

Lab 2

Relatório Final



Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

Grupo 3:

Francisco Rodrigues - 201305627

João Nogueira - up201303882

Marta Lopes - 201208067

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

21 de Dezembro de 2015

1 Sumário

Este relatório tem como objetivo explicar o segundo projeto da Unidade Curricular de Redes de Computadores bem como analisar os resultados obtidos na realização das experiências especificadas no enunciado do mesmo.

Conteúdo

1	Sumário	2
2	Introdução	4
3	Parte 1 - Aplicação de download	4
3.1	Arquitetura	4
3.2	Resultados	5
4	Parte 2 - Configuração de Redes	6
4.1	Configuração de um IP de rede	6
4.2	Configuração de duas redes <i>LAN</i> virtuais num <i>switch</i>	6
4.3	Configuração de um <i>router</i> em <i>Linux</i>	7
4.4	Configuração de duas redes <i>LAN</i> virtuais num <i>switch</i>	7
4.5	Configuração de duas redes <i>LAN</i> virtuais num <i>switch</i>	7
4.6	Configuração de duas redes <i>LAN</i> virtuais num <i>switch</i>	7
5	Conclusões	8
6	Anexos	9
6.1	Headers	9
6.1.1	conection.h	9
6.1.2	url.h	9
6.1.3	utilities.h	9
6.2	*.c files	10
6.2.1	main.c	10
6.2.2	conection.c	11
6.2.3	url.c	16
6.2.4	utilities.c	18
6.3	Makefile	19
6.4	Configuration Scripts	19
6.4.1	Router Configuration	19
6.4.2	Switch Configuration	20
6.4.3	tux1 Configuration	21
6.4.4	tux2 Configuration	21
6.4.5	tux4 Configuration	21
6.5	Wireshark Logs	22
6.5.1	Experiência 1	22
6.5.2	Experiência 2	22
6.5.3	Experiência 3	24
6.5.4	Experiência 4	24
6.5.5	Experiência 5	24
6.5.6	Experiência 6	24

2 Introdução

Este projeto encontra-se dividido em duas grandes partes. Em primeiro lugar, é-nos pedido que desenvolvamos uma aplicação de *download* que proceda à transferência de um ficheiro e que implemente o protocolo *FTP*. Em segundo lugar, é-nos pedido que configuremos e estudemos uma Rede de Computadores seguindo a estrutura das experiências abaixo enumeradas:

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 Parte 1 - Aplicação de download

Como referido anteriormente, a primeira parte deste trabalho consiste numa aplicação que transfere um ficheiro utilizando o protocolo *FTP* descrito no ficheiro RFC959. Como método de *input* é utilizada a sintaxe mostrada na figura abaixo como descrito no ficheiro RFC1738.

```
joao@joao-VirtualBox:~/Documents/Github/RCOM-code/ftpDownloader$ ./download ftp://
PROGRAM: Entering anonymous Mode
ERROR - Wrong paramater -> URL | Expected something like: ftp://[<user>:<password>@]<host>/<url-path>
```

Figura 1: Input

A aplicação desenvolvida permite que seja feito um download em modo anónimo. Para tal basta não colocar os caracteres '@' e ':' e não colocar nome de utilizador e password. Neste caso a aplicação irá assumir o utilizador *anonymous* e a palavra-passe vazia.

3.1 Arquitetura

A *UrlStruct* é a estrutura definida responsável por guardar a informação necessária que depende do *input* do utilizador.

```
1 typedef struct{
2     char * user;
3     char * password;
4     struct hostent * h;
5     char * urlPath;
6     char * hostIp;
7 }urlStruct;
```

urlStruct

Ao correr o programa é chamada a função *getUrlInfo* que é responsável por pegar na *string* que o utilizador forneceu como argumento e interpretar toda a informação necessária.

```
1 #include "utilities.h"
2
3 void getUrlInfo(char * completeString, urlStruct * url);
```

Url Header

Depois de interpretar a informação introduzida pelo utilizador, e após verificar que esta informação é válida é chamada a função *startConection* responsável por ligar o cliente *FTP* ao servidor através de um *socket*. Com a ligação estabelecida é então necessário chamar função *getControl*, responsável por enviar a informação necessária para o *login* e por enviar o comando **PASV**, o que vai permitir que haja comunicação em ambos os sentidos.

```

1 int getControl(FTP * ftp, urlStruct * url, FTP * receiverFtp) {
2     if(sendAndReceiveControl(CMD_USER, ftp, receiverFtp, url) != 0) {
3         return -1;
4     }
5     if (strlen(url->password) != 0) {
6         msg("Entering in anonymous mode");
7     }
8     if(sendAndReceiveControl(CMD_PASS, ftp, receiverFtp, url) != 0) {
9         return -1;
10    }
11    if(sendAndReceiveControl(CMD_PASSV, ftp, receiverFtp, url) != 0) {
12        return -1;
13    }
14    return 0;
15 }

```

getControl

É também feita uma nova conexão através da função *startReceiverConection* para permitir a receção do ficheiro. pedido pelo utilizador. Por fim é enviado o comando **RETR** e recebido o ficheiro a ser guardado. A função *receiveFile* é responsável por enviar o comando, receber o ficheiro e escrevê-lo no disco.

Terminada a receção do ficheiro resta apenas fechar os *sockets* abertos e libertar a memória alocada para terminar o programa.

