Faculdade de Engenharia da Universidade do Porto
Redes de Computadores - 1º Semestre 2022/2023

Computer Networks

Final Report

Class 3LEIC02

Henrique Silva up202007242@fe.up.pt
Tiago Branquinho up202005567@fe.up.pt

Index

Sumário

Este relatório descreve o segundo projeto realizado na unidade curricular de Redes de Computadores. Todas as experiências mencionadas foram realizadas nos computadores do laboratório I320 e I321. Nas secções seguintes é documentado o procedimento e explicados os resultados.

Introdução

O segundo projeto da unidade curricular de Redes de Computadores foi dividido em duas partes.

Na primeira parte, foi desenvolvida uma aplicação de *download*, capaz de descarregar um ficheiro por FTP, dado o seu URL. Foi também necessária a utilização de *sockets* e do TCP.

Na segunda parte, foi configurada uma rede de computadores, seguindo os guiões fornecidos e registando as experiências realizadas para poder responder às questões colocadas.

Aplicação de *Download*

Arquitetura

A primeira tarefa realizada pela aplicação é a leitura e processamento do URL que lhe é passado no formato ftp://[<user>:<password>@]<host>/<url-path>. Esta é realizada pela função parseURL que guardará cada uma das secções do URL, sendo elas: username, password, hostname, file path e file name. De seguida é chamada a função já fornecida getIPAddress, a qual sendo dado o parâmetro host name retornará o endereço de IP correspondente. A outra função fornecida, clientTCP, utilizará de seguida este endereço para conectar com o socket que irá criar, na porta porta padrão, ou seja a porta 21.

Passando à comunicação com o servidor, a função utilizada para a intermediar é sendCommandReceiveResponse que delega a tarefa de enviar comandos à função sendCommand e a de receber respostas à função receiveReponse. O processamento de cada resposta é baseado no primeiro digito da mesma, que pode variar entre 1 e 5. Caso a resposta começe por 1, a resposta indicará uma ação bem sucedida se estiver a ser recebido um ficheiro, se não será necessário esperar por nova resposta. Caso seja 2 a ação foi bem sucedida, caso seja 3 o servidor está a requerer mais informação, caso seja 4 o servidor requer que o comando seja reenviado, e caso seja 5 é porque ocorreu um erro e o socket será fechado e a aplicação terminada.

Depois de estabelecida a ligação com o socket é necessário efetuar o login, responsabilidade da função login. De seguida, é necessário percorrer os diretórios do servidor para chegar ao ficheiro pretendido, utilizando a função CWD. Agora, é necessário enviar o comando 'pasv' ao servidor para indicar a entrada em modo passivo, e receber a porta onde será aberto o data socket para receber o ficheiro, através da função getServerPort. Para indicar a prontidão para receber o ficheiro é necessário enviar o comando retr, através da função retr, podendo de seguida efetuar o *download* utilizando a função downloadFile. Após finalizado o *download* é enviado o comando QUIT ao servidor, através da função disconnect, para fechar a ligação.

As funções principais estão incluídas na secção dos anexos.

Resultados

A aplicação efetuou todas as tarefas propostas com sucesso, funcionando tanto em servidores em que é necessário login como naqueles em que não é.

Os resultados estão presentes na secção dos anexos.

Configuração de uma Rede de Computadores

NOTA: As capturas referenciadas em cada análise foram omitidas do relatório por questões de brevidade e podem ser encontradas no git, na sua respetiva pasta.

Experiência 1 - Configurar uma Rede de Endereços de IP

Nesta experiência foi necessário configurar as máquinas tux3 e tux4 e conectá-las ao switch, assim como verificar a comunicação entre as duas máquinas

Comandos utilizados:

- ➔ ifconfig <ethx> <IP address>/<netmask> (para configurar o endereço de IP de uma máquina)
- ➔ route -n (para verificar a lista de rotas já existentes em cada máquina)
- ➔ arp –a (para verificar tabelas ARP)
- ➔ arp -d <IP address> (para eliminar as entradas correspondentes a este endereço de IP nas tabelas ARP)
- ➔ ping <IP address> (para verificar a existência de comunicação com este endereço de IP)

O que são os pacotes ARP, e para que são usados?

Uma máquina, ao tentar enviar um pacote a outra, na mesma rede local , irá enviar um pacote ARP, por broadcast, para todas as máquinas dessa mesma rede perguntando qual delas tem um endereço MAC que corresponde ao endereço IP do destinatário. Por sua vez, o destinatário irá enviar outro pacote ARP que indica à máquina fonte qual o seu endereço MAC. Deste modo, a transferência de pacotes pode ser efetuada.

