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

Conteúdo

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.

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 tranalho 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 respponsável 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.

```c
#include "utilities.h"

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.

```c
int getControl(FTP * ftp, urlStruct * url, FTP * receiverFtp) {
    if(sendAndReceiveControl(CMD_USER, ftp, receiverFtp, url) != 0) {
        return -1;
    }
    if (strlen(url->password) != 0) {
        msg("Entering in anonymous mode");
    }
    if(sendAndReceiveControl(CMD_PASS, ftp, receiverFtp, url) != 0) {
        return -1;
    }
    if(sendAndReceiveControl(CMD_PASSV, ftp, receiverFtp, url) != 0) {
        return -1;
    }
    return 0;
}
```

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 definifdas abaixo, bem como nos anexos.

```c
#include "url.h"

typedef struct
{
    int socketFd; // file descriptor to control socket
    int dataSocketFd; // file descriptor to data socket

    int passvAnswer[6];
    int port;
    char ip[MAX_STRING_DEBUG_SIZE];

} FTP;

int startConection(urlStruct * url, FTP * ftp);
int showResponse(FTP * ftp);
int sendAndReceiveControl(int cmd, FTP * ftp, FTP * receiverFtp, urlStruct * url);
int receivePassvAnswer(FTP * ftp);
int getControl(FTP * ftp, urlStruct * url, FTP * receiverFTP);
int startReceiverConection(urlStruct * url, FTP * ftp);
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.

```c
#define DEBUG_MODE 0
#define MAX_STRING_DEBUG_SIZE 100
#define PORT_FTP 21

#define CMD_USER 0
#define CMD_PASS 1
#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

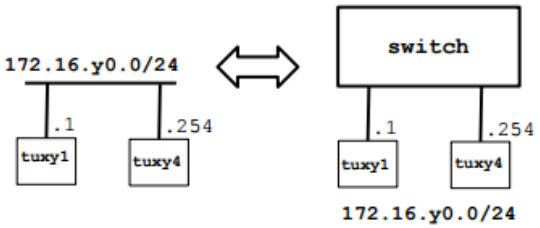


Figura 2: Experiment 1

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 8 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*

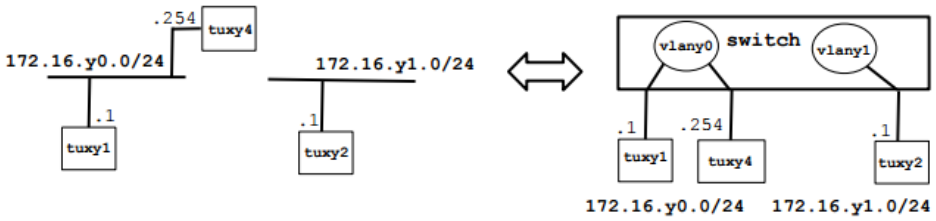


Figura 3: Experiment 2

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 9.

Pode também verificar-se, na análise das figuras 10, 11, 12, 13, 14 e 15 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 reder 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 41*.

4.3 Configuração de um *router* em *Linux*

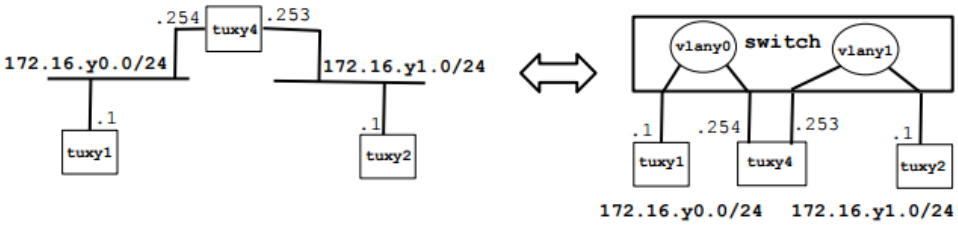


Figura 4: Experiment 3

Esta experiência consiste na configuração do computador *tux4* como *router* por forma a ligar as duas *vlan's* existentes:

- **172.16.40.0/24** - *vlan 40*;
- **172.16.41.0/24** - *vlan 41*.

Em primeiro lugar ativa-se a porta *eth1* do *tux4* i liga-se ao *switch*. Esta porta será a que será ligada à *vlan 41*. Configura-se esta mesma porta com o endereço *IP* **172.16.41.253/24**. Sendo este computadro aquele que pretendemos que sirva de *router*, é necessário ativar o reencaminhamento de *IP's* através do comando:

```
1   echo 1 > /proc/sys/net/ipv4/ip_forward
```

Shell Command

Por fim, adicionou-se as rotas necessárias no *tux1* e no *tux2* por forma a que estes, através do *tux4* pudessem aceder à rede a que não pertencem. Estas rotas forma adicionadas utilizando o comando:

```
1   route add -net _ _
```

Shell Command

Como é possível analisar através da figura 16, agora é possível a comunicação entre qualquer uma das três máquinas.

4.4 Configuração de um *router* comercial implementando *NAT*

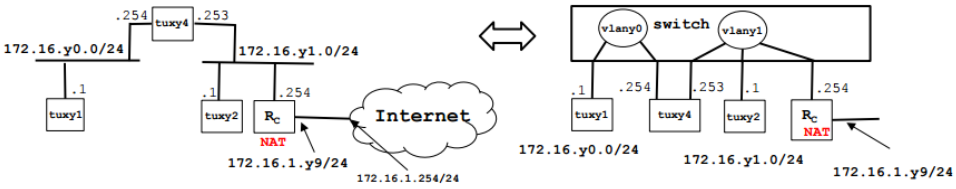


Figura 5: Experiment 4

Esta experiência tem como objetivo a configuração do *CISCO* dentro da rede 41 de forma a que tanto os computadores na *vlan40* como na *vlan41* tenham acesso à Internet.

Para configurar o router, depois de fazer *login* na linha de comandos corre-se o *script* do Anexo 9. Os comandos referidos em anexo começam por configurar duas interfaces do *router*, atribuindo corretamente as configurações *NAT*. A configuração correta do *NAT* é essencial pois a falta desta poderia resultar em falta de acesso à Internet em qualquer um dos computadores pois, como sabemos, *NAT* tem a função de traduzir endereços, resultando neste caso na tradução do endereço de sub-rede de cada computador no endereço do *router* comercial.

Depois de permitir que os computadores das redes criadas tenham acesso, é configurada uma rota predefinida para o endereço da Internet. Da mesma forma é adicionada aos computadores a rota predefinida para o endereço **172.16.41.254**, endereço do *router* comercial.

Como é possível verificar na figura 17, mesmo o computador da rede 10 tem acesso ao *router* comercial.

4.5 *DNS*

```
tuxy1$ vi /etc/resolv.conf
        search netlab.fe.up.pt
        nameserver  172.16.1.2
```

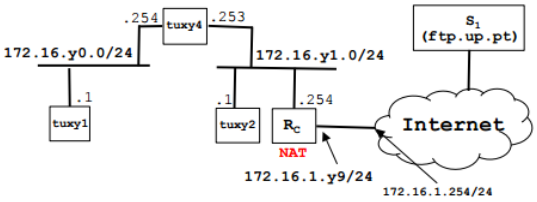


Figura 6: Experiment 5

4.6 Conexões *TCP*

```
tuxy1$ vi /etc/resolv.conf
        search netlab.fe.up.pt
        nameserver  172.16.1.1
```

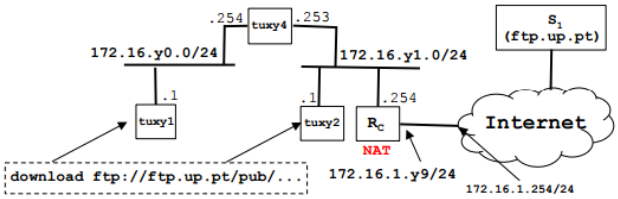


Figura 7: Experiment 6

5 Conclusões

6 Anexos

6.1 Headers

6.1.1 conection.h

```
#include "url.h"

typedef struct
{
    int socketFd; // file descriptor to control socket
    int dataSocketFd; // file descriptor to data socket

    int passvAnswer[6];
    int port;
    char ip[MAX_STRING_DEBUG_SIZE];

} FTP;

int startConection(urlStruct * url, FTP * ftp);
int showResponse(FTP * ftp);
int sendAndReceiveControl(int cmd, FTP * ftp, FTP * receiverFtp, urlStruct *
    url);
int receivePassvAnswer(FTP * ftp);
int getControl(FTP * ftp, urlStruct * url, FTP * receiverFTP);
int startReceiverConection(urlStruct * url, FTP * ftp);
int receiveFile(urlStruct * url, FTP * ftp, FTP * receiverFtp);
```

Anexo 1 - conection.h

6.1.2 url.h

