Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

Lab 2 - Rede de computadores

23 de Dezembro de 2018

Nuno Tiago Tavares Lopes, up201605337@fe.up.pt
Amadeu Prazeres Pereira, up201605646@fe.up.pt
João Carlos Parada Alves, up201605236@fe.up.pt
Mateus Pedroza Cortes Marques, up201601876@fe.up.pt

Conteúdo

1 Introdução

No âmbito da unidade curricular de Redes de Computadores foi nos pedido que realizássemos um trabalho com duas finalidades: a configuração de uma rede e o desenvolvimento de uma aplicação de donwload.

Relativamente à configuração de uma rede, o seu objetivo é permitir a execução de uma aplicação, a partir de duas VLAN's dentro de um switch. De seguida, foi desenvolvida uma aplicação download de acordo com o protocolo FTP e com a ajuda de ligações TCP (Transmission Control Protocol) a partir de sockets.

Quanto a este relatório está dividido em duas partes: uma parte onde é descrita a arquitetura e o funcionamento da aplicação de donwload, e uma segunda parte onde são analisadas as várias experiências feitas nas aulas teórico-práticas.

2 Parte 1 - Aplicação de download

2.1 Arquitetura

A arquitetura do programa de download é dividida em três camadas, high-level, intermediária e low-level, representadas pelos ficheiros *clientTCP.c/.h*, *clientFTP.c/.h* e *download.c*.

2.1.1 Camada high-level (download.c)

Camada responsável por fazer parse do argumento recebido pela linha de comandos (string com url do servidor) e chamar a função **downloadFile** da camada intermediária, responsável por fazer download de um ficheiro via FTP.

```
// Download File
  downloadFile(ip, user, password, urlPath);
```

*Função **downloadFile** da camada intermediária é chamada no ficheiro download.c após calcular os argumentos.*

2.1.2 Camada intermediária (clientFTP.c/.h)

Responsável por chamar funções da camada low-level para abrir, fechar, escrever e ler de sockets TCPs (**openTcpSocket, closeTcpSocket, writeTcp e readTcp**). Escreve nos sockets as funções e os argumentos FTPs necessários para fazer download de um ficheiro (ex: "USER"). Além disso essa camada abre um novo processo para escrever um novo ficheiro com os dados recebidos para replicar o ficheiro localmente.

```
int downloadFTP(char *filePath)
{
 // write download file command
 if (writeFTP(primaryFtpSocket, "RETR", filePath == NULL ?
     "" : filePath))
 {
    printf("Error writing to server.\n");
    return -1;
 }

 char *msg = readTcp(primaryFtpSocket);

 printf("%s\n", msg);
...
```

*Função **writeTcp** da camada low-level é chamada no ficheiro clientFTP.c para pedir ao servidor FTP para fazer download de um ficheiro.*

```
int pchild, status;

pchild = fork();

if (pchild == 0)
{
    // Child process
 secondaryFtpSocket = openTcpSocket(ip, secondaryPort);

 if (secondaryFtpSocket < 0)
 {
    printf("Error opening secondary socket.\n");
    return -1;
 }

 if (receiveFile(filePath))
    {
       printf("Error receiving file.\n");
       return -1;
    }

    printf("Finished child process.\n");
 }
 else if (pchild > 0){
    // Parent process

    // download file
    if (downloadFTP(filePath))
    {
       printf("Error downloading file.\n");
       kill(pchild, SIGKILL);
       free(msg);
       return -1;
    }

    // wait for child process to finish (download process)
    waitpid(pchild, &status, 0);
 }
 ...
```

Um processo filho é iniciado para salvar o ficheiro recebido localmente, enquanto o processo pai pede ao servidor para baixar o ficheiro.

2.1.3 Camada low-level (clientTCP.c/.h)

Camada representadas por funções capazes de fazer interface com sockets TCP. Possui uma função para obter o IP baseado no hostname (**getIP** - Providenciada pela cadeira) e funções para abrir, fechar, escrever e ler de sockets TCP.

```
int writeTcp(int socket, char *msg)
{
 int bytes;

 strcat(msg, "\r\n");

 /* send a string to the server */
 bytes = send(socket, msg, strlen(msg) * sizeof(char), 0);

 printf("%s", msg);

 /* read response */
 return bytes;
}
```

*Função **writeFtp** presente no ficheiro clientTCP.c faz interface com a escrite em sockets TCP através da função **send**.*

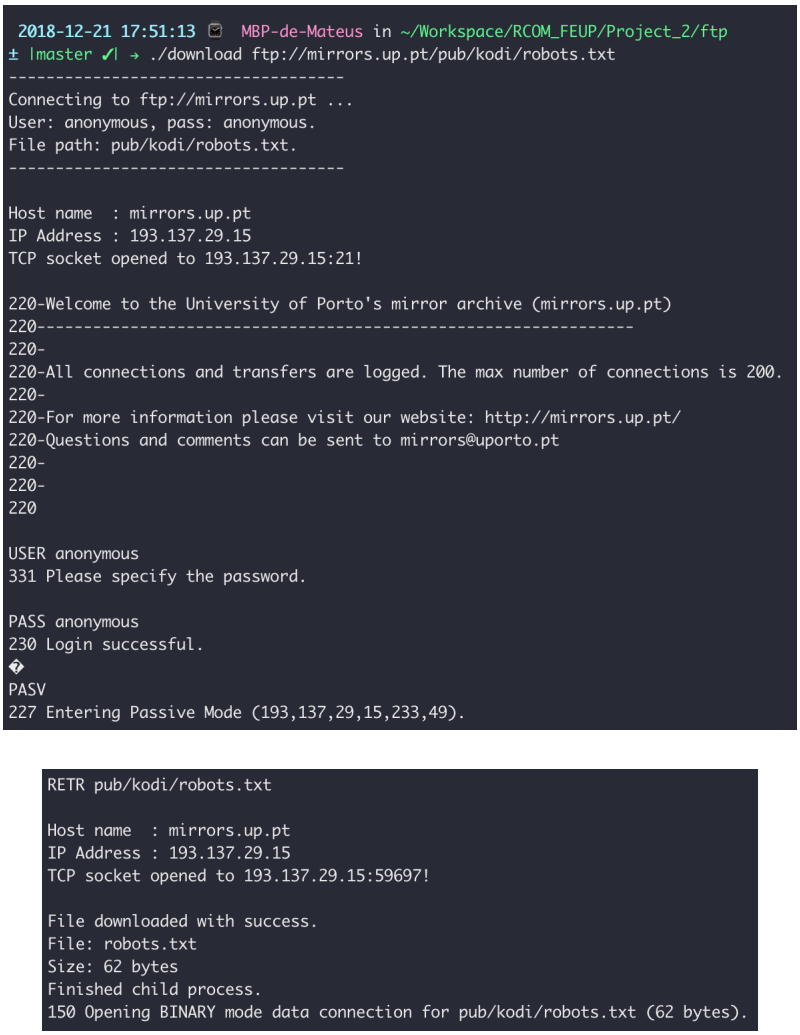
2.2 Relatório de um download com sucesso

