Faculdade de Engenharia da Universidade do Porto

2º Trabalho Laboratorial

Redes de Computadores 2020/2021

3MIEIC01 - 1ºsemestre

23 Dezembro 2020

Grupo 7

Ana Teresa Dias Silva (up201606703) - up201606703@fe.up.pt

Rodrigo Campos Reis (up201806534) - up201806534@fe.up.pt

Sumário

O presente relatório, elaborado como complemento do segundo trabalho laboratorial da Unidade Curricular de Redes de Computadores, está dividido em duas partes: aplicação de download e configuração de uma rede.

Este trabalho consiste no desenvolvimento de uma aplicação para download de um ficheiro utilizando o protocolo FTP. Adicionalmente, foi feita a configuração e análise de uma rede de computadores ao longo de seis experiências laboratoriais que serão detalhadas posteriormente neste documento.

Ambas as etapas do trabalho foram concluídas com sucesso tendo em conta que a aplicação de download é capaz de transferir qualquer ficheiro de diferentes servidores FTP e que a configuração da rede foi feita corretamente.

Índice

1. Introdução

O trabalho desenvolvido teve como base de objetivos: **desenvolver uma aplicação** capaz de transferir um ficheiro de um servidor FTP; **configurar e analisar uma rede** de computadores.

Este relatório é uma análise e reflexão do trabalho desenvolvido, procurando abordar os seguintes tópicos: **descrição da arquitetura da aplicação** e apresentação de resultados da sua execução; **descrição da arquitetura da rede**, apresentação dos objetivos de cada experiência laboratorial e dos principais comandos utilizados e análise dos logs recolhidos durante a execução das atividades; **considerações finais** sobre o desenvolvimento do projeto e o respetivo processo de aprendizagem.

2. Parte 1 - Aplicação de Download

Esta etapa do trabalho consiste no desenvolvimento de uma aplicação de download de ficheiros utilizando o protocolo FTP (File Transfer Protocol). O ficheiro RFC959 revelou-se bastante útil pela informação disponibilizada sobre este protocolo.

2.2. Arquitetura

A aplicação recebe como argumento um link com o formato **ftp://[<user>:<password>@]<host>/<url-path>**. O processamento deste URL é feito na função ***parseURL*** e os elementos *user*, *password*, *host*, *filePath* e *filename* são guardados numa struct ***arguments***.

Caso o *user* não seja indicado no URL, visto ser opcional, a conexão é feita em modo anonymous e a password é uma string vazia.

O endereço IP é obtido na função ***get_ip_address***, cujo código foi fornecido, recebendo como argumento o *host* previamente armazenado na estrutura de dados. O número da porta de controlo do protocolo FTP é sempre 21.

Na camada **clientFTP**, foi utilizada uma estrutura de dados ***ftp*** que armazena o descritor de ficheiro para o socket de controlo e outro descritor de ficheiro para o socket de dados. Primeiramente, na função ***open_socket***, é estabelecida a conexão entre o cliente FTP e o servidor FTP através de um socket TCP. Após feita a ligação, são enviados para o servidor, por esta ordem, os seguintes comandos: **USER user** e **PASS pass** (na função ***login***); **CWD filePath** (na função ***change_directory***, onde o diretório é alterado para aquele onde se encontra o ficheiro); **PASV** (na função ***passive_mode***, onde é feita a entrada no modo passivo e a abertura de um novo socket, numa porta diferente, de forma a permitir a troca de dados); **RETR filename** (na função ***retrieve***, para se obter do servidor o ficheiro que se pretende transferir).
Depois de se obter o ficheiro, é feito o download do mesmo na função ***download***. Por fim, a conexão é encerrada na função ***close_socket*** onde é enviado ao servidor o comando **QUIT**.

2.2. Resultados

A aplicação foi testada nos dois modos (anonymous e com um user/password), e em servidores FTP diferentes (nomeadamente os servidores *netlab1.fe.up.pt* e *ftp.up.pt*). Para efeitos de teste, foram transferidos ficheiros de tamanhos e formatos variados.

Em todas as situações o programa comportou-se da forma esperada, só encerrando no caso de existir algum erro - nestas situações, o respetivo erro é impresso na consola. Ao longo da execução, todas as respostas vão sendo também mostradas para que seja mais fácil para o utilizador acompanhar o processo.

Os resultados são apresentados em anexo na Figura 1.

3. Parte 2 - Configuração e Análise de Rede

Todos os comandos principais de configuração utilizados encontram-se disponíveis em anexo.

3.1. Experiência I - Configurar uma Rede IP

Nesta experiência os tux3 e tux4 são conectados através do switch, sendo atribuindo um endereço IP a cada uma das máquinas.

1. O que são os pacotes ARP e para que são usados?

O Protocolo de Resolução de Endereços (ARP) é um padrão de telecomunicação usado para mapear um endereço de rede (IPv4) de uma máquina num endereço físico (MAC) de outra máquina na rede local. Assim, quando uma máquina tenta enviar um pacote a outra dentro da mesma rede local, enviará um pacote ARP, por broadcast, para todas as máquinas ligadas a essa mesma rede e, simultaneamente, "questionando" qual das máquinas tem um endereço MAC correspondente ao endereço IP do destinatário. Por sua vez, o destinatário envia outro pacote ARP que indica ao computador emissor qual o seu endereço MAC. Consequentemente, a transferência de pacotes passa a poder ser efetuada.

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

Quando o tux3 tenta enviar um pacote ao tux4, como a entrada da tabela ARP referente ao tux4 foi apagada, o tux3 não sabe qual é o endereço MAC associado ao endereço IP do tux4 (172.16.10.254). Deste modo, irá enviar um pacote ARP para toda a rede local, sendo que este pacote contém o seu endereço IP (172.16.10.1) e o seu endereço MAC (00:21:5a:5a:7d:16). O endereço MAC do destinatário, sendo desconhecido, assume o valor de 00:00:00:00:00:00 (Figura 2).

De seguida, o tux4 irá enviar um pacote ARP para o tux3, com o endereço MAC dele (00:21:5a:5a:7b:3f), e o seu endereço IP (172.16.10.254). Portanto, pode-se concluir que cada pacote ARP contém campos para os endereços MAC e IP da máquina que envia, e para os endereços MAC e IP da máquina que recebe.

