Trabalho Prático 2
Rede de Computadores

Feito por

Ana Sofia Teixeira

João Henrique Pinho

Unidade Curricular de Redes de Computadores

January 24, 2022

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Departamento de Engenharia Eletrotécnica e de Computadores

Índice

1 Sumário

Este relatório ilustra e descreve o trabalho desenvolvido no 2º trabalho prático da cadeira de Redes de Computadores, do 3º ano do curso LEIC, na FEUP. As experiências foram realizadas maioritariamente nas aulas práticas da cadeira, com o equipamento fornecido pelos laboratórios da faculdade. Nas secções seguintes todo o trabalho é documentado e explicado, tanto a nível do processo como dos resultados obtidos.

2 Introdução

O presente relatório foi elaborado como parte integrante da unidade curricular de Redes de Computadores, lecionada no âmbito da Licenciatura em Engenharia Informática e Computação e supervisionada pelo Departamento de Engenharia Eletrotécnica e de Computadores.

Foi proposto o desenvolvimento de um projeto com duas partes. A primeira incide sobre o desenvolvimento de uma aplicação de *download* de ficheiros por FTP, e a segunda incide sobre uma série de experiências com as várias estapas da configuração de uma rede. Quanto à primeira parte do projeto, a aplicação foi desenvolvida na linguagem, e é capaz de transferir ficheiros de modo pretendido, não descurando a necessária robustez à ocorrência de erros ou a *input* inválidos do utilizador. Em relação à segunda parte, as experiências foram realizadas com sucesso, tendo os objetivos sido alcançados, e tendo as principais dúvidas/questões relativas a estas sido respondidas.

Com o objetivo de detalhar a componente teórica do projeto, este relatório aborda os seguintes temas:

- **Aplicação *download***
  - Arquitetura da aplicação
  - Resultados
- **Configuração e análise de redes** - Análise de cada uma das experiências
- **Conclusões** - Síntese da informação apresentada nas secções anteriores; Reflexão sobre os objetivos de aprendizagem alcançados.

3 Parte 1 - Aplicação *Download*

A primeira parte deste trabalho consiste no desenvolvimento de uma aplicação de *download* de ficheiros de acordo com o protocolo FTP (*File Transfer Protocol*) descrito no RFC959 e com ligações TCP (*Transmission Control Protocol*) a partir de *sockets*, sendo assim a aplicação capaz de fazer transferência de qualquer tipo de ficheiros de um servidor FTP. A aplicação aceita como argumento um link com a seguinte configuração:

```
ftp://[<user>:<password>@]<host>/<url-path>
```

Este *link* está de acordo com a sintaxe *url* descrita no RFC1738. No <***url-path***> foi colocado como meio de teste o servidor FTP ***ftp.fe.up.pt*** seguido do *path* do ficheiro que se pretende transferir (***ftp.fe.up.pt/path-do-ficheiro***). Assim, foi necessário estudar o RFC959 "*Internet message protocol*", respetivo a FTP, e o RFC1738 "*Uniform Resource Locators (URL)*)", respetivo ao tratamento de informação contida num *URL*.

3.1 Arquitetura da aplicação

- **Processamento do *URL*** - O processamento é efetuado através da função *parseUrlInfo* que guarda o *username*, a *password*, o *host*, o *path* e o *filename* na *struct urlinfo*.
  - Caso o *username* e a *password* não sejam fornecidos, estes são colocados com os valores *anonymous* e *anypassword*, respetivamente.
  - Caso apenas seja fornecido o *username*, a aplicação pergunta ao utilizador a *password* respetiva.
- **Endereço IP** - Utiliza-se a função *getIpAddressFromHost* que retorna o endereço IP.
- **Abertura do *socket*** - Através da função *ftpStartConnection*, é aberto o *socket* pelo qual o cliente e o servidor vão comunicar.
- ***Login*** - É efetuado o *login* com os comandos *User user* e *Pass password*, utilizando a função *ftpLoginIn*.
- **Entrada em modo passivo** - A partir do comando *Pasv* enviado para o servidor, é efetuada a entrada em modo passivo e processada a resposta na qual é fornecido o endereço IP e os valores necessários para o cálculo da porta a utilizar para abrir a *socket* responsável pela troca de dados. Para tal, é usada a função *ftpPassiveMode*.
- **Pedir ficheiro** - Após a entrada em modo passivo, é chamada a função *ftpRetrieveFile* que envia o comando *RETR filename* para pedir a transferência do ficheiro respetivo e o *download* é efetuado pela função *ftpDownloadAndCreateFile*.
- **Fecho de ligação** - As ligações são todas encerradas.