Para realizar download de um ficheiro, basta executar o ficheiro **download** com a url do servidor, credenciais de autênticação (opcionais - *anonymous* caso deixe em branco) e o caminho até o ficheiro desejado.

Em caso de sucesso, será apresentado as mensagens de boas vindas do servidor. Após mostrar a mensagem, é realizado a autênticação com os comandos *USER* e *PASSWORD* e o pedido para o servidor responder em modo passivo (*PASV*).

Caso o ficheiro exista, será aberta uma nova conexão em um processo filho para receber os dados do ficheiro desejado e salvá-lo no disco rígido.

Após terminar o download do ficheiro, o programa termina.



Exemplo de um pedido com sucesso para fazer download de um ficheiro por FTP.

3 Parte 2 - Configuração de rede e análise

3.1 Experiência 1 - Configurar um IP de rede

A objetivo desta experiência foi configurar uma ligação entre o tuxy1 e o tuxy4 recorrendo ao switch. Para tal, foram utilizados comandos como **ifconfig** e **route** para ligar as portas eth0 dos dois computadores. Após a configuração foi enviado o sinal "ping"de um computador para o outro para verificar que, de facto, as ligações foram bem realizadas e as máquinas possuíam uma conexão.

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

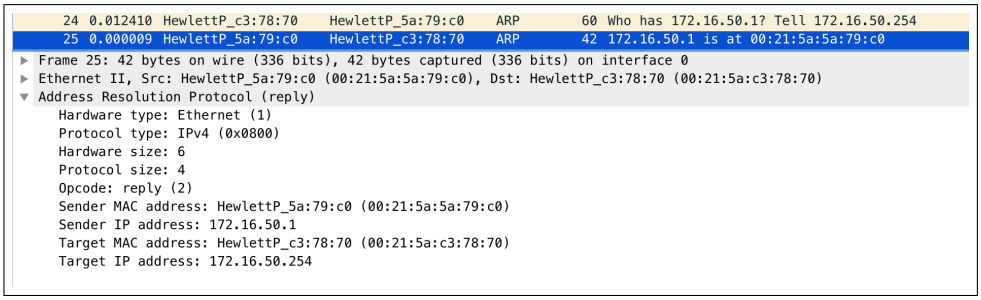
O ARP (Address Resolution Protocol) é um protocolo de comunicação que serve para descobrir o endereço da camada de ligação associado ao endereço IPv4. Serve para mapear o endereço de rede a um endereço físico como o endereço MAC.

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

Os pacotes ARP são emitidos para toda a LAN. Assim sendo, quando, por exemplo o tux X dá ping ao tux Y, pergunta a todos os tuxes ligados à LAN qual é que possui o IP do tux Y. A pergunta em si é um pacote ARP, que possui o endereço MAC e o endereço IP do tux X, bem como o endereço IP do tux Y.

```
    24 17.125482     HewlettP_c3:78:70    HewlettP_5a:79:c0    ARP         60 Who has 172.16.50.1? Tell 172.16.50.254
    25 17.125491     HewlettP_5a:79:c0    HewlettP_c3:78:70    ARP         42 172.16.50.1 is at 00:21:5a:5a:79:c0
▶ Frame 24: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▶ Ethernet II, Src: HewlettP_c3:78:70 (00:21:5a:c3:78:70), Dst: HewlettP_5a:79:c0 (00:21:5a:5a:79:c0)
▼ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: HewlettP_c3:78:70 (00:21:5a:c3:78:70)
    Sender IP address: 172.16.50.254
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 172.16.50.1
```

Após a pergunta, o tux X fica à espera da resposta do tux Y que irá informa-lo que o IP Y se encontra no MAC Y.

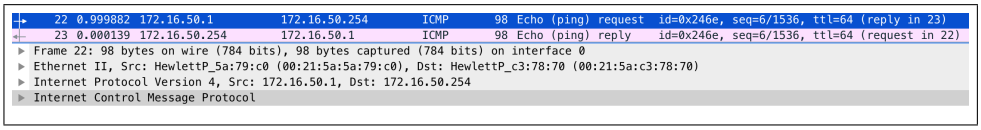


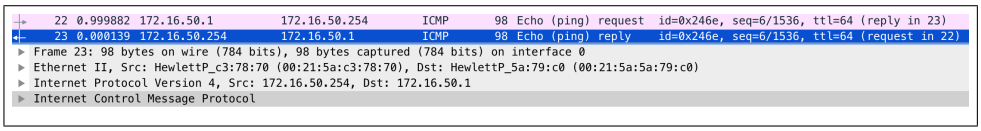
- Que pacotes são gerados pelo comando ping?