3. Que pacotes gera o comando *ping*?

O comando ping permite testar a conectividade, ao nível do endereço IP, entre equipamentos. Este gera primeiro pacotes ARP para saber qual o endereço MAC do destinatário e, posteriormente, pacotes ICMP (Internet Control Message Protocol).

4. Quais são os endereços MAC e IP dos pacotes *ping*?

Quando é efetuado um ping do tux3 para o tux4, os endereços dos pacotes enviados são os seguintes:

- Pacote request (de tux3 para tux4 → Figura 3):
  - IP address da source: 172.16.10.1 (tux3)
  - MAC address da source: 00:21:5a:5a:7d:16 (tux3)
  - IP address do destinatário: 172.16.10.254 (tux4)
  - MAC address do destinatário: 00:21:5a:5a:7b:3f (tux4)
- Pacote reply (de tux4 para tux3 → Figura 4):
  - IP address da source: 172.16.10.254 (tux4)
  - MAC address da source: 00:21:5a:5a:7b:3f (tux4)
  - IP address do destinatário: 172.16.10.1 (tux3)
  - MAC address do destinatário: 00:21:5a:5a:7d:16 (tux3)

5. Como determinar se uma frame recetora Ethernet é ARP, IP ou ICMP?

Conseguimos determinar o tipo de frame recetora através do Ethernet header da mesma. Se o seu valor for 0x0800, significa que a trama é do tipo IP. Se for 0x0806, a trama é do tipo ARP. Caso a trama seja do tipo IP, podemos analisar o seu IP header. Se este tiver valor igual a 1, então o tipo de protocolo é ICMP.

6. Como determinar o comprimento de uma frame recetora?

O comprimento de uma frame recetora pode ser visualizado no software Wireshark (Figura 5).

7. O que é a interface *loopback* e porque é importante?

O loopback pode ser um canal de comunicação com apenas um ponto final de comunicação. Assim, qualquer mensagem transmitida por meio desse canal é imediatamente recebida pelo mesmo canal.

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 (Figura 6).

3.2. Experiência II - Implementar duas VLANs virtuais num switch

Nesta experiência são criadas duas vlans virtuais (vlan0 e vlan1). Os tux 3 e 4 são associados à vlan0 e o tux2 à vlan1.

1. Como configurar vlany0?

Depois de aceder ao switch, no gtkterm efetuar os seguintes comandos:

- Criação das vlans (gtkterm):
  » configure terminal

» vlan 10 (vlan0)
» end

- Adicionar as portas do tux3 e do tux4 à respetiva vlan:
  » configure terminal
  » interface fastethernet 0/N (N é a porta no switch a que o tux está ligado)
  » switchport mode access
  » switchport access vlan 10
  » end

2. Quantos domínios broadcast existem? Como se conclui isso pelos logs?

Existem 2 domínios de broadcast já que ao fazer broadcast apenas abrange as portas pertencentes a essa virtual lan, tal como foi visto na experiência e nos respetivos logs: no tux3, quando é feito o ping tux2, é apresentado um erro "network is unreachable" pois este está num domínio diferente do tux3. Já o mesmo não acontece quando, também no tux3, é feito ping tux4 pois ambos pertencem ao mesmo domínio de broadcast.

3.3. Experiência III - Configurar um Router em Linux

Nesta experiência o tux4 é configurado como um router de forma a permitir a comunicação entre as duas VLANs criadas previamente na segunda experiência.

1. Que rotas existem nos *tuxes*? Qual o seu significado?

- Rotas de acesso às vlans:
  - tux3→vlan0 (172.16.y0.0): gateway 172.16.y0.1
  - tux4→vlan0 (172.16.y0.0): gateway 172.16.y0.254
  - tux4→vlan1 (172.16.y1.0): gateway 172.16.y1.253
  - tux2→vlan1 (172.16.y1.0): gateway 172.16.y1.1
- Rotas criadas na experiência - permitem comunicação entre vlans pelo tux4:
  - tux3→vlan1 (172.16.y1.0): gateway 172.16.y0.254
  - tux2→vlan0 (172.16.y0.0): gateway 172.16.y1.253

2. Que informação contém uma entrada de uma tabela de *forwarding*?

Em cada entrada pode ser verificado a **Destination** - IP da máquina de destino, o **Gateway** - endereço da máquina pela qual será transmitida a mensagem entre a origem e o destino, a **Interface** - placa de rede (eth0, por exemplo), a **Netmask** (utilizado para determinar o ID da rede a partir do endereço IP do destino), as **Flags** que contêm informações sobre as rotas, o valor de **Metric** (custo da rota), de **Ref** (número de referências para a rota - não utilizado no kernel do Linux) e de **Use** (contador de pesquisas pela rota que, dependendo do uso de -F e -C corresponderá ao número de falhas da cache ou ao número de sucessos.

3. Que mensagens ARP e endereços MAC associados são observados? Porquê?

Quando um tux envia um ping a outro tux e o tux emissor desconhece o endereço MAC do tux recetor, este envia uma mensagem ARP na qual pergunta qual o

endereço MAC do tux com aquele IP. Essa mensagem vai ter o MAC do tux de origem associado e 00:00:00:00:00:00 pois ainda não sabe qual o tux de destino. De seguida, o tux de destino responde com uma mensagem ARP na qual envia o seu endereço MAC. Esta mensagem vai ter associado tanto o endereço MAC do tux de destino como o de origem.

4. Que pacotes ICMP são observados e porquê?

São observados pacotes ICMP do tipo *request* e *reply* porque foram adicionadas todas as rotas, o que permite que todos os *tuxes* consigam reconhecer os restantes. Se tal não acontecesse, seriam enviados pacotes ICMP do tipo *Host Unreachable*.

5. Quais os endereços IP e MAC associados aos pacotes ICMP? Porquê?

Os endereços IP e MAC associados aos pacotes ICMP são os endereços dos tuxes de origem e destino. Por exemplo, ao fazer ping do tux3 para o tux4 (.254), os endereços IP e MAC de origem correspondem aos do tux3 (**IP**: 172.16.10.1; **MAC**: 00:21:5a:5a:7d:16) e os endereços IP e MAC de destino correspondem aos do tux4 na interface eth0 (**IP**: 172.16.10.254; **MAC**: 00:21:5a:5a:7b:3f).