Ao longo da execução do programa, todas as respostas do servidor são processadas e em caso de erro, o programa termina com a mensagem respetiva à situação ocorrida.

3.2 Resultados

A aplicação desenvolvida foi testada em diversas situações:

- com *username* e *password* fornecidos;
- apenas com *username* fornecido;
- sem *username* nem *password* fornecidos;
- com diferentes tipos de ficheiros;
- com um ficheiro que não existia.

Todos os testes a que o program foi submetido foram superados com sucesso.

4 Parte 2 - Configuração e análise de redes

4.1 Experiência 1 - Configuração de uma rede IP

4.1.1 O que são pacotes ARP e para que são usados?

ARP (*Address Resolution Protocol*) é um protocolo da camada de rede utilizado para converter um endereço IP num endereço físico designado endereço MAC (*Media Access Control*). Este endereço de *hardware* é único e identifica um específico nó numa rede.
Os pacotes ARP são utilizados para descobrir o endereço físico de um *host*. Quando um *host* pretende obter o endereço MAC de outro *host* na mesma rede, tendo apenas o endereço IP, envia em *broadcast* um pacote para a rede TCP/IP. Assim sendo, o *host* com o respetivo endereço IP envia uma resposta com o seu endereço físico.

4.1.2 Quais são os endereços MAC e IP dos pacotes ARP e porquê?

Os pacotes ARP contêm o endereço MAC e IP do transmissor e do recetor. No pacote enviado pelo host que pretende obter o endereço físico de outro host, o endereço MAC do recetor é ignorado, uma vez que não se sabe o valor.
Deverão ser verificados os logs registados nas Figuras 1 e 2 para consultar estes endereços.

4.1.3 Que pacotes são gerados pelo comando *ping*?

O comando *ping* gera pacotes ICMP (*Internet Control Message Protocol*). Inicialmente, se o endereço físico de destino não estiver contido na tabela ARP, são enviados pacotes ARP para obter esse endereço.
Este comando é utilizado para identificar o alcance de um *host*, indicando a existência de erros na rede estabelecida, perda de pacotes e ainda estatísticas dos resultados.

4.1.4 Quais são os endereços MAC e IP nos pacotes do comando *ping*?

Os pacotes ICMP, gerados pelo comando *ping*, contêm os endereços MAC e IP do transmissor e do recetor.
Deverão ser verificados os logs registados nas Figuras 3 e 4 para consultar estes endereços.

4.1.5 Como determinar se uma trama de *Ethernet* recebida é ARP, IP, ICMP?

Analisando o *Ethernet header* da trama recebida, obtém-se o *EtherType*, composto por 2 octetos.

- Caso o *header* tenha o valor 0x0800, significa que a trama é do tipo IP. Já se tiver o valor 0x0806, então a trama é do tipo ARP.
- Caso a trama seja do tipo IP, podemos analisar o seu IP *header*. Se este *header* tomar o valor 1, então o tipo de protocolo é ICMP.

Deverão ser verificados os logs registados nas Figuras 5 e 6.

4.1.6 Como determinar o tamanho de uma trama recebida?

No *Ethernet header* de uma trama recebida existem 2 *bytes* que contêm o valor relativo ao tamanho total da trama. O comprimento de uma trama pode ser visualizado através do *software Wireshark*.
Deverá ser verificado o log registado na Figura 7.