O comando ping gera primeiro pacotes ARP para obter os endereços MAC e de seguida gera pacotes ICMP (Internet Control Message Protocol).

- Quais são os endereços MAC e IP dos pacotes ping?

Os endereços (origem e destino) IP e MAC dos pacotes, quando fazemos "ping"no tuxy1 do tuxy4, podem ser vistos pelos resultados obtidos no wireshark.



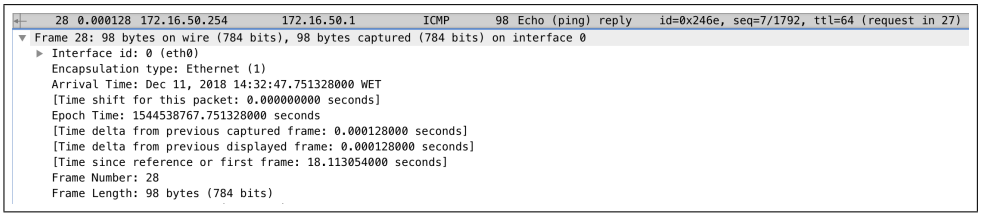


- Como determinar se a trama recetora Ethernet é ARP, IP ou ICMP?

Tal como é referido no guião, inspecionando o cabeçalho de um pacote conseguimos determinar o tipo da trama. Cada tipo de trama tem cabeçalho diferente. Por exemplo, no caso das tramas do tipo ARP, ocabeçalho é 0x0806 e se o valor do cabeçalho for 0x0800 é uma trama do tipoIPv4.

- Como determinar o comprimento da trama recetora?

Para determinar o comprimento da trama recetora temos de consultar o wireshark. Como podemos verificar a trama recetora tem um comprimento de 98 bytes.



- O que é a interface loopback e porque é que é tão importante?

A interface loopback é uma interface virtual da rede que permite ao computador receber respostas de si mesmo. É usada para testar se a carta de rede está configurada corretamente.

3.2 Experiência 2 - Implementar duas LAN's virtuais no switch

Para a realização desta experiência tivemos de configurar duas LANs virtuais no switch, sendo que na VLANY0 foram associados o tuxy1 e o tuxy4 e na VLANY1 foi associado o tuxy2. O objetivo desta experiência era verificar que com esta configuração o tuxy2 deixaria de ter acesso aos tuxy1 e tuxy4, uma vez que estas máquinas se encontrariam em sub-redes diferentes.

- Como configurar a vlany0?

Depois de ligar um dos tuxes ao switch é só realizar os seguintes comandos no GTKTerm:

```
configure terminal
vlan y0
end
```

De seguida, tem que se adicionar as ligações nos computados que já tinham sido criadas fisicamente, ligado os tux 1 e 4:

```
configure terminal
interface fastethernet 0/y, em que y  a porta
switchport mode access
switchport access vlan y0
end
```

- Quantos broadcast domains existem? Como é que podemos concluir isto, partindo dos logs?

Existem dois domínios de broadcast, visto que o tuxy1 recebe resposta do tuxy4 quando se faz "ping broadcast", mas não do tuxy2. Ou seja, existem dois domínios de broadcast: o que contém o tuxy1 e tuxy4 e o que contém o tuxy2.

3.3 Experiência 3 - Configurar um router em Linux

Esta experiência teve como objectivo configurar o tux4 como um router entre as duas sub-redes previamente criadas. Para este efeito, foi necessário ligar a interface *eth1* do tux4 e configurá-la com um IP dentro da mesma gama do tux2; adicionando, de seguida, esta interface à sub-rede do tux2.

Após este passo, adicionou-se uma rota ao tux1 utilizando o comando ***<route add –net 172.16.y1.0/24 gw 172.16.y0.254>***.

De seguida, repetiu-se o procedimento para o tux2, mas utilizando os seguintes endereços: ***<route add –net 172.16.y0.0/24 gw 172.16.y1.253>***

No final foi possível enviar um comando *ping* do tux1 para o tux2 com sucesso.

- Que rotas podem ser encontradas nos tuxes e qual é o significado destas?

Inicialmente cada tux apresenta uma rota para a vlan associada, o tux1 para a vlany0, o tux2 para a vlany1 e, por fim o tux4 para a vlany0 e para a vlany1. Durante esta experiência foram adicionadas rotas que permitam o tux1 e o tux2 contactarem-se, ou seja foi adicionada uma rota no tux1 para a vlany1 através do tux4 e uma rota no tux2 para a vlany0 através do tux4 também.

- Que informação é contida numa entrada de uma forwarding table?

Uma forwarding table guarda informação acerca da Destionation, Netmask (estes 2 definem a network id), gateway, indica por onde a rede pode ser acedida, interface, indica a interface responsável por atingir o gateway e, por fim, a métrica indica o custo associado à rota.

- Quais mensagens ARP e endereços MAC associados são observados e porquê?

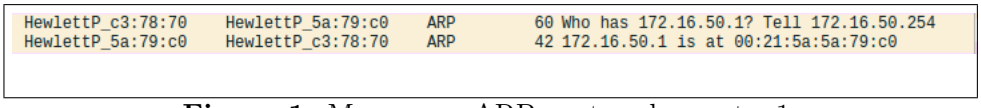


Figura 1: Mensagens ARP capturadas no tux1.

No início da comunicação, quer o tux1, quer o tux2 não sabem qual o endereço MAC do respetivo tux4 e, portanto, também emitem um pacote ARP para descobrir esse endereço (no caso do tuxy1 o .254).