3.4. Experiência IV - Configurar um router comercial e implementar NAT

Nesta experiência começou-se por configurar um router comercial, sem NAT, que foi ligado à rede do laboratório. De seguida, o mesmo router foi configurado com NAT de forma a permitir o acesso das máquinas daquela rede à internet.

1. Como implementar uma rota estática num router comercial?

Após ligar o cabo de série S0 de um tux à entrada de configuração do router, inicia-se sessão no router no gtkterm. Para configurar as rotas, usam-se os comandos:

» configure terminal;

» ip route [destino] [máscara] [gateway];

» exit

2. Quais os caminhos seguidos pelos pacotes nas experiências feitas? Porquê?

Quando as rotas existem, os pacotes seguem esse caminho. Caso contrário, seguem a rota default (pelo router).

3. Como configurar o NAT num router comercial?

O NAT foi configurado com base nas instruções fornecidas no slide 46 do guião, que pode ser consultado na Figura 7, tendo sido adaptados alguns passos, nomeadamente a substituição de *gigabitethernet* por *fastEthernet* e a alteração dos endereços IP de forma a corresponderem ao laboratório onde foi feita a experiência.

4. O que faz o NAT?

O NAT (Network Address Translation) permite traduzir IP locais num IP que permite acesso a uma rede pública, permitindo o mecanismo de troca de informação entre computadores de uma rede privada e uma rede pública. Como pode ser verificado na Figura 8, se o NAT não estiver configurado não é possível aceder a uma rede pública, através de uma máquina numa rede privada.

3.5. Experiência V - DNS

Nesta experiência, o DNS (Domain Name System) foi configurado para os tux 2, 3 e 4. Este sistema permite a tradução de um *hostname* para o respetivo endereço IP. Assim, foi possível verificar a alteração na forma como as máquinas conseguem conectar-se à internet.

1. Como configurar o serviço DNS num host?

No ficheiro ***resolv.conf***, que se encontra no diretório ***/etc***, colocar a seguinte informação: **search netlab1.fe.up.pt** e **nameserver 172.16.1.1** (nome e IP do servidor).

2. Que pacotes são trocados por DNS? Que informação é transportada?

O host envia um pacote com o hostname para o servidor esperando receber o seu IP. O recetor envia um pacote com o endereço IP do host.

3.6. Experiência VI - Conexões TCP

Nesta experiência foi observado o funcionamento do protocolo TCP recorrendo, para isso, à aplicação de download desenvolvida previamente e detalhada na parte 1.

1. Quantas conexões TCP foram abertas pela aplicação FTP?

Foram abertas duas conexões TCP: uma para o envio e receção de comandos para o servidor e outra para a transferência do ficheiro (troca de dados).

2. Em que conexão é transportado o controlo de informação?

O controlo de informação é transportado na conexão da troca de comandos.

3. Quais as fases da conexão TCP?

Na conexão TCP estabelece-se a conexão, posteriormente dá-se a troca de dados e, por fim, a conexão é encerrada.

4. Como funciona o mecanismo ARQ TCP? Quais os campos TCP relevantes? Qual a informação relevante pode ser observada nos logs?

O TCP utiliza o mecanismo ARQ com o método da janela deslizante que faz o controlo de erros durante a transmissão de dados. Para tal, são utilizados *acknowledgment numbers* (trama recebida com sucesso), *window size* (gama de pacotes que podem ser enviados) e *sequence number* (número do pacote a ser enviado).

5. Como funciona o mecanismo de controlo de congestionamento TCP? Como evolui o fluxo de dados da conexão ao longo do tempo? Está de acordo com o mecanismo de controlo de congestão TCP?

O mecanismo de controlo de congestionamento limita ou aumenta a taxa de envio de dados em função do congestionamento atual da rede. Na situação em que foram testados dois downloads em simultâneo, foi analisado o fluxo de dados (Figura 9). No início da conexão ocorre uma "partida lenta" e, posteriormente, a variação de fluxo sofre um aumento exponencial. Apesar de algumas variações, a taxa de transferência acaba por estabilizar durante alguns segundos, sendo que as descidas

mais abruptas se dão quando são detetados erros (assinalados a vermelho). Esta informação está de acordo com o mecanismo de congestionamento referido.

6. De que forma é afetada a conexão de dados TCP pelo surgimento de uma segunda conexão TCP? Como?

O surgimento de uma segunda conexão pode diminuir a taxa de transmissão da primeira conexão (taxa de transferência é distribuída de igual forma pelas duas conexões).

4. Conclusões

Neste segundo trabalho da Unidade Curricular, cujos objetivos eram desenvolver uma aplicação de download através do protocolo FTP e, adicionalmente, configurar uma rede IP de forma faseada, consideramos que os mesmos foram atingidos com sucesso. Os conhecimentos teóricos foram aprofundados e mais facilmente compreendidos ao longo do desenvolvimento do trabalho.

5. Referências

Durante o desenvolvimento deste trabalho foram consultados os slides teóricos da Unidade Curricular, o guião e também os RFC fornecidos, com destaque para o RFC959.

6. Anexos

6.1. Comandos de configuração (bancada 1)

- **tux12:**
  ifconfig eth0 172.16.11.1/24
  route add -net 172.16.10.0/24 gw 172.16.11.253
  route add default gw 172.16.11.254

- **tux13:**
  ifconfig eth0 172.16.10.1/24
  route add -net 172.16.11.0/24 gw 172.16.10.254
  route add -net 172.16.31.0/24 gw 172.16.30.254
  route add default gw 172.16.10.254
  echo -e 'search netlab.fe.up.pt\nnameserver 172.16.1.1' > /etc/resolv.conf

- **tux14:**
  ifconfig eth0 172.16.10.254/24
  ifconfig eth1 172.16.11.253/24
  route add default gw 172.16.11.254
  echo 1 > /proc/sys/net/ipv4/ip_forward
  echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