```
#include "utilities.h"

void getUrlInfo(char * completeString, urlStruct * url);
```

Anexo 2 - url.h

6.1.3 utilities.h

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

#define DEBUG_MODE 0
```

```c
#define MAX_STRING_DEBUG_SIZE 100
#define PORT_FTP 21

#define CMD_USER 0
#define CMD_PASS 1
#define CMD_PASSV 2


typedef struct{
    char * user;
    char * password;
    struct hostent * h;
    char * urlPath;
    char * hostIp;
}urlStruct;

void debug(char * msg1, char * msg);
void msg(char * m);
void stringMsg(char * m, char * m2);
void getName(char * url, char ** filename);
void responseMsg(char * m, char * m2);
```

Anexo 3 - utilities.h

6.2 *.c files

6.2.1 main.c

```c
#include "conection.h"

int main(int argc, char *argv[])
{
    debug("", "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");

    urlStruct * url = malloc(sizeof(urlStruct));
    urlStruct * receiverUrl = malloc(sizeof(urlStruct));

    if (argc != 2) {
        fprintf(stderr,"usage: getip address\n");
        exit(1);
    }
    getUrlInfo(argv[1], url);

    FTP ftp;
    FTP receiverFtp;

    if (startConection(url, &ftp) < 0) {
        msg("Error starting connection\n");
        return -1;
    }

    if (getControl(&ftp, url, &receiverFtp) < 0) {
        msg("ERROR - getControl()");
        return -1;
    }
```

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

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

```
76
77      if (strlen(url->password) != 0) {
78          msg("Entering in anonymous mode");
79      }
80
81      if(sendAndReceiveControl(CMD_PASS, ftp, receiverFtp, url) != 0) {
82          return -1;
83      }
84
85      if(sendAndReceiveControl(CMD_PASSV, ftp, receiverFtp, url) != 0) {
86          return -1;
87      }
88
89      return 0;
90  }
91
92  int sendAndReceiveControl(int cmdSelector, FTP * ftp, FTP * receiverFtp,
        urlStruct * url) {
93      debug("##################### DEBUG SEND AND RECEIVE CONTROL
            ####################", "BEGIN");
94      char cmd[MAX_STRING_DEBUG_SIZE];
95      switch(cmdSelector) {
96          case CMD_USER:
97              strcpy(cmd, "user \0");
98              strcat(cmd, url->user);
99              break;
100         case CMD_PASS:
101             strcpy(cmd, "pass \0");
102             strcat(cmd, url->password);
103             break;
104         case CMD_PASSV:
105             strcpy(cmd, "pasv \0");
106             break;
107         default:
108             break;
109     }
110     strcat(cmd, "\n");
111     debug("Command to Send", cmd);
112     if(write(ftp->socketFd, cmd, strlen(cmd)) < 0) {
113         perror("write()");
114         return -1;
115     } else {
116         debug("Command Sent", cmd);
117     }
118     sleep(1);
119     switch(cmdSelector) {
120         case CMD_PASSV:
121             if(0 == receivePassvAnswer(ftp)) {
122                 // Parse new IP address
123                 receiverFtp->port = ftp->passvAnswer[4] * 256 + ftp->
                        passvAnswer[5];
124                 memset(receiverFtp->ip, 0, MAX_STRING_DEBUG_SIZE); // clearing
                        the array, "just in case"
125                 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->
                        ip);
127                 char temp[MAX_STRING_DEBUG_SIZE];
128                 sprintf(temp, "%d", receiverFtp->port);
129                 stringMsg("Port to receive file      ", temp);
130 
131                 debug("#################### DEBUG SEND AND RECEIVE CONTROL
