Aplicação de download e configuração de rede



Relatório

Redes de Computadores

Francisco Teixeira Lopes

ei11056

Nelson André Garrido da Costa

up201403128

**Sumário**

Este relatório incide sobre a criação de uma aplicação de *download* e configuração de uma rede.

A aplicação utiliza o protocolo FTP com ligações TCP para efetuar a transferência, implementando uma versão leve do RFC959.

A rede consiste numa série de experiências que culminaram em configurar um *router* comercial com NAT, um *switch* com duas LAN virtuais, e cada computador da rede com respetivo endereço IP e DNS.

O objetivo final, foi efetuar uma transferência usando a aplicação na rede configurada.

Índice

[Introdução 1](#_Toc501729546)

[Parte 1 – Aplicação de *download* 2](#_Toc501729547)

[Arquitetura da aplicação 2](#_Toc501729548)

[Exemplo de *download* 3](#_Toc501729549)

[Parte 2 – Configuração de rede 4](#_Toc501729550)

[Experiência 1 – Configurar uma rede IP 4](#_Toc501729551)

[Experiência 2 – Implementar duas LAN virtuais num *switch* 5](#_Toc501729552)

[Experiência 3 – Configurar um *router* em Linux 5](#_Toc501729553)

[Experiência 4 – Configurar um *router* comercial e implementar NAT 6](#_Toc501729554)

[Experiência 5 – DNS 6](#_Toc501729555)

[Experiência 6 – Ligações TCP 7](#_Toc501729556)

[Conclusão 9](#_Toc501729557)

[Anexos 10](#_Toc501729558)

[Anexo 1 – Comandos de configuração 10](#_Toc501729559)

[Experiência 1 10](#_Toc501729560)

[Experiência 2 10](#_Toc501729561)

[Experiência 3 11](#_Toc501729562)

[Experiência 4 (sem NAT) 11](#_Toc501729563)

[Experiência 4 (NAT) 12](#_Toc501729564)

[Experiência 5 12](#_Toc501729565)

[Anexo 2 – Capturas 13](#_Toc501729566)

[Experiência 1 13](#_Toc501729567)

[Experiência 2 13](#_Toc501729568)

[Experiência 3 15](#_Toc501729569)

[Experiência 4 15](#_Toc501729570)

[Experiência 5 16](#_Toc501729571)

[Experiência 6 16](#_Toc501729572)

[Anexo 3 – Código da aplicação de download 17](#_Toc501729573)

[Download.h 17](#_Toc501729574)

[Download.c 20](#_Toc501729575)

[Makefile 32](#_Toc501729576)

# Introdução

O segundo trabalho laboratorial de Redes de Computadores divide-se em duas partes: criação de uma aplicação de *download* e configuração de uma rede.

A primeira parte, tem por objetivo efetuar a transferência de um ficheiro a partir de um servidor, para isto, é utilizado o protocolo FTP com implementação do RFC959 e conexões TCP ao servidor.

A segunda parte, tem por objetivo configurar uma rede de forma a poder efetuar a transferência de um ficheiro. Foram efetuadas várias experiências incrementais, que eventualmente, permitiram acesso à Internet a partir da rede configurada.

Este relatório encontra-se dividido em duas seções principais:

* Aplicação de *download*: nesta seção descreve-se a arquitetura da aplicação e apresenta-se um relato de uma transferência feita com sucesso.
* Configuração de rede: nesta seção apresentam-se os passos para configuração da rede e reflete-se sobre os objetivos de aprendizagem de cada experiência.

# Parte 1 – Aplicação de *download*

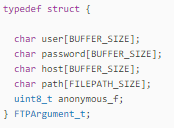
## Arquitetura da aplicação

O processamento da aplicação acontece na função main do programa, esta chama, sequencialmente, todas as funções auxiliares necessárias para descodificar o argumento passado pelo utilizador, a obtenção dos dados necessários para poder fazer a ligação TCP ao servidor FTP, e ainda, as operações necessárias para conseguir fazer o *download* do ficheiro pedido.

Para este efeito são usadas duas *structs* com dados utéis ao programa, a primeira *struct*, FTPFile\_t, contém o caminho para onde guardar o ficheiro localmente, e também, o descritor de ficheiro do *socket* de dados.

A segunda *struct*, FTPArgument\_t, contém os comandos FTP necessários para obter o ficheiro, bem como, um valor booleano que determina se o modo é *anonymous*.

Figura 1 - estruturas de dados da aplicação



A aplicação é executada recorrendo a um argumento, que indica o nome de utilizador e a palavra-passe, juntamente com, o URL do servidor e ficheiro a transferir. Pode-se também omitir o utilizador e palavra-passe, sendo que, é assumido o modo *anonymous*.

Figura 2 - utilização da aplicação

Após executar a aplicação, é chamada uma sequência de funções com a finalidade de transferir o ficheiro pedido. Primeiramente, é validado o argumento fornecido, para esse efeito, a função “*parseArgument*” verifica que todos os dados necessários estão presentes. A *struct* FTPArgument\_t é preenchida por esta função, no caso de argumento válido. De seguida, é obtido o endereço IP do servidor através da função “*getAddress*”, o qual é fornecido à função “*getTCPSocket*”. Esta, cria uma ligação TCP à porta 21 do servidor, isto é, a porta de controlo de FTP. Após estabelecer ligação ao servidor, são enviados os comandos necessários para transferir o ficheiro, a função “*FTPLogin*” envia os comandos “USER” e “PASS”, de seguida “*FTPCommand*” envia “TYPE I” para usar modo binário e finalmente “*FTPPassive*” envia o comando “PASV” e retorna a porta de dados. Sabendo a porta de dados, “*getTCPSocket*” cria outra ligação TCP com o mesmo endereço IP mas porta distinta. Criada esta ligação, “*FTPCommand*” envia “RETR” e inicia a transferência do ficheiro, isto é feito através de um valor booleano na função, que indica a possibilidade do comando ativar uma transferência. Termina, assim, a aplicação com o ficheiro transferido na pasta onde foi executada, qualquer erro é devidamente mostrado ao utilizador e, tratando-se da seção FTP, é tratado de acordo com o RFC959.

## Exemplo de *download*

Para fazer transferência de um ficheiro, basta fornecer à aplicação um URL correto e, opcionalmente, o utilizador e palavra-passe. Seguem-se imagens de testes bem sucedidos, um em modo anónimo, e o outro com dados de utilizador.

Figura 3 - download com utilizador

Os comandos enviados foram:

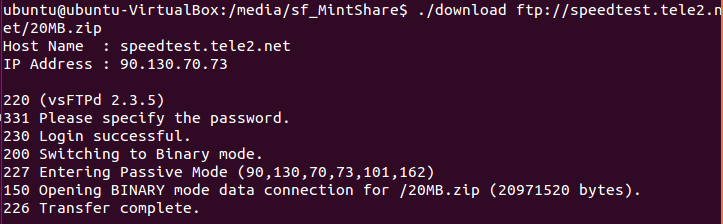
* ./download ftp://demo:password@test.rebex.net/pub/example/mail-editor.png
* ./download ftp://speedtest.tele2.net/20MB.zip

Figura 4 - download em modo anónimo

# Parte 2 – Configuração de rede

Nos seguintes subcapítulos são elucidados os objetivos de cada experiência, sendo que, a configuração exaustiva é apresentada em anexo no final do relatório, assim como, os respetivos *logs* de captura de pacotes.

