Lab2 - Computer Networks

Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

Turma 6:
Duarte Miguel de Novo Faria - 201607176
Maria Teresa Queiroz Machado Urbano Ferreira - 201603811
Maria João Senra Viana - 201604751

Faculdade de Engenharia da Universidade do Porto

23 de Dezembro de 2018

Conteúdo

1 Sumário

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

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

2 Introdução

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

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

3 Download Application

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

3.1 Arquitetura

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

3.1.1 Parser

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

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

3.1.2 Client

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

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

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

3.2 Resultados

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

4 Configuração e estudo da rede

4.1 Configurar uma rede IP

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

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

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

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

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

- O ARP (Address Resolution Protocol) é um protocolo de comunicação utilizado para obter o endereço MAC associado a um endereço IP fornecido.
- Num pacote ARP o endereço MAC corresponde ao endereço físico do emissor do pacote original para que possa depois receber a resposta da mensagem, enquanto o endereço IP corresponde ao endereço IP da máquina cujo endereço MAC se pretende determinar.
- O comando "ping" gera pacotes ICMP com o objetivo de captar respostas (eco) de um destinatário, assim determinando se ele se encontra conectado e ativo.
- Para conseguir distinguir os pacotes ARP dos IP e dos ICMP temos de analisar os 2 bytes do cabeçalho da trama Ethernet. Se esses 2 bytes tiverem o valor 0x0800 então sabemos que se trata de um IP, se tiver o valor 0x0806 sabemos que se trata de uma trama do tipo ARP.

  Como a trama ICMP é um sub-protocolo do protocolo IP para distinguir um protocolo ICMP dos restantes temos de analisar o valor do IP header. Se este for 1 então trata-se de uma trama ICMP.
- Para ver o tamanho dos pacotes basta analisar no wireshark o campo frame length.
- A interface loopback é uma interface virtual que permite que o computador receba respostas de si mesmo. É fundamental pois testa se a carta da rede está corretamente configurada.

4.2 Implementar duas LAN's virtuais no switch

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

4.2.1 Como se configura a VLANy0?

Criação da VLAN

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

Adicionar as portas do tuxy1 e tuxy2

```
configure terminal
interface fastethernet 0/#numero da porta
switchport mode access
switchmode access vlan y0
end
```

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

4.3 Configurar um router em Linux

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

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

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

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

```
configure terminal
interface fastethernet 0/#numero da porta
switchport mode access
switchmode access vlan y1
end
```

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

1. No tuxy1

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

2. No tuxy2

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

- Cada route tem uma *destination* e um *gateway*, em que o primeiro identifica a gama de endereços a que se vai adicionar a rota e o segundo é o IP para qual o *packet* vai ser reencaminhado.
- A informação mais importante para a experiência presente nas tabelas de *forwarding* é: Destination (destino da rota); Gateway (ip para onde o packet esta a ser reencaminhado); Netmask (juntamente com a *destination* formam o ID da rede) e Interface (placa de rede responsavel pela *gateway* (eth0 ou rth1)
- As mensagens de ARP mostram que o tux que recebeu o ping nao reconheceu o MAC address do emissor e por isso pergunta a quem pertence o MAC address do tux que tem aquele IP.
- São observados pacotes ICMP de request e reply devido a se ter adicionado rotas a todos os tuxs para que todos consigam comunicar entre si.
- Consultando os logs do Wireshark podemos verificar que ao fazermos ping do tuxy1 para o tuxy2 os endereços IP e MAC associados aos pacotes ICMP contêm a informação do tuxy4, porque este é o que esta a fazer de "ponte"entre os emissor e o recetor.
- Esta experiência tinha como objetivo configurar um router comercial, primeiramente sem NAT e posteriormente com NAT, permitindo a possibilidade de comunicar com a Internet. Sendo assim o router foi configurado para que fizesse parte da VLANy1.