Na tentativa do tux1 dar ping ao tux2, é possível verificar a mensagem ARP em que o tux4 pede o endereço MAC relativo ao IP do tux2. O tux1 envia a informação para o tux2, sendo esta redirecionada para o tux4 através da tabela de rooting. Visto o tux4 não possuir na sua tabela ARP o endereço MAC do tux2 este necessita de transmitir (broadcast) um pacote ARP para descobrir esse endereço MAC.

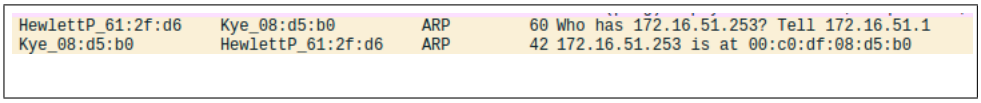


Figura 2: Mensagens ARP capturadas no tux4, interface eth1.

- Quais pacotes ICMP são observados e porquê?

Na sequência da execução do comando ping no tux1, com as interfaces (172.16.y0.254, 172.16.y1.253, 172.16.y1.1) como destino é possível verificar os diferentes pacotes ICMP associados a cada IP.

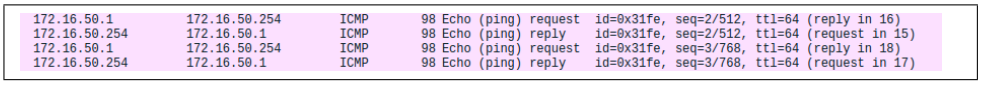


Figura 3: Pacotes ICMP capturados no tux1.

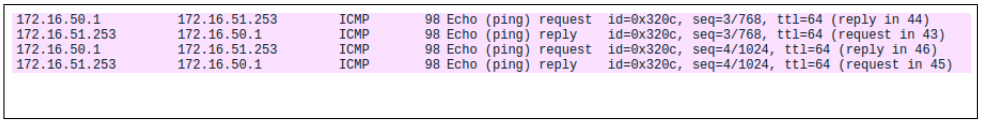


Figura 4: Pacotes ICMP capturados no tux1.

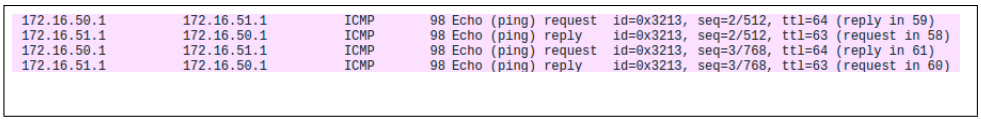


Figura 5: Pacotes ICMP capturados no tux1.

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

Os endereços MAC associados aos pacotes observados são os das respetivas máquinas de destino e origem. No entanto, devido ao *forwarding* de pacotes pelo tux4, os pacotes podem ter endereço IP de destino correspondente ao do tux4 (terminado em 50.254 ou 51.253).

3.4 Experiência 4 - Configurar um router comercial e implementar o NAT

Nesta experiência pretendia-se fazer a configuração de um router comercial com NAT devidamente implementado, sendo necessária a configuração de duas interfaces deste mesmo router (gigabitEthernet 0/0 ligada à vlany1 e gigabitEthernet 0/1 ligada à rede exterior da sala).

- Como configurar um router estático num router comercial?

Para configurar um router, como falado previamente, é necessário configurar duas interfaces deste (acedendo a elas através do comando <***interface gigabitEthernet 0/[interface]***>, definindo o seu endereço IP e a sua máscara de bits com <***ip address*** <***ip***> <***mascara***>>, com o ip 172.16.51.254 para a interface 0/0 e 172.16.1.y9 para a interface 0/1 e mascara 255.255.255.0 para as duas. Seguidamente foi definido o ponto de entrada de entrada e saída de NAT, na interface 0/0 foi utilizado o comando ***nat inside*** e o na interface 0/1 ***nat outside***. Também é necessário utilizar o ***no shutdown*** em ambas as interfaces para as configurações não serem perdidas caso o router seja desligado. Após esta configuração das interfaces é necessário utilizar os comandos <***ip nat pool ovrld 172.16.1.y9 172.16.1.y9 prefix 24***> e <***ip nat inside source list 1 pool ovrld overload***> para garantir a

gama de endereços. Para criar a lista de acessos e permissões de pacotes utilizou-se <***acesslist 1 permit*** <***ip***> <***max***>>, este comando teve que ser executado 2 vezes, para os ips 172.16.y0.0 e 172.16.y1.0 com max igual a 0.0.0.255 (apesar de no exemplo do lab eles utilizarem 0.0.0.7 nós pretendiamos que o tux4 também tivesse acesso à internet). Faltava só definir rotas no router com o comando <***ip route*** <***dest***> <***mask***> <***gateway***>>. Foram definidas duas rotas: uma interna com dest 0.0.0.0, mask 0.0.0.0 e gateway 172.16.1.254 e uma externa com dest 172.16.y0.0, mask 255.255.255.0 e gateway 172.16.y1.253. Desta forma garante-se que os pacotes têm forma de chegar à sub-rede pretendida.

- O que é que o NAT faz?

O NAT (Network Address Translation) possibilita a comunicação entre os computadores da rede criada e redes externas. Como estamos a trabalhar numa rede privada os ip's definidos nunca seriam reconhecidos externamente, para isso é necessário reescreve-los.

3.5 Experiência 5 - DNS

Nesta experiência pretendia-se configurar o servidor de DNS para permitir ligação à Internet através da procura de nomes de domínios. Domain Name System (DNS) é responsável por associar e traduzir diversa informação associada aos nomes dos domínios em particular o seu IP.

- Como se configura um serviço DNS a um host?