- **switch:**
  configure terminal
  vlan 10
  end
  configure terminal
  interface fastEthernet 0/1
  switchport mode access
  switchport access vlan 10
  end
  configure terminal
  interface fastEthernet 0/2
  switchport mode access
  switchport access vlan 10
  end
  configure terminal
  vlan 11
  end
  configure terminal
  interface fastEthernet 0/3
  switchport mode access
  switchport access vlan 11
  end

configure terminal
interface fastEthernet 0/4
switchport mode access
switchport access vlan 11
end
configure terminal
interface fastEthernet 0/5
switchport mode access
switchport access vlan 11
end

- **router (sem NAT):**
  configure terminal
  interface fastEthernet 0/0
  ip address 172.16.11.254 255.255.255.0
  no shutdown
  exit
  interface fastEthernet 0/1
  ip address 172.16.2.19 255.255.255.0
  no shutdown
  exit

6.2. Aplicação download

```
pc@home:~/Desktop/FEUP-RCOM/Project2/src$ ./download ftp://ftp.up.pt/pub/CPAN/RECENT-6h.json
User: anonymous
Password:
Host: ftp.up.pt
File Path: pub/CPAN
File Name: RECENT-6h.json
IP Address: 193.137.29.15
Socket successfully opened!

220-Welcome to the University of Porto's mirror archive (mirrors.up.pt)
220----------------------------------------------------------------
220-
220-All connections and transfers are logged. The max number of connections is 200.
220-
220-For more information please visit our website: http://mirrors.up.pt/
220-Questions and comments can be sent to mirrors@uporto.pt
220-
220-
220
Send info: USER anonymous

331 Please specify the password.
Send info: PASS

230 Login successful.
Send info: CWD pub/CPAN

250-The Comprehensive Perl Archive Network (http://www.cpan.org/)
250-master site has been from the very beginning (1995) hosted at FUNET,
250-the Finnish University NETwork.
250-
250-
250 Directory successfully changed.
Send info: PASV

227 Entering Passive Mode (193,137,29,15,229,66).
IP Address: 193.137.29.15
Port: 58690
Socket successfully opened!

Entered passive mode successfully.
Send info: RETR RECENT-6h.json

150 Opening BINARY mode data connection for RECENT-6h.json (14660 bytes).
Starting to download file with name RECENT-6h.json
226 Transfer complete.
Send info: QUIT
```

```
pc@home:~/Desktop/FEUP-RCOM/Project2/src$ ./download ftp://rcom:rcom@netlab1.fe.up.pt/files/crab.mp4
User: rcom
Password: rcom
Host: netlab1.fe.up.pt
File Path: files
File Name: crab.mp4
IP Address: 192.168.109.136
Socket successfully opened!

220 Welcome to netlab-FTP server
Send info: USER rcom

331 Please specify the password.
Send info: PASS rcom

230 Login successful.
Send info: CWD files

250 Directory successfully changed.
Send info: PASV

227 Entering Passive Mode (192,168,109,136,172,198).
IP Address: 192.168.109.136
Port: 44230
Socket successfully opened!

Entered passive mode successfully.
Send info: RETR crab.mp4

150 Opening BINARY mode data connection for crab.mp4 (88123184 bytes).
Starting to download file with name crab.mp4
226 Transfer complete.
Send info: QUIT
```

Figura 1 - Resultados da aplicação de download

6.3. Experiências Laboratoriais

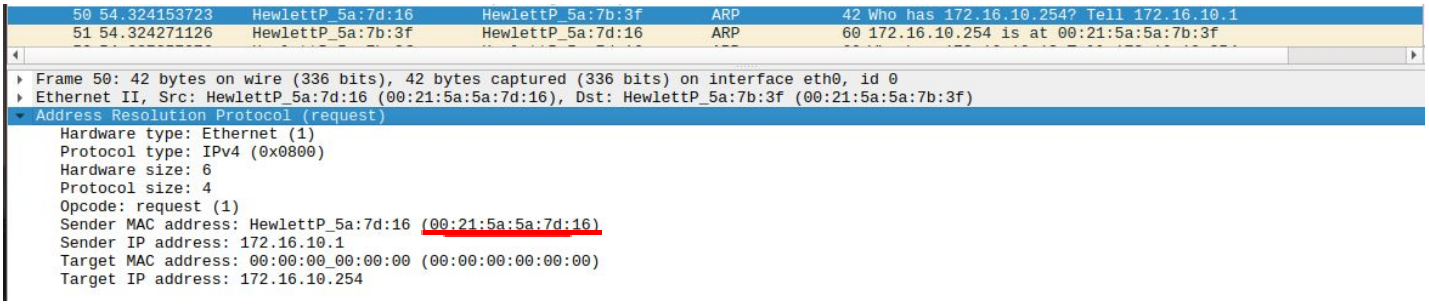


Figura 2 - Pacote ARP (endereço MAC desconhecido)