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

4.4 DNS

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

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

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

- O *host* manda para o servidor DNS o *hostname* que pretende aceder, pedindo o respetivo IP.

  O DNS responde com um pacote contendo varias informações incluindo o IP necessário.

4.5 Ligações TCP

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

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

  O *header* do TCP tem como principais informações

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

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

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

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

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

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

5 Conclusão

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

A Código fonte

A.1 parser.h

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

#define MAX_SIZE 256

typedef struct{
  char  user[MAX_SIZE];
  char  password[MAX_SIZE];
  char  host[MAX_SIZE];
  char  filePath[MAX_SIZE];
  char  fileName[MAX_SIZE];
  char  ip[MAX_SIZE];
} urlInfo;

int parseUsernamePassword(urlInfo * infoStruct, char * completeUrl);
int parseUrl(char completeUrl[], urlInfo * infoStruct);
int getIp(urlInfo * infoStruct);
```

parser.h

A.2 parser.c

```
#include "parser.h"

int parseUsernamePassword(urlInfo * infoStruct, char * completeUrl){
    //ftp://[<user>:<password>@]<host>/<url-path>

  char * at = strrchr(completeUrl, '@');
  char* firstSlash = strchr(completeUrl, '/');
  firstSlash += 2; // pointing to the beggining of the username

  char* password = strchr(firstSlash, ':'); //password pointing to :
  if(password == NULL){
    fprintf(stderr, "Your link must contain a ':' separating the username and password!'\n");
    return 1;
  }

  memcpy(infoStruct->user, firstSlash, password - firstSlash); //password - slash it's the
      size of username in bytes
  infoStruct->user[password-firstSlash]=0;
  password++; // now pointing to the first character of the password

  memcpy(infoStruct->password, password, at - password);
  infoStruct->password[at-password] = 0; //string end character
  return 0;
}

int parseUrl(char completeUrl[], urlInfo * infoStruct){
    //see if it begins with ftp://
    if(strncmp(completeUrl, "ftp://", strlen("ftp://")) != 0){
    fprintf(stderr, "The link does not begin with 'ftp://'\n");
    return 1;
    }

  char* slashAfterHost;

  if(!strchr(completeUrl, '@')){ //if it doesnt find @ it's password and username are
      anonymous
    memcpy(infoStruct->user, "anonymous", strlen("anonymous") + 1);
    memcpy(infoStruct->password, "anonymous", strlen("anonymous") + 1);

    char * s1 = strchr(completeUrl,'/');
    s1++;
    s1++;

    slashAfterHost = strchr(s1, '/');
    memcpy(infoStruct->host, s1, slashAfterHost-s1);
    infoStruct->host[slashAfterHost-s1] = 0;
  }

  else{
    if(parseUsernamePassword(infoStruct,completeUrl)!=0)
      return 1;

      char * at = strrchr(completeUrl, '@');
      at++;

      slashAfterHost = strchr(at, '/');
      memcpy(infoStruct->host, at, slashAfterHost-at);
      infoStruct->host[slashAfterHost-at] = 0;
  }

  char* lastSlash = strrchr(completeUrl, '/');
  lastSlash++; //to point to the element after the slash
  memcpy(infoStruct->filePath, slashAfterHost, lastSlash-slashAfterHost);
```

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

parser.c

A.3 client.h

```
#include <stdio.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>
#include <netdb.h>
#include <string.h>

#include "parser.h"

typedef struct {
    int controlSocketFd; // file descriptor to control socket
    int dataSocketFd; // file descriptor to data socket
} FTPInfo;

int initConnection(FTPInfo * ftp, char * ip, int port);
int connectSocket(char * ip, int port);
void login(FTPInfo ftp, urlInfo url);
void passiveMode(FTPInfo ftp, char * ip, int * port);
void retrieve(FTPInfo ftp, urlInfo url);
int readMessage(int socketfd, char * repply);
int sendMessage(int socketfd, char * cmd);
int download(FTPInfo ftp, urlInfo url);
int endConnection(FTPInfo ftp);
```

client.h

A.4 client.c