4.1.7 O que é a interface *loopback* e porque é que é importante?

O termo *loopback* refere-se ao encaminhamento de sinais digitais ou fluxo de dados para a fonte de origem do envio, sem qualquer processamento ou modificação. Este método é utilizado para testar a rede estabelecida para comunicação.
A interface *loopback* é uma interface virtual da rede que permite ao computador receber respostas de si próprio, para testar a correta configuração da carta de rede.
Deverá ser verificado o log registado na Figura 8.

4.2 Experiência 2 - VLAN

4.2.1 Como configurar a *vlany0*?

Estando ligado ao *switch* pela porta série /dev/tty/S0, de modo a ser possível executar comandos no GTKTerm, e correspondendo y ao número da bancada:

- Criar as VLANs:

```
> configure terminal
> vlan y0
> vlan y1
> end
```

- Adicionar as portas às respetivas VLANs:

```
> configure terminal
> interface fastethernet 0/X
> switchport mode access
> switchport access vlan yY
> end
```

- Eliminar as VLANs no fim do procedimento:

```
> configure terminal
> no vlan y0
> no vlan y1
> end
```

4.2.2 Quantos domínios de transmissão existem? Como se pode concluir a partir dos *logs*?

Existem dois domínios de transmissão já que ao fazer *broadcast* apenas abrange as portas pertencentes a essa *virtual lan*, tal como foi visto na experiência.

4.3 Experiência 3 - Configuração de um *router* (*online*)

4.3.1 Como configurar uma rota estática num *router* comercial?

É necessário iniciar sessão no *router* através do *GTKTerm*, para tal finalidade é necessário ligar através do cabo de série S0 de um TUX da bancada à entrada de configuração do *router*. Para configurar as rotas temos que executar o comando *ip route* dentro do *GTKTerm*. O comando *ip route* segue a seguinte estrutura:

```
ip route prefix mask {ip-address | interface-type interface-number [ip-address]}
```

Exemplo de adicionar uma rota ao *router*:

```
> configure terminal
> ip route [destino] [máscara] [gateway]
> exit
```

4.3.2 Como configurar NAT num *router* comercial?

Para configurar a NAT num *router* comercial, foi necessário configurar a interface interna no processo de NAT de acordo com o guião fornecido. Foi ainda necessário reconfigurar as rotas IP. Assim sendo, a partir do terminal GTK foram inseridos os seguintes comandos:

```
> conf t
> interface gigabitethernet 0/0
```

```
> ip address 172.16.61.254 255.255.255.0
> no shutdown
> ip nat inside
> exit
> interface gigabitethernet 0/1
> ip address 172.16.1.69 255.255.255.0
> no shutdown
> ip nat outsider
> exit
> ip nat pool ovrld 172.16.1.69 172.16.1.69 prefix 24
> ip nat inside source list 1 pool ovrld overload
> access-list 1 permit 172.16.60.0 0.0.0.7
> access-list 1 permit 172.16.61.0 0.0.0.7
> end
```

4.3.3 O que é que faz o NAT?

O NAT (*Network Address Translation*) é uma técnica utilizada para a conservação de endereços IP. Esta técnica permite que redes IP privadas que utilizam endereços IP não registados se possam conectar à Internet. O NAT é aplicado num *router* que conecta, geralmente, duas redes e traduz os endereços privados (não globalmente únicos) na rede interna em endereços legais, antes de os pacotes serem encaminhados para outra rede. O NAT pode ser configurado para anunciar um endereço para toda a rede para a rede externa. Deste modo, fornece segurança adicional ao ocultar toda a rede por de trás desse endereço. O NAT oferece uma função dupla de segurança e de conservação de endereços e é, geralmente, utilizado em ambientes de acesso remoto.

4.3.4 Como configurar um serviço DNS num *host*?

O serviço DNS é configurado no ficheiro *resolv.conf*, localizado no diretório /etc/ do anfitrião *tux* em questão. A configuração é feita através de dois comandos, um que representa o nome do servidor DNS, e um com o respetivo endereço IP:

```
> search netlab.fe.up.pt
> nameserver 172.16.1.1
```

4.3.5 Que pacotes são trocados pelo DNS e que informação transportam?

Primeiramente, o *host* envia um pacote para o servidor que contém o *hostname* desejado, como por exemplo *google.com*, pedindo o seu endereço IP. De seguida, o servidor responde enviando um pacote contendo o IP do *hostname*.