As funções acima referidas e outras auxiliares estão definidas abaixo, bem como nos anexos.

```

1 #include "url.h"
2
3 typedef struct
4 {
5     int socketFd; // file descriptor to control socket
6     int dataSocketFd; // file descriptor to data socket
7
8     int passvAnswer[6];
9     int port;
10    char ip[MAX_STRING_DEBUG_SIZE];
11 } FTP;
12
13
14 int startConection(urlStruct * url, FTP * ftp);
15 int showResponse(FTP * ftp);
16 int sendAndReceiveControl(int cmd, FTP * ftp, FTP * receiverFtp, urlStruct * url);
17 int receivePassvAnswer(FTP * ftp);
18 int getControl(FTP * ftp, urlStruct * url, FTP * receiverFtp);
19 int startReceiverConection(urlStruct * url, FTP * ftp);
20 int receiveFile(urlStruct * url, FTP * ftp, FTP * receiverFtp);

```

conection.h

Durante o desenvolvimento da aplicação foi implementado um modo de *debug* que é ativo ao alterar a Macro *DEBUG* de 0 para 1. Este modo faz com que haja mais impressões na consola, o que permite controlar com maior exatidão o modo como a aplicação está a funcionar.

```

1 #define DEBUG_MODE 0
2 #define MAX_STRING_DEBUG_SIZE 100
3 #define PORT_FTP 21
4
5 #define CMD_USER 0
6 #define CMD_PASS 1
7 #define CMD_PASSV 2

```

Macros

3.2 Resultados

Esta aplicação foi testada com diversos ficheiros, tanto em modo anónimo como em modo não anónimo. A transferência dos vários ficheiros foi verificada tendo sido o máximo ficheiro testado um ficheiro de vídeo com cerca de 200MB.

Em caso de erro, para além da aplicação terminar é impresso na consola o erro em causa, de modo a que o utilizador tenha o máximo controlo possível sobre o sucedido.

4 Parte 2 - Configuração de Redes

4.1 Configuração de um IP de rede

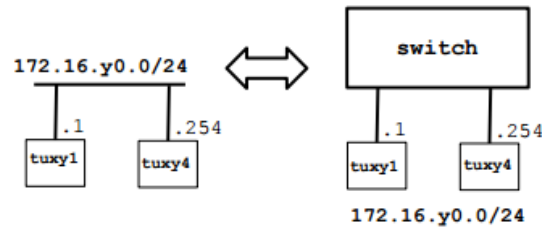


Figura 2: Experiment 1

Esta primeira experiência tem como objetivo configurar duas máquinas numa só rede e compreender o seu funcionamento. Foram então configurados os dois computadores **tux41** e **tux44** para que estes assumissem os endereços de **IP** de **172.16.40.1** e **172.16.40.254**, respetivamente.

Para tal, utilizamos o comando **ifconfig**, atribuindo estes mesmos valores e ativando as portas **eth0** às quais foram ligados os cabos de rede.

Apos a configuração, através do comando **ping** verificou-se que existia a ligação entre as duas máquinas. Após esta verificação foram apagadas todas as entradas na tabela **ARP** através do comando **arp -d 'ip address'**. Por fim repetiu-se o comando **ping** registando o processo através do programa *wireshark*.

Analisando o *log* do *wireshark* da figura 4 nos Anexos, podemos verificar que, tendo apagado as entradas na tabela **ARP** é perguntado à rede qual o endereço **MAC** com um endereço de **IP** igual a **172.16.40.254**. Este computador responde com o seu endereço **MAC** e, a partir de aí, sempre que o primeiro faz um request **ICMP**, este é seguido de uma resposta do segundo. Verifica-se também que, como o endereço **MAC** do segundo se encontra na tabela **ARP** do primeiro, não é necessário haver mais nenhum **broadcast** como o da linha 5 do *log* acima.

4.2 Configuração de duas redes *LAN* virtuais num *switch*

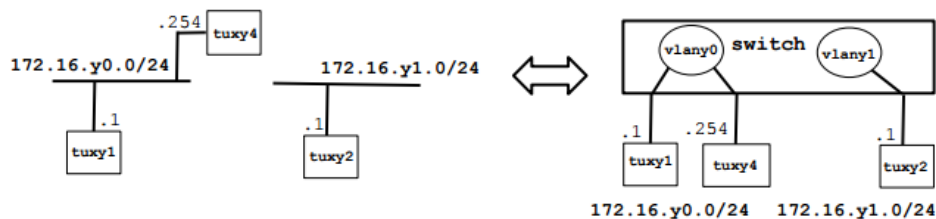


Figura 3: Experiment 2

Esta experiência consiste na criação de duas *vlan's* diferentes:

- *VLAN 40* - *172.16.40.0/24* - à qual pertencem os computadores *tux1* e *tux4*;
- *VLAN 41* - *172.16.41.0/24* - à qual pertence o computador *tux2*.

De notar que no final da configuração das máquinas e do *switch* o computador *tux2* deixará de ter acesso aos computadores que pertencem à rede *VLAN 40*, à qual não pertence.

Para configurar os computadores da forma referida basta proceder do mesmo modo que se procedeu para a primeira experiência, mas desta vez fazê-lo no computador *tux2* e atribuindo **172.16.41.1** como endereço de *IP* (tendo em conta que os outros dois computadores continuam com a configuração da primeira experiência).

Para configurar o *switch* acede-se à sua consola através da aplicação *gkterm* e corre-se os comandos especificados em anexo.

Estando os dois computadores na mesma rede, então, ao fazer **ping** do *tux1* para o *tux4*, não é enviado o pacote *ARP* para saber o endereço *MAC*, como se pode verificar na figura 5.