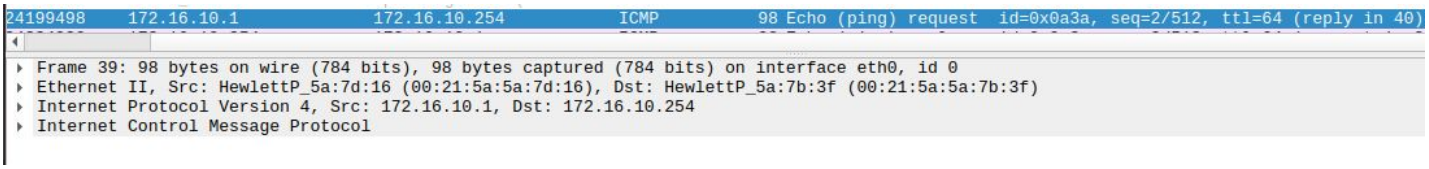


Figura 3 - Pacote Pedido

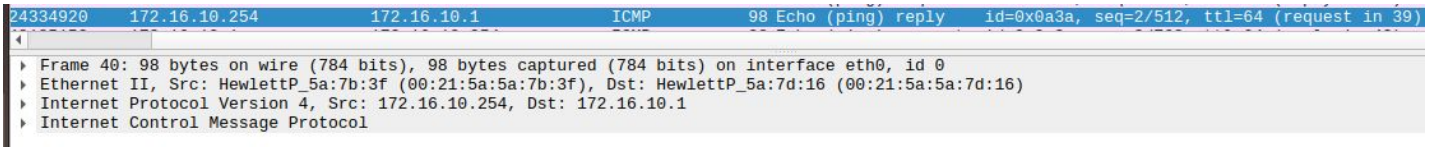


Figura 4 - Pacote Resposta

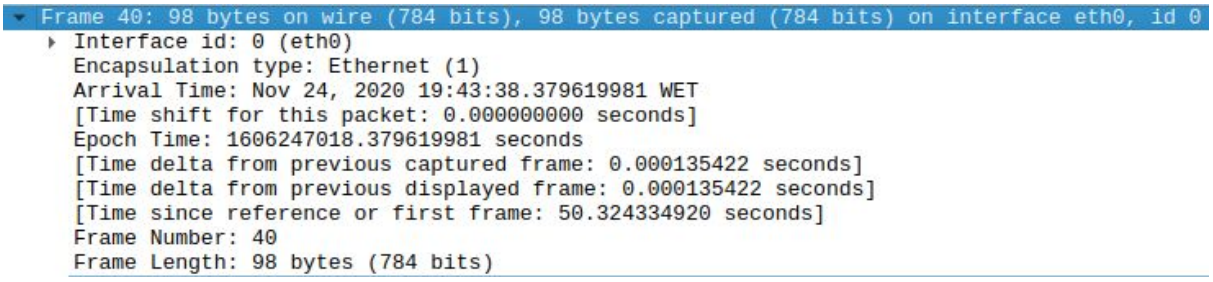


Figura 5 - Informação de uma Frame

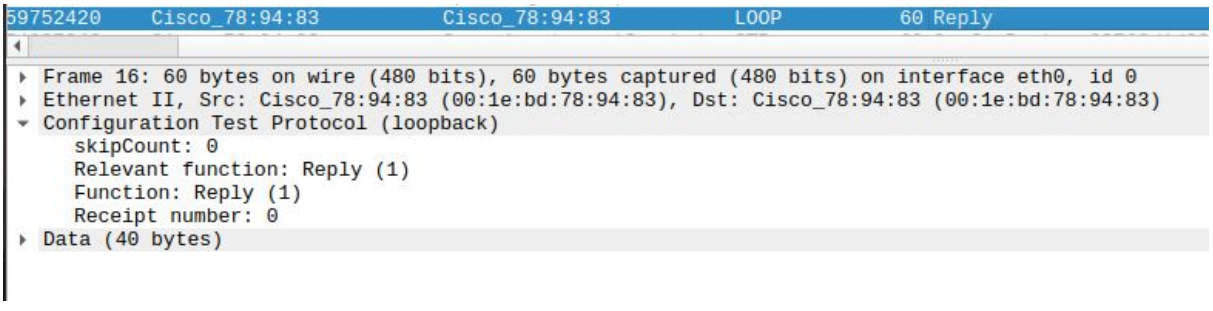
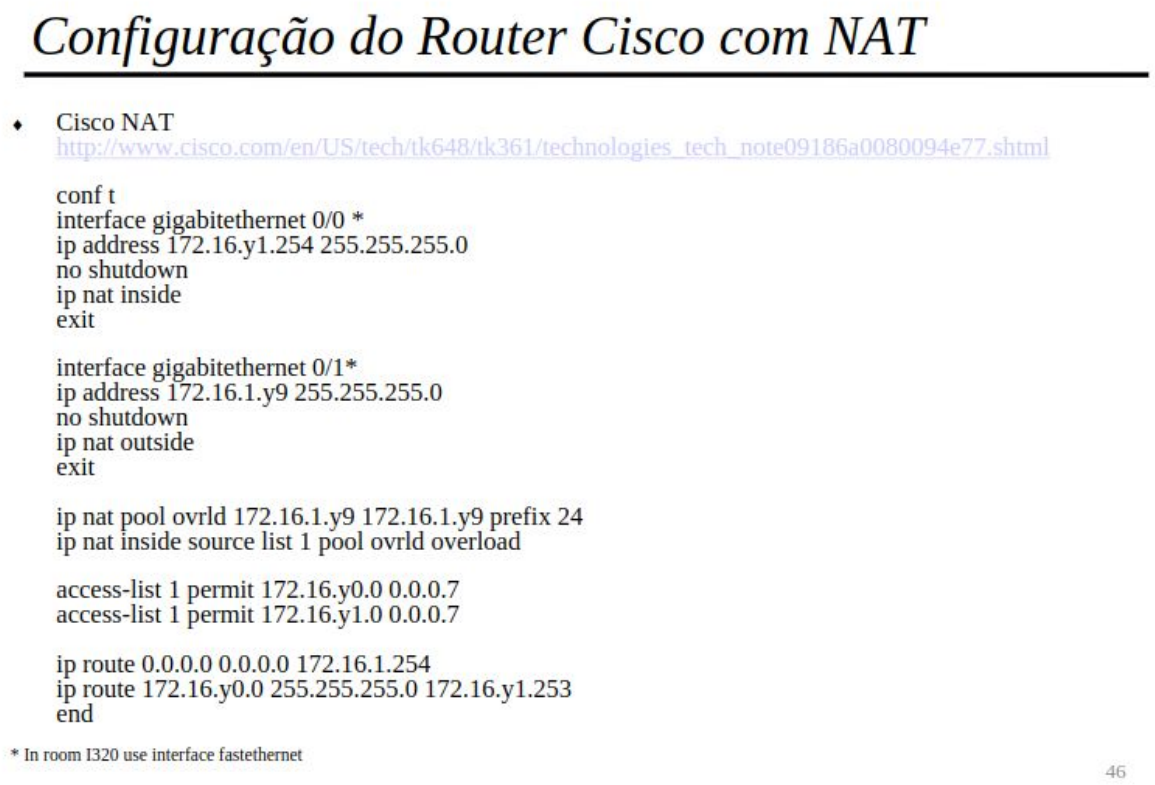