4.3.6 Que pacotes ICMP são observados e porquê?

São observados pacotes ICMP de *request* e *reply* caso a rede tenha sido corretamente estabelecida. Caso contrário, em vez de pacotes ICMP de *reply* seriam pacotes ICMP de *Destination Unreachable* indicando um problema na rede.

4.3.7 Quais os endereços IP e MAC associados aos pacotes ICMP e porquê?

Os endereços IP e MAC de origem e destino associados com os pacotes ICMP são os endereços MAC e IP das máquinas/interfaces que recebem/enviam os pacotes.
Se as duas máquinas não estiverem conectadas à mesma *virtual network*, não é possível efetuar o envio de informação entre as duas máquinas com apenas um pacote ICMP sem este ser modificado.

4.4 Experiência 4 - Configuração de um *router* (*lab*)

Não conseguimos realizar esta experiência.

5 Conclusões

Este trabalho teve como objetivo o desenvolvimento de uma aplicação *download* e a configuração de uma rede de computadores. Esta aplicação *download* foi, posteriormente, testada nesta rede configurada no laboratório, tendo superado com sucesso todos os testes a que esteve sujeita.
Em suma, as experiências na rede estabelecida foram bem-sucedidas assim como o programa desenvolvido para a transferência de dados. Ao longo da realização do projeto, foram adquiridos importantes conhecimentos teórico-práticos em relação ao tema abordado, aprofundando a aprendizagem relativos a redes de computadores, algo presente no nosso quotidiano de forma relevante.