Pode também verificar-se, na análise das figuras 6, 7, 8, 9, 10 e 11 que cada *vlan* tem um *broadcast domain* diferente, tendo em conta que o *tux2* não detetou o pacote enviado, como tal, conclui-se que a configuração destas rede foi feita da forma correta pois verifica-se a falta de comunicação entre as redes, como foi referido no início desta experiência. Nas primeiras três figuras foi feito um *broadcast* na *vlan 41* enquanto que nas últimas três foi feito na *vlan 40*.

4.3 Configuração de um *router* em *Linux*

4.4 Configuração de duas redes *LAN* virtuais num *switch*

4.5 Configuração de duas redes *LAN* virtuais num *switch*

4.6 Configuração de duas redes *LAN* virtuais num *switch*

5 Conclusões

6 Anexos

6.1 Headers

6.1.1 conection.h

```
1 #include "url.h"
2
3 typedef struct
4 {
5     int socketFd; // file descriptor to control socket
6     int dataSocketFd; // file descriptor to data socket
7
8     int passvAnswer[6];
9     int port;
10    char ip[MAX_STRING_DEBUG_SIZE];
11 } FTP;
12
13
14 int startConection(urlStruct * url, FTP * ftp);
15 int showResponse(FTP * ftp);
16 int sendAndReceiveControl(int cmd, FTP * ftp, FTP * receiverFtp, urlStruct *
    url);
17 int receivePassvAnswer(FTP * ftp);
18 int getControl(FTP * ftp, urlStruct * url, FTP * receiverFTP);
19 int startReceiverConection(urlStruct * url, FTP * ftp);
20 int receiveFile(urlStruct * url, FTP * ftp, FTP * receiverFtp);
```

Anexo 1 - conection.h

6.1.2 url.h

```
1 #include "utilities.h"
2
3 void getUrlInfo(char * completeString, urlStruct * url);
```

Anexo 2 - url.h

6.1.3 utilities.h

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <errno.h>
4 #include <netdb.h>
5 #include <sys/types.h>
6 #include <netinet/in.h>
7 #include <string.h>
8 #include <unistd.h>
9 #include <signal.h>
10 #include <sys/types.h>
11 #include <sys/socket.h>
12 #include <arpa/inet.h>
13
14 #define DEBUG_MODE 0
```

```

15 #define MAX_STRING_DEBUG_SIZE 100
16 #define PORT_FTP 21
17
18 #define CMD_USER 0
19 #define CMD_PASS 1
20 #define CMD_PASSV 2
21
22
23 typedef struct{
24     char * user;
25     char * password;
26     struct hostent * h;
27     char * urlPath;
28     char * hostIp;
29 }urlStruct;
30
31 void debug(char * msg1, char * msg);
32 void msg(char * m);
33 void stringMsg(char * m, char * m2);
34 void getName(char * url, char ** filename);
35 void responseMsg(char * m, char * m2);

```

Anexo 3 - utilities.h

6.2 *.c files

6.2.1 main.c

[illegible]

```

28
29     if (startReceiverConection(receiverUrl, &receiverFtp) < 0) {
30         msg("Error starting receiver connection");
31         return -1;
32     }
33
34     if (receiveFile(url, &ftp, &receiverFtp) < 0) {
35         msg("Error receiving file");
36         return -1;
37     }
38
39     close(receiverFtp.socketFd);
40     close(ftp.socketFd);
41     msg("Sockets Closed, Terminating...");
42
43     free(url->user);
44     free(url->password);
45     free(url->urlPath);
46     free(url);
47
48     return 0;
49 }

```

Anexo 4 - main.c

6.2.2 conection.c

```

1  #include "conection.h"
2
3  int startConection(urlStruct * url, FTP * ftp) {
4      debug("##### DEBUG START CONNECTION #####", "
5          BEGIN");
6      int socket_fd;
7      struct sockaddr_in server_addr;
8
9      // Configuring server address
10     bzero((char*)&server_addr, sizeof(server_addr));
11     server_addr.sin_family = AF_INET;
12     server_addr.sin_addr.s_addr = inet_addr(url->hostIp);           // 32
13     // bit Internet address network byte ordered
14     server_addr.sin_port = htons(PORT_FTP);                       // server
15     // TCP port (21) must be network byte ordered
16
17     // Opening the control TCP socket
18     socket_fd = socket(AF_INET, SOCK_STREAM, 0);
19     if(socket_fd < 0) {
20         printf("Error opening control TCP socket\n");
21         return -1;
22     } else {
23         char temp[MAX_STRING_DEBUG_SIZE];
24         sprintf(temp, "%d", socket_fd);
25         debug("Opened control TCP socket with Fd      ", temp);
26     }
27
28     // Connecting to the server...
29     int connectRet;

```