Figura 6 - Interface Loopback



Configuração do Router Cisco com NAT

- Cisco NAT
  http://www.cisco.com/en/US/tech/tk648/tk361/technologies_tech_note09186a0080094e77.shtml

```
conf t
interface gigabitethernet 0/0 *
ip address 172.16.y1.254 255.255.255.0
no shutdown
ip nat inside
exit

interface gigabitethernet 0/1*
ip address 172.16.1.y9 255.255.255.0
no shutdown
ip nat outside
exit

ip nat pool ovrld 172.16.1.y9 172.16.1.y9 prefix 24
ip nat inside source list 1 pool ovrld overload

access-list 1 permit 172.16.y0.0 0.0.0.7
access-list 1 permit 172.16.y1.0 0.0.0.7

ip route 0.0.0.0 0.0.0.0 172.16.1.254
ip route 172.16.y0.0 255.255.255.0 172.16.y1.253
end
```

* In room I320 use interface fastethernet

46

Figura 7 - Instruções de configuração de um Router Cisco com NAT

```
root@gnu63:~# ping 172.16.1.254
PING 172.16.1.254 (172.16.1.254) 56(84) bytes of data.
^C
--- 172.16.1.254 ping statistics ---
13 packets transmitted, 0 received, 100% packet loss, time 288ms
```

Figura 8 - Resultado do ping da rede do laboratório (sem NAT)