```
#include "client.h"

int initConnection(FTPInfo* ftp, char* ip, int port){
  int socketfd;

  if ((socketfd = connectSocket(ip, port)) < 0) {
        printf("Error connecting socket.\n");
        return 1;
    }

    ftp->controlSocketFd = socketfd;
  ftp->dataSocketFd = 0;

  return 0;

}

int connectSocket(char * ip, int port){
  int   socketfd;
  struct    sockaddr_in server_addr;

  /*server address handling*/
  bzero((char*)&server_addr,sizeof(server_addr));
  server_addr.sin_family = AF_INET;
  server_addr.sin_addr.s_addr = inet_addr(ip);  /*32 bit Internet address network byte ordered
      */
  server_addr.sin_port = htons(port);       /*server TCP port must be network byte ordered */

  /*open an TCP socket*/
  if ((socketfd = socket(AF_INET,SOCK_STREAM,0)) < 0) {
        perror("socket()");
          return -1;
      }
  /*connect to the server*/
      if(connect(socketfd,(struct sockaddr *)&server_addr,sizeof(server_addr)) < 0){
          perror("connect()");
          return -1;
        }
  return socketfd;
}

void login(FTPInfo ftp, urlInfo url){
  char user[MAX_SIZE];
  char password[MAX_SIZE];

  readMessage(ftp.controlSocketFd, NULL);

  sprintf(user, "USER %s\r\n", url.user);
  printf(">%s",user);

  sendMessage(ftp.controlSocketFd,user);
  readMessage(ftp.controlSocketFd,NULL);

  sprintf(password, "PASS %s\r\n", url.password);
  printf(">%s",password);

  sendMessage(ftp.controlSocketFd, password);
  if(readMessage(ftp.controlSocketFd,NULL) !=0){
    fprintf(stderr, "Wrong credentials. Exiting...\n");
    exit(1);
  }
}

```

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

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

client.c

A.5 main.c

```
#include <stdio.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>
#include <netdb.h>
#include <string.h>
#include "client.h"

#define PORT 21

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

  if(argc != 2){
    fprintf(stderr, "Usage: %s <address>\n", argv[0]);
    exit(1);
  }

  urlInfo url;
  FTPInfo ftp;

  if(parseUrl(argv[1], &url) != 0){
    fprintf(stderr, "Invalid URL\n");
    exit(1);
  }

  printf("\nusername:%s\n", url.user);
  printf("password:%s\n", url.password);
  printf("ip:%s\n", url.ip);
  printf("path:%s\n", url.filePath);
  printf("file name:%s\n", url.fileName);
  printf("host:%s\n", url.host);
  printf("\n\n");

  if(initConnection(&ftp, url.ip, PORT) !=0){
    fprintf(stderr, "Error opening control connection\n");
    exit(1);
  }

  login(ftp,url);

  char ip[MAX_SIZE];
  int port;

  passiveMode(ftp, ip, &port);

  if ((ftp.dataSocketFd = connectSocket(ip,port))<0){
    fprintf(stderr, "Error opening data connection\n");
    exit(1);
  }

  retrieve(ftp,url);
  download(ftp,url);
  endConnection(ftp);