```

27     if((connectRet = connect(socket_fd, (struct sockaddr *)&server_addr, sizeof
28         (server_addr))) < 0) {
29         perror("connect()");
30         printf("Error connecting to the server to open the control connection\n
31             ");
32         return -1;
33     } else {
34         char temp[MAX_STRING_DEBUG_SIZE];
35         sprintf(temp, "%d", connectRet);
36         debug("Connected to server with return      ", temp);
37     }
38
39     ftp->socketFd = socket_fd;
40
41     msg("Connected");
42     debug("##### DEBUG START CONNECTION #####", "
43         END");
44
45     return 0;
46 }
47
48 int showResponse(FTP * ftp) {
49     char answerFromServer[MAX_STRING_DEBUG_SIZE] = "";
50     if(read(ftp->socketFd, answerFromServer, MAX_STRING_DEBUG_SIZE) > 0) {
51         responseMsg("Server response", answerFromServer);
52         return 0;
53     }
54     msg("Could not read response form server");
55     return -1;
56 }
57
58 int receivePassvAnswer(FTP * ftp) {
59     char passvAnswer[MAX_STRING_DEBUG_SIZE];
60     if(read(ftp->socketFd, passvAnswer, MAX_STRING_DEBUG_SIZE) > 0) {
61         responseMsg("Server response", passvAnswer);
62         if(6 != sscanf(passvAnswer, "%*[^()(%d,%d,%d,%d,%d,%d)\n", &(ftp->
63             passvAnswer[0]), &(ftp->passvAnswer[1]), &(ftp->passvAnswer[2]), &(
64             ftp->passvAnswer[3]), &(ftp->passvAnswer[4]), &(ftp->passvAnswer[5])
65             ))
66         {
67             stringMsg("Could not read the 6 bytes from the server response",
68                 passvAnswer);
69             return -1;
70         }
71         return 0;
72     }
73
74     msg("Could not read response form server socket - PASSV");
75     return -1;
76 }
77
78 int getControl(FTP * ftp, urlStruct * url, FTP * receiverFtp) {
79     if(sendAndReceiveControl(CMD_USER, ftp, receiverFtp, url) != 0) {
80         return -1;
81     }
82 }

```

```

76     if (strlen(url->password) != 0) {
77         msg("Entering in anonymous mode");
78     }
79
80     if(sendAndReceiveControl(CMD_PASS, ftp, receiverFtp, url) != 0) {
81         return -1;
82     }
83
84     if(sendAndReceiveControl(CMD_PASSV, ftp, receiverFtp, url) != 0) {
85         return -1;
86     }
87
88     return 0;
89 }
90
91
92 int sendAndReceiveControl(int cmdSelector, FTP * ftp, FTP * receiverFtp,
93     urlStruct * url) {
94     debug("##### DEBUG SEND AND RECEIVE CONTROL
95         #####", "BEGIN");
96     char cmd[MAX_STRING_DEBUG_SIZE];
97     switch(cmdSelector) {
98         case CMD_USER:
99             strcpy(cmd, "user \0");
100             strcat(cmd, url->user);
101             break;
102         case CMD_PASS:
103             strcpy(cmd, "pass \0");
104             strcat(cmd, url->password);
105             break;
106         case CMD_PASSV:
107             strcpy(cmd, "pasv \0");
108             break;
109         default:
110             break;
111     }
112     strcat(cmd, "\n");
113     debug("Command to Send", cmd);
114     if(write(ftp->socketFd, cmd, strlen(cmd)) < 0) {
115         perror("write()");
116         return -1;
117     } else {
118         debug("Command Sent", cmd);
119     }
120     sleep(1);
121     switch(cmdSelector) {
122         case CMD_PASSV:
123             if(0 == receivePassvAnswer(ftp)) {
124                 // Parse new IP address
125                 receiverFtp->port = ftp->passvAnswer[4] * 256 + ftp->
                    passvAnswer[5];
126                 memset(receiverFtp->ip, 0, MAX_STRING_DEBUG_SIZE); // clearing
                    the array, "just in case"
127                 sprintf(receiverFtp->ip, "%d.%d.%d.%d", ftp->passvAnswer[0],
                    ftp->passvAnswer[1], ftp->passvAnswer[2], ftp->passvAnswer
                    [3]);

```

```

126         stringMsg("IP address to receive file", (char *) &receiverFtp->
127             ip);
128         char temp[MAX_STRING_DEBUG_SIZE];
129         sprintf(temp, "%d", receiverFtp->port);
130         stringMsg("Port to receive file", temp);
131
132         debug("##### DEBUG SEND AND RECEIVE CONTROL
133             #####", "END");
134         return 0;
135     }
136
137     msg("Could not receive the 'passvAnswer' response");
138     return -1;
139     default:
140         showResponse(ftp);
141         break;
142 }
143
144 int startReceiverConection(urlStruct * url, FTP * ftp) {
145     debug("##### DEBUG START RECEIVER CONECTION
146         #####", "BEGIN");
147     struct sockaddr_in server_addr;
148     char *host_ip;
149     int socket_fd;
150
151     if (NULL == (url->h = gethostbyname(ftp->ip))) {
152         msg("Could not get host");
153         return -1;
154     }
155
156     host_ip = inet_ntoa(*(struct in_addr *)url->h->h_addr));
157
158     stringMsg("Host name ", url->h->h_name);
159     stringMsg("IP Address ", host_ip);
160
161     // Configuring server address
162     bzero((char*)&server_addr, sizeof(server_addr));
163     server_addr.sin_family = AF_INET;
164     server_addr.sin_addr.s_addr = inet_addr(host_ip); //32 bit Internet
165         address network byte ordered
166     server_addr.sin_port = htons(ftp->port); //server TCP port
167         must be network byte ordered | this is the new port received from the
168         control TCP connection!
169
170     // Opening the receiver TCP socket
171     socket_fd = socket(AF_INET, SOCK_STREAM, 0);
172     if (socket_fd < 0) {
173         msg("Error opening receiver TCP socket");
174         return -1;
175     }
176
177     int connectRet;

```

