura 22: Tráfego TCP no Tuxy4 Eth1, transferência FTP do exterior para o Tuxy1

C Comandos de configuração

C.1 Repôr configurações originais

```

1 switch$ configure terminal
2 switch$ no vlan 2-4094
3 switch$ exit
4 switch$ copy flash:tuxY-clean startup-config # Y e o num da bancada
5 switch$ reload
6
7 router$ copy flash:tuxY-clean startup-config # Y e o num da bancada
8 router$ reload

```

C.2 Experiência 1

```

1 tuxy1$ ifconfig eth0 172.16.10.1/24
2 tuxy4$ ifconfig eth0 172.16.10.254/24

```

C.3 Experiência 2

```

1 tuxy2$ ifconfig eth0 172.16.11.1/24
2
3 switch$ configure terminal
4 switch$ vlan 10
5 switch$ vlan 11
6 # Tux1 E0 na porta 1, VLAN 10
7 switch$ interface fastethernet 0/1
8 switch$ switchport mode access
9 switch$ switchport access vlan 10

```

```

10 # Tux4 E0 na porta 2, VLAN 10
11 switch$ interface fastethernet 0/2
12 switch$ switchport mode access
13 switch$ switchport access vlan 10
14 # Tux2 E0 na porta 3, VLAN 11
15 switch$ interface fastethernet 0/3
16 switch$ switchport mode access
17 switch$ switchport access vlan 11
18 switch$ end

```

C.4 Experiência 3

```

1 tuxy1$ route add default gw 172.16.10.254
2
3 tuxy2$ route add -net 172.16.10.0/24 gw 172.16.11.253
4 tuxy2$ echo 1 > /proc/sys/net/ipv4/conf/eth0/accept_redirects
5 tuxy2$ echo 1 > /proc/sys/net/ipv4/conf/all/accept_redirects
6
7 tuxy4$ ifconfig eth1 172.16.11.253/24
8 # Configurar como router
9 tuxy4$ echo 1 > /proc/sys/net/ipv4/ip_forward
10 tuxy4$ echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
11
12 # Tux4 E1 na porta 4, VLAN 11
13 switch$ configure terminal
14 switch$ interface fastethernet 0/4
15 switch$ switchport mode access
16 switch$ switchport access vlan 11
17 switch$ end

```

C.5 Experiência 4

```

1 tuxy2$ route add default gw 172.16.11.254
2
3 tuxy4$ route add default gw 172.16.11.254
4
5 # Router da bancada na porta 5, VLAN 11
6 switch$ configure terminal
7 switch$ interface fastethernet 0/5
8 switch$ switchport mode access
9 switch$ switchport access vlan 11
10 switch$ end
11
12 router$ configure terminal
13 router$ interface gigabitethernet 0/0
14 router$ ip address 172.16.11.254 255.255.255.0
15 router$ no shutdown
16 router$ ip nat inside
17 router$ exit
18 router$ interface gigabitethernet 0/1
19 router$ ip address 172.16.1.19 255.255.255.0
20 router$ no shutdown
21 router$ ip nat outside
22 router$ exit
23 router$ ip nat pool ovrld 172.16.1.19 172.16.1.19 prefix 24
24 router$ ip nat inside source list 1 pool ovrld overload

```

```

25 router$ access-list 1 permit 172.16.10.0 0.0.0.255
26 router$ access-list 1 permit 172.16.11.0 0.0.0.255
27 router$ ip route 172.16.10.0 255.255.255.0 172.16.11.253 # mandar para o
    Tux4 se forem para o Tux1
28 router$ ip route 0.0.0.0 0.0.0.0 172.16.1.254
29 router$ end

```

C.6 Experiência 5

```

1 tuxyY$ gedit /etc/resolv.conf # gedit ou outro editor a escolha
2     search netlab.fe.up.pt
3     nameserver 172.16.1.1

```

C.7 Experiência 6

Não há comandos adicionais nesta experiência.

C.8 Experiência 7

```

1 tuxy4$ iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
2 tuxy4$ iptables -A FORWARD -i eth1 -m state --state NEW,INVALID -j DROP
3
4 # Substituir IP's do Tuxy1 e Tuxy4 E0 por IP's privados
5 tuxy1$ ifconfig eth0 10.10.0.1/24
6 tuxy4$ ifconfig eth0 10.10.0.254/24
7
8 tuxy1$ route add default gw 10.10.0.254

```