  return 0;
  }
```

main.c

A.6 makefile

```
gs: client.c parser.c main.c
	gcc -Wall -o t2 client.c parser.c main.c

clean:
	rm -f t2
```

makefile

B Anexos:

B.1 Arquitetura das experiências:

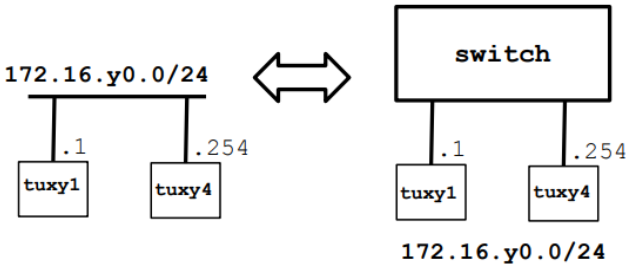


Figura 1: Arquitetura da experiência 1

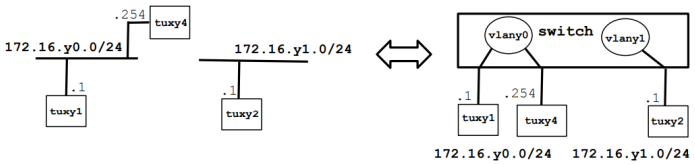


Figura 2: Arquitetura da experiência 2

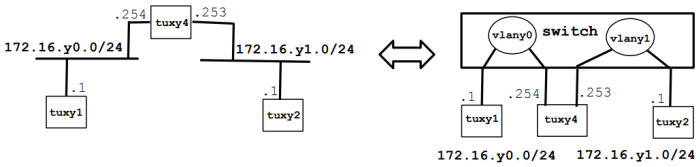


Figura 3: Arquitetura da experiência 3

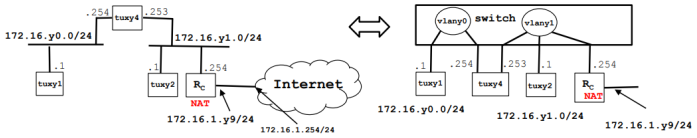


Figura 4: Arquitetura da experiência 4

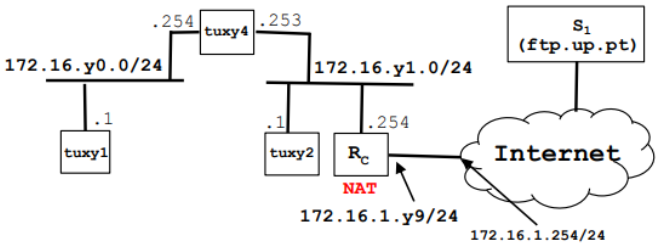


Figura 5: Arquitetura da experiência 5

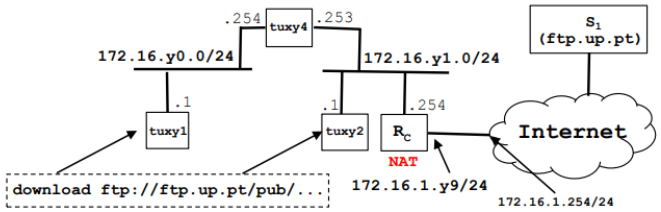
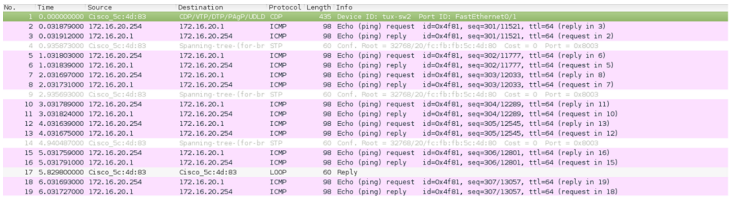


Figura 6: Arquitetura da experiência 6

B.2 Logs das Experiências

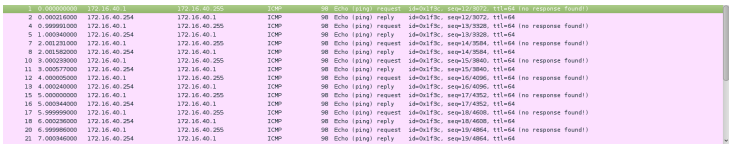
B.2.1 Exp. 1



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Cisco_5c:4d:83	CDP/VTP/DTP/PAgP/UDLD	CDP	458	Device ID: tuxy4 Port ID: FastEthernet0/1
2	0.031879000	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) request id=0x4f81, seq=301/11521, ttl=64 (reply in 3)
3	0.031912000	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) reply id=0x4f81, seq=301/11521, ttl=64 (request in 2)
4	0.935073000	Cisco_5c:4d:83	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/20/fc:fb:fb:5c:4d:80 Cost = 0 Port = 0x802d
5	1.031803000	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) request id=0x4f81, seq=302/11777, ttl=64 (reply in 6)
6	1.031839000	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) reply id=0x4f81, seq=302/11777, ttl=64 (request in 5)
7	2.031697000	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) request id=0x4f81, seq=303/12033, ttl=64 (reply in 8)
8	2.031731000	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) reply id=0x4f81, seq=303/12033, ttl=64 (request in 7)
9	2.935693000	Cisco_5c:4d:83	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/20/fc:fb:fb:5c:4d:80 Cost = 0 Port = 0x802d