```

175     if((connectRet = connect(socket_fd, (struct sockaddr *)&server_addr, sizeof
176         (server_addr))) < 0) {
177         perror("connect()");
178         msg("Error connecting to the server to open the control connection");
179         return -1;
180     } else {
181         char temp[MAX_STRING_DEBUG_SIZE];
182         sprintf(temp, "%d", connectRet);
183         debug("Connected to server with return      ", temp);
184     }
185
186     ftp->socketFd = socket_fd;
187     ftp->dataSocketFd = socket_fd;
188     msg("Connected to receiver");
189     debug("##### DEBUG START RECEIVER CONECTION
190         #####", "END");
191
192     return 0;
193 }
194
195 int receiveFile(urlStruct * url, FTP * ftp, FTP * receiverFtp) {
196     debug("##### DEBUG RECEIVE FILE #####", "
197         BEGIN");
198     msg("Receiving File...");
199     char cmd[MAX_STRING_DEBUG_SIZE] = "";
200     strcpy(cmd, "retr ");
201     strcat(cmd, url->urlPath);
202     strcat(cmd, "\n");
203     stringMsg("Command to be sent", cmd);
204     if(write(ftp->socketFd, cmd, strlen(cmd)) < 0){
205         msg("ERROR - retr command could not be sent");
206         return -1;
207     }
208     stringMsg("Command sent", cmd);
209
210     char * filename;
211     getName(url->urlPath, &filename);
212     debug("Filename is", filename);
213
214     FILE* file;
215     int bytes;
216
217     if (!(file = fopen(filename, "w"))) {
218         msg("ERROR: Cannot open file.");
219         return -1;
220     }
221
222     char temp[MAX_STRING_DEBUG_SIZE];
223     sprintf(temp, "%d", receiverFtp->dataSocketFd);
224     debug("Receiver Data Socket fd", temp);
225
226     char buf[1024];
227     while ((bytes = read(receiverFtp->dataSocketFd, buf, sizeof(buf)))) {
228         if (bytes < 0) {
229             msg("ERROR: Nothing was received.");

```

```

228         return -1;
229     }
230
231     if ((bytes = fwrite(buf, bytes, 1, file)) < 0) {
232         msg("ERROR: Cannot write data in file.\n");
233         return -1;
234     }
235     debug("In read cycle", "Read 1 byte");
236 }
237
238 fclose(file);
239
240 msg("File received");
241 debug("##### DEBUG RECEIVE FILE #####", "END");
242
243 return 0;
244 }

```

Anexo 5 - conection.c

6.2.3 url.c

```

1  #include "url.h"
2
3  void getUrlInfo(char * completeString, urlStruct * url) {
4      debug("##### DEBUG URL INFO #####", "BEGIN");
5
6      if(strncmp(completeString, "ftp://", 6)) {
7          printf("Wrong Url on argument, expected begining like: 'ftp://'\n");
8          exit(1);
9      }
10     //##### debug code #####
11     char debugString[MAX_STRING_DEBUG_SIZE];
12     char debugString_2[MAX_STRING_DEBUG_SIZE];
13     char debugString_3[MAX_STRING_DEBUG_SIZE];
14     char debugString_4[MAX_STRING_DEBUG_SIZE];
15     char debugString_5[MAX_STRING_DEBUG_SIZE];
16     //##### debug code #####
17
18     char * at = strchr(completeString, '@');
19     if (at == NULL) {
20         msg("Entering anonymous Mode");
21     }
22
23     char * toTwoPoints = strchr(completeString + 6, ':');
24     char * slashAfterAt = strchr(completeString + 7, '/');
25
26     if (slashAfterAt == NULL) {
27         printf("ERROR - Wrong paramater -> URL | Expected something like: ftp
28             ://[<user>:<password>@]<host>/<url-path>\n");
29         exit(1);
30     }
31
32     int lengthOfUserAndPassword;
33     int lengthOfUser;

```



```

33     int lengthOfPassword;
34     int lengthOfHost;
35     int lengthOfUrlPath;
36
37     if (at != NULL) {
38         lengthOfUserAndPassword = (int) (at - completeString - 6);
39         lengthOfUser = (int) (toTwoPoints - completeString - 6);
40         lengthOfPassword = lengthOfUserAndPassword - lengthOfUser - 1;
41         lengthOfHost = (int) (slashAfterAt - at - 1);
42         lengthOfUrlPath = strlen(completeString) - (9 + lengthOfUser +
            lengthOfPassword + lengthOfHost);
43     } else {
44         lengthOfUserAndPassword = 0;
45         lengthOfUser = (int) 0;
46         lengthOfPassword = 0;
47         lengthOfHost = (int) (slashAfterAt - completeString - 6);
48         lengthOfUrlPath = strlen(completeString) - (7 + lengthOfUser +
            lengthOfPassword + lengthOfHost);
49     }
50
51     if (lengthOfHost <= 0 || lengthOfUrlPath <= 0) {
52         printf("ERROR - Wrong paramater -> URL | Expected something like: ftp
            ://[<user>:<password>@]<host>/<url-path>\n");
53         exit(1);
54     }
55
56     //##### debug code #####
57     sprintf(debugString, "%d", lengthOfUser);
58     sprintf(debugString_2, "%d", lengthOfPassword);
59     sprintf(debugString_3, "%d", lengthOfHost);
60     sprintf(debugString_5, "%d", lengthOfUrlPath);
61     debug("Number of characters of the User ", debugString);
62     debug("Number of characters of the Password ", debugString_2);
63     debug("Number of characters of the Host ", debugString_3);
64     debug("Number of characters of the Url Path ", debugString_5);
65     //##### debug code #####
66
67     url->password = malloc(sizeof(char) * lengthOfPassword);
68     url->urlPath = malloc(sizeof(char) * lengthOfUrlPath);
69     char hostTemp[MAX_STRING_DEBUG_SIZE];
70     if (at != NULL) {
71         url->user = malloc(sizeof(char) * lengthOfUser);
72         strncpy(url->user, completeString + 6, lengthOfUser);
73         strncpy(url->password, completeString + lengthOfUser + 7,
            lengthOfPassword);
74         strncpy(hostTemp, at + 1, lengthOfHost);
75     } else {
76         url->user = malloc(sizeof(char) * strlen("anonymous"));
77         strncpy(url->user, "anonymous", strlen("anonymous"));
78         strncpy(hostTemp, completeString + 6, lengthOfHost);
79     }
80     strncpy(url->urlPath, slashAfterAt + 1, lengthOfUrlPath);
81     hostTemp[lengthOfHost] = '\0';
82
83     //##### debug code #####
84     debug("User field ", url->user);

```