6 Anexo

6.1 Código

```c
#include "ftp.h"


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

    if(argc != 2){
        perror("Wrong number of arguments!\n");
        return -1;
    }

    urlInfo url;
    int fdSocket;
    int fdDataSocket;
    int fileSize;

    initializeUrlInfo(&url);

    if(parseUrlInfo(&url,argv[1]) < 0){
        printf("Error parsing url!\n");
        return -1;
    }

    if(getIpAddressFromHost(&url) < 0){
        printf("Error gettting IP Address!\n");
        return -1;
    }

    if(ftpStartConnection(&fdSocket,&url) < 0){
        printf("Error initializing connection!\n");
        return -1;
    }

    if(ftpLoginIn(&url,fdSocket) < 0){
        printf("Error while logging in!\n");
        return -1;
    }


    if(ftpPassiveMode(&url,fdSocket,&fdDataSocket) < 0){
        printf("Error entering in passive mode!\n");
        return -1;
    }


    if(ftpRetrieveFile(&url,fdSocket,&fileSize) < 0){
        printf("Error retrieving file!\n");
```

```
        return -1;
    }

    if(ftpDownloadAndCreateFile(&url,fdDataSocket, fileSize) < 0){
        printf("Error downloading/creating file!\n");
        return -1;
    }

    close(fdSocket);

    return 0;
}
```

```
#include "ftp.h"

int ftpStartConnection(int* fdSocket, urlInfo* url){

    char response[1024] = {0};
    int code;
    char aux[1024] = {0};

    if((*fdSocket = openSocket(url->ipAddress,url->port)) < 0){
        perror("Error opening socket!\n");
        return -1;
    }

    readSocketResponse(*fdSocket,response);

    memcpy(aux,response,3);
    code = atoi(aux);

    if(code != 220){
        return -1;
    }

    return 0;

}

int ftpLoginIn(urlInfo* url, int fdSocket){
    char userCommand[512] = {0};
    char passwordCommand[512] = {0};
    char responseToUserCommand[1024] = {0};
    char responseToPasswordCommand[1024] = {0};
    char aux[1024] = {0};
    int codeUserCommand;
    int codePasswordCommand;

    sprintf(userCommand,"user %s\n",url->user);
    sprintf(passwordCommand,"pass %s\n",url->password);
```

```
38
39
40      if(writeCommandToSocket(fdSocket,userCommand) < 0){
41          return -1;
42      }
43
44      if(readSocketResponse(fdSocket,responseToUserCommand) < 0){
45          return -1;
46      }
47
48      memcpy(aux,responseToUserCommand,3);
49
50      codeUserCommand = atoi(aux);
51
52
53      if(codeUserCommand != 230 && codeUserCommand != 331){
54          return -1;
55      }
56
57
58      if(writeCommandToSocket(fdSocket,passwordCommand) < 0){
59          return -1;
60      }
61
62      if(readSocketResponse(fdSocket,responseToPasswordCommand) < 0){
63          return -1;
64      }
65
66      memcpy(aux,responseToPasswordCommand,3);
67      codePasswordCommand = atoi(aux);
68
69      if(codePasswordCommand == 530){
70          return -1;
71      }
72
73
74      return 0;
75  }
76
77  int ftpPassiveMode(urlInfo* url, int fdSocket, int* dataSocket){
78
79      char responseToPassiveMode[1024] = {0};
80
81      if(writeCommandToSocket(fdSocket,"pasv\n") < 0){
82          return -1;
83      }
84
85      if(readSocketResponse(fdSocket,responseToPassiveMode) < 0){
86          return -1;
87      }
```

```
88
89
90      int firstIpElement, secondIpElement, thirdIpElement, fourthIpElement;
91      int firstPortElement, secondPortElement;
92      int codePassiveMode;
93
94      char serverIp[1024] = {0};
95      int serverPort;
96      char aux[1024] = {0};
97
98      memcpy(aux,responseToPassiveMode,3);
99      codePassiveMode = atoi(aux);
100
101     if(codePassiveMode != 227){
102         return -1;
103     }
104
105
106     sscanf(responseToPassiveMode,"227 Entering Passive Mode (%d,%d,%d,%d,%d
        ,%d)",&firstIpElement,&secondIpElement,&thirdIpElement,&fourthIpElement
        ,&firstPortElement,&secondPortElement);
107
108     sprintf(serverIp,"%d.%d.%d.%d",firstIpElement,secondIpElement,
        thirdIpElement,fourthIpElement);
109
110     serverPort = 256*firstPortElement+secondPortElement;
111
112
113     if((*dataSocket = openSocket(serverIp,serverPort)) < 0){
114         return -1;
115     }
116
117     return 0;