10	3.031789000	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) request id=0x4f81, seq=304/12289, ttl=64 (reply in 11)
11	3.031824000	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) reply id=0x4f81, seq=304/12289, ttl=64 (request in 10)
12	4.031639000	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) request id=0x4f81, seq=305/12545, ttl=64 (reply in 13)
13	4.031675000	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) reply id=0x4f81, seq=305/12545, ttl=64 (request in 12)
14	4.936867000	Cisco_5c:4d:83	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/20/fc:fb:fb:5c:4d:80 Cost = 0 Port = 0x802d
15	5.031759000	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) request id=0x4f81, seq=306/12801, ttl=64 (reply in 16)
16	5.031791000	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) reply id=0x4f81, seq=306/12801, ttl=64 (request in 15)
17	5.829800000	Cisco_5c:4d:83	Cisco_5c:4d:83	LOOP	60	Reply
18	6.031693000	172.16.20.254	172.16.20.1	ICMP	98	Echo (ping) request id=0x4f81, seq=307/13057, ttl=64 (reply in 19)
19	6.031727000	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) reply id=0x4f81, seq=307/13057, ttl=64 (request in 18)

Figura 7: Ping do tuxy1 para o tuxy4

B.2.2 Exp. 2



1	0.000000000	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1f3c, seq=12/3072, ttl=64 (no response found!)
2	0.000216000	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1f3c, seq=12/3072, ttl=64
4	0.999981000	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1f3c, seq=13/3328, ttl=64 (no response found!)
5	1.000340000	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1f3c, seq=13/3328, ttl=64
7	2.001281000	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1f3c, seq=14/3584, ttl=64 (no response found!)
8	2.001582000	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1f3c, seq=14/3584, ttl=64