```

85     debug("Password field", "<password>");
86     debug("Host field", hostTemp);
87     debug("Url Path field", url->urlPath);
88     //##### debug code #####
89
90     if ((url->h=gethostbyname(hostTemp)) == NULL) {
91         perror("gethostbyname");
92         exit(1);
93     }
94
95     sprintf(debugString_4, "%s", inet_ntoa(*(struct in_addr *)url->h->h_addr));
96     url->hostIp = malloc(sizeof(char) * strlen(debugString_4));
97     strncpy(url->hostIp, debugString_4, strlen(debugString_4));
98
99     //##### debug code #####
100    debug("Host name", url->h->h_name);
101    debug("IP Address", url->hostIp);
102    //##### debug code #####
103
104    debug("##### DEBUG URL INFO #####", "END");
105    return;
106 }

```

Anexo 6 - url.c

6.2.4 utilities.c

```

1  #include "utilities.h"
2
3  void debug(char * msg1, char * msg) {
4      if (DEBUG_MODE == 1) {
5          printf("DEBUG: %s: %s\n", msg1, msg);
6      }
7  }
8
9  void msg(char * m) {
10     printf("PROGRAM: %s\n", m);
11 }
12
13 void stringMsg(char * m, char * m2) {
14     printf("PROGRAM: %s: %s\n", m, m2);
15 }
16
17 void getName(char * url, char ** filename) {
18     char * temp = url;
19     int i = 0;
20     while(temp != NULL) {
21         if(i > 0)
22             *filename = temp + 1;
23         else
24             *filename = temp;
25         temp = strchr(*filename, '/');
26         i++;
27     }
28 }

```

```

29
30 void responseMsg(char * m, char * m2) {
31     printf("PROGRAM: %s\n%sRESPONSE END\n", m, m2);
32 }

```

Anexo 7 - utilities.c

6.3 Makefile

```

1 all: main.c utilities.c url.c conection.c
2     gcc -Wall -o download main.c utilities.c url.c conection.c

```

Anexo 8 - Makefile

6.4 Configuration Scripts

6.4.1 Router Configuration

```

1 conf t
2 interface gigabitethernet 0/0
3 ip address 172.16.11.254 255.255.255.0
4 no shutdown
5 ip nat inside
6 exit
7 interface gigabitethernet 0/1
8 ip address 172.16.1.19 255.255.255.0
9 no shutdown
10 ip nat outside
11 exit
12
13 ip nat pool ovrlld 172.16.1.19 172.16.1.19 prefix 24
14 ip nat inside source list 1 pool ovrlld overload
15 access-list 1 permit 172.16.10.0 0.0.0.255
16 access-list 1 permit 172.16.11.0 0.0.0.255
17 ip route 0.0.0.0 0.0.0.0 172.16.1.254
18 ip route 172.16.10.0 255.255.255.0 172.16.11.253
19 end

```

Anexo 9 - Router Configuration

6.4.2 Switch Configuration

```
1  conf t
2  vlan 10
3  end
4
5  conf t
6  vlan 11
7  end
8
9  conf t
10 interface fastethernet 0/1
11 switchport mode access
12 switchport access vlan 10
13 end
14
15 conf t
16 interface fastethernet 0/4
17 switchport mode access
18 switchport access vlan 10
19 end
20
21 conf t
22 interface fastethernet 0/2
23 switchport mode access
24 switchport access vlan 11
25 end
26
27 conf t
28 interface fastethernet 0/5
29 switchport mode access
30 switchport access vlan 11
31 end
32
33 conf t
34 interface gigabitethernet 0/1
35 switchport mode access
36 switchport access vlan 11
37 end
```

Anexo 10 - Switch Configuration

6.4.3 tux1 Configuration

```
1 #!/bin/bash
2
3 ifconfig eth0 up
4 ifconfig eth0 172.16.40.1/24
5
6 route add default gw 172.16.40.254
```

Anexo 11 - tux1 Final Configuration

6.4.4 tux2 Configuration

```
1 #!/bin/bash
2
3 ifconfig eth0 up
4 ifconfig eth0 172.16.11.1/24
5
6 ifconfig eth1 down
7 ifconfig eth2 down
8
9 route add default gw 172.16.11.254
10 route add -net 172.16.10.0/24 gw 172.16.11.253
```

Anexo 12 - tux2 Final Configuration

6.4.5 tux4 Configuration