118 }
119
120 int ftpRetrieveFile(urlInfo* url, int fdSocket, int * fileSize){
121
122     char retrieveCommand[1024] = {0};
123     char responseToRetrieve[1024] = {0};
124     char aux[1024] = {0};
125     int codeRetrieve;
126
127     sprintf(retrieveCommand,"retr ./%s\n",url->urlPath);
128
129     if(writeCommandToSocket(fdSocket,retrieveCommand) < 0){
130         return -1;
131     }
132
133
134     if(readSocketResponse(fdSocket,responseToRetrieve) < 0){
```

```
135         return -1;
136     }
137 
138     memcpy(aux,responseToRetrieve,3);
139     codeRetrieve = atoi(aux);
140 
141     if(codeRetrieve != 150){
142         return -1;
143     }
144 
145     char auxSize[1024];
146     memcpy(auxSize,&responseToRetrieve[48],(strlen(responseToRetrieve)
    -48+1));
147 
148 
149     char path[1024];
150     sscanf(auxSize,"%s (%d bytes)",path,fileSize);
151 
152 
153     return 0;
154 }
155 
156 int ftpDownloadAndCreateFile(urlInfo* url, int fdDataSocket, int fileSize){
157 
158     int fd;
159     int bytesRead;
160     double totalSize = 0.0;
161     char buffer[1024];
162 
163     if((fd = open(url->fileName, O_WRONLY | O_CREAT, 0666)) < 0) {
164     perror("open()\n");
165     return -1;
166   }
167 
168     printf("\n");
169 
170     while ((bytesRead = read(fdDataSocket, buffer, 1024))) {
171 
172     if (bytesRead < 0) {
173       perror("read()\n");
174       return -1;
175     }
176 
177     if (write(fd, buffer, bytesRead) < 0) {
178       perror("write()\n");
179       return -1;
180     }
181 
182         totalSize += bytesRead;
183 
```

```
184         //progress bar
185         double percentage = totalSize/fileSize;
186         int val = (int) (percentage * 100);
187         int lpad = (int) (percentage * 60);
188         int rpad = 60 - lpad;
189         printf("\r%3d%% [%.*s%*s]", val, lpad, "
    ############################################################", rpad, ""
    );
190         fflush(stdout);
191   }
192
193     printf("\n");
194
195     if(totalSize != fileSize){
196         return -1;
197     }
198
199     close(fd);
200     close(fdDataSocket);
201
202     return 0;
203 }
```

```
1  #include "socket.h"
2  #include <sys/types.h>
3  #include <sys/stat.h>
4  #include <fcntl.h>
5
6  int ftpStartConnection(int* fdSocket, urlInfo* url);
7
8  int ftpLoginIn(urlInfo* url, int fdSocket);
9
10 int ftpPassiveMode(urlInfo* url, int fdSocket, int* dataSocket);
11
12 int ftpRetrieveFile(urlInfo* url, int fdSocket, int * fileSize);
13
14 int ftpDownloadAndCreateFile(urlInfo* url, int fdDataSocket, int fileSize);
```

```
1  #include "socket.h"
2
3
4  int openSocket(char* ipAddress, int port){
5      int sockfd;
6    struct  sockaddr_in server_addr;
7
8      bzero((char*)&server_addr,sizeof(server_addr));
9    server_addr.sin_family = AF_INET;
10   server_addr.sin_addr.s_addr = inet_addr(ipAddress);
11   server_addr.sin_port = htons(port);
```

```
12
13      if ((sockfd = socket(AF_INET,SOCK_STREAM,0)) < 0) {
14          perror("socket()");
15          return -1;
16      }
17    /*connect to the server*/
18      if(connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)
        ) < 0){
19          perror("connect()");
20          return -2;
21    }
22
23    return sockfd;
24  }
25
26  int writeCommandToSocket(int fdSocket,char* command){
27
28      int commandSize = strlen(command);
29      int bytesWritten;
30
31      if((bytesWritten = write(fdSocket,command,commandSize)) < commandSize){
32          perror("Error writing command to socket\n");
33          return -1;
34      }
35
36      return 0;
37  }
38
39  int readSocketResponse(int fdSocket,char * response){
40      FILE* file = fdopen(fdSocket,"r");
41
42      do {
43      memset(response, 0, 1024);
44      response = fgets(response, 1024, file);
45      printf("%s", response);
46
47    } while (!('1' <= response[0] && response[0] <= '5') || response[3] != '
        ');
48
49
50      return 0;
51  }
```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/types.h>
4  #include <sys/socket.h>
5  #include <netinet/in.h>
6  #include <arpa/inet.h>
7  #include <unistd.h>
```