                        ####################", "END");
132                 return 0;
133             }
134 
135             msg("Could not receive the 'passvAnswer' response");
136             return -1;
137         default:
138             showResponse(ftp);
139             break;
140     }
141     debug("#################### DEBUG SEND AND RECEIVE CONTROL
            ####################", "END");
142     return 0;
143 }
144 
145 int startReceiverConection(urlStruct * url, FTP * ftp) {
146     debug("#################### DEBUG START RECEIVER CONECTION
            ####################", "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
            address network byte ordered
165     server_addr.sin_port = htons(ftp->port);                //server TCP port
            must be network byte ordered | this is the new port received from the
            control TCP connection!
166 
167     // Opening the receiver TCP socket
168     socket_fd = socket(AF_INET, SOCK_STREAM,0);
169     if (socket_fd < 0) {
170         msg("Error opening receiver TCP socket");
171         return -1;
172     }
173 
174     int connectRet;
```

```
175     if((connectRet = connect(socket_fd, (struct sockaddr *)&server_addr, sizeof
            (server_addr))) < 0) {
176         perror("connect()");
177         msg("Error connecting to the server to open the control connection");
178         return -1;
179     } else {
180         char temp[MAX_STRING_DEBUG_SIZE];
181         sprintf(temp, "%d", connectRet);
182         debug("Connected to server with return      ", temp);
183     }
184
185
186     ftp->socketFd = socket_fd;
187     ftp->dataSocketFd = socket_fd;
188     msg("Connected to receiver");
189     debug("#################### DEBUG START RECEIVER CONECTION
            ####################", "END");
190
191     return 0;
192 }
193
194 int receiveFile(urlStruct * url, FTP * ftp, FTP * receiverFtp) {
195     debug("#################### DEBUG RECEIVE FILE ####################", "
            BEGIN");
196     msg("Receiving File...");
197     char cmd[MAX_STRING_DEBUG_SIZE] = "";
198     strcpy(cmd, "retr ");
199     strcat(cmd, url->urlPath);
200     strcat(cmd, "\n");
201     stringMsg("Command to be sent", cmd);
202     if(write(ftp->socketFd, cmd, strlen(cmd)) < 0){
203         msg("ERROR - retr command could not be sent");
204         return -1;
205     }
206     stringMsg("Command sent", cmd);
207
208     char * filename;
209     getName(url->urlPath, &filename);
210     debug("Filename is", filename);
211
212     FILE* file;
213     int bytes;
214
215     if (!(file = fopen(filename, "w"))) {
216         msg("ERROR: Cannot open file.");
217         return -1;
218     }
219
220     char temp[MAX_STRING_DEBUG_SIZE];
221     sprintf(temp,"%d",  receiverFtp->dataSocketFd);
222     debug("Receiver Data Socket fd", temp);
223
224     char buf[1024];
225     while ((bytes = read(receiverFtp->dataSocketFd, buf, sizeof(buf)))) {
226         if (bytes < 0) {
227             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
           ://[<user>:<password>@]<host>/<url-path>\n");
28         exit(1);
29     }
30 
31     int lengthOfUserAndPassword;
32     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          herror("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 ovrld 172.16.1.19 172.16.1.19 prefix 24
14 ip nat inside source list 1 pool ovrld 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