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 (172.16.30.254). Deste modo, irá enviar um pacote ARP em broadcast (para toda a rede local), sendo que este pacote contém o endereço IP (172.16.30.1) e o endereço MAC do tux3 (00:0f:fe:8b:e4:4d). O endereço MAC do destinatário, como é desconhecido, tem o valor de 00:00:00:00:00:00. De seguida, o tux4 irá enviar um pacote ARP para o tux3, com seu o endereço MAC(00:21:5a:5a:7d:74), e o seu endereço IP (172.16.30.254). Portanto, pode-se concluir que cada pacote ARP contém campos para os endereços MAC e IP da máquina fonte, e para os endereços MAC e IP da máquina destinatária.

Que pacotes gera o comando ping?

Tal como foi analisado nas capturas, o comando ping gera primeiro pacotes ARP para saber qual o endereço MAC do destinatário, e de seguida, gera pacotes ICMP (Internet Control Message Protocol).

Quais são os endereços MAC e IP dos pacotes ping?

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

- ➢ Pacote request (de tux3 para tux4):
  - – IP address da source: 172.16.30.1 (tux3)
  - – MAC address da source: 00:0f:fe:8b:e4:4d (tux3)
  - – IP address do destinatário: 172.16.30.254 (tux4)
  - – MAC address do destinatário: 00:21:5a:5a:7d:74 (tux4)
- ➢ Pacote reply (de tux4 para tux3):

- – IP address da source: 172.16.30.254 (tux4)
- – MAC address da source: 00:21:5a:5a:7d:74 (tux4)
- – IP address do destinatário: 172.16.30.1 (tux3)
- – MAC address do destinatário: 00:0f:fe:8b:e4:4d (tux3)

Como determinar se uma trama Ethernet é ARP, IP, ICMP?

Conseguimos determinar o tipo da trama recetora analisando o Ethernet header da trama. Caso este tenha o valor 0x0800 a trama é do tipo IP. No entanto, se tiver o valor 0x0806 é do tipo ARP. No caso de a trama ser do tipo IP, podemos analisar o seu IP header. Se este header tiver valor 1, então o tipo de protocolo é ICMP.

Como determinar o tamanho de uma trama recebida?

O comprimento de uma trama pode ser verificado recorrendo ao Wireshark.

No.	Time	Source	Destination	Protocol	Length	Info
33	39.353447759	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x1cfb, seq=1/256, ttl=64 (request in 32)

O que é a interface loopback, e porque é que é importante?

A interface loopback é uma interface virtual de rede que é utilizada para testes de rede e para fins de configuração. Ela é importante porque permite testar a configuração de rede e as aplicações de rede sem precisar de uma conexão de rede real.

A captura na qual esta análise foi baseada foi a picap-exp1-step9-tux3.

Experiência 2 - Implementação de Duas Bridges num Switch

Nesta experiência foi necessário associar as máquinas tux3 e tux4 a uma bridge0, e a máquina tux2 a uma bridge1, depois de configurar cada um dos elementos, para depois verificar como é que as bridges influenciam a comunicação entres as diferentes máquinas.

Principais comandos (omitindo a configuração das bridges que será abordada posteriormente):

➔ echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts (efetuado em todas as máquinas)

- Como configurar a bridge0 ?

  Começar por conectar ao switch:

  -Abrir GTKterm@115200
  -Carregar ENTER
  -user: admin
  -pass: (blank)

  Criar a bridge0 no switch:

  -/interface bridge add name=bridgeY0

  Remover as portas onde tux3 e tux4 estão conectados à bridge padrão (bridge) e adicionar-lhes a porta correspondente à bridge0.

  -/interface bridge port remove [find interface=etherXX]
  -/interface bridge port add interface=etherXX bridge=bridgeY0

  O processo seria semelhante para a bridge1 e a máquina tux2.

Quantos domínios de transmissão existem? Como é que isso pode ser concluído a partir das capturas?

Existem 2 domínios de broadcast já que ao fazer broadcast apenas são abrangidas as portas pertencentes a essa bridge. Tal foi confirmado ao analisar as capturas do passo 9 da experiência e ao verificar que, quando realizado o comando ping -b 172.16.Y0.255 no tux3, apenas houve comunicação entres as máquinas tux3 e tux4, e nunca na máquina tux2. Ao repetir a experiência na outra bridge o resultado foi o inverso.

As capturas nas quais esta análise foi baseada foram picap-exp2-step9-tux2, picap-exp2-step9-tux3 e picap-exp2-step9-tux4.

Experiência 3 - Configurar um Router em Linux

Nesta experiência, foi necessário configurar um segundo eth na máquina tux4 (eth1), de modo a que esta pudesse funcionar como um router, ligado a duas bridges, que permite a comunicação entre entre máquinas em bridges distintas, no caso entre tux3 e tux2.

Principais comandos:

- ➔ ifconfig <ethx> <IP address>/<netmask>
- ➔ echo 1 > /proc/sys/net/ipv4/ip-forward (para ativar o reencaminhamento de endereços de IP)
- ➔ echo 1 > /proc/sys/net/ipv4/icmp echo ignore broadcasts (para desativar ICMP echo-ignore-broadcast)
- ➔ route add default gw <IP address> (para adicionar uma rota numa máquina que obriga à passagem pela máquina associada a este endereço de IP)

Que rotas existem em cada máquina? Qual é o seu significado?