```
#include <signal.h>
#include <netdb.h>
#include <string.h>
#include <strings.h>
#include "url.h"

int openSocket(char* ipAddress, int port);

int writeCommandToSocket(int fdSocket,char* command);

int readSocketResponse(int fdSocket,char* response);
```

```
#include "url.h"


void initializeUrlInfo(urlInfo* url){
    memset(url->user,0,256);
    memset(url->password,0,256);
    memset(url->host,0,256);
    memset(url->urlPath,0,256);
    memset(url->ipAddress,0,256);
    memset(url->fileName,0,256);
    url->port = 21;
}

char* getStringBeforeCharacther(char* str, char chr){

    char* aux = (char*)malloc(strlen(str));
    strcpy(aux, strchr(str, chr));
    int index = strlen(str) - strlen(aux);

    aux[index] = '\0';

    strncpy(aux,str,index);

    return aux;

}

int getIpAddressFromHost(urlInfo* url){
    struct hostent* h;

    if ((h=gethostbyname(url->host)) == NULL) {
        herror("gethostbyname");
        return -1;
    }

    strcpy(url->ipAddress,inet_ntoa(*((struct in_addr *)h->h_addr)));

    return 0;
```

```
}


int parseUrlInfo(urlInfo* url, char * urlGiven){
    char* protocol = "ftp://";
    char firstPart[7];
    memcpy(firstPart,urlGiven,6);

    if(strcmp(protocol,firstPart) != 0){
        return -1;
    }

    char * auxUrl = (char*) malloc(strlen(urlGiven));
    char * urlPath = (char*) malloc(strlen(urlGiven));

    strcpy(auxUrl,urlGiven+6);

    char* userNameGiven  = strchr(auxUrl,'@');

    //username given
    if(userNameGiven != NULL){
        strcpy(urlPath,userNameGiven+1);

        if(strchr(auxUrl,':') == NULL){
            char* aux;
            strcpy(url->user,getStringBeforeCharacther(auxUrl,'@'));

            printf("Please put the user password: \n");
            fgets(url->password,256,stdin);

            aux = strchr(url->password,'\n');
            *aux = '\0';

        }
        else{
            strcpy(url->user,getStringBeforeCharacther(auxUrl,':'));

            strcpy(auxUrl,auxUrl + strlen(url->user)+1);

            strcpy(url->password,getStringBeforeCharacther(auxUrl,'@'));
        }
    }
    else{
        strcpy(urlPath,auxUrl);

        strcpy(url->user,"anonymous");
        strcpy(url->password,"anypassword");
    }


```

```
89
90      strcpy(url->host,getStringBeforeCharacther(urlPath,'/'));
91
92      strcpy(urlPath,urlPath + strlen(url->host)+1);
93      int index = -1;
94
95
96      for (int i = strlen(urlPath)-1; i >= 0; i--)
97      {
98
99          if(urlPath[i] == '/'){
100             index = i;
101             break;
102         }
103     }
104
105     strcpy(url->urlPath,urlPath);
106     if(index == -1){
107         strcpy(url->fileName,urlPath);
108     }
109     else
110     {
111         strcpy(url->fileName,urlPath+index+1);
112     }
113
114     return 0;
115
116 }
```

```
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 <arpa/inet.h>
8  #include <string.h>
9
10 #define h_addr h_addr_list[0]
11
12
13 typedef struct {
14     char user[256];
15     char password[256];
16     char host[256];
17     char urlPath[256];
18     char ipAddress[256];
19     char fileName[256];
20     int port;
21
```

```
}urlInfo;

void initializeUrlInfo(urlInfo* url);

char* getStringBeforeCharacther(char* str, char chr);

int getIpAddressFromHost(urlInfo* url);