```
conf t
vlan 10
end

conf t
vlan 11
end

conf t
interface fastethernet 0/1
switchport mode access
switchport access vlan 10
end

conf t
interface fastethernet 0/4
switchport mode access
switchport access vlan 10
end

conf t
interface fastethernet 0/2
switchport mode access
switchport access vlan 11
end

conf t
interface fastethernet 0/5
switchport mode access
switchport access vlan 11
end

conf t
interface gigabitethernet 0/1
switchport mode access
switchport access vlan 11
end
```

Anexo 10 - Switch Configuration

6.4.3 tux1 Configuration

```
#!/bin/bash

ifconfig eth0 up
ifconfig eth0 172.16.40.1/24

route add default gw 172.16.40.254
```

Anexo 11 - tux1 Final Configuration

6.4.4 tux2 Configuration

```
#!/bin/bash

ifconfig eth0 up
ifconfig eth0 172.16.11.1/24

ifconfig eth1 down
ifconfig eth2 down

route add default gw 172.16.11.254
route add -net 172.16.10.0/24 gw 172.16.11.253
```

Anexo 12 - tux2 Final Configuration

6.4.5 tux4 Configuration

```
#!/bin/bash

ifconfig eth0 up
ifconfig eth0 172.16.40.254/24

ifconfig eth1 up
ifconfig eth1 172.16.41.253/24

route add default gw 172.16.41.254

echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
```

Anexo 13 - tux4 Final Configuration

6.5 Wireshark Logs

6.5.1 Configuração de um *IP* de rede

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 8: Experiment 1 - log

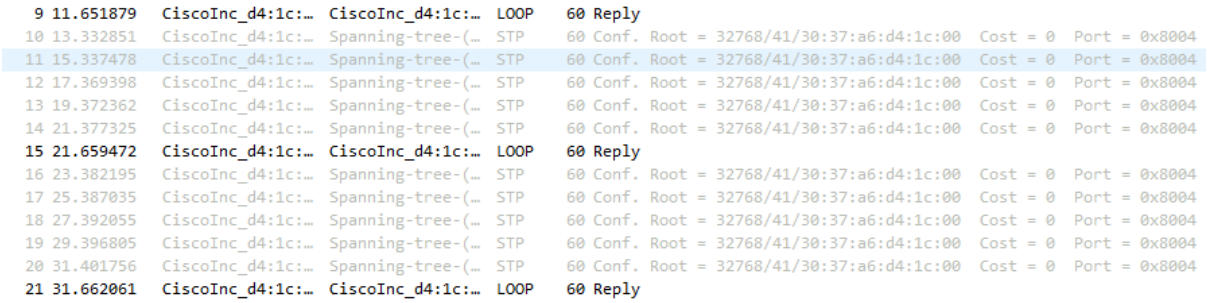
6.5.2 Configuração de duas Redes *LAN* virtuais num *switch*

8	10.913047	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 9: Experiment 2 - Point 6

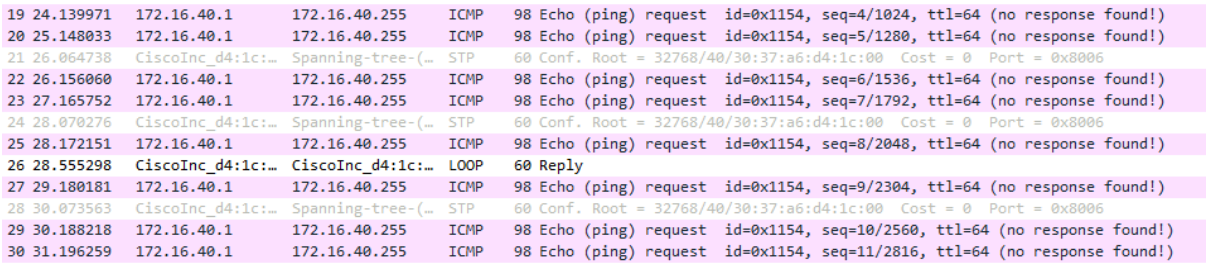
30	46.111027	CiscoInc_d4:1c:…	Spanning-tree-(…	STP	60 Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
31	47.179101	172.16.40.1	172.16.40.255	ICMP	98 Echo (ping) request id=0x1154, seq=1/256, ttl=64 (no response found!)
32	48.115818	CiscoInc_d4:1c:…	Spanning-tree-(…	STP	60 Conf. Root = 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 10: Experiment 2 - Point 9 - tux1



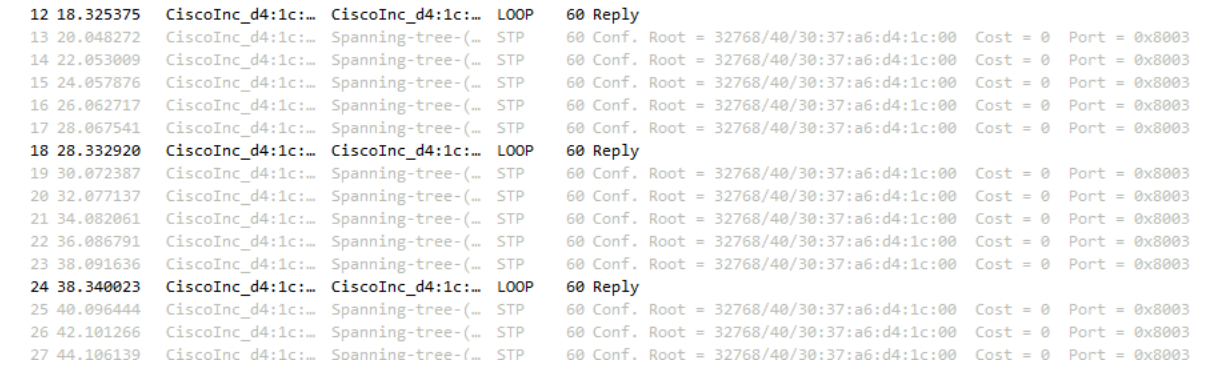
9	11.651879	CiscoInc_d4:1c:…	CiscoInc_d4:1c:…	LOOP	60 Reply
10	13.332851	CiscoInc_d4:1c:…	Spanning-tree-(…	STP	60 Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8004
11	15.337478	CiscoInc_d4:1c:…	Spanning-tree-(…	STP	60 Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8004
12	17.369398	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 11: Experiment 2 - Point 9 - tux2



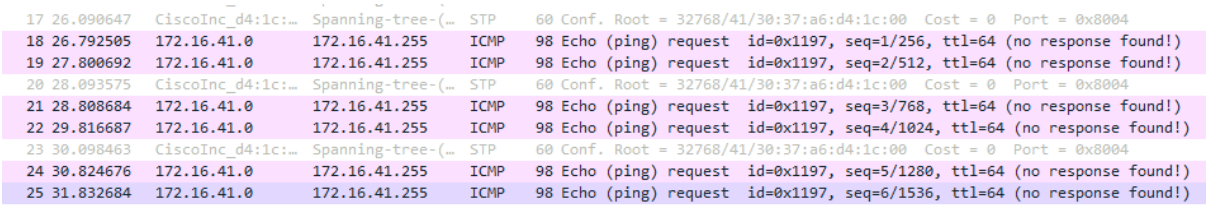
19	24.139971	172.16.40.1	172.16.40.255	ICMP	98 Echo (ping) request id=0x1154, seq=4/1024, ttl=64 (no response found!)
20	25.148033	172.16.40.1	172.16.40.255	ICMP	98 Echo (ping) request id=0x1154, seq=5/1280, ttl=64 (no response found!)
