Redes de Computadores

2.º Trabalho Prático - Rede de Computadores

Gonçalo Teixeira e Gonçalo Alves

Mestrado Integrado em Engenharia
Informática e Computação

20 de dezembro de 2020
Porto

Sumário

Sumário aqui -> dizer para que é que serve este relatório

Conteúdo

Introdução

Sumário aqui -> introduzir o tema
Dizer do que se trata a aplicação download
Dizer do que se trata a configurar uma rede de computadores

Parte I - Aplicação Download

Pequeno texto acerca da aplicação download

Arquitetura

Arquitetura da Aplicação Download

Resultados

Pequeno texto acerca de como foi testada a aplicação e um log de resultados

Parte II - Configuração e Análise de Rede

Pequeno texto acerca da rede a configurar

Experiência 1 - Configuração IP de Rede

Explicar sucintamente o objetivo desta experiência.

1. 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 IP numa LAN (*Local Area Network*). O endereço da camada de ligação é também conhecido por Endereço MAC (*Media Access Control*).

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

Executando o comando *ping* do tux3 para o tux4, o tux3 envia uma pergunta para saber qual é o endereço MAC associado ao IP do tux4. A "pergunta"é feita através de um pacote ARP que contém o endereço IP e MAC do tux3 (`172.16.30.1` e `00:21:5a:5a:7d:74`) e o endereço IP do tux4 (`172.16.1.254`), uma vez que se quer descobrir o endereço MAC do tux4, o campo dedicado a esse efeito está a `00:00:00:00:00:00`. De seguida é enviada uma resposta, também sob a forma de um pacote ARP, do tux4 para o tux3, indicando o seu endereço MAC (`00:21:5a:5a:7d:74`). Figura 1

3. Quais os pacotes gerados pelo comando ping?

Primeiro o comando *ping* gera pacotes ARP para fazer a relação entre endereços IP e MAC, de seguida gera pacotes ICMP (*Internet Control Message Protocol*).

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

Quando se executa o comando *ping* no tux3 para o tux4, os endereços (IP e MAC) vão ser os endereços dos tux. Podemos ver de seguida os endereços registados nos pacotes de pedido e reposta, respetivamente.

	tux	MAC	IP
Origem	3	00:21:5a:61:24:92	172.16.30.1
Destino	4	00:21:5a:5a:7d:74	172.16.30.254

Tabela 1: Pacote de Pedido

	tux	MAC	IP
Origem	4	00:21:5a:5a:7d:74	172.16.30.254
Destino	3	00:21:5a:61:24:92	172.16.30.1

Tabela 2: Pacote de Resposta

Devem-se consultar as figuras 2 e 3 para referência.

5. Como determinar a trama recetora Ethernet é ARP, IP, ICMP?

O *Ethernet Header* de um pacote contém a informação acerca do tipo da trama. Para as tramas IP, o valor do tipo será `0x0800`, se o *IP Header* tiver o valor 1 então o tipo de protocolo é ICMP. Para as tramas ARP o valor do tipo será `0x0806`.
Para referência devem-se consultar as figuras 4 e 5.

6. Como determinar o comprimento de uma trama recetora?

O comprimento de uma trama recetora pode ser determinado inspecionando a entrada no registo do *Wireshark*, tal como se pode observar na figura 6.