int parseUrlInfo(urlInfo* url, char * urlGiven);
```

6.2 Imagens

Figura 1

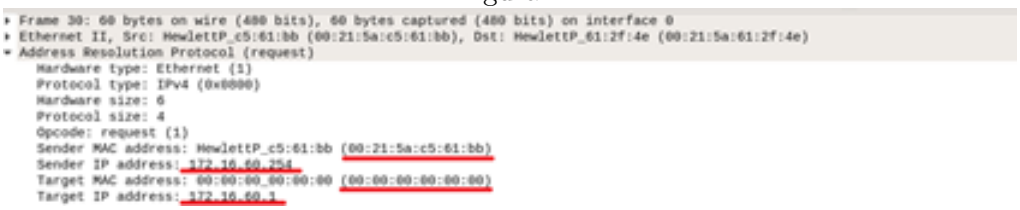

Figura 2

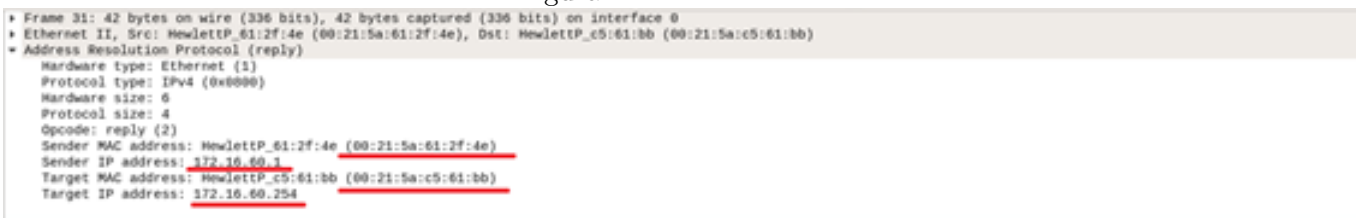

Figura 3

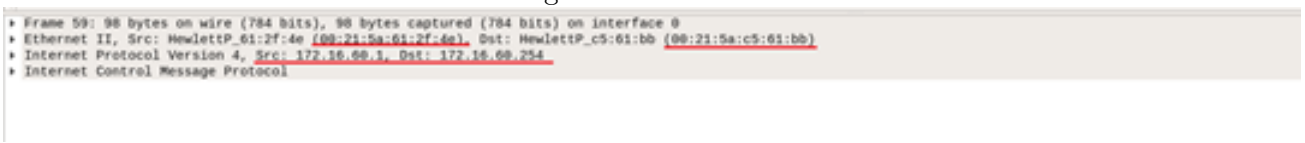

Figura 4

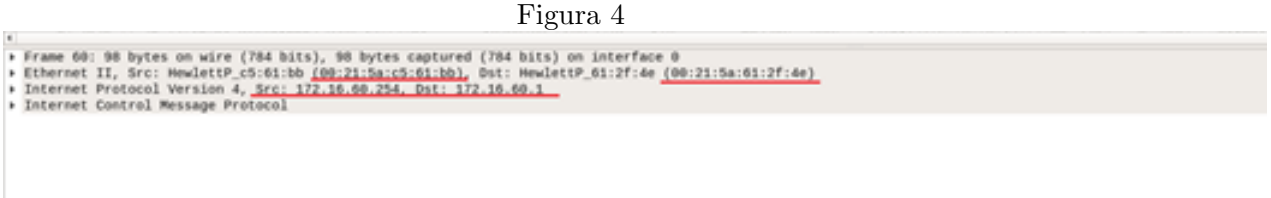

Figura 5

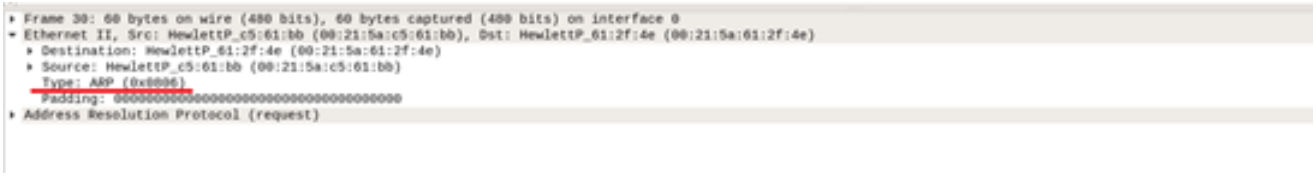

Figura 6

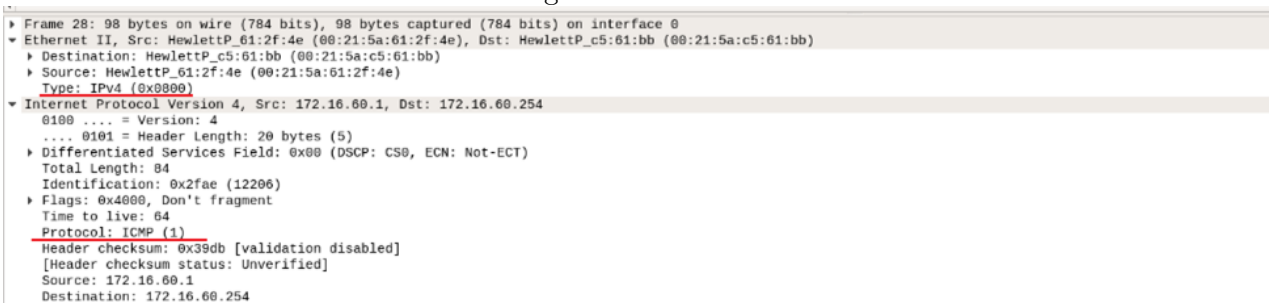

Figura 7

Figura 8