Para configurar um serviço DNS é necessário aceder e editar o ficheiro resolv.conf. Este ficheiro é lido sempre que são invocadas rotinas que fornecem acesso à Internet. Neste caso adicionamos duas entradas ao ficheiro: "search netlab.fe.up.pt"e "nameserver 172.16.1.1"que representam o servidor a ser acedido.Para tal, foi utilizado o formato **search new-page-name <nameserver><IP-page>**, sendo que *new-page-name* representa o nome da página, e *IP-page* o endereço IP da página.

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

Para testar esta funcionalidade foi feito um ping ao domínio www.google.com.

Através da análise dos resultados do *Wireshark* podemos observar que no início da comunicação, antes do envio/receção de pacotes ICMP (enviados pelo comando **ping <ip>**), foram enviados pacotes DNS para identificar o

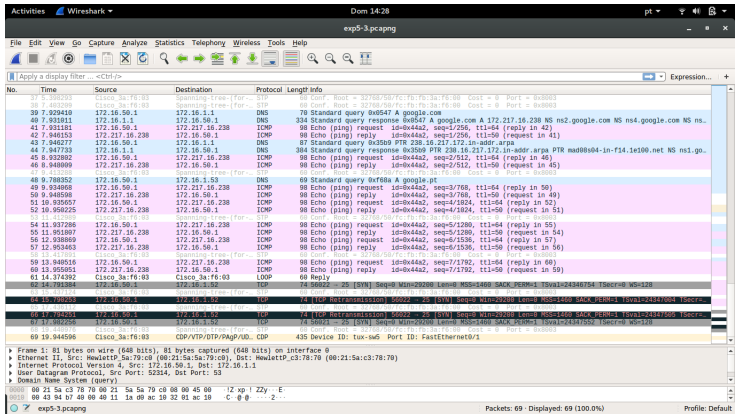


Figura 6: Pacotes registados no ping feito a www.google.com

endereço IP do destino (neste caso www.google.com). Foi pedido ao DNS que enviasse os atributos do domínio através do pacote 39. Como resposta o servidor enviou o pacote 40 que inclui entre outras informações o endereço IP do destino. Posteriormente, foi também feito um pedido *reverse* DNS (rDNS) nos pacotes 43 e 44 que processa o pedido contrário, ou seja, obter o nome domínio a partir do enderenço IP.

3.6 Experiência 6 - Ligações TCP

Nesta experiência executou-se a aplicação desenvolvida na primeira parte do trabalho, analisando as suas características de comunicação.

- Quantas conexões TCP são abertas pela aplicação FTP?

A aplicação abriu 2 ligações TCP, uma recebia e mandava os comandos FTP para o servidor e a outra recebia os dados enviados pelo servidor.

- Em qual conexão FTP é transportada a sua informação de controlo?

A informação de controlo vai na conexão TCP responável pela troca de comandos.

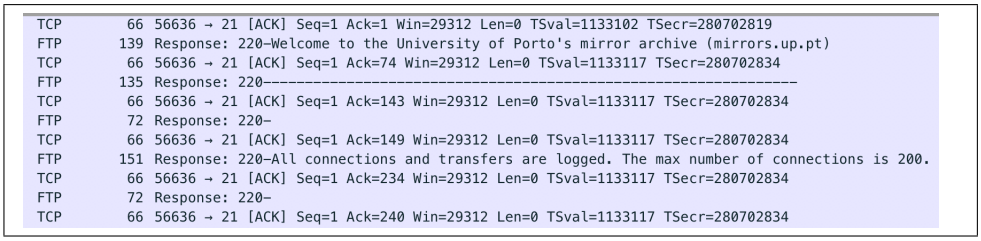


Figura 7: Pacotes capturados no tux1.

- Quantas fases tem uma conexão TCP?

Uma conexão TCP apresenta 3 fase: a conexão ao servidor, a troca de dados/informção e o terminação da conexão.

- Como é que o mecanismo ARQ TCP funciona? Quais são os campos TCP relevantes? Que informação relevante pode ser retirada dos logs?

O mecanismo ARQ (Automatic Repeat Request) TCP é uma variação de Go-Back-N com a diferença que o recetor não deixa de processar os frames recebidos quando deteta um erro. O recetor continua a receber as frames seguintes enviando no ACK o número da frame que falhou até a conseguir receber. O emissor verifica os ACK e reenvia a frames perdidas.

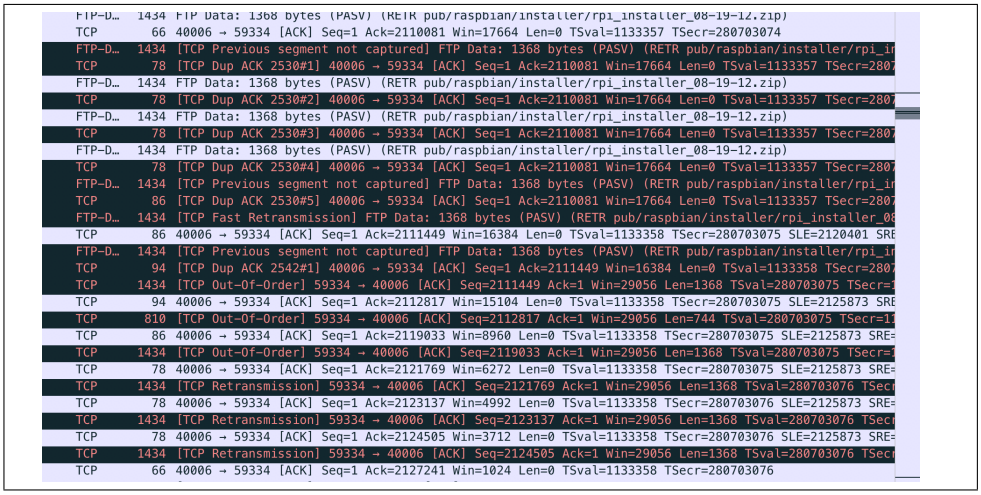


Figura 8: Pacotes Dup ACK

- Como é que o mecanismo de controlo de congestionamento TCP funciona? Quais são os seus campos relevantes? Como é que o fluxo de

dados da conexão evolui ao longo do tempo? Está de acordo com o mecanismo de controlo de congestionamento TCP?