Algumas rotas são geradas automaticamente, ao ligar as máquinas às bridges a que pertencem: o tux3 tem uma rota para a bridge0, o tux2 tem uma rota para a bridge1, e o tux4 tem rotas para as duas. Estas rotas têm o gateway 0.0.0.0. Para além disso, no tux3 foi adicionada a rota - route add -net 172.16.41.0/24 gw 172.16.40.254, isto é, quando o tux3 (172.16.40.1) quer comunicar com uma máquina na bridge1 (172.16.41.0) ele vai utilizar como gateway o router que é o tux4 (172.16.40.254). Este tux vai ter uma tabela de reencaminhamento que torna isto possível. No tux2 fez-se algo similar ao que se fez no tux3, criou-se uma rota - route add -net 172.16.40.1/24 gw 172.16.41.253, isto é, quando este comunica com uma máquina na bridge0, o router (172.16.41.253) será um intermediário da comunicação.

Que informação contém uma entrada na tabela de reencaminhamento?

Cada entrada na tabela de reencaminhamento contém informações sobre o destino do pacote, o endereço de rede ou o endereço IP de destino, e o gateway, ou seja, o caminho ou o próximo salto para chegar ao destino. Além disso, a entrada da tabela de reencaminhamento pode conter informações adicionais, como a máscara de sub-rede, a interface de saída (ethX) e o número de saltos máximos permitidos antes que o pacote seja descartado.

Que mensagens ARP, e endereços MAC associados, são observados, e porquê?

São verificadas mensagens ARP quando um tux comunica com outro tux, e o tux fonte não conhece o endereço MAC do tux destino. No nosso caso, acontece quando o tux3 tenta comunicar com o tux2. O tux3 envia uma mensagem ARP, na qual pede o endereço MAC do tux2, através do seu IP. Quando uma mensagem ARP é enviada, o tux que a envia (neste caso, tux3) associa o seu endereço MAC à mensagem, para que o recetor da comunicação (tux2) saiba a que tux responder (”Who has [IP de tux2]? Tell [próprio IP]”). Esta mensagem será enviada no modo broadcast (endereço MAC do recetor tem o valor 00:00:00:00:00:00), porque o endereço MAC de tux2 ainda é desconhecido. Quando recebe este broadcast, o tux2 responde com uma mensagem ARP, na qual indica o seu endereço MAC(”[IP de tux2] is at [endereço MAC de tux2]”). Esta mensagem é enviada apenas para o tux3.

Que pacotes ICMP são observados, e porquê?

São observado pacotes ICMP do tipo request e reply, visto que, cada máquina tem associadas as routes necessárias para a comunicação. Caso não, os pacotes seriam do tipo Host Unreachable.

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

Os endereços IP e MAC de origem e destino associados aos pacotes ICMP são os das máquinas ou interfaces que os recebem ou enviam. Por exemplo, quando um pacote ICMP é enviado do tux3 para a interface do tux4 conectada à mesma sub-rede (neste caso a eth0), os endereços de origem serão os do tux3 (IP: 172.16.40.1 e MAC: 00:21:5a:61:2f:13) e os endereços de destino serão os associados à interface eth0 do tux4 (IP: 172.16.40.254 e MAC: 00:21:5a:c3:78:76). Se as duas máquinas não estiverem conectadas à mesma bridge (como é o caso do tux3 e tux2), não é possível efetuar a comunicação entre ambas com apenas um pacote ICMP sem este ser modificado. Neste caso, o que acontece é que o tux3 irá enviar um pacote ICMP para a interface eth0 do tux4, uma vez que o tux4 está ligado a ambas as bridges e consegue interagir diretamente com o tux2. Os endereços IP do pacote serão os IPs do tux3 e tux2 (origem e destino, respetivamente), e os endereços MAC serão os do tux3 e o da interface eth0 do tux4. Ao receber este pacote, o tux4 irá reencaminha-lo para o tux2, mantendo os endereços IP, mas os endereços MAC serão diferentes: o de origem será o da interface eth1 do tux4 , e o de destino será o endereço MAC do tux2.

As capturas nas quais esta análise foi baseada foram picap-exp3-step7-tux3, picap-exp3-step11-tux4-eth0 e picap-exp3-step11-tux4-eth1.

Experiência 4 - Configurar um Router comercial e implementar NAT

Para a realização desta experiência foi necessário ligar um router comercial à rede do laboratório e a uma das bridges, neste caso, *bridgeY1*. Este está predefinido para utilizar a NAT de modo a fornecer internet para todas as máquinas ligadas ao switch. O objetivo desta experiência foi também perceber o caminho realizado por pacotes ICMP.