```
1 #!/bin/bash
2
3 ifconfig eth0 up
4 ifconfig eth0 172.16.40.254/24
5
6 ifconfig eth1 up
7 ifconfig eth1 172.16.41.253/24
8
9 route add default gw 172.16.41.254
10
11 echo 1 > /proc/sys/net/ipv4/ip_forward
12 echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
```

Anexo 13 - tux4 Final Configuration

6.5 Wireshark Logs

6.5.1 Experiência 1

3	4.009607	CiscoInc_d4:1c:...	Spanning-tree-...	STP	60	Conf. Root = 32768/1/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
4	6.014517	CiscoInc_d4:1c:...	Spanning-tree-...	STP	60	Conf. Root = 32768/1/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
5	6.330459	G-ProCom_8c:af:...	Broadcast	ARP	42	Who has 172.16.40.254? Tell 172.16.40.1
6	6.330803	HewlettP_5a:7b:...	G-ProCom_8c:af:...	ARP	60	172.16.40.254 is at 00:21:5a:5a:7b:ea
7	6.330822	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x0f19, seq=1/256, ttl=64 (reply in 8)
8	6.331078	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x0f19, seq=1/256, ttl=64 (request in 7)
9	7.329464	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x0f19, seq=2/512, ttl=64 (reply in 10)
10	7.329678	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x0f19, seq=2/512, ttl=64 (request in 9)

Figura 4: Experiment 1 - log

6.5.2 Experiência 2

8	10.913044	CiscoInc_d4:1c:...	Spanning-tree-...	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
9	12.917839	CiscoInc_d4:1c:...	Spanning-tree-...	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
10	13.017550	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x10eb, seq=1/256, ttl=64 (reply in 11)
11	13.017761	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x10eb, seq=1/256, ttl=64 (request in 10)
12	13.706702	CiscoInc_d4:1c:...	CiscoInc_d4:1c:...	LOOP	60	Reply
13	14.016555	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x10eb, seq=2/512, ttl=64 (reply in 14)
14	14.016923	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x10eb, seq=2/512, ttl=64 (request in 13)
15	14.922868	CiscoInc_d4:1c:...	Spanning-tree-...	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
16	15.015557	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x10eb, seq=3/768, ttl=64 (reply in 17)
17	15.015821	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x10eb, seq=3/768, ttl=64 (request in 16)
18	16.014559	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x10eb, seq=4/1024, ttl=64 (reply in 19)
19	16.014805	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x10eb, seq=4/1024, ttl=64 (request in 18)
20	16.927594	CiscoInc_d4:1c:...	Spanning-tree-...	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
21	17.014059	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x10eb, seq=5/1280, ttl=64 (reply in 22)
22	17.014269	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x10eb, seq=5/1280, ttl=64 (request in 21)
23	18.014063	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x10eb, seq=6/1536, ttl=64 (reply in 24)
24	18.014303	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x10eb, seq=6/1536, ttl=64 (request in 23)

Figura 5: Experiment 2 - Point 6

30	46.111027	CiscoInc_d4:1c:...	Spanning-tree-...	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
31	47.179101	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1154, seq=1/256, ttl=64 (no response found!)
32	48.115818	CiscoInc_d4:1c:...	Spanning-tree-...	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
33	48.186156	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1154, seq=2/512, ttl=64 (no response found!)
34	49.194152	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1154, seq=3/768, ttl=64 (no response found!)
35	50.120827	CiscoInc_d4:1c:...	Spanning-tree-...	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
36	50.202147	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1154, seq=4/1024, ttl=64 (no response found!)
37	51.210165	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1154, seq=5/1280, ttl=64 (no response found!)
38	52.125525	CiscoInc_d4:1c:...	Spanning-tree-...	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
39	52.218157	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1154, seq=6/1536, ttl=64 (no response found!)
40	53.227813	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1154, seq=7/1792, ttl=64 (no response found!)
41	54.130383	CiscoInc_d4:1c:...	Spanning-tree-...	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
42	54.234169	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1154, seq=8/2048, ttl=64 (no response found!)
43	54.617024	CiscoInc_d4:1c:...	CiscoInc_d4:1c:...	LOOP	60	Reply
44	55.242166	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1154, seq=9/2304, ttl=64 (no response found!)
45	56.135077	CiscoInc_d4:1c:...	Spanning-tree-...	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003

Figura 6: Experiment 2 - Point 9 - tux1

9	11.651879	CiscoInc_d4:1c:...	CiscoInc_d4:1c:...	LOOP	60 Reply			
10	13.332851	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/41/30:37:a6:d4:1c:00	Cost = 0	Port = 0x8004	
11	15.337478	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/41/30:37:a6:d4:1c:00	Cost = 0	Port = 0x8004	
12	17.369398	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/41/30:37:a6:d4:1c:00	Cost = 0	Port = 0x8004	
13	19.372362	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/41/30:37:a6:d4:1c:00	Cost = 0	Port = 0x8004	
14	21.377325	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/41/30:37:a6:d4:1c:00	Cost = 0	Port = 0x8004	
15	21.659472	CiscoInc_d4:1c:...	CiscoInc_d4:1c:...	LOOP	60 Reply			
16	23.382195	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/41/30:37:a6:d4:1c:00	Cost = 0	Port = 0x8004	
17	25.387035	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/41/30:37:a6:d4:1c:00	Cost = 0	Port = 0x8004	