O TCP utiliza um mecanismo de controlo de congestionamento, ou seja ele limita ou aumenta a taxa de envio de dados em função do congestionamento da rede. Durante o início de uma conexão TCP temos a fase de partida lenta, que depois aumenta sua taxa exponencialmente. Seguidamente sofre uma descida e acaba por estabilizar, mas ainda sofrendo algumas alterações na taxa de transferência. A informação recolhida está de acordo com o mecanismo de controlo de congestionamento TCP.

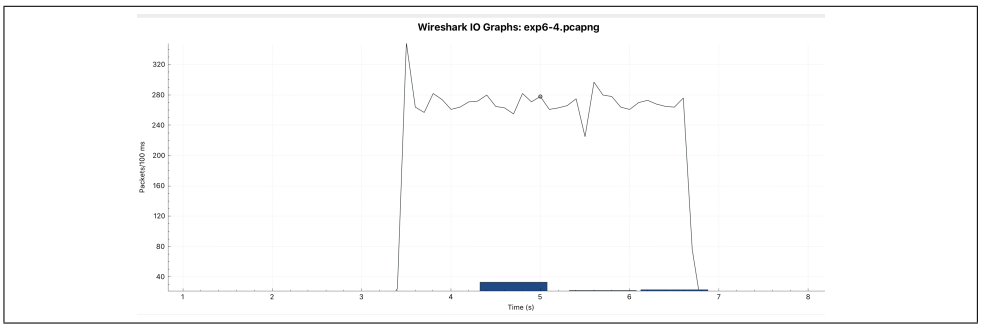


Figura 9: Variação do fluxo de dados

- O fluxo de dados da conexão é afetada pelo aparecimento de uma segunda conexão TCP? Como?

Ao iniciar uma segunda conexão TCP o fluxo de dados é afetado. Apesar da média de transeferência de pacotes se ter mantido ao iniciar uma segunda conexão verificou-se um aumento de decréscimos, levando, assim, a que o download do ficheiro do servidor TCP demora-se mais tempo.

4 Conclusão

No final, foi possível concluir que a aplicação implementada era robusta e permitia um *download* de ficheiros na sua integra e sem erros. Consideramos que a realização e compreensão exaustiva das experiências traduziu-se posteriormente na facilidade de desenvolvimento da aplicação e na compreensão de todos os conceitos relativos ao projeto. Assim, é de realçar que as experiências se revelaram uma grande fonte de conhecimento visto que correspondiam à parte mais complexa e trabalhosa do projeto. Foram dominados os objetivos implícitos no trabalho tais como: compreensão do conceito de 'cliente - servidor', compreensão do protocolo de comunicação *TCP / IP*, compreensão do protocolo de comunicação *FTP*, compreensão do serviço *DNS*. O relatório é sinónimo do descrito em cima estando o grupo orgulhoso do que conseguiu desenvolver.

5 Referências

- Catalyst 3560 Switch Software Configuration Guide
- Linux Network Administrators Guide
- Transmission Control Protocol

6 Anexos

6.1 Comandos de configuração

6.1.1 tux51

```
ifconfig eth0 172.16.50.1/24
route add -net 172.16.51.0/24 gw 172.16.50.254
route add default gw 172.16.50.254
```

6.1.2 tux52

```
ifconfig eth0 172.16.51.1/24
route add -net 172.16.50.0/24 gw 172.16.51.253
route add default gw 172.16.51.254
```

6.1.3 tux54

```
ifconfig eth0 172.16.50.254/24
ifconfig eth1 172.16.51.253/24
route add default gw 172.16.51.254

echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
```

6.1.4 switch

```
configure terminal
vlan 50
end

configure terminal
interface fastethernet 0/1
switchport mode access
switchport access vlan 50
end

configure terminal
interface fastethernet 0/2
```

```
switchport mode access
switchport access vlan 50
end

configure terminal
vlan 51
end

configure terminal
interface fastethernet 0/3
switchport mode access
switchport access vlan 51
end

configure terminal
interface fastethernet 0/4
switchport mode access
switchport access vlan 51
end

configure terminal
interface fastethernet 0/5
switchport mode access
switchport access vlan 51
end
```

6.1.5 router

```
configure terminal

interface gigabitEthernet 0/0
ip address 172.16.51.254 255.255.255.0
no shutdown
ip nat inside
exit

interface gigabitEthernet 0/1
ip address 172.16.1.59 255.255.255.0
no shutdown
ip nat outside
exit

ip nat pool ovrld 172.16.1.59 172.16.1.59 prefix 24
```

```
        ip nat inside source list 1 pool ovrld overload

        access-list 1 permit 172.16.50.0 0.0.0.255
        access-list 1 permit 172.16.51.0 0.0.0.255

        ip route 0.0.0.0 0.0.0.0 172.16.1.254
        ip route 172.16.50.0 255.255.255.0 172.16.51.253

        end
```