10	3.000239000	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1f3c, seq=15/3840, ttl=64 (no response found!)
11	3.000577000	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1f3c, seq=15/3840, ttl=64
12	4.000005000	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1f3c, seq=16/4096, ttl=64 (no response found!)
13	4.000240000	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1f3c, seq=16/4096, ttl=64
15	5.000000000	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1f3c, seq=17/4352, ttl=64 (no response found!)
16	5.000344000	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1f3c, seq=17/4352, ttl=64
17	5.999999000	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1f3c, seq=18/4608, ttl=64 (no response found!)
18	6.000236000	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1f3c, seq=18/4608, ttl=64
20	6.999986000	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x1f3c, seq=19/4864, ttl=64 (no response found!)
21	7.000346000	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1f3c, seq=19/4864, ttl=64

Figura 8: Ping do tuxy1 para o broadcast 172.16.40.255

B.2.3 Exp. 3

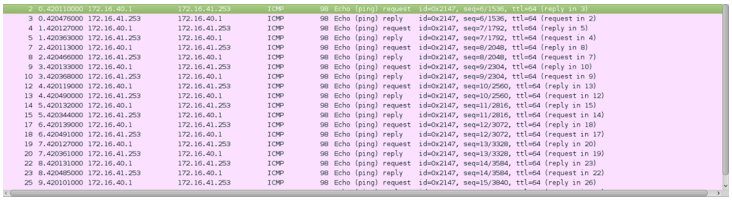


Figura 9: Ping do tuxy1 para o tuxy4 na interface eth1

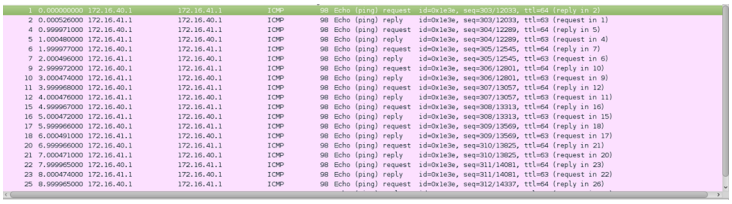


Figura 10: Ping do tuxy1 para o tuxy2

B.2.4 Exp. 4

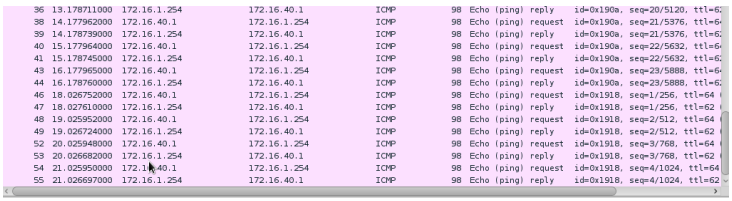


Figura 11: Ping do tuxy1 para o tuxy4 na interface eth1

B.2.5 Exp. 5

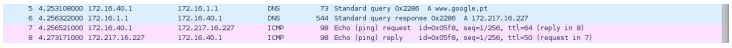


Figura 12: Resposta do comando DNS

B.2.6 Exp. 6

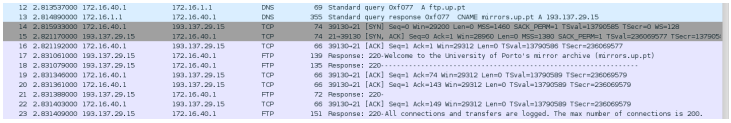


Figura 13: Estabelecimento de conexão através do protocolo TCP

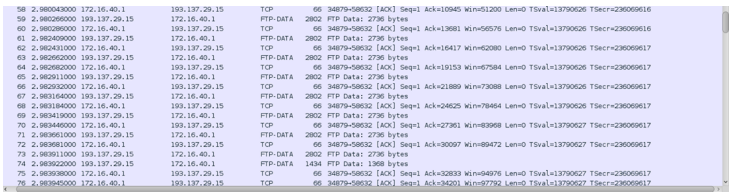


Figura 14: Transferência de dados através do protocolo TCP

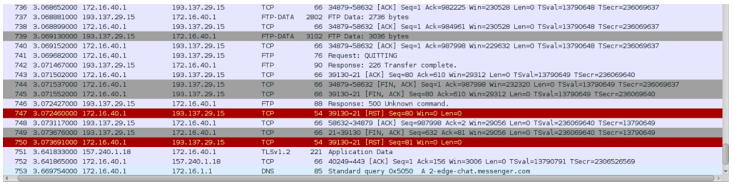


Figura 15: Terminação da conexão através do protocolo TCP

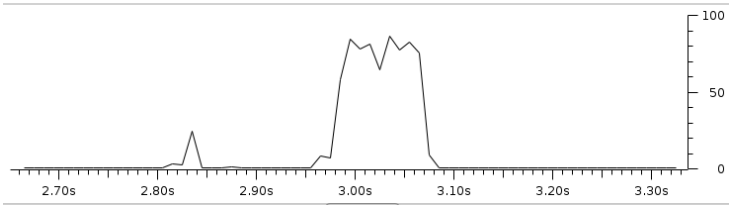


Figura 16: Gráfico de transferência dos pacotes por unidade de tempo