Principais comandos:

- ➔ /ip address add address=172.16.2.Y9/24 interface=ether1
- ➔ /ip address add address=172.16.Y1.254/24 interface=ether2
- ➔ route add default gw 172.16.Y1.254 (no tux2 e tux4)
- ➔ ip route add 172.16.Y0.0/24 via 172.16.Y1.253 (tux2 e rc)
- ➔ /ip route add dst-address=0.0.0.0/0 gateway=172.16.X.254 X=(1,2)
- ➔ /ip route add dst-address=172.16.Y0.0/24 gateway=172.16.Y1.253
- ➔ echo 0 > /proc/sys/net/piv4/conf/eth0/accept_redirects
- ➔ echo 0 > /proc/sys/net/ipv4/conf/all/accept_redirects
- ➔ /ip firewall nat disable 0

Como configurar uma rota estática num router comercial?

Em primeiro lugar temos de aceder ao gtkterm. Para isso ligamos o S0 de um tux à porta Router MKTIK, podendo passar pelas portas RS232. Após iniciarmos sessão com as credenciais na bancada, temos de executar o comando ip route add <address> <gateway>, como por exemplo ip route add 172.16.Y0.0/24 via 172.16.Y1.253.

Quais são os caminhos seguidos pelos pacotes nas experiências efetuadas, e porquê?

Os caminhos seguidos pelos pacotes variam de acordo com a configuração do tux2, que, por sua vez, dita a existência de uma rota conhecida ou não. Ao executar os comandos para desativar os redirects (listados em cima) no tux2, verificou-se que o tux2 deixou de guardar informações relativas a entradas resultantes de redirects de outras

máquinas. Neste cenário, quando fazemos ping do tux3 no tux2, os pacotes são dirigidos ao router, que por sua vez envia-os à interface do tux4 que está na rede do tux2 e, por fim, estes alcançam o tux2. Isto ocorre porque as rotas do tux 2 para o router (route default do tux2) e deste para o tux2 (via tux4) já estavam previamente definidas. Por outro lado, caso os redirects estejam ativos, no primeiro ping, o tux2 entrará em contacto com o router, e, posteriormente, receberá um redirect do mesmo com a informação de que pode aceder ao tux3 através do tux4. A partir desse momento, não haverão mais redirects porque o tux2 enviará pacotes diretamente para o tux3 (através do tux4).

Como configurar a NAT num router comercial?

Para tal, inserimos os seguintes comandos no gtkterm: **/ip address add address=172.16.X.Y9/24 X=(1,2)**, dependendo do laboratório - para configurar uma interface do router, associando um IP a uma porta, neste caso 1. **/ip route add dst-address=0.0.0.0/0 gateway=172.16.X.254** - para configurar a rota default para 172.16.X.254 e **/ip route add dst-address=172.16.Y0.0/24 gateway=172.16.Y1.253** - configuração da rota para a rede Y0 (172.16.Y0.0), através do endereço 172.16.Y1.253 (interface eth1 do tux4)

O que faz a NAT?

A NAT (Network Address Translation) é um protocolo que faz o mapeamento de um endereço IP de uma rede privada ou pública. Esse mapeamento consiste em mascarar o remetente e destinatário de pacotes, alterando essas informações no cabeçalho dos mesmo, de modo a assegurar uma maior privacidade. Para além disso, a NAT também permite que redes de IP privadas se conectem com redes públicas, incluindo a Internet.
As capturas nas quais esta análise foi baseada foram picap-exp4-step4.6-tux2, picap-exp4-step5-tux3 e picap-exp4-step7-tux3.

Experiência 5 - DNS

O objetivo desta experiência foi visualizar o impacto que a conexão das máquinas a um servidor DNS, que traduz hostnames para endereços IP, tem nas comunicações necessárias para as conectar à Internet.

Principais comandos: sudo nano /etc/resolv.conf

Como configurar o serviço DNS num host?

Primeiro abrimos o ficheiro resolv.conf, através do seguinte comando sudo nano /etc/resolv.conf. Após isso, apenas temos de escrever nameserver 172.16.X.1 X=(1,2) e sair do ficheiro, guardando as alterações.

Quais pacotes são trocados por DNS e que informação é transportada?

O host envia para o servidor um pacote com o hostname selecionado e o servidor responde com um pacote correspondente ao endereço IP do hostname.

A captura na qual esta análise foi baseada foi picap-exp5-step3-tux3.

Experiência 6 - Conexões TCP

O objetivo desta experiência foi compreender o comportamento do protocolo TCP desenvolvido anteriormente.

Principais comandos: Compilação da aplicação, captura de pacotes e execução da aplicação

Quantas conexões são abertas pela aplicação FTP?

São abertas duas conexões FTP. A primeira, "control socket", é utilizada para receber e enviar comandos ao servidor, de modo a aprontar tudo para a transferência do ficheiro. A segunda é a "data socket", e é utilizada para fazer a transferência em si.

Em que conexão é transportada a informação de controle do FTP?

Esta informação é transportada na primeira conexão.

Quais são as fases de uma conexão TCP?

Uma conexão TCP tem 3 fases: o estabelecimento da conexão, a transferência de dados e, por fim, o encerramento da conexão.