6.2 Código fonte da aplicação

6.2.1 download.c

```
#include "clientFTP.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void removeSubstr(char *string, char *sub)
{
   char *match;
   int len = strlen(sub);
   while ((match = strstr(string, sub)))
   {
      *match = '\0';
      strcat(string, match + len);
   }
}

int main(int argc, char **argv)
{
   char *urlPath, *ip, *user, *password, *pch, *url;

   if (argc != 2)
   {
      printf("Not enought arguments.\n");
      printf("Usage: ftp://[<user>:<password>@]<host>/<url-path>\n");
      return -1;
   }

   // remove ftp tag from full argument
   url = malloc(strlen(argv[1]) - 5);
```

```
strncpy(url, &(argv[1][6]), strlen(argv[1]) - 5);
//protocol = malloc()

// parse info from url argument
int credentials = 0; //0 - no credentials, 1 - credentials given
char *at = strchr(url, '@');
char *two_dots = strchr(url, ':');
if (at != NULL && two_dots != NULL)
{
   credentials = 1;
}
else if ((at != NULL && two_dots == NULL) || (at == NULL &&
    two_dots != NULL))
{
   printf("User or/and password wrong\n");
   exit(-1);
}

pch = strtok(url, "/:@");

int count = 0;

while (count != 4)
{
   int size = sizeof(char) * strlen(pch);

   if (credentials == 0)
   {
      // ip
      if (count == 0)
      {
         ip = malloc(size);
         strcpy(ip, pch);
         pch = strtok(NULL, "");
      }

      // file path
      else
      {
         urlPath = malloc(size);
         strcpy(urlPath, pch);
         count = 3;
      }
   }
   else
```

```
    {
        // user
        if (count == 0)
        {
            user = malloc(size);
            strcpy(user, pch);
            pch = strtok(NULL, "/:@");
        }

        // password
        else if (count == 1)
        {
            password = malloc(size);
            strcpy(password, pch);
            pch = strtok(NULL, "/:@");
        }

        // ip
        else if (count == 2)
        {
            ip = malloc(size);
            strcpy(ip, pch);
            pch = strtok(NULL, "");
        }

        // file path
        else if (count == 3)
        {
            urlPath = malloc(size);
            strcpy(urlPath, pch);
        }
    }

    count++;
}

if (credentials == 0)
{
    user = "anonymous";
    password = "anonymous";
}

if (user == NULL)
{
    printf("User can't be null.\n");
```

```
        exit(-1);
    }

    if (ip == NULL)
    {
        printf("Ip can't be null.\n");
        exit(-1);
    }

    if (password == NULL)
    {
        printf("Password can't be null.\n");
        exit(-1);
    }

    if(urlPath == NULL){
        printf("URL Path can't be null.\n");
        exit(-1);
    }

    printf("-------------------------------------\n");
    printf("Connecting to ftp://%s ...\n", ip);
    printf("User: %s, pass: %s.\n", user, password);
    printf("File path: %s.\n", urlPath);
    printf("-------------------------------------\n");

    // Download File
    downloadFile(ip, user, password, urlPath);

    free(ip);
    if (credentials == 1)
    {
        free(user);
        free(password);
    }
    free(urlPath);
    free(url);

    // close ftp program
    closeFTP();

    return 0;
}
```

6.2.2 clientFTP.h

```
#ifndef _CLIENT_FTP_H_
#define _CLIENT_FTP_H_

int closeFTP();

int downloadFile(char *ip, char *user, char *password, char
    *filePath);

#endif // _CLIENT_FTP_H_
```

6.2.3 clientFTP.c

```
#define _POSIX_SOURCE
#include "clientTCP.h"
#include "clientFTP.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <signal.h>
#include <sys/types.h>

#define PORT 21
#define USER "USER"
#define PASS "PASS"

int primaryFtpSocket;
int secondaryFtpSocket;

int writeFTP(int ftpSocket, char *cmd, char *arg)
{
   char buffer[256] = "";

   strcat(buffer, cmd);
   strcat(buffer, " ");
   strcat(buffer, arg);
   // write msg to tcp socket
   return writeTcp(ftpSocket, buffer) > 0 ? 0 : -1;
}
```

```
int auth(char *user, char *password)
{
   // send user
   if (writeFTP(primaryFtpSocket, USER, user) < 0)
   {
      printf("Error on sending USER.");
      return -1;
   }

   char *msg = readTcp(primaryFtpSocket);
   printf("%s\n", msg);

   if(strncmp(msg, "331 Please specify the password.", 32) != 0){
      printf("Error on authentication: wrong USER\n");
      return -1;
   }

   // send password
   if (writeFTP(primaryFtpSocket, PASS, password) < 0)
   {
      printf("Error on sending PASS.");
      return -1;
   }

   msg = readTcp(primaryFtpSocket);
   printf("%s\n", msg);

   if(strncmp(msg, "230 Login successful.", 21) != 0){
      printf("Error on authentication: wrong PASS\n");
      return -1;
   }

   free(msg);
   return 0;
}

int downloadFTP(char *filePath)
{
   // write download file command
   if (writeFTP(primaryFtpSocket, "RETR", filePath == NULL ? "" :
       filePath))
   {
      printf("Error writing to server.\n");
      return -1;
```

```
    }

    char *msg = readTcp(primaryFtpSocket);

    printf("%s\n", msg);

    char *token = "";

    token = strtok(msg, " ");

    int code = strtol(token, NULL, 10);

    switch (code)
    {
    case 550:
        token = strtok(NULL, "");
        free(msg);
        return -1;
    default:
        break;
    }

    free(msg);

    return 0;
}

int parsePasvMsg(char *msg)
{
    char *token = "";
    int firstNumber, secondNumber;

    token = strtok(msg, "(,)");
    int i= 0;
    for (; i < 7; i++)
    {
        token = strtok(NULL, "(,)");

        if (i == 4)
        {
            firstNumber = strtol(token, NULL, 10);
        }

        else if (i == 5)
        {
```

```
         secondNumber = strtol(token, NULL, 10);
      }
   }
   return firstNumber * 256 + secondNumber;
}

char *getFileName(char *filePath)
{
   char *token = strtok(filePath, "/");
   char *previousToken = token;

   /* walk through other tokens */
   while( token != NULL ) {
      previousToken = token;
      token = strtok(NULL, "/");
   }

   return previousToken;
}

int receiveFile(char *filePath)
{
   // receive file
   char * fileName = getFileName(filePath);
   FILE *f = fopen(fileName, "w");

   if (f == NULL)
   {
      return -1;
   }

   char buffer[1];
   int bytes;
   int counter = 0;

   while((bytes = read(secondaryFtpSocket, &buffer,
       sizeof(buffer))) > 0){

      //Write to file
      fwrite(buffer, sizeof(char), sizeof(buffer), f);

      counter += bytes;

      if(bytes == -1){
         printf("Error reading socket.\n");
```

```
            return -1;
        }
    }

    if(counter == 0){
        printf("Nothing written in file.\n");
        return -1;
    }
    else{
        printf("File downloaded with success.\n");
        printf("File: %s\n", fileName);
        printf("Size: %d bytes\n", counter);
    }

    fclose(f);

    return 0;
}

int downloadFile(char *ip, char *user, char *password, char
    *filePath)
{
    // connect to primary socket
    primaryFtpSocket = openTcpSocket(ip, PORT);
    if (primaryFtpSocket < 0)
    {
        printf("Error opening socket.\n");
        return -1;
    }

    char *msg = readTcp(primaryFtpSocket);
    printf("%s\n", msg);

    // login to ftp server
    if (auth(user, password))
    {
        return -1;
    }

    // set passive mode
    if (writeFTP(primaryFtpSocket, "PASV", ""))
    {
        printf("Error entering passive mode.\n");
        return -1;
    }
```

```
msg = readTcp(primaryFtpSocket);
printf("%s\n", msg);

// Connect to secondary socket
int secondaryPort = parsePasvMsg(msg);

int pchild, status;

pchild = fork();

if (pchild == 0)
{
   // Child process
   secondaryFtpSocket = openTcpSocket(ip, secondaryPort);

   if (secondaryFtpSocket < 0)
   {
      printf("Error opening secondary socket.\n");
      return -1;
   }

   if (receiveFile(filePath))
   {
      printf("Error receiving file.\n");
      return -1;
   }

   printf("Finished child process.\n");
}
else if (pchild > 0){
   // Parent process

   // download file
   if (downloadFTP(filePath))
   {
      printf("Error downloading file.\n");
      kill(pchild, SIGKILL);
      free(msg);
      return -1;
   }

   // wait for child process to finish (download process)
   waitpid(pchild, &status, 0);
}
```

```
   else
   {
      // Error
      status = -1;
      free(msg);
      return -1;
   }

   free(msg);
   return 0;
}

int closeFTP()
{
   closeTcpSocket(primaryFtpSocket);
   closeTcpSocket(secondaryFtpSocket);
   return 0;
}
```