## Experiência 1 – Configurar uma rede IP

Esta experiência teve por objetivo a configuração de uma rede local, na qual, se ligava um computador a outro. Para este efeito, ligaram-se os computadores diretamente através da porta eth0, e de seguida, configuraram-se os seus endereços IP de forma a situarem-se na mesma sub-rede (172.16.10.x).

**O que são pacotes ARP e para que servem?**

Os pacotes ARP servem para mapear um endereço de rede (IPv4) a um endereço físico (MAC).

**Qual é o MAC e endereço IP de pacotes ARP?**

Existem pacotes ARP de pedido e de resposta, num pacote de pedido, é enviado o endereço IP do computador que fez o pedido, e também, o endereço IP do computador do qual se quer saber o MAC. Num pacote de resposta, é enviado ao computador que fez o pedido, o endereço IP e MAC do computador destino.

Figura 5 - exemplo de pacotes ARP

**Que pacotes gera o comando *ping*?**

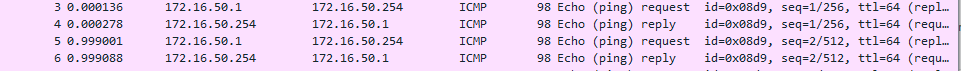
O comando *ping* gera pacotes ICMP, estes podem ser do tipo *echo request* ou do tipo *echo reply*.

Figura 6 - exemplo de pacotes ICMP

**Qual é o MAC e endereço IP de pacotes ping?**

Os pacotes ICMP contêm o endereço IP do computador origem e do computador destino, isto é válido tanto para *echo request*, como para *echo reply*.

**Como se determina se uma trama Ethernet é do tipo ARP, IP, ICMP?**

No cabeçalho da trama Ethernet, no campo *EtherType*, é possível verificar se se trata de um pacote ARP (0x0806) ou IP (0x0800). Para pacotes ICMP (tipo 1), esta informação, vem no cabeçalho do pacote IP no campo *Protocol*.

**Como se determina o tamanho de uma trama Ethernet?**

O tamanho da trama é indicado no campo *EtherType* da trama, para evitar ambiguidade com o outro uso deste campo, utilizam-se valores até 1500 para referir o tamanho, e valores acima de 1536 para o tipo.

**O que é a interface de *loopback* e para que serve?**

A interface de *loopback* é utilizada para um computador poder comunicar consigo mesmo, isto tem a utilidade de poder realizar testes de diagnóstico, e também, de poder aceder a servidores no próprio computador.

## Experiência 2 – Implementar duas LAN virtuais num *switch*

Esta experiência teve por objetivo a implementação de LAN virtual, dois computadores seriam ligados a uma VLAN e um terceiro a outra VLAN sem rota entre as duas VLAN.

Para este efeito, configurou-se um *switch* da Cisco e respetivas portas para ligar o computador 1 e 4 numa VLAN, e o computador 2 noutra VLAN.

**Quantos domínios de *broadcast* existem? Como se pode concluir isso a partir dos *logs*?**

Existem dois domínios de *broadcast* na rede configurada, pois a divisão em VLAN cria duas sub-redes. Nos *logs* pode-se ver isto pois ao fazer *ping* *broadcast* a partir do computador 1, chega-se apenas ao computador 4, e ao fazer ping broadcast do computador 2, não se chega a outro computador. Isto deve-se às redes criadas préviamente.

Figura 7 - computador 1 fazendo broadcast, resposta de computador 4

## Experiência 3 – Configurar um *router* em Linux

Esta experiência teve por objetivo utilizar o computador 4 como *router*, de forma a ligar as duas sub-redes criadas na experiência anterior.

Para este efeito, configurou-se a porta eth1 do computador 4 com um endereço IP da sub-rede do computador 2, e também, alteraram-se algumas configurações de forma a ativar *IP Forwarding* e desativar o *ICMP\_echo\_ignore\_broadcast*. Definiram-se também as rotas necessárias para cada sub-rede ter acesso à outra, usando o computador 4 como *gateway*.

**Que rotas existem nos computadores? Qual o seu significado?**

O computador 1 tem uma rota para a sub-rede do computador 2 e vice-versa, por exemplo, “172.16.11.1 172.16.10.254 255.255.255.0”. Esta rota permite ao computador 1 saber que para o destino 172.16.11.1 (computador 2), deve usar o endereço 172.16.10.254 como *gateway*, neste caso é o computador 4 (*router*).

**Que informação contém uma entrada na tabela de encaminhamento?**

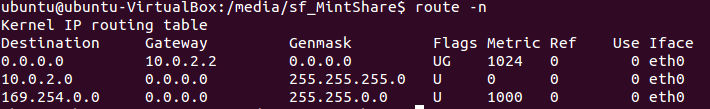
As tabelas de encaminhamento contém a informação necessária para fazer chegar pacotes ao endereço IP destino. Para isto contêm informação de destino e para onde deve ser encaminhado o pacote para chegar ao destino.

Figura 8 - exemplo de tabela de encaminhamento

**Qual o comportamento dos pacotes ARP e ICMP nesta rede?**

Estando no computador 1 e fazendo ping ao computador 2, verifica-se que o computador 1 envia para o computador 4 um pacote ICMP com o endereço IP do computador 2 no campo de destino. O computador 4, caso não tenha em cache o MAC do computador 2, envia um pacote ARP para descobrir o MAC associado ao endereço IP do ping recebido, após descobrir este MAC envia o ping original ao computador 2. O processo de resposta é idêntico com os valores invertidos, o computador 4 serve então de *router*, como é demonstrado por este comportamento.

## Experiência 4 – Configurar um *router* comercial e implementar NAT

Esta experiência teve por objetivo adicionar à rede um *router* comercial, este, configurado com NAT para permitir acesso à Internet.

Para este efeito, adicionou-se à VLAN do computador 2/4, o *router* comercial, de seguida, configuraram-se as rotas necessárias para ser possível receber pacotes de qualquer combinação de computadores. Finalmente, configurou-se NAT no *router*, para permitir acesso a redes externas.

**Qual o caminho seguido pelos pacotes na experiência?**

Inicialmente, o computador 2 tem apenas uma rota através do *router* comercial, fazendo *ping* a um endereço da outra sub-rede tem por resultado o pacote parar primeiro no *router* comercial, de seguida pára no computador 4 e daí é feita a ligação à outra sub-rede. Adicionando a rota à outra sub-rede, usando o computador 4 como *gateway*, faz com que o caminho anterior se torne mais curto, pois não tem de passar no *router* comercial primeiro.

**O que faz o NAT?**

NAT permite mapear um espaço de endereçamento IP a outro espaço, modificando o cabeçalho de datagramas, enquanto estes transitam num dispositivo de *routing*. O seu uso atual consiste em possibilitar que uma rede interna seja endereçada apenas por um único endereço IP público, sendo que esta utilização é denomeada de *IP masquerading*.

## Experiência 5 – DNS

Esta experiência teve por objetivo a configuração de DNS nos computadores da rede.

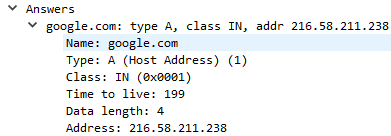
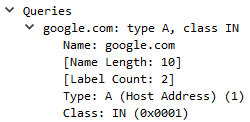
Para este efeito, bastou editar um ficheiro de configuração, para adicionar um *nameserver*, o qual traduz um *hostname* para um endereço IP.