Como funciona o mecanismo ARQ TCP? Quais são os campos TCP relevantes? Que informação relevante pode ser observada nos logs?

Este é utilizado pelo TCP para garantir a entrega confiável de dados. Quando um pacote é enviado, o emissor espera pela receção de um ACK do destinatário, que funciona como uma confirmação que este recebeu o pacote em questão. Caso o emissor não receba o ACK, reenvia o pacote inicial. São também utilizados o window size, que indica a gama de pacotes recebidos e o sequence number, que é o número do pacote a ser enviado.

Como funciona o mecanismo de controle de congestionamento do TCP? Que campos são relevantes? Como evoluiu o throughput da conexão de dados? Está de acordo com o mecanismo de controle de congestionamento do TCP?

Este é um sistema cujo objetivo é evitar o congestionamento de uma rede. O mecanismo opera através do uso de dois campos: o tamanho da janela de congestão e o tamanho da janela de transmissão. Relativamente ao primeiro campo, o seu valor é regulado de acordo com a congestão da conexão, sendo incrementado caso a congestão da rede aumente, e decrementado caso diminua. O recebimento de um ACK faz com que a janela de congestão aumente, enquanto que a receção de um segmento que não é identificado dessa forma faz com que essa janela diminua. Quando o tamanho da janela de congestão atinge o tamanho da janela de transmissão, a taxa de transmissão diminui, de modo a evitar o congestionamento da rede. A taxa de transmissão de pacotes do primeiro download diminui quando começa o segundo download, pelo que o throughput estabilizou num nível mais baixo do que era anteriormente (antes do segundo download ter começado).

O throughput de uma conexão de dados TCP é influenciado pela aparição de uma segunda conexão TCP? Como?

Sim, a taxa de transmissão de pacotes do primeiro download diminui quando começa o segundo download, pelo que o throughput estabilizou num nível mais baixo do que era anteriormente (antes do segundo download ter começado). Isto ocorre porque foi atribuída uma capacidade para a transmissão de pacotes do novo download.

As capturas nas quais esta análise foi baseada foram picap-exp6-step2-tux3.pcapng, picap-exp6-step5-tux3.pcapng, picap-exp6-step5-tux2.pcapng.

Anexos

Principais funções da aplicação

```
/**
```

```
* Struct that contains the necessary fields to parse the command line arguments
passed
*/
struct arguments {
   char user[MAX_IP_LENGTH]; /**< user string */
   char password[MAX_IP_LENGTH]; /**< password string */
   char host_name[MAX_IP_LENGTH]; /**< host name string */
   char file_path[MAX_IP_LENGTH]; /**< file path string */
   char file_name[MAX_IP_LENGTH]; /**< file name string */
};

/**
* Struct that contains the control and data file descriptors for the FTP
*/
struct ftp {
   int control_socket_fd; /**< file descriptor to control socket */
   int data_socket_fd; /**< file descriptor to data socket */
};

int getIPAddress(char *ipAddress, char *hostName);

int clientTCP(char *address, int port);

int parseURL(struct arguments* args, char* commandLineArg);

int receiveFromControlSocket(struct ftp *ftp, char* string, size_t size);

int sendToControlSocket(struct ftp* ftp, char* cmdHeader, char* cmdBody);

int sendCommandReceiveResponse(struct ftp* ftp, char* cmdHeader,  char* cmdBody,
char* response, size_t responseLength, int readingFile);

int login(struct ftp* ftp, char* username, char* password);

int cwd(struct ftp* ftp, char* path);

int getServerPortForFile(struct ftp *ftp);

int retr(struct ftp* ftp, char* fileName);

int downloadFile(struct ftp* ftp, char* fileName);

int disconnect(struct ftp* ftp);
```

Resultados da aplicação

```
> ./main ftp://ftp.gnu.org/gnu/gdb/gdb-5.2.1.tar.gz
Parsing command line arguments
Parsed command line arguments.
User: anonymous
Password:
Host name: ftp.gnu.org
File path: gnu/gdb
File name: gdb-5.2.1.tar.gz
Receiving from control socket: 220 GNU FTP server ready.
Expecting username
Sending Username
Sending to control Socket: user anonymous
Receiving from control socket: 230-NOTICE (Updated October 15 2021):
230-
230-If you maintain scripts used to access ftp.gnu.org over FTP,
230-we strongly encourage you to change them to use HTTPS instead.
230-
230-Eventually we hope to shut down FTP protocol access, but plan
230-to give notice here and other places for several months ahead
230-of time.
230-
230----
230-
230-Due to U.S. Export Regulations, all cryptographic software on this
230-site is subject to the following legal notice:
230-
230-    This site includes publicly available encryption source code
230-    which, together with object code resulting from the compiling of
230-    publicly available source code, may be exported from the United
230-    States under License Exception "TSU" pursuant to 15 C.F.R. Section
230-    740.13(e).
230-
230-This legal notice applies to cryptographic software only. Please see
230-the Bureau of Industry and Security (www.bxa.doc.gov) for more
230-information about current U.S. regulations.
230 Login successful.
Sent Username
Sending to control Socket: CWD gnu/gdb
Receiving from control socket: 250 Directory successfully changed.
Sending to control Socket: pasv
Receiving from control socket: 227 Entering Passive Mode (209,51,188,20,103,182).
Port number 26550
Sending to control Socket: RETR gdb-5.2.1.tar.gz
Receiving from control socket: 150 Opening BINARY mode data connection for gdb-5.2.1.tar.
gz (14715792 bytes).
Starting to download file with name gdb-5.2.1.tar.gz
Finished dowloading file
Receiving from control socket: 226 Transfer complete.
Sending to control Socket: QUIT
Receiving from control socket: 221 Goodbye.
```