18	27.392055	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/41/30:37:a6:d4:1c:00	Cost = 0	Port = 0x8004	
19	29.396805	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/41/30:37:a6:d4:1c:00	Cost = 0	Port = 0x8004	
20	31.401756	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/41/30:37:a6:d4:1c:00	Cost = 0	Port = 0x8004	
21	31.662061	CiscoInc_d4:1c:...	CiscoInc_d4:1c:...	LOOP	60 Reply			

Figura 7: Experiment 2 - Point 9 - tux2

19	24.139971	172.16.40.1	172.16.40.255	ICMP	98 Echo (ping) request id=0x1154, seq=4/1024, ttl=64 (no response found!)
20	25.148033	172.16.40.1	172.16.40.255	ICMP	98 Echo (ping) request id=0x1154, seq=5/1280, ttl=64 (no response found!)
21	26.064738	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8006
22	26.156060	172.16.40.1	172.16.40.255	ICMP	98 Echo (ping) request id=0x1154, seq=6/1536, ttl=64 (no response found!)
23	27.165752	172.16.40.1	172.16.40.255	ICMP	98 Echo (ping) request id=0x1154, seq=7/1792, ttl=64 (no response found!)
24	28.070276	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8006
25	28.172151	172.16.40.1	172.16.40.255	ICMP	98 Echo (ping) request id=0x1154, seq=8/2048, ttl=64 (no response found!)
26	28.555298	CiscoInc_d4:1c:...	CiscoInc_d4:1c:...	LOOP	60 Reply
27	29.180181	172.16.40.1	172.16.40.255	ICMP	98 Echo (ping) request id=0x1154, seq=9/2304, ttl=64 (no response found!)
28	30.073563	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8006
29	30.188218	172.16.40.1	172.16.40.255	ICMP	98 Echo (ping) request id=0x1154, seq=10/2560, ttl=64 (no response found!)
30	31.196259	172.16.40.1	172.16.40.255	ICMP	98 Echo (ping) request id=0x1154, seq=11/2816, ttl=64 (no response found!)

Figura 8: Experiment 2 - Point 9 - tux4

12	18.325375	CiscoInc_d4:1c:...	CiscoInc_d4:1c:...	LOOP	60 Reply
13	20.048272	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
14	22.053009	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
15	24.057876	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
16	26.062717	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
17	28.067541	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
18	28.332920	CiscoInc_d4:1c:...	CiscoInc_d4:1c:...	LOOP	60 Reply
19	30.072387	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
20	32.077137	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
21	34.082061	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
22	36.086791	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
23	38.091636	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
24	38.340023	CiscoInc_d4:1c:...	CiscoInc_d4:1c:...	LOOP	60 Reply
25	40.096444	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
26	42.101266	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
27	44.106139	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003

Figura 9: Experiment 2 - Point 10 - tux1

17	26.090647	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8004
18	26.792505	172.16.41.0	172.16.41.255	ICMP	98 Echo (ping) request id=0x1197, seq=1/256, ttl=64 (no response found!)
19	27.800692	172.16.41.0	172.16.41.255	ICMP	98 Echo (ping) request id=0x1197, seq=2/512, ttl=64 (no response found!)
20	28.093575	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8004
21	28.808684	172.16.41.0	172.16.41.255	ICMP	98 Echo (ping) request id=0x1197, seq=3/768, ttl=64 (no response found!)
22	29.816687	172.16.41.0	172.16.41.255	ICMP	98 Echo (ping) request id=0x1197, seq=4/1024, ttl=64 (no response found!)
23	30.098463	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60 Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8004
24	30.824676	172.16.41.0	172.16.41.255	ICMP	98 Echo (ping) request id=0x1197, seq=5/1280, ttl=64 (no response found!)
25	31.832684	172.16.41.0	172.16.41.255	ICMP	98 Echo (ping) request id=0x1197, seq=6/1536, ttl=64 (no response found!)

Figura 10: Experiment 2 - Point 10 - tux2

5	6.296122	CiscoInc_d4:1c:...	CiscoInc_d4:1c:...	LOOP	60	Reply						
6	8.019163	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60	Conf.	Root = 32768/40/30:37:a6:d4:1c:00	Cost = 0	Port = 0x8006			
7	10.024252	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60	Conf.	Root = 32768/40/30:37:a6:d4:1c:00	Cost = 0	Port = 0x8006			
8	12.029523	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60	Conf.	Root = 32768/40/30:37:a6:d4:1c:00	Cost = 0	Port = 0x8006			
9	14.034519	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60	Conf.	Root = 32768/40/30:37:a6:d4:1c:00	Cost = 0	Port = 0x8006			
10	16.039552	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60	Conf.	Root = 32768/40/30:37:a6:d4:1c:00	Cost = 0	Port = 0x8006			
11	16.304033	CiscoInc_d4:1c:...	CiscoInc_d4:1c:...	LOOP	60	Reply						
12	18.044825	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60	Conf.	Root = 32768/40/30:37:a6:d4:1c:00	Cost = 0	Port = 0x8006			
13	20.048210	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60	Conf.	Root = 32768/40/30:37:a6:d4:1c:00	Cost = 0	Port = 0x8006			
14	22.053299	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60	Conf.	Root = 32768/40/30:37:a6:d4:1c:00	Cost = 0	Port = 0x8006			
15	24.058352	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60	Conf.	Root = 32768/40/30:37:a6:d4:1c:00	Cost = 0	Port = 0x8006			
16	26.063820	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60	Conf.	Root = 32768/40/30:37:a6:d4:1c:00	Cost = 0	Port = 0x8006			
17	26.311496	CiscoInc_d4:1c:...	CiscoInc_d4:1c:...	LOOP	60	Reply						
18	28.068834	CiscoInc_d4:1c:...	Spanning-tree-(...	STP	60	Conf.	Root = 32768/40/30:37:a6:d4:1c:00	Cost = 0	Port = 0x8006			

Figura 11: Experiment 2 - Point 10 - tux4

6.5.3 Experiência 3

6.5.4 Experiência 4

6.5.5 Experiência 5

6.5.6 Experiência 6