Figura 9 - resolv.conf

**Que pacotes são trocados pelo DNS e que informação é transportada?**

Quando se executa um *ping* a um *hostname*, é enviado um pacote “DNS *standard query*” que contém parâmetros *name*, *type* e *class*. A resposta a este pacote, é um pacote “DNS *standard query response*” que contém, além dos parâmetros mencionados, um campo *Time to live* e *Data length*.

Figura 10 - exemplo de pacotes DNS



## Experiência 6 – Ligações TCP

Esta experiência teve por objetivo testar a aplicação de *download* na rede configurada. Para este efeito, foram testados diversos servidores FTP, nos quais, se experimentou fazer transferência de diferentes tipos de ficheiro, como por exemplo, zip, png e txt.

**Quantas ligações TCP são criadas pela aplicação?**

O protocolo FTP funciona com uma ligação de dados e uma ligação de controlo, por isso, são abertas duas ligações TCP por transferência.

**Em que ligação é transportada a informação de controlo de FTP?**

A informação de controlo é transportada na ligação TCP da porta 21, que é a porta por defeito de serviços FTP.

**Quais são as fases de uma ligação TCP?**

Uma ligação TCP pode consistir de três fases, as quais são:

* Um processo de *handshake* com vários passos, SYN -> SYN-ACK -> ACK.
* Uma fase de transferência de dados.
* Finalmente, uma fase de término de ligação, que consiste noutro *handshake*.

**Como funciona o mecanismo ARQ de TPC? Quais sãos os campos relevantes a TCP?**

O mecanismo ARQ de TCP funciona com uma “janela”, o recetor envia o tamanho desta “janela” no pacote TCP, e o emissor envia no máximo, até este valor de bytes. Após enviar os bytes, espera por um ACK do recetor e nova “janela”. Os campos relevantes no pacote TCP a este mecanismo são *Sequence number*, *Acknowledgment number* e *Window Size*.

**Como funciona o mecanismo de controlo de congestão de TCP? Quais são os campos relevantes?**

O mecanismo de congestão de TCP usa o sistema *slow-start*, o qual começa por enviar dados em número reduzido. Este número é duplicado a cada ACK recebido até ocorrer uma falha na transmissão, ao fim da qual, a ligação entra em fase de *congestion avoidance*, na qual o algoritmo utilizado varia com a implementação,o mais simplista sendo reduzir o número de bytes enviados de volta ao ínicio do *slow-start*. Os campos relevantes no pacote TCP são as *flags* CWR, ECE e URG, juntamente com o campo *Urgent pointer*.

**A taxa de transferência de uma ligação de dados TCP é perturbada pela aparência de uma segunda ligação TCP? Em que sentido?**

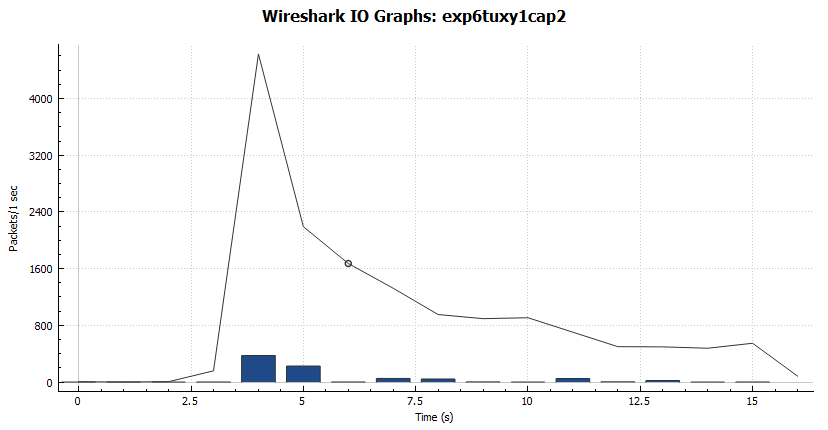
A aparência de uma segunda ligação reduz para cerca de metade a taxa de transferência, isto pode ser visualizado na captura seguinte.

Figura 11 - taxa de transferência

# Conclusão

Com a conclusão do relatório e da análise da aplicação de *download*, como das experiências laboratoriais, considera-se que o grupo conseguiu alcançar os objetivos do segundo trabalho.

Como descrito na primeira seção, o grupo implementou uma aplicação que utiliza o protocolo FTP, de acordo com o RFC959. Isto permitiu consolidar o conhecimento sobre FTP e sobre ligações TCP.

Na segunda seção foi descrita a configuração de uma rede, sendo que, o grupo aprendeu, os conhecimentos base necessários, para conseguir configurar redes complexas. Estes conhecimentos passaram por configurar endereços IP de computadores, implementação de VLAN num *switch* e configuração de rotas e NAT num *router* comercial. Finalizando, com a utilização da aplicação para efetuar uma transferência na rede, confirmando assim, a apreensão dos conhecimentos do trabalho.

Posto isto, é considerado que o trabalho foi bem sucedido e os seus objetivos, maioritariamente, compreendidos.

# Anexos

## Anexo 1 – Comandos de configuração

Para todos os comandos seguintes, y é o número de bancada da sala I321. Os comandos assumem uma configuração incremental começando na primeira experiência.

### Experiência 1

**Computador 1**

ifconfig eth0 down

ifconfig eth1 down

ifconfig eth0 172.16.y0.1/24

**Computador 4**

ifconfig eth0 down

ifconfig eth1 down

ifconfig eth0 172.16.y0.254/24

### Experiência 2

**Computador 2**

ifconfig eth0 down

ifconfig eth1 down

ifconfig eth0 172.16.y1.1/24

***Switch***

conf t

vlan y0

exit

vlan y1

end

conf t

interface fastethernet 0/1

switchport mode access

switchport access vlan y0

end

conf t

interface fastethernet 0/4

switchport mode access

switchport access vlan y0

end

conf t

interface fastethernet 0/2

switchport mode access

switchport access vlan y1

end

### Experiência 3

**Computador 1**

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

**Computador 2**

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

**Computador 4**

ifconfig eth1 172.16.y1.253/24

echo 1 > /proc/sys/net/ipv4/ip\_forward

echo 0 > /proc/sys/net/ipv4/icmp\_echo\_ignore\_broadcasts

***Switch***

conf t

interface fastethernet 0/14

switchport mode access

switchport access vlan y1

end

### Experiência 4 (sem NAT)

***Switch***

conf t

interface fastethernet 0/20

switchport mode access

switchport access vlan y1

end

**Computador 1**

route add -net default gw 172.16.y0.254

route del -net 172.16.y1.0/24

**Computador 2**

route add -net default gw 172.16.y1.254

**Computador 4**

route add -net default gw 172.16.y1.254

***Router***

conf t

interface gigabitethernet 0/0

ip address 172.16.y1.254 255.255.255.0

no shutdown

end

conf t

ip route 172.16.y0.0 255.255.255.0 172.16.y1.253

end

conf t

interface gigabitethernet 0/1

ip address 172.16.1.y9 255.255.255.0

no shutdown

end

conf t

ip route 0.0.0.0 0.0.0.0 172.16.1.254

end

### Experiência 4 (NAT)

***Router***

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

end

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 0.0.0.255

access-list 1 permit 172.16.y1.0.0 0.0.0.255

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

### Experiência 5

**Computador 1/2/4**

Editar /etc/resolv.conf para conter:

"search netlab.fe.up.pt

nameserver 172.16.1.1"