Codigo fonte

ficheiro main.c

```c
#include <stdio.h>
#include "include/auxiliar.h"

//./main ftp://ftp.up.pt/pub/...

// TO DO
/* - unique use case: connect, login host, passive, get path, success (file saved
in CWD) or un-success (indicating failing phase)
- challenging programming aspects: gethostbyname, sockets, control connection,
passive, data connection */

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s %s\n", argv[0],
"ftp://[<user>:<password>@]<host>/<url-path>");
        return -1;
    }

    // parse command line arguments
    struct arguments args;
    if (parseURL(&args, argv[1]) != 0) {
        return -1;
    }
    printf("User: %s\n", args.user);
    printf("Password: %s\n", args.password);
    printf("Host name: %s\n", args.host_name);
    printf("File path: %s\n", args.file_path);
    printf("File name: %s\n", args.file_name);
```

```
struct ftp ftp;
char response[MAX_IP_LENGTH];    // buffer to read commands

// get IP Address
char ipAddress[MAX_IP_LENGTH];
if (getIPAddress(ipAddress, args.host_name) < 0) {
    return -1;
}

// create and connect socket to server
if ((ftp.control_socket_fd = clientTCP(ipAddress, FTP_PORT_NUMBER)) < 0) {
    printf("Error creating new socket\n");
    return -1;
}

// receive confirmation from server
receiveFromControlSocket(&ftp, response, MAX_IP_LENGTH);

// checking confirmation from server
if (response[0] == '2') {
    printf("Expecting username\n");
}
else
{
    printf("Error in conection\n");
    return -1;
}

// login in the server
if (login(&ftp, args.user, args.password) < 0) {
    printf("Login failed\n");
    return -1;
}

// change working directory in server
if (strlen(args.file_path) > 0) {
    if (cwd(&ftp, args.file_path) < 0)
    {
        printf("Error changing directory\n");
        return -1;
    }
}

// sends pasv command to get ip address and port to receive the file
if (getServerPortForFile(&ftp) < 0){
    printf("Error getting server Port for file\n");
    return -1;
}

// sends retr command to begin file transfer
if(retr(&ftp, args.file_name) < 0){
    printf("Error sending comand retr\n");
    return -1;
}

// downloads file
if(downloadFile(&ftp, args.file_name) < 0){
    printf("Error downloading file\n");
    return -1;
}

// disconnects from server
if(disconnect(&ftp) < 0){
    printf("Error disconnecting from server\n");
    return -1;
}

return 0;
```

```
}
```

ficheiro auxiliar.h

```
#ifndef _AUXILIAR_H_
#define _AUXILIAR_H_

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>


#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include "macros.h"

/**
* Struct that contains the necessary fields to parse the command line arguments
passed
*/
struct arguments {
   char user[MAX_IP_LENGTH]; /**< user string */
   char password[MAX_IP_LENGTH]; /**< password string */
   char host_name[MAX_IP_LENGTH]; /**< host name string */
   char file_path[MAX_IP_LENGTH]; /**< file path string */
   char file_name[MAX_IP_LENGTH]; /**< file name string */
};

/**
* Struct that contains the control and data file descriptors for the FTP
*/
struct ftp {
   int control_socket_fd; /**< file descriptor to control socket */
   int data_socket_fd; /**< file descriptor to data socket */
};

int getIPAddress(char *ipAddress, char *hostName);

int clientTCP(char *address, int port);

int parseURL(struct arguments* args, char* commandLineArg);

int receiveFromControlSocket(struct ftp *ftp, char* string, size_t size);

int sendToControlSocket(struct ftp* ftp, char* cmdHeader, char* cmdBody);

int sendCommandReceiveResponse(struct ftp* ftp, char* cmdHeader,  char* cmdBody,
char* response, size_t responseLength, int readingFile);

int login(struct ftp* ftp, char* username, char* password);

int cwd(struct ftp* ftp, char* path);

int getServerPortForFile(struct ftp *ftp);

int retr(struct ftp* ftp, char* fileName);

int downloadFile(struct ftp* ftp, char* fileName);

int disconnect(struct ftp* ftp);


#endif // _AUXILIAR_H_
```