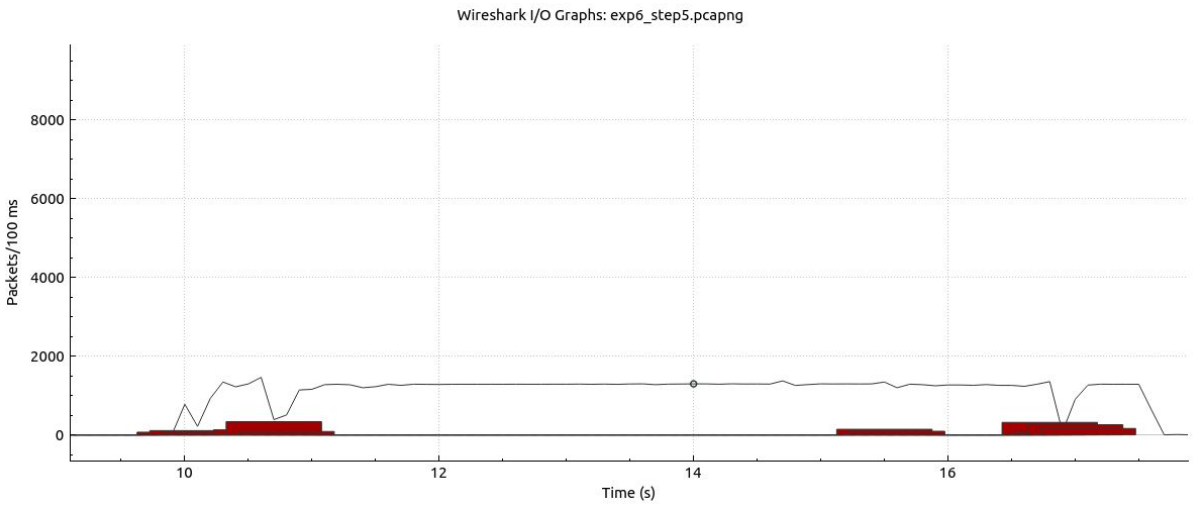


Figura 9 - Variação do fluxo de dados

6.4. Código Fonte - Aplicação de Download

```
//download.c
------------------------------------------------------------------

#include "utils.h"
#include "clientTCP.h"

//Example: ./download ftp://ftp.up.pt/pub/file.md

int main(int argc, char *argv[])
{
    char buffer[MAX_SIZE];

    if (argc != 2)
    {
        printf("Usage: ./download
ftp://[<user>:<password>@]<host>/<url-path>\n");
        return -1;
    }

    //Parse arguments
    struct arguments args;
    if (parseURL(argv[1], &args) != 0)
    {
```

```
        return -1;
    }

    printInfo(&args);

    char *ip_address;
    ip_address = get_ip_address(args.host);

    printf("IP Address: %s\n", ip_address);

    //Create and open socket
    struct ftp ftp_connection;

    ftp_connection.sockfd = open_socket(21, ip_address);

    if (ftp_connection.sockfd < 0)
    {
        printf("ERROR: Opening socket!\n");
        return -1;
    }

    if (read_from_socket(ftp_connection.sockfd, buffer,
sizeof(buffer)))
    {
        perror("ERROR: Reading from socket!");
        return -1;
    }

    if (login(&ftp_connection, args.user, args.password))
    {
        perror("ERROR: Login!");
        return -1;
    }

    if (strlen(args.filePath) > 0)
    {
        if (change_directory(&ftp_connection, args.filePath))
        {
            perror("ERROR: Changing directory");
            return -1;
        }
    }
```

```
    if (passive_mode(&ftp_connection))
    {
        perror("ERROR: Entering in passive mode!");
        return -1;
    }

    if (retrieve(&ftp_connection, args.fileName))
    {
        perror("ERROR: retrieve!");
        return -1;
    }

    if (download(&ftp_connection, args.fileName))
    {
        perror("ERROR: download file!");
        return -1;
    }

    if (close_socket(&ftp_connection))
    {
        perror("ERROR: Disconnect!");
        return -1;
    }

    return 0;
}
```

```
//clientTCP.h
-----------------------------------------------------------------
#include "utils.h"

/**
* @brief Open new socket using specified port and IP address
*
* @param port Port
* @param address IP address
```

```
* @return int 0 in case of success or -1 if any error occurs
*/
int open_socket(int port, char *address);

/**
* @brief Send a message to the socket with specified file
descriptor
*
* @param fd Socket file descriptor
* @param buf Message to send
* @param size Size of message
* @return int 0 in case of success or -1 if any error occurs
*/
int write_to_socket(int fd, char *buf, size_t size);

/**
* @brief Read from socket with specified file descriptor
*
* @param fd Socket file descriptor
* @param buf Message to be read
* @param buf_size Size of message
* @return int 0 in case of success or -1 if any error occurs
*/
int read_from_socket(int fd, char *buf, size_t buf_size);

/**
* @brief Login into the FTP server with received credentials
*
* @param ftp_connnection Struct with sockets file descriptors
* @param user User
* @param password User's password
* @return int 0 in case of success or -1 if any error occurs
*/
int login(struct ftp *ftp_connnection, char *user, char
*password);

/**
* @brief Change server directory
*
* @param ftp_connection Struct with sockets file descriptors
* @param path New path
* @return int 0 in case of success or -1 if any error occurs
```

```
*/
int change_directory(struct ftp *ftp_connection, char *path);

/**
* @brief Enter in passive mode
*
* @param ftp_connection Struct with sockets file descriptors
* @return int 0 in case of success or -1 if any error occurs
*/
int passive_mode(struct ftp *ftp_connection);

/**
* @brief File starts being transmited in the file socket
*
* @param ftp_connection Struct with sockets file descriptors
* @param file File name
* @return int 0 in case of success or -1 if any error occurs
*/
int retrieve(struct ftp *ftp_connection, char *file);

/**
* @brief Download file from FTP server
*
* @param ftp_connection Struct with sockets file descriptors
* @param file Transferred file name
* @return int 0 in case of success or -1 if any error occurs
*/
int download(struct ftp *ftp_connection, char *file);

/**
* @brief Close connection and sockets
*
* @param ftp_connection Struct with sockets file descriptors
* @return int 0 in case of success or -1 if any error occurs
*/
int close_socket(struct ftp *ftp_connection);
```

```
//clientTCP.c
-----------------------------------------------------------------

#include "clientTCP.h"

int open_socket(int port, char *address)
{
   int sockfd;
   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(address); /*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 ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
   {
       perror("socket()");
       return -1;
   }
   /*connect to the server*/
   if (connect(sockfd, (struct sockaddr *)&server_addr,
sizeof(server_addr)) < 0)
   {
       perror("connect()");
       return -1;
   }

   printf("Socket successfully opened!\n\n");

   return sockfd;
}

int write_to_socket(int fd, char *buf, size_t size)
{
   int bytes;

   if ((bytes = write(fd, buf, size)) <= 0)
```

```
    {
        printf("WARNING: Error sending to server.\n");
        return -1;
    }

    //printf("Bytes send: %d\nInfo: %s\n", bytes, buf);
    printf("Send info: %s\n", buf);
    return 0;
}

int read_from_socket(int fd, char *buf, size_t buf_size)
{
    FILE *fp = fdopen(fd, "r");

    do
    {
        memset(buf, 0, buf_size);
        buf = fgets(buf, buf_size, fp);
        printf("%s", buf);

    } while (!('1' <= buf[0] && buf[0] <= '5') || buf[3] != ' ');

    return 0;
}

int login(struct ftp *ftp_connection, char *user, char *password)
{

    char buffer[MAX_SIZE];

    sprintf(buffer, "USER %s\r\n", user);

    if (write_to_socket(ftp_connection->sockfd, buffer,
strlen(buffer)))
    {
        printf("ERROR: TCP send fail\n");
        return -1;
    }

    if (read_from_socket(ftp_connection->sockfd, buffer,
sizeof(buffer)))
    {
```

```
        printf("ERROR: Access denied- failed reading from
server!\n");
        return -1;
    }

    memset(buffer, 0, sizeof(buffer));
    sprintf(buffer, "PASS %s\r\n", password);

    if (write_to_socket(ftp_connection->sockfd, buffer,
strlen(buffer)))
    {
        printf("ERROR: TCP send fail\n");
        return -1;
    }

    if (read_from_socket(ftp_connection->sockfd, buffer,
sizeof(buffer)))
    {
        printf("ERROR: Access denied - failed reading from
server!\n");
        return -1;
    }

    return 0;
}

int change_directory(struct ftp *ftp_connection, char *path)
{
    char change_directory[MAX_SIZE];

    sprintf(change_directory, "CWD %s\r\n", path);
    if (write_to_socket(ftp_connection->sockfd, change_directory,
strlen(change_directory)))
    {
        printf("ERROR: Failed sending path to CWD.\n");
        return -1;
    }

    if (read_from_socket(ftp_connection->sockfd, change_directory,
sizeof(change_directory)))
    {
        printf("ERROR: Failed changing directory.\n");
```

```
        return -1;
    }

    return 0;
}

int passive_mode(struct ftp *ftp_connection)
{
    char passive[MAX_SIZE] = "PASV\r\n";
    char ip_address[MAX_SIZE];
    int port;

    if (write_to_socket(ftp_connection->sockfd, passive,
strlen(passive)))
    {
        printf("ERROR: Failed entering passive mode!\n");
        return -1;
    }

    if (read_from_socket(ftp_connection->sockfd, passive,
sizeof(passive)))
    {
        printf("ERROR: Not getting response entering in passive