## Anexo 2 – Capturas

### Experiência 1

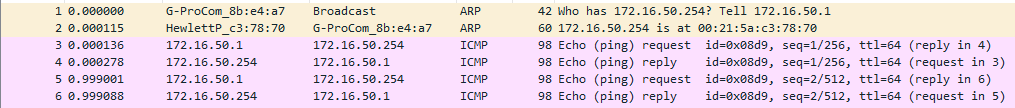


Figura 12 – ping computador 4

### Experiência 2

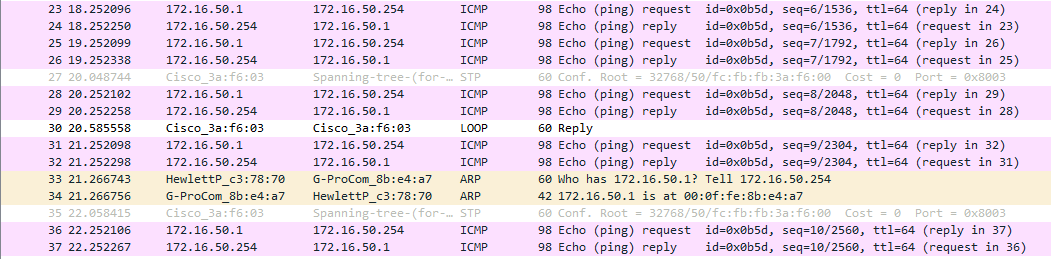


Figura 13 - ping computador 4 em VLAN

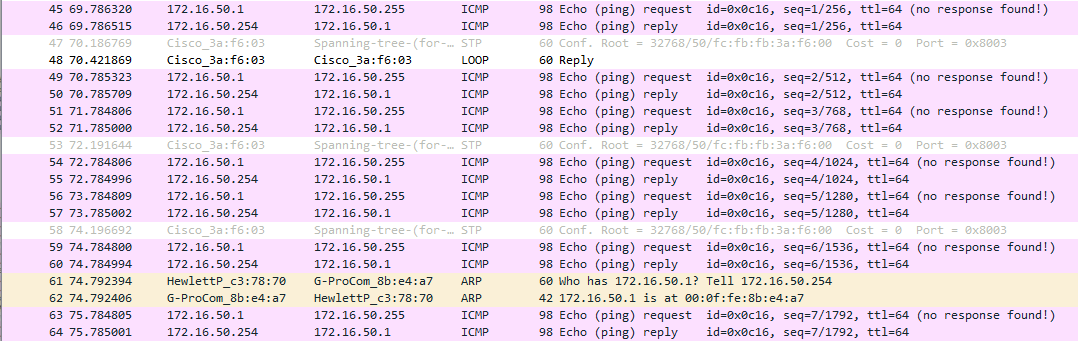


Figura 14 - pacotes no computador 1 recebidos por fazer ping broadcast no computador 1

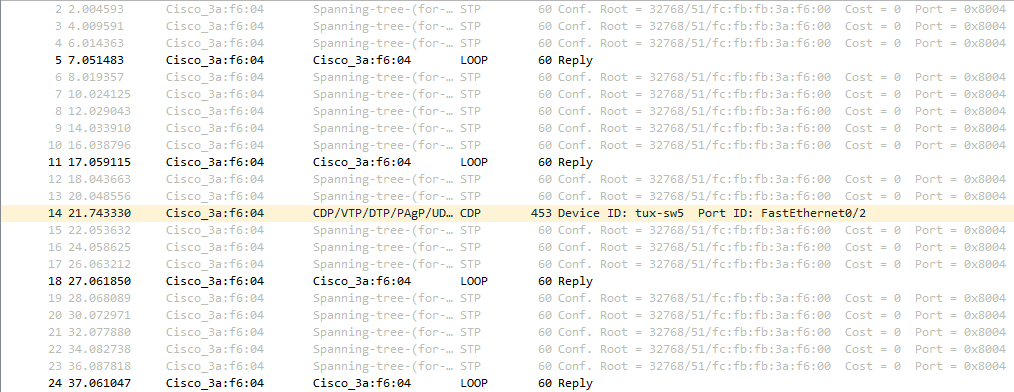


Figura 15 - pacotes no computador 2 recebidos por fazer ping broadcast no computador 1

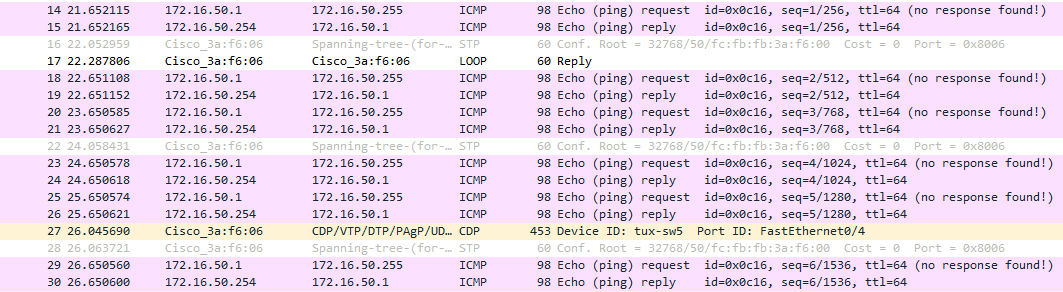


Figura 16 - pacotes no computador 4 recebidos por fazer ping broadcast no computador 1

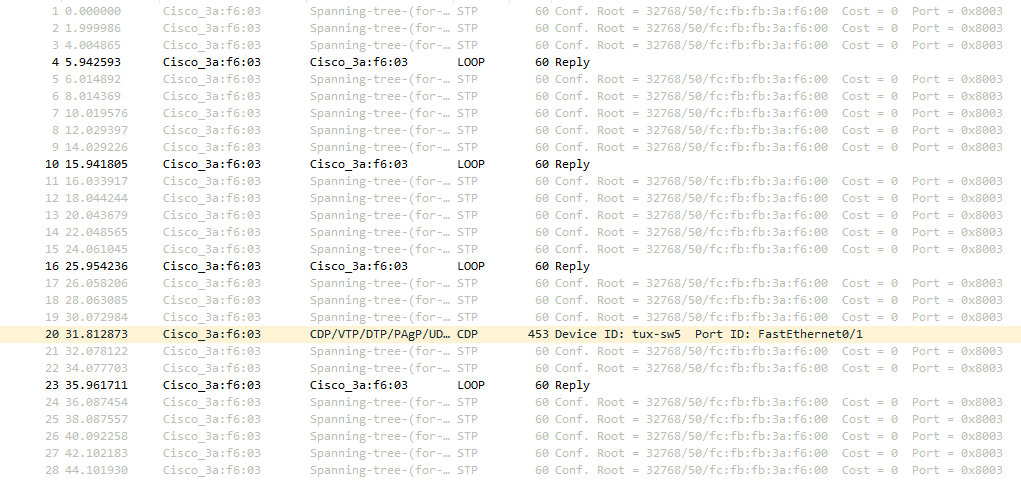


Figura 17 - pacotes no computador 1 recebidos por fazer ping broadcast no computador 2