ficheiro macros.h

```
#ifndef _MACROS_H_
#define _MACROS_H_

#define MAX_IP_LENGTH 200
#define FTP_PORT_NUMBER 21

#endif // _MACROS_H_
```

ficheiro auxiliar.c

```
#include "../include/auxiliar.h"

int getIPAddress(char *ipAddress, char *hostName){
   struct hostent *h;

/**
* The struct hostent (host entry) with its terms documented

   struct hostent {
       char *h_name;    // Official name of the host.
       char **h_aliases;    // A NULL-terminated array of alternate names for the
host.
       int h_addrtype;    // The type of address being returned; usually AF_INET.
       int h_length;    // The length of the address in bytes.
       char **h_addr_list;    // A zero-terminated array of network addresses for
the host.
       // Host addresses are in Network Byte Order.
   };

   #define h_addr h_addr_list[0]   The first address in h_addr_list.
*/

   if ((h = gethostbyname(hostName)) == NULL) {
       herror("gethostbyname()");
       return -1;
   }

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

   /* printf("Host name  : %s\n", h->h_name);
   printf("IP Address : %s\n", inet_ntoa(*((struct in_addr *) h->h_addr))); */

   return 0;
}

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

```
        perror("connect()");
        exit(-1);
    }

    return sockfd;
}

int parseURL(struct arguments *args, char *commandLineArg) {

    printf("Parsing command line arguments\n");

    // verifying FTP protocol
    char *token = strtok(commandLineArg, ":");
    if ((token == NULL) || (strcmp(token, "ftp") != 0)) {
        printf("Protocol is not FTP\n");
        return -1;
    }

    token = strtok(NULL, "\0");
    char arguments[MAX_IP_LENGTH];
    strcpy(arguments, token);

    // parsing user name
    char aux[MAX_IP_LENGTH];
    strcpy(aux, arguments);
    token = strtok(aux, ":");

    if (token == NULL || (strlen(token) < 3) || (token[0] != '/') || (token[1] !=
'/')) {
        printf("Error parsing the user's name\n");
        return -1;
    }
    else if (strcmp(token, arguments) == 0) {
        memset(args->user, 0, sizeof(args->user));
        strcpy(args->user, "anonymous");
        memset(args->password, 0, sizeof(args->password));
        strcpy(args->password, "");

        strcpy(aux, &arguments[2]);
        strcpy(arguments, aux);
    }
    else {
        memset(args->user, 0, sizeof(args->user));
        strcpy(args->user, &token[2]);

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

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

    // parsing hostname
    token = strtok(arguments, "/");
    if (token == NULL) {
        printf("Error parsing the hostname\n");
        return -1;
    }
    memset(args->host_name, 0, sizeof(args->host_name));
    strcpy(args->host_name, token);

    // parsing path and name
```

```
    token = strtok(NULL, "\0");
    if (token == NULL) {
        printf("Error parsing the file path\n");
        return -1;
    }
    char* file_name = strrchr(token, '/');
    if (file_name != NULL) {
        memset(args->file_path, 0, sizeof(args->file_path));
        strncpy(args->file_path, token, file_name - token);
        memset(args->file_name, 0, sizeof(args->file_name));
        strcpy(args->file_name, file_name + 1);
    }
    else {
        memset(args->file_path, 0, sizeof(args->file_path));
        strcpy(args->file_path, "");
        memset(args->file_name, 0, sizeof(args->file_name));
        strcpy(args->file_name, token);
    }

    printf("Parsed command line arguments.\n");

    return 0;

}

int receiveFromControlSocket(struct ftp *ftp, char *response, size_t size) {
    printf("Receiving from control socket: ");
    FILE *fp = fdopen(ftp->control_socket_fd, "r");
    do {
        memset(response, 0, size);
        response = fgets(response, size, fp);
        printf("%s", response);
    } while (('1' > response[0] || response[0] > '5') || response[3] != ' ');
    return 0;
}

int sendToControlSocket(struct ftp *ftp, char *cmdHeader, char *cmdBody) {
    printf("Sending to control Socket: %s %s\n", cmdHeader, cmdBody);
    int bytes = write(ftp->control_socket_fd, cmdHeader, strlen(cmdHeader));
    if (bytes != strlen(cmdHeader))
        return -1;
    bytes = write(ftp->control_socket_fd, " ", 1);
    if (bytes != 1)
        return -1;
    bytes = write(ftp->control_socket_fd, cmdBody, strlen(cmdBody));
    if (bytes != strlen(cmdBody))
        return -1;
    bytes = write(ftp->control_socket_fd, "\n", 1);
    if (bytes != 1)
        return -1;
    return 0;
}

int sendCommandReceiveResponse(struct ftp *ftp, char *cmdHeader, char *cmdBody,
char *response, size_t responseLength, int readingFile) {
    if (sendToControlSocket(ftp, cmdHeader, cmdBody) < 0) {
        printf("Error Sending Command  %s %s\n", cmdHeader, cmdBody);
        return -1;
    }
    int code;
    while (1) {
        receiveFromControlSocket(ftp, response, responseLength);
        code = response[0] - '0';
        switch (code) {
            case 1:
                // expecting another reply
                if (readingFile)
                    return 2;
```