mode!\n");
        return -1;
    }

    if (parse_passive_response(passive, ip_address, &port))
    {
        perror("ERROR: Passive response failed!\n");
        return -1;
    }

    printf("IP Address: %s\n", ip_address);
    printf("Port: %d\n", port);

    ftp_connection->datafd = open_socket(port, ip_address);
    if (ftp_connection->datafd < 0)
    {
        printf("ERROR: Cannot open socket!\n");
        return -1;
    }
```

```
    printf("Entered in passive mode successfully.\n");
    return 0;
}

int retrieve(struct ftp *ftp_connection, char *file)
{
    char retr[MAX_SIZE];

    sprintf(retr, "RETR %s\r\n", file);
    if (write_to_socket(ftp_connection->sockfd, retr,
strlen(retr)))
    {
        printf("ERROR: Sending filename\n");
        return -1;
    }

    if (read_from_socket(ftp_connection->sockfd, retr,
sizeof(retr)))
    {
        printf("ERROR: Filename not received!\n");
        return -1;
    }

    return 0;
}

int download(struct ftp *ftp_connection, char *filename)
{
    FILE *file;
    int bytes;

    if (!(file = fopen(filename, "w")))
    {
        printf("ERROR: Cannot open file.\n");
        return -1;
    }

    char buf[MAX_SIZE];
    printf("Starting to download file with name %s\n", filename);
    while ((bytes = read(ftp_connection->datafd, buf,
sizeof(buf))))
```

```
    {
        if (bytes < 0)
        {
            printf("ERROR: Nothing was received from data socket
fd.\n");
            return -1;
        }

        if ((bytes = fwrite(buf, bytes, 1, file)) < 0)
        {
            printf("ERROR: Cannot write data in file.\n");
            return -1;
        }
    }
    if (fclose(file) < 0)
    {
        printf("ERROR: closing file\n");
        return -1;
    }

    close(ftp_connection->datafd);

    return 0;
}

int close_socket(struct ftp *ftp_connection)
{
    char buffer[MAX_SIZE];

    if (read_from_socket(ftp_connection->sockfd, buffer,
sizeof(buffer)))
    {
        printf("ERROR: Cannot disconnect.\n");
        return -1;
    }

    sprintf(buffer, "QUIT\r\n");

    if (write_to_socket(ftp_connection->sockfd, buffer,
strlen(buffer)))
    {
        printf("ERROR: Failed disconnection!\n");
```

```
        return -1;
    }

    if (ftp_connection->sockfd)
        close(ftp_connection->sockfd);

    return 0;
}


//utils.h
-----------------------------------------------------------------------

#pragma once

#include <stdio.h>
#include <string.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 <strings.h>
#include <errno.h>

#define MAX_SIZE 256

/**
* @brief Struct to store information about elements received in
the URL
*
*/
struct arguments
{
    char user[MAX_SIZE];
    char password[MAX_SIZE];
    char host[MAX_SIZE];
    char filePath[MAX_SIZE];
    char fileName[MAX_SIZE];
```

```
};

/**
* @brief Struct to store sockets file descriptors
*
*/
struct ftp
{
   int sockfd;
   int datafd;
};

/**
* @brief Verify if a string matches ftp://
*
* @param str String to compare
* @return int 0 if str is not equal to ftp://
*/
int verify_ftp(const char *str);

/**
* @brief Parse all elements of the received URL and fills the args
struct with parsed information
*
* @param complete_url URL to parse
* @param args Struct to store information about all the elements
received in the URL
* @return int 0 in case of sucessful parse or -1 if any error
occurs
*/
int parseURL(char *complete_url, struct arguments *args);

/**
* @brief Print on the screen information stored on args
*
* @param args Struct with information
*/
void printInfo(struct arguments *args);

/**
* @brief Get the ip address of host
*
```

```
* @param host_name Hostname
* @return char* IP address
*/
char *get_ip_address(char *host_name);

/**
* @brief Send command to server in order to enter on passive mode
*
* @param response Server response
* @param ip_address New ip address for passive mode
* @param port New port to open socket on passive mode
* @return int 0 in case of success or -1 if any error occurs
*/
int parse_passive_response(char *response, char *ip_address, int
*port);


//utils.c
-----------------------------------------------------------------
#include "utils.h"

int verify_ftp(const char *str)
{
   char *ftp_str = "ftp://";
   char aux[7];

   memcpy(aux, str, 6);

   return !strcmp(ftp_str, aux);
}

int parseURL(char *complete_url, struct arguments *args)
{
   // verify ftp
   char start[6];
   strncpy(start, complete_url, 6);
   complete_url += 6;
   if (!verify_ftp(start))
   {
       printf("URL must start with ftp://\n");
       return -1;
   }
```

```
char *token = strtok(complete_url, "\0");
char url[MAX_SIZE];
strcpy(url, token);

//reads user
char aux[MAX_SIZE];
strcpy(aux, url);

token = strtok(aux, ":");

if (token == NULL)
{
    printf("Failed parsing user!\n");
    return -1;
}
else if (strcmp(token, url) == 0) //User is not indicated
{
    memset(args->user, 0, sizeof(args->user));
    strcpy(args->user, "anonymous");
    memset(args->password, 0, sizeof(args->password));
    strcpy(args->password, "");

    char aux2[MAX_SIZE];
    strcpy(aux2, &url[0]);
    strcpy(url, aux2);
}
else
{
    memset(args->user, 0, sizeof(args->user));
    strcpy(args->user, &token[0]);

    //read passowrd
    token = strtok(NULL, "@");
    if (token == NULL || (strlen(token) == 0))
    {
        printf("Failed parsing password\n");
        return -1;
    }
    memset(args->password, 0, sizeof(args->password));
    strcpy(args->password, token);
```

```
        token = strtok(NULL, "\0");
        strcpy(url, token);
    }

    //read host
    token = strtok(url, "/");
    if (token == NULL)
    {
        printf("Failed parsing host\n");
        return -1;
    }
    memset(args->host, 0, sizeof(args->host));
    strcpy(args->host, token);

    //get path and name
    token = strtok(NULL, "\0");
    if (token == NULL)
    {
        printf("Failed parsing file path\n");
        return -1;
    }
    char *name = strrchr(token, '/');
    if (name != NULL)
    {
        //get path
        memset(args->filePath, 0, sizeof(args->filePath));
        strncpy(args->filePath, token, name - token);

        //get name
        memset(args->fileName, 0, sizeof(args->fileName));
        strcpy(args->fileName, name + 1);
    }
    else //only filename is indicated
    {
        memset(args->filePath, 0, sizeof(args->filePath));
        strcpy(args->filePath, "");
        memset(args->fileName, 0, sizeof(args->fileName));
        strcpy(args->fileName, token);
    }

    return 0;
}
```

```
void printInfo(struct arguments *args)
{
   printf("User: %s\n", args->user);
   printf("Password: %s\n", args->password);
   printf("Host: %s\n", args->host);
   printf("File Path: %s\n", args->filePath);
   printf("File Name: %s\n", args->fileName);
}

char *get_ip_address(char *host_name)
{
   struct hostent *h;

   if ((h = gethostbyname(host_name)) == NULL)
   {
       herror("gethostbyname");
       exit(1);
   }

   return inet_ntoa(*((struct in_addr *)h->h_addr_list[0]));
}

int parse_passive_response(char *response, char *ip_address, int
*port)
{
   int h1, h2, h3, h4, p1, p2;

   //227 Entering Passive Mode (h1,h2,h3,h4,p1,p2).
   if (sscanf(response, "227 Entering Passive Mode
(%d,%d,%d,%d,%d,%d)", &h1, &h2, &h3, &h4, &p1, &p2) < 0)
   {
       perror("Passive response!\n");
       return -1;
   }

   //Create new address
   sprintf(ip_address, "%d.%d.%d.%d", h1, h2, h3, h4);

   //Calculate new port
   *port = p1 * 256 + p2;
```

```
    return 0;
}
```