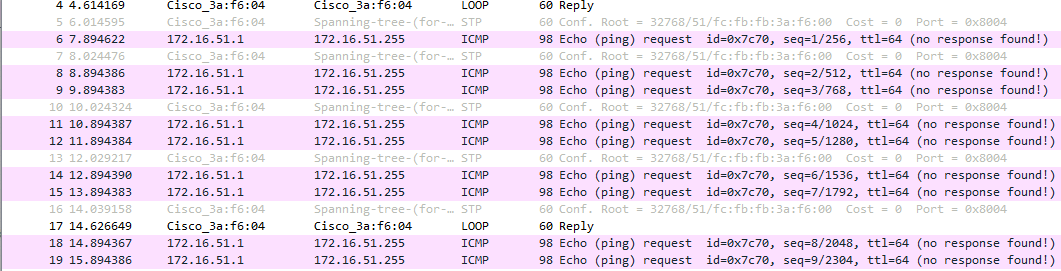


Figura 18 - pacotes no computador 2 recebidos por fazer ping broadcast no computador 2

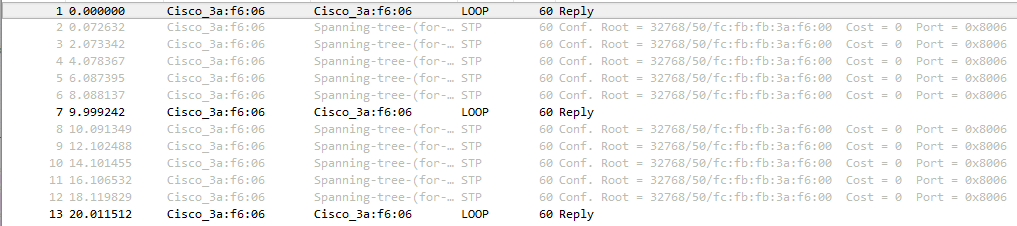


Figura 19 - pacotes no computador 4 recebidos por fazer ping broadcast no computador 2

### Experiência 3

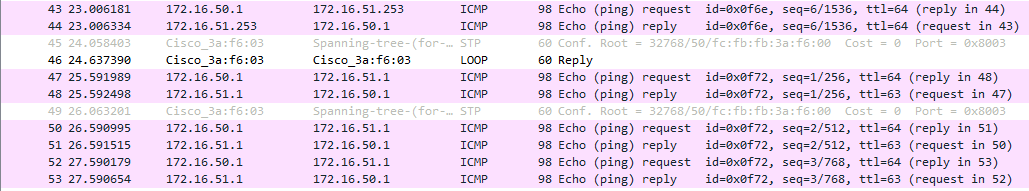


Figura 20 - ping da outra sub-rede a partir do computador 1

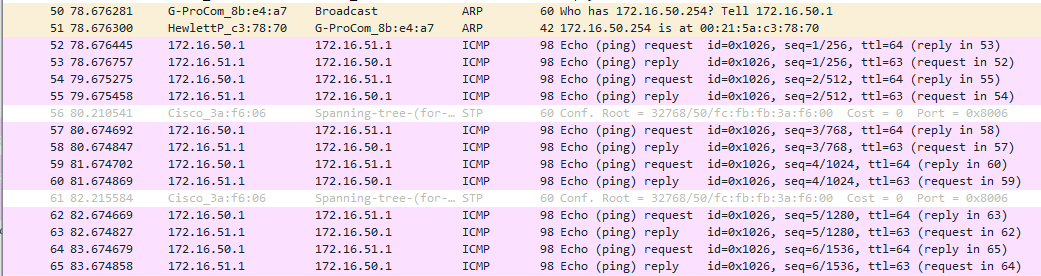


Figura 21 - pacotes no computador 4 ethernet 0 quando o computador 1 faz ping ao computador 2

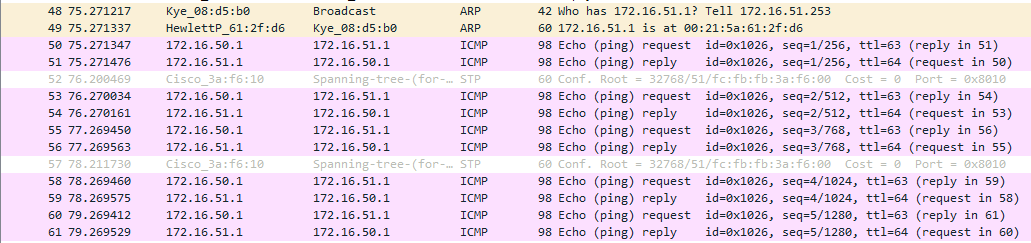


Figura 22 - pacotes no computador 4 ethernet 1 quando o computador 1 faz ping ao computador 2

### Experiência 4

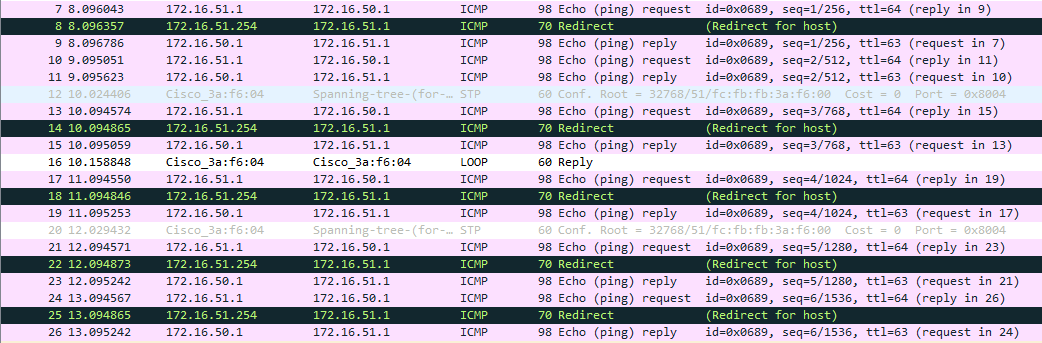


Figura 23 - pacotes no computador 2 quando faz ping ao computador 1 sem rota direta para a sub-rede deste