21	26.064738	CiscoInc_d4:1c:…	Spanning-tree-(…	STP	60 Conf. Root = 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 12: 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 13: 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 14: 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 15: Experiment 2 - Point 10 - tux4

6.5.3 Configuração de um *router* em *Linux*

41	28.876088	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0x12e3, seq=10/2560, ttl=64 (reply in 42)
42	28.876432	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x12e3, seq=10/2560, ttl=64 (request in 41)
43	30.072100	CiscoInc_3a:fa:…	Spanning-tree-(…	STP	60 Conf. Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
44	32.085006	CiscoInc_3a:fa:…	Spanning-tree-(…	STP	60 Conf. Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
45	33.821157	172.16.30.1	172.16.31.253	ICMP	98 Echo (ping) request id=0x12ed, seq=1/256, ttl=64 (reply in 46)
46	33.821520	172.16.31.253	172.16.30.1	ICMP	98 Echo (ping) reply id=0x12ed, seq=1/256, ttl=64 (request in 45)
47	33.855344	CiscoInc_3a:fa:…	CiscoInc_3a:fa:…	LOOP	60 Reply
48	34.081667	CiscoInc_3a:fa:…	Spanning-tree-(…	STP	60 Conf. Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
49	34.820164	172.16.30.1	172.16.31.253	ICMP	98 Echo (ping) request id=0x12ed, seq=2/512, ttl=64 (reply in 50)
50	34.820308	172.16.31.253	172.16.30.1	ICMP	98 Echo (ping) reply id=0x12ed, seq=2/512, ttl=64 (request in 49)
51	35.820087	172.16.30.1	172.16.31.253	ICMP	98 Echo (ping) request id=0x12ed, seq=3/768, ttl=64 (reply in 52)
52	35.820320	172.16.31.253	172.16.30.1	ICMP	98 Echo (ping) reply id=0x12ed, seq=3/768, ttl=64 (request in 51)
53	36.086620	CiscoInc_3a:fa:…	Spanning-tree-(…	STP	60 Conf. Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
54	36.820095	172.16.30.1	172.16.31.253	ICMP	98 Echo (ping) request id=0x12ed, seq=4/1024, ttl=64 (reply in 55)
55	36.820460	172.16.31.253	172.16.30.1	ICMP	98 Echo (ping) reply id=0x12ed, seq=4/1024, ttl=64 (request in 54)
56	37.820110	172.16.30.1	172.16.31.253	ICMP	98 Echo (ping) request id=0x12ed, seq=5/1280, ttl=64 (reply in 57)
57	37.820349	172.16.31.253	172.16.30.1	ICMP	98 Echo (ping) reply id=0x12ed, seq=5/1280, ttl=64 (request in 56)
58	38.096347	CiscoInc_3a:fa:…	Spanning-tree-(…	STP	60 Conf. Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
59	38.820092	172.16.30.1	172.16.31.253	ICMP	98 Echo (ping) request id=0x12ed, seq=6/1536, ttl=64 (reply in 60)
60	38.820454	172.16.31.253	172.16.30.1	ICMP	98 Echo (ping) reply id=0x12ed, seq=6/1536, ttl=64 (request in 59)
61	39.820092	172.16.30.1	172.16.31.253	ICMP	98 Echo (ping) request id=0x12ed, seq=7/1792, ttl=64 (reply in 62)
62	39.820328	172.16.31.253	172.16.30.1	ICMP	98 Echo (ping) reply id=0x12ed, seq=7/1792, ttl=64 (request in 61)
63	40.096161	CiscoInc_3a:fa:…	Spanning-tree-(…	STP	60 Conf. Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