6.2.4 clientTCP.h

```
#ifndef _CLIENT_TCP_
#define _CLIENT_TCP_

/**
 * @brief Open tcp socket.
 *
 * @param hostname host url (ex: fe.up.pt).
 * @param port port (ex: 21).
 * @return int socket in success, -1 in error.
 */

#define h_addr h_addr_list[0]

int openTcpSocket(char *hostname, int port);

int writeTcp(int socket, char *msg);

char *readTcp(int socket);

int closeTcpSocket(int socket);

#endif
```

6.2.5 clientTCP.c

```
/*      (C)2000 FEUP */
#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 <string.h>
#include <stdio.h>
#include <errno.h>
#include "clientTCP.h"

char *getIP(char *hostname)
{
   struct hostent *h;

   if ((h = gethostbyname(hostname)) == NULL)
   {
      perror("gethostbyname");
      return NULL;
   }

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

   return inet_ntoa(*((struct in_addr *)h->h_addr));
}
char *readTcp(int socket)
{
   char *msg = malloc(sizeof(char));
   int bytes;
   char buffer;
   char line[256] = "";
   int state = 0;
   int count = 0, msgLength = 0;

   /* read server response */
   do
```

```
{
   bytes = recv(socket, &buffer, sizeof(buffer), 0);

   if(bytes == -1){
      printf("Error on reading TCP\n");
      return NULL;
   }
   line[count++] = buffer;

   // copy buffer to final message
   msg = realloc(msg, sizeof(msg) + sizeof(char) * (msgLength +
       1));
   msg[msgLength++] = buffer;

   switch (state)
   {
   case 0:
      if (buffer == '\r')
      {
         state = 1;
      }
      break;
   case 1:
      if (buffer == '\n')
      {
         if (line[3] == ' ')
         {
            state = 2;
         }
         else
         {
            state = 0;
            strcpy(line, "");
            count = 0;
         }
      }
      else
      {
         state = 0;
      }
      break;
   default:
      break;
   }
```

```
   } while (state != 2);

   return msg;
}

int openTcpSocket(char *hostname, int port)
{
   int tcpSocket = 0;
   struct sockaddr_in server_addr;

   // get ip from hostname
   char *ip = getIP(hostname);

   if (ip == NULL)
   {
      return -1;
   }

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

   /*open an TCP socket*/
   if ((tcpSocket = socket(AF_INET, SOCK_STREAM, 0)) < 0)
   {
      perror("socket()");
      return -1;
   }
   else
   {
      printf("TCP socket opened to %s:%d!\n\n", ip, port);
   }

   /*connect to the server*/
   if (connect(tcpSocket, (struct sockaddr *)&server_addr,
       sizeof(server_addr)) < 0)
   {
      perror("connect()");
      return -1;
   }
```

```
   return tcpSocket;
}

int writeTcp(int socket, char *msg)
{
   int bytes;

   strcat(msg, "\r\n");

   /* send a string to the server */
   bytes = send(socket, msg, strlen(msg) * sizeof(char), 0);

   printf("%s", msg);

   /* read response */
   return bytes;
}

int closeTcpSocket(int socket)
{
   return close(socket);
}
```