### Experiência 5

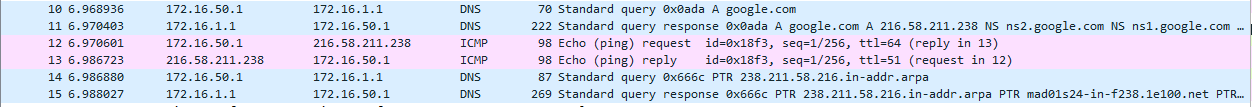


Figura 24 - pacotes DNS no computador 1 quando se faz ping a www.google.com

### Experiência 6

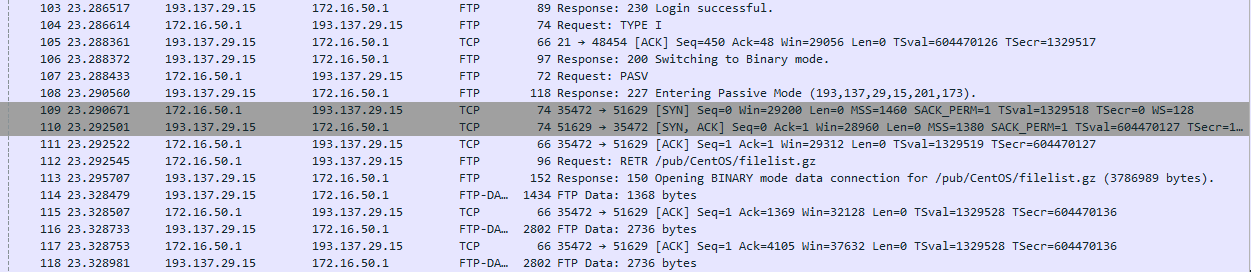


Figura 25 - pacotes TCP no computador 1 quando transfere um ficheiro por FTP

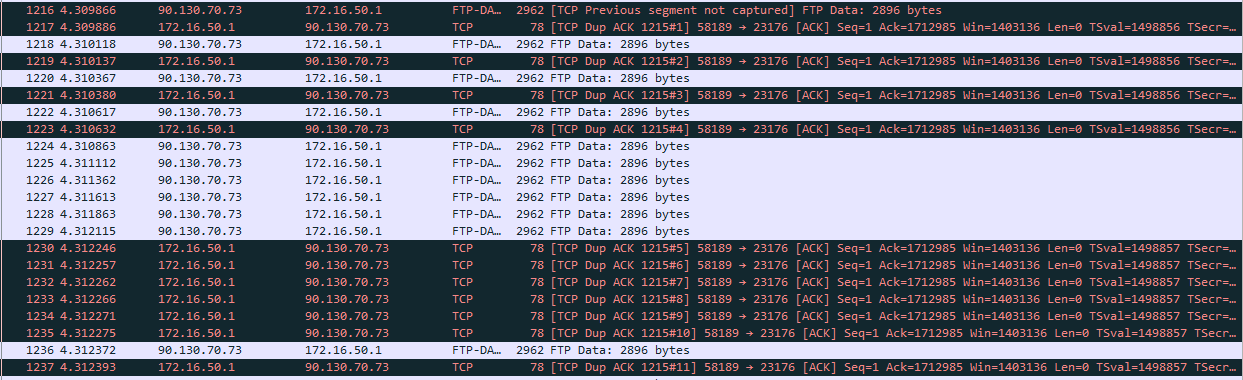


Figura 26 - pacotes TCP no computador 1 quando transfere um ficheiro por FTP ao mesmo tempo que o computador 2

## Anexo 3 – Código da aplicação de download

Download.h

#ifndef \_\_DOWNLOAD\_H

#define \_\_DOWNLOAD\_H

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MACROS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef FALSE

#define FALSE 0

#endif

#ifndef TRUE

#define TRUE 1

#endif

#define FTP\_PORT 21 //Default FTP port

#define FTP\_MSG\_SIZE 512 //FTP Message buffer size

#define R1XX 1 //FTP Reply code hundreds digit

#define R2XX 2 //FTP Reply code hundreds digit

#define R3XX 3 //FTP Reply code hundreds digit

#define R4XX 4 //FTP Reply code hundreds digit

#define R5XX 5 //FTP Reply code hundreds digit

#define FTP\_USER\_START 6 //User argument start index

#define BUFFER\_SIZE 256

#define FILEPATH\_SIZE 1024

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ENUMERATORS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

typedef enum {MULTILINE, NON\_TERMINATED, REPLY\_OVER, REPLY\_TOO\_LONG} FTPLineType\_t; //FTP Reply types

typedef enum {WAIT\_CR, WAIT\_LF, OVER} FTPLineState\_t; //FTP Line states

typedef enum {WAIT\_REPLY, CONTINUE, REPEAT, ABORT} FTPReplyState\_t; //FTP Reply code states

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

STRUCTS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Stores data socket descriptor and file path needed for download

typedef struct {

char path[FILEPATH\_SIZE];

int datafd;

} FTPFile\_t;

//Stores command line arguments passed

typedef struct {

char user[BUFFER\_SIZE];

char password[BUFFER\_SIZE];

char host[BUFFER\_SIZE];

char path[FILEPATH\_SIZE];

uint8\_t anonymous\_f;

} FTPArgument\_t;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FUNCTIONS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* Gets the hostname and IPv4 address of a given host

\*

\* @param address host string

\* @return pointer to hostent struct that stores host info

\*/

struct hostent\* getAddress(char\* address);

/\*\*

\* Gets a TCP socket to specified address at specified port

\*

\* @param address 32 bit Internet address network byte ordered

\* @param port port number

\* @return TCP socket file descriptor

\*/

int getTCPSocket(in\_addr\_t address, uint16\_t port);

/\*\*

\* Gets a TCP socket to specified address at specified port

\*

\* @param address 32 bit Internet address network byte ordered

\* @param port port number

\* @return TCP socket file descriptor

\*/

FTPLineType\_t readFTPLine(int sockfd, char\* reply, size\_t reply\_size);

/\*\*

\* Parses final line of FTP reply message for reply code

\*

\* @param reply reply message text

\* @return FTP reply code

\*/

uint16\_t parseFTPReplyCode(char\* reply);

/\*\*

\* Aborts program by closing the TCP socket and printing error message before exiting

\*

\* @param sockfd TCP socket file descriptor connected to FTP port

\* @param message error message to print

\*/

void FTPAbort(int sockfd, char\* message);

/\*\*

\* Handles FTP reply code and returns action to take

\*

\* @param reply\_code numeric value of the FTP reply code

\* @return enumerator specifying action caller should take

\*/

FTPReplyState\_t handleFTPReplyCode(uint16\_t reply\_code);

/\*\*

\* Sends a command through the TCP connection, if command triggers download calls appropriate function

\*

\* @param sockfd TCP socket file descriptor connected to FTP port

\* @param command FTP command to send

\* @param download\_f whether command will trigger a download from FTP server

\* @result 0 on success

\*/

int8\_t FTPCommand(int sockfd, char\* command, uint8\_t download\_f);

/\*\*

\* Performs FTP login command sequence

\*

\* @param sockfd TCP socket file descriptor connected to FTP port

\* @param user FTP USER command ready to send

\* @param password FTP PASS command ready to send

\* @return non 0 on failure

\*/

int8\_t FTPLogin(int sockfd, char\* user, char\* password);

/\*\*

\* Reads full FTP message and parses the reply code

\*

\* @param sockfd TCP socket file descriptor connected to FTP port

\* @return FTP reply code

\*/

uint16\_t readFTPReply(int sockfd);

/\*\*

\* Gets numeric representation of data port from FTP message containing it

\*

\* @param message FTP message containing data port

\* @return data port on success, -1 otherwise

\*/

int32\_t getDataPort(char\* message);

/\*\*

\* Sends PASV command and parses the data port received

\*

\* @param sockfd TCP socket file descriptor connected to FTP port

\* @return data port to use

\*/

uint16\_t FTPPassive(int sockfd);

/\*\*

\* Parses command line argument and fills FTPArgument\_t struct

\*

\* @param argument command line argument received

\* @return 0 on success

\*/

int8\_t parseArgument(char\* argument);

/\*\*

\* Parses path to file to get the file name, copies it to FTPFile\_t struct

\*

\* @param path file path

\*/

void parseFilePath(char\* path);

/\*\*

\* Uses FTP data socket connection to download file requested

\*/

void FTPDownload();

#endif /\*\_\_DOWNLOAD\_H \*/

### Download.c

#include <stdio.h>

#include <stdlib.h>

#include <netdb.h>

#include <ctype.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

#include <signal.h>

#include <string.h>

#include "Download.h"

static char last\_message[FTP\_MSG\_SIZE]; //Most recent message received

static FTPFile\_t file\_info; //Information about data socket and local file path

static FTPArgument\_t argument\_info; //Information needed for FTP handling

int main(int argc, char\* argv[]) {

/\* Check program parameter count \*/

if(argc != 2) {

printf("download: Wrong number of arguments.\n");

printf("download: usage: ./download ftp://[<user>:<password>@]<host>/<url-path>\n");

exit(1);