64	40.820095	172.16.30.1	172.16.31.253	ICMP	98 Echo (ping) request id=0x12ed, seq=8/2048, ttl=64 (reply in 65)
65	40.820237	172.16.31.253	172.16.30.1	ICMP	98 Echo (ping) reply id=0x12ed, seq=8/2048, ttl=64 (request in 64)
66	41.820089	172.16.30.1	172.16.31.253	ICMP	98 Echo (ping) request id=0x12ed, seq=9/2304, ttl=64 (reply in 67)
67	41.820320	172.16.31.253	172.16.30.1	ICMP	98 Echo (ping) reply id=0x12ed, seq=9/2304, ttl=64 (request in 66)
68	42.101008	CiscoInc_3a:fa:…	Spanning-tree-(…	STP	60 Conf. Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
69	42.820103	172.16.30.1	172.16.31.253	ICMP	98 Echo (ping) request id=0x12ed, seq=10/2560, ttl=64 (reply in 70)
70	42.820247	172.16.31.253	172.16.30.1	ICMP	98 Echo (ping) reply id=0x12ed, seq=10/2560, ttl=64 (request in 69)
71	43.862613	CiscoInc_3a:fa:…	CiscoInc_3a:fa:…	LOOP	60 Reply
72	44.110814	CiscoInc_3a:fa:…	Spanning-tree-(…	STP	60 Conf. Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
73	46.110572	CiscoInc_3a:fa:…	Spanning-tree-(…	STP	60 Conf. Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
74	47.549000	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0x12f4, seq=1/256, ttl=64 (reply in 75)
75	47.549619	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0x12f4, seq=1/256, ttl=63 (request in 74)

Figura 16: Experiment 3 - tux1

6.5.4 Configuração de um *router* comercial implementando *NAT*

5	2.572720	172.16.30.1	172.16.31.254	ICMP	98 Echo (ping) request id=0x1733, seq=2/512, ttl=64 (reply in 6)
6	2.573395	172.16.31.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1733, seq=2/512, ttl=254 (request in 5)
7	2.734763	CiscoInc_3a:fa:…	CiscoInc_3a:fa:…	LOOP	60 Reply
8	3.571761	172.16.30.1	172.16.31.254	ICMP	98 Echo (ping) request id=0x1733, seq=3/768, ttl=64 (reply in 9)
9	3.572531	172.16.31.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1733, seq=3/768, ttl=254 (request in 8)
10	4.009679	CiscoInc_3a:fa:…	Spanning-tree-(…	STP	60 Conf. Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
11	4.571761	172.16.30.1	172.16.31.254	ICMP	98 Echo (ping) request id=0x1733, seq=4/1024, ttl=64 (reply in 12)
12	4.572440	172.16.31.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1733, seq=4/1024, ttl=254 (request in 11)
13	5.571762	172.16.30.1	172.16.31.254	ICMP	98 Echo (ping) request id=0x1733, seq=5/1280, ttl=64 (reply in 14)
14	5.572454	172.16.31.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1733, seq=5/1280, ttl=254 (request in 13)
15	6.014340	CiscoInc_3a:fa:…	Spanning-tree-(…	STP	60 Conf. Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
16	6.571759	172.16.30.1	172.16.31.254	ICMP	98 Echo (ping) request id=0x1733, seq=6/1536, ttl=64 (reply in 17)
17	6.572434	172.16.31.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1733, seq=6/1536, ttl=254 (request in 16)

Figura 17: Experiment 4 - tux1

6.5.5 *DNS*

6.5.6 Conexões *TCP*