```
                else
                    break;
            case 2:
                // request action success
                return 2;
            case 3:
                // needs aditional information
                return 3;
            case 4:
                // try again
                if (sendToControlSocket(ftp, cmdHeader, cmdBody) < 0) {
                    printf("Error Sending Command  %s %s\n", cmdHeader, cmdBody);
                    return -1;
                }
                break;
            case 5:
                // error in sending command, closing control socket , exiting
application
                printf("Command wasn\'t accepted\n");
                close(ftp->control_socket_fd);
                exit(-1);
                break;
            default:
                break;
        }
    }
}

int login(struct ftp *ftp, char *username, char *password) {
    printf("Sending Username\n");
    char response[MAX_IP_LENGTH];
    int rtr = sendCommandReceiveResponse(ftp, "user", username, response,
MAX_IP_LENGTH, 0);
    if (rtr == 3 || rtr == 2) {
        printf("Sent Username\n");
    }
    else {
        printf("Error sending Username\n");
        return -1;
    }
    if (rtr == 3){
        printf("Sending Password\n");
        rtr = sendCommandReceiveResponse(ftp, "pass", password, response,
MAX_IP_LENGTH, 0);
        if (rtr == 2) {
            printf("Sent Password\n");
        }
        else {
            printf("Error sending Password\n");
            return -1;
        }
    }
    return 0;
}

int cwd(struct ftp* ftp, char* path) {
    char response[MAX_IP_LENGTH];
    if(sendCommandReceiveResponse(ftp, "CWD", path, response, MAX_IP_LENGTH, 0) <
0){
        printf("Error sending cwd command\n");
        return -1;
    }
    return 0;
}

int getServerPortForFile(struct ftp *ftp) {
    char firstByte[4];
    char secondByte[4];
```

```
    memset(firstByte, 0, 4);
    memset(secondByte, 0, 4);
    char response[MAX_IP_LENGTH];
    int ipPart1, ipPart2, ipPart3, ipPart4;
    int port1, port2;
    int rtr = sendCommandReceiveResponse(ftp, "pasv", "", response, MAX_IP_LENGTH,
0);
    if (rtr < 0) {
        printf("Error sending pasv command\n");
        return -1;
    }
    else if (rtr == 2) {
        // starting to process information
        if ((sscanf(response, "227 Entering Passive Mode (%d,%d,%d,%d,%d,%d)",
                    &ipPart1, &ipPart2, &ipPart3, &ipPart4, &port1, &port2)) < 0) {
            printf("ERROR: Cannot process information to calculating port.\n");
            return -1;
        }
    }
    else {
        printf("Error receiving pasv command response from server\n");
        return -1;
    }

    char ip[MAX_IP_LENGTH];
    sprintf(ip, "%d.%d.%d.%d", ipPart1, ipPart2, ipPart3, ipPart4);
    int port = port1 * 256 + port2;
    printf("Port number %d\n", port);
    if ((ftp->data_socket_fd = clientTCP(ip, port)) < 0) {
        printf("Error creating new socket\n");
        return -1;
    }
    return 0;
}

int retr(struct ftp* ftp, char* fileName){
    char response[MAX_IP_LENGTH];
    if(sendCommandReceiveResponse(ftp, "RETR", fileName, response, MAX_IP_LENGTH, 1)
< 0){
        printf("Error sending retr command\n");
        return -1;
    }
    return 0;
}

int downloadFile(struct ftp* ftp, char * fileName){
    FILE *fp = fopen(fileName, "w");
    if (fp == NULL){
        printf("Error opening or creating file\n");
        return -1;
    }
    char buf[1024];
    int bytes;
    printf("Starting to download file with name %s\n", fileName);
    while((bytes = read(ftp->data_socket_fd, buf, sizeof(buf)))){
        if(bytes < 0){
            printf("Error reading from data socket\n");
            return -1;
        }
        if((bytes = fwrite(buf, bytes, 1, fp)) < 0){
            printf("Error writing data to file\n");
            return -1;
        }
    }

    printf("Finished dowloading file\n");

    if(fclose(fp)){
```

```
        printf("Error closing file\n");
        return -1;
    }
    close(ftp->data_socket_fd);
    char response[MAX_IP_LENGTH];
    receiveFromControlSocket(ftp, response, MAX_IP_LENGTH);
    if (response[0] != '2')
        return -1;
    return 0;
}

int disconnect(struct ftp* ftp) {
    char response[MAX_IP_LENGTH];
    if(sendCommandReceiveResponse(ftp, "QUIT", "", response, MAX_IP_LENGTH, 0) !=
2){
        printf("Error sending quit command\n");
        return -1;
    }
    return 0;
}
```

Todos os ficheiros pcapng estão presentes na root do repositório GitLab ,na pasta “picaps”.