}

/\* Parse argument \*/

if(parseArgument(argv[1]) != 0) {

printf("download: Invalid argument.\n");

printf("download: usage: ./download ftp://[<user>:<password>@]<host>/<url-path>\n");

exit(1);

}

/\* Default values for anonymous user \*/

if(argument\_info.anonymous\_f == TRUE) {

strncpy(argument\_info.user, "USER ", strlen("USER "));

strcpy(argument\_info.user + strlen("USER "), "anonymous\r\n");

strncpy(argument\_info.password, "PASS ", strlen("PASS "));

strcpy(argument\_info.password + strlen("PASS "), "ei11056@fe.up.pt\r\n");

}

/\* Get host IP address \*/

struct hostent\* host;

host = getAddress(argument\_info.host);

if(host == NULL) {

printf("download: Couldn't get info on host provided.\n");

exit(1);

}

printf("Host Name : %s\n", host->h\_name);

printf("IP Address : %s\n\n", inet\_ntoa(\*((struct in\_addr\*)host->h\_addr)));

/\* Get TCP socket to host \*/

int sockfd = getTCPSocket(inet\_addr(inet\_ntoa(\*((struct in\_addr\*)host->h\_addr))), FTP\_PORT);

if(sockfd < 0) FTPAbort(sockfd, "download: Failure connecting to FTP server control port.\n");

/\* Login to FTP server \*/

FTPLogin(sockfd, argument\_info.user, argument\_info.password);

/\* Use BINARY mode \*/

if(FTPCommand(sockfd, "TYPE I\r\n", FALSE) != 0) FTPAbort(sockfd, "download: TYPE command received 5XX code.\n");

/\* Enter passive mode and get data port \*/

uint16\_t data\_port = FTPPassive(sockfd);

file\_info.datafd = getTCPSocket(inet\_addr(inet\_ntoa(\*((struct in\_addr\*)host->h\_addr))), data\_port);

if(file\_info.datafd < 0) FTPAbort(sockfd, "download: Failure connecting to FTP server data port.\n");

/\* Send command to download file and download it \*/

if(FTPCommand(sockfd, argument\_info.path, TRUE) != 0) FTPAbort(sockfd, "download: RETR command received 5XX code.\n");

printf("\n");

close(file\_info.datafd);

close(sockfd);

return 0;

}

/\*\*

\* Parses path to file to get the file name, copies it to FTPFile\_t struct

\*

\* @param path file path

\*/

void parseFilePath(char\* path) {

size\_t path\_i = 0;

size\_t last\_slash\_i = 0;

while(path[path\_i] != '\0') {

if(path[path\_i] == '/') {

last\_slash\_i = path\_i;

}

path\_i++;

}

file\_info.path[0] = '.';

strcpy(file\_info.path + 1, path + last\_slash\_i);

}

/\*\*

\* Parses command line argument and fills FTPArgument\_t struct

\*

\* @param argument command line argument received

\* @return 0 on success

\*/

int8\_t parseArgument(char\* argument) {

/\* Check that start of argument is as expected \*/

if(strncmp(argument, "ftp://", FTP\_USER\_START) != 0) {

printf("download: Could not find \"ftp://\" in argument.\n");

return -1;

}

size\_t arg\_i = 0;

size\_t param\_i = 0;

uint8\_t at\_f = FALSE;

uint8\_t separator\_f = FALSE;

/\* Search for @ symbol in argument \*/

while(argument[arg\_i] != '\0') {

if(argument[arg\_i] == '@') {

at\_f = TRUE;

break;

}

arg\_i++;

}

arg\_i = FTP\_USER\_START;

argument\_info.anonymous\_f = TRUE;

/\* Parse user and password if @ found \*/

if(at\_f) {

argument\_info.anonymous\_f = FALSE;

/\* Parse user \*/

strncpy(argument\_info.user, "USER ", strlen("USER "));

param\_i += strlen("USER ");

while(argument[arg\_i] != '\0') {

if(argument[arg\_i] != ':') {

argument\_info.user[param\_i] = argument[arg\_i];

arg\_i++;

param\_i++;

} else {

arg\_i++;

separator\_f = TRUE;

break;

}

}

if(!separator\_f) {

printf("download: Could not find \":\" separator in argument.\n");

return -1;

} else if(param\_i <= strlen("USER ")) {

printf("download: Could not find user in argument.\n");

return -1;

}

strcpy(argument\_info.user + param\_i, "\r\n");

separator\_f = FALSE;

param\_i = 0;

/\* Parse password \*/

strncpy(argument\_info.password, "PASS ", strlen("PASS "));

param\_i += strlen("PASS ");

while(argument[arg\_i] != '\0') {

if(argument[arg\_i] != '@') {

argument\_info.password[param\_i] = argument[arg\_i];

arg\_i++;

param\_i++;

} else {

arg\_i++;

separator\_f = TRUE;

break;

}

}

if(!separator\_f) {

printf("download: Could not find \"@\" separator in argument.\n");

return -1;

} else if(param\_i <= strlen("PASS ")) {

printf("download: Could not find password in argument.\n");

return -1;

}

strcpy(argument\_info.password + param\_i, "\r\n");

separator\_f = FALSE;

param\_i = 0;

}

/\* Parse host name \*/

while(argument[arg\_i] != '\0') {

if(argument[arg\_i] != '/') {

argument\_info.host[param\_i] = argument[arg\_i];

arg\_i++;

param\_i++;

} else {

separator\_f = TRUE;

break;

}

}

if(!separator\_f) {

printf("download: Could not find \"/\" separator in argument for host.\n");

return -1;

} else if(param\_i <= 0) {

printf("download: Could not find host in argument.\n");

return -1;

}

argument\_info.host[param\_i] = '\0';

separator\_f = FALSE;

param\_i = 0;

/\* Parse file path \*/

strncpy(argument\_info.path, "RETR ", strlen("RETR "));

param\_i += strlen("RETR ");

while(argument[arg\_i] != '\0') {

argument\_info.path[param\_i] = argument[arg\_i];

arg\_i++;

param\_i++;

}

if(param\_i <= 0) {

printf("download: Could not find file path in argument.\n");

return -1;

}

argument\_info.path[param\_i] = '\0';

/\* Extract file name from path \*/

parseFilePath(argument\_info.path);

strcpy(argument\_info.path + param\_i, "\r\n");

return 0;

}

/\*\*

\* Uses FTP data socket connection to download file requested

\*/

void FTPDownload() {

/\* Open file with file name parsed previously \*/

FILE\* fd = fopen(file\_info.path, "wb");

char file\_data[FTP\_MSG\_SIZE];

int n;

/\* Read until no more data is returned \*/

while((n = read(file\_info.datafd, file\_data, FTP\_MSG\_SIZE)) > 0) {

fwrite(file\_data, 1, n, fd);

}

fclose(fd);

}

/\*\*

\* Gets numeric representation of data port from FTP message containing it

\*

\* @param message FTP message containing data port

\* @return data port on success, -1 otherwise

\*/

int32\_t getDataPort(char\* message) {

size\_t msg\_i = 0;

size\_t number\_i = 0;

size\_t comma\_count = 0;

char msb[6];

char lsb[6];

uint8\_t success\_f = FALSE;

/\* Process message up to null terminator \*/

while(message[msg\_i] != '\0') {

/\* Count commas up to data port values \*/

if(message[msg\_i] == ',') comma\_count++;

msg\_i++;