Conclusão

Conclusão aqui

Referências

Referências aqui

Anexos

Imagens

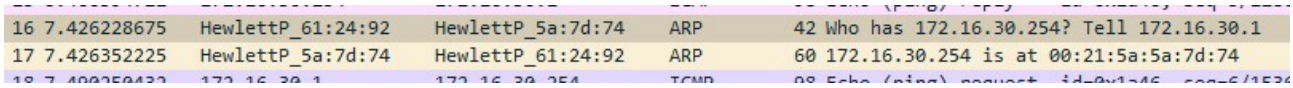

Figura 1: Pacotes ARP

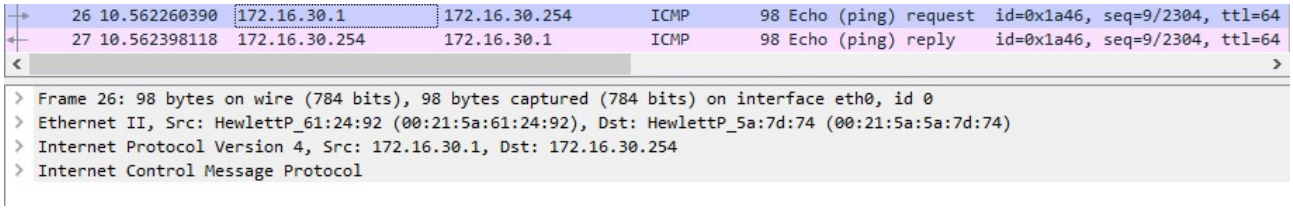

Figura 2: Pacote de Pedido

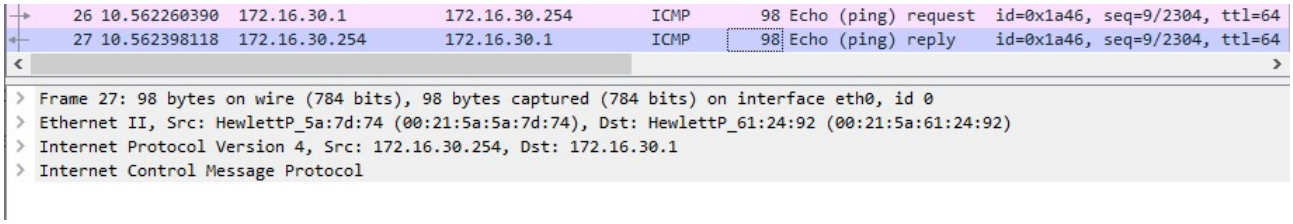

Figura 3: Pacote de Resposta

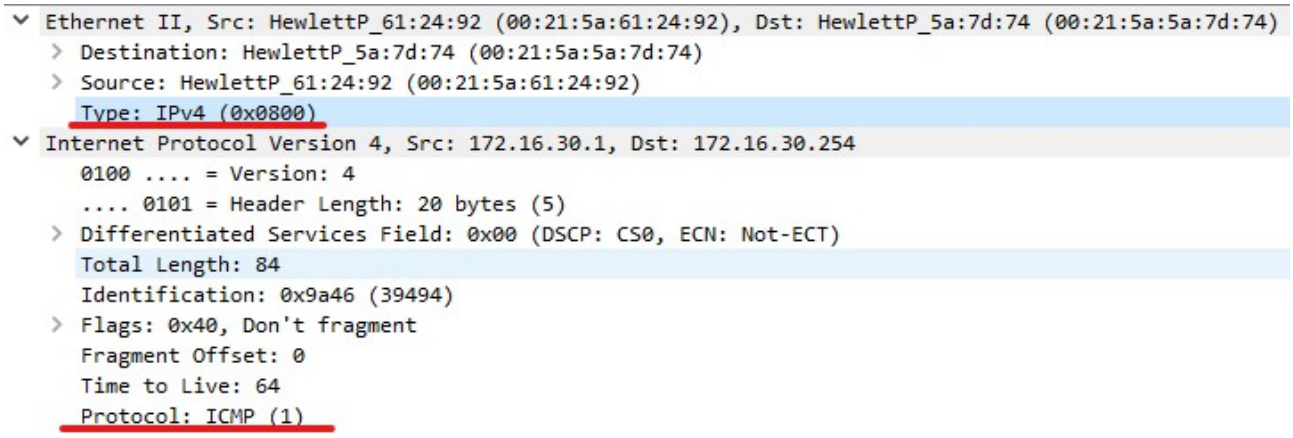

Figura 4: Campo Type Pacote ICMP

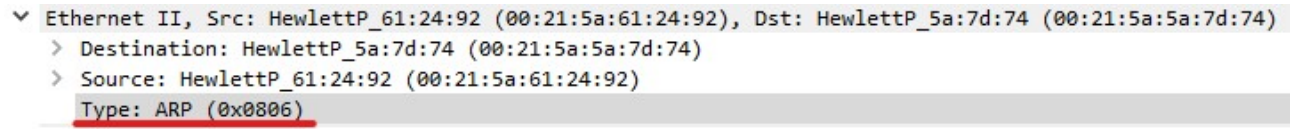

Figura 5: Campo Type Pacote ARP

Figura 6: Tamanho de uma trama Recetora

Aplicação Download

download.c

```c
/*      (C)2000 FEUP  */

#include "connection.h"
#include "getip.h"

#define SERVER_PORT 6000
#define SERVER_ADDR "192.168.28.96"


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

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

    char user[1024], password[1024], host[1024], url_path[1024], *ip;

    parseArg(argv[1], user, password, host, url_path);
    printArg(user, password, host, url_path);
    ip = getIP(host);
    printf("IP - %s\n", ip);
    if (ftp_init_connection(ip) == -1)return -1;
    if (ftp_login(user, password) == -1)return -1;
    if (ftp_download(url_path) == -1) return -1;

    return 0;
}

void printArg(char *user, char *password, char *host, char *url_path) {
    printf("User - %s\n", user);
    printf("Password - %s\n", password);
    printf("Host - %s\n", host);
    printf("URL - %s\n", url_path);
}

// ./download ftp://user:1234@sftp.up.pt/pub/ficheiro.zip
void parseArg(char *arg, char *user, char *password, char *host, char *url_path) {


    char *args = strtok(arg, "/");
    args = strtok(NULL, "/:");
    strcpy(user, args);

    args = strtok(NULL, "@");
    strcpy(password, args);

    args = strtok(NULL, "/");
    if (args == NULL) {
        printf("No User\nSetting Default- anonymous\n");
        strcpy(host, user);
        strcpy(user, "anonymous");
    } else {
        strcpy(host, args);
    }

    args = strtok(NULL, "\0");
    if (args == NULL) {
```

```
        printf("No Password\nSetting Default- 1234\n");
        strcpy(url_path, password);
        strcpy(password, "1234");
    } else {
        strcpy(url_path, args);
    }

}
```

getip.h

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include<arpa/inet.h>

char *getIP(char host[]);
```

getip.c

```

#include "getip.h"

char *getIP(char host[]) {
    struct hostent *h;

/*
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(host)) == NULL) {
        herror("gethostbyname");
        exit(1);
    }
    char *IP = 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 IP;
}
```

connection.h

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include<arpa/inet.h>

char *getIP(char host[]);
```

connection.c

```

#include "getip.h"

char *getIP(char host[]) {
    struct hostent *h;

/*
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(host)) == NULL) {
        herror("gethostbyname");
        exit(1);
    }
    char *IP = 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 IP;
}
```

Makefile

```
CC = gcc
CFLAGS = -Wall -pthread -Wno-pointer-sign -g
DEPS = connection.h getip.h
OBJ = connection.o getip.o
TARGETS = download

all: download

%.o: %.c $(DEPS)
  @$(CC) $(CFLAGS) -c -o $@ $<
  @echo $@

download: $(OBJ)
  @$(CC) $(CFLAGS) -o $@ $@.c $(OBJ) -lm
  @echo $@

clean:
  @rm *.o $(TARGETS)
```