/\* Next 2 number values will be the data port \*/

if(comma\_count >= 4) {

/\* Read MSB of data port \*/

while(isdigit(message[msg\_i])) {

msb[number\_i] = message[msg\_i];

number\_i++;

msg\_i++;

}

msb[number\_i] = '\0';

number\_i = 0;

msg\_i++;

/\* Read LSB of data port \*/

while(isdigit(message[msg\_i])) {

lsb[number\_i] = message[msg\_i];

number\_i++;

msg\_i++;

}

lsb[number\_i] = '\0';

success\_f = TRUE;

break;

}

}

if(!success\_f) return -1;

unsigned long msb\_value = strtoul(msb, NULL, 10);

unsigned long lsb\_value = strtoul(lsb, NULL, 10);

return (msb\_value \* 256 + lsb\_value);

}

/\*\*

\* Sends PASV command and parses the data port received

\*

\* @param sockfd TCP socket file descriptor connected to FTP port

\* @return data port to use

\*/

uint16\_t FTPPassive(int sockfd) {

if(FTPCommand(sockfd, "PASV\r\n", FALSE) != 0) FTPAbort(sockfd, "download: PASV command received 5XX code.\n");

int32\_t data\_port = getDataPort(last\_message);

if(data\_port < 0) FTPAbort(sockfd, "download: data port could not be parsed.\n");

return data\_port;

}

/\*\*

\* Aborts program by closing the TCP socket and printing error message before exiting

\*

\* @param sockfd TCP socket file descriptor connected to FTP port

\* @param message error message to print

\*/

void FTPAbort(int sockfd, char\* message) {

printf("%s", message);

close(file\_info.datafd);

close(sockfd);

exit(1);

}

/\*\*

\* Handles FTP reply code and returns action to take

\*

\* @param reply\_code numeric value of the FTP reply code

\* @return enumerator specifying action caller should take

\*/

FTPReplyState\_t handleFTPReplyCode(uint16\_t reply\_code) {

uint16\_t hundreds = (reply\_code / 100) % 100;

switch(hundreds) {

case R1XX:

return WAIT\_REPLY;

break;

case R2XX:

return CONTINUE;

break;

case R3XX:

return CONTINUE;

break;

case R4XX:

return REPEAT;

break;

case R5XX:

return ABORT;

break;

default:

return ABORT;

break;

}

}

/\*\*

\* Sends a command through the TCP connection, if command triggers download calls appropriate function

\*

\* @param sockfd TCP socket file descriptor connected to FTP port

\* @param command FTP command to send

\* @param download\_f whether command will trigger a download from FTP server

\* @result 0 on success

\*/

int8\_t FTPCommand(int sockfd, char\* command, uint8\_t download\_f) {

FTPReplyState\_t reply\_state;

/\* Repeat until reply code signals completion \*/

do {

/\* Write command and read response \*/

write(sockfd, command, strlen(command));

reply\_state = handleFTPReplyCode(readFTPReply(sockfd));

/\* Handle reply code \*/

if(reply\_state == ABORT) return -1;

else if(reply\_state == WAIT\_REPLY) {

if(download\_f) FTPDownload();

reply\_state = handleFTPReplyCode(readFTPReply(sockfd));

if(reply\_state == ABORT) return -1;

}

} while(reply\_state != CONTINUE);

return 0;

}

/\*\*

\* Performs FTP login command sequence

\*

\* @param sockfd TCP socket file descriptor connected to FTP port

\* @param user FTP USER command ready to send

\* @param password FTP PASS command ready to send

\* @return non 0 on failure

\*/

int8\_t FTPLogin(int sockfd, char\* user, char\* password) {

FTPReplyState\_t reply\_state;

/\* Read message of the day \*/

reply\_state = handleFTPReplyCode(readFTPReply(sockfd));

if(reply\_state != CONTINUE) FTPAbort(sockfd, "download: Server is not ready.\n");

/\* Send USER command \*/

if(FTPCommand(sockfd, user, FALSE) != 0) FTPAbort(sockfd, "download: USER command received 5XX code.\n");

/\* Send PASS command \*/

if(FTPCommand(sockfd, password, FALSE) != 0) FTPAbort(sockfd, "download: PASS command received 5XX code.\n");

return 0;

}

/\*\*

\* Reads full FTP message and parses the reply code

\*

\* @param sockfd TCP socket file descriptor connected to FTP port

\* @return FTP reply code

\*/

uint16\_t readFTPReply(int sockfd) {

FTPLineType\_t status;

char message[FTP\_MSG\_SIZE];

do {

status = readFTPLine(sockfd, message, sizeof(message));

printf("%s", message);

} while(status != REPLY\_OVER);

strcpy(last\_message, message);

return parseFTPReplyCode(message);

}

/\*\*

\* Parses final line of FTP reply message for reply code

\*

\* @param reply reply message text

\* @return FTP reply code

\*/

uint16\_t parseFTPReplyCode(char\* reply) {

/\* Copy first 3 values of reply message which should be the reply code \*/

char code[4];

strncpy(code, reply, 3);

code[3] = '\0';

unsigned long number = strtoul(code, NULL, 10);

return number;

}

/\*\*

\* Reads a single FTP reply message line, waits for Telnet end-of-line

\*

\* @param sockfd TCP socket file descriptor connected to FTP port

\* @param reply reply message text filled in by the function

\* @param reply\_size buffer size for reply message text

\* @return state of the reply message (finished, multiline, buffer overflow, non terminated reply)

\*/

FTPLineType\_t readFTPLine(int sockfd, char\* reply, size\_t reply\_size) {

FTPLineState\_t state = WAIT\_CR;

char receive[1];

size\_t counter = 0;

/\* Read 1 byte at a time until Telnet end-of-line (CRLF) \*/

while(read(sockfd, receive, 1) > 0) {

/\* State machine \*/

if(receive[0] == '\r') state = WAIT\_LF;

else if(receive[0] == '\n' && state == WAIT\_LF) state = OVER;

reply[counter] = receive[0];

counter++;

/\* Check if reply is larger than buffer \*/

if(counter >= reply\_size - 1) {

reply[counter] = '\0';

return REPLY\_TOO\_LONG;

}

/\* Check if reply message is over by looking for "xyz " string at the start \*/

if(state == OVER) {

reply[counter] = '\0';

if(isdigit(reply[0]) && isdigit(reply[1]) && isdigit(reply[2]) && reply[3] == ' ') return REPLY\_OVER;

else return MULTILINE;

}

}

return NON\_TERMINATED;

}

/\*\*

\* Gets a TCP socket to specified address at specified port

\*

\* @param address 32 bit Internet address network byte ordered

\* @param port port number

\* @return TCP socket file descriptor

\*/

int getTCPSocket(in\_addr\_t address, uint16\_t port) {

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 = address;

server\_addr.sin\_port = htons(port);

/\* Open 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;

}

return sockfd;

}

/\*\*

\* Gets the hostname and IPv4 address of a given host

\*

\* @param address host string

\* @return pointer to hostent struct that stores host info

\*/

struct hostent\* getAddress(char\* address) {

struct hostent\* host;

if((host = gethostbyname(address)) == NULL) {

herror("download");

return NULL;

}

return host;

}

### Makefile

CC = gcc

CFLAGS = -Wall

SRCS = Download.c

all: $(SRCS)

$(CC) $(CFLAGS) $(SRCS) -lm -o download