Redes de Computadores

2º Trabalho - Rede de computadores

Nuno Miguel Fernandes Marques - 201708997 - MIEIC

December 22, 2020

Resumo

O segundo trabalho laboratorial de Redes de Computadores consistiu no desenvolvimento de uma aplicação de download usando o protocolo FTP, e numa série de experiências em configuração de redes.

Aplicação de Download

- **Arquitectura**
  A arquitectura da aplicação consiste nos seguintes elementos:
  - Parse e processamento do url dado como argumento do programa em utils.c
  - E aberta uma conexão TCP e feito o login com os dados processados no primeiro passo, com consideração do utilizador anonimo.
  - A conexão é colocada em modo passivo.
  - Outra conexão é aberta, a conexão original envia um comando do tipo "retr".
  - O ficheiro é criado e a segunda conexão transmite os dados que são guardados no ficheiro.
- **Resultados**
  O programa foi testado em modo anónimo, não anónimo, com vários URLs e diferentes tamanhos de ficheiro com sucesso. Foi também testado o processamento de erros em casos como o ficheiro não ser encontrado, o login falhar, etc.

Configuração e análise de rede

- **Experiência 1** - Configurar uma rede IP
  - Tux 3 - MAC: 00:21:5a:61:2d:df IP: 172.16.60.1
  - Tux 4 - MAC: 00:21:5a:5a:79:97 IP: 172.16.60.254

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

     APR(Addresss Resolution Protocol) é uma protocolo de comunicação que é usado para descobrir um endereço físico (MAC) associado a um endereço IP.

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

   É enviado um pacote ARP do tux3 a pedir o MAC do tux4. Este pacote apresenta o IP e MAC do tux3(origem), e IP do tux4 com MAC de destino "00:00:00:00:00:00", Visto que, inicialmente, o MAC do tux4 não é conhecido. De seguida o tux3 recebe um pacote que da a conhecer o MAC do tux4 e vice-versa. Estes pacotes estão demonstrados na figura 1 e 2 dos anexos.

3. Que pacotes são gerados pelo comando **Ping**?
   O Ping gera pacotes ARP e ICMP.
4. Quais são os endereços **IP e MAC** dos pacotes **Ping**?

   Pacote de pedido apresenta o IP e MAC de origem do tux3 e IP e MAC de destino do tux4. Pacote de resposta apresenta o IP e MAC de origem do tux4 e IP e MAC de destino do tux3. Pode ser observado na figura 3 e 4.
5. Como determinar se a trama Ethernet recebida é APR, IP, ICMP?

   No campo **type** do header da trama de Ethernet é apresentada a informação se a trama é do tipo IP ou ARP. No header do IP tiver o valor 1 a trama é do tipo ICMP. Pode ser observado na figura 5.
6. Como determinar o tamanho de uma trama recebida?

   No **wireshark** podemos observar o tamanho da trama no campo **frame length**, pode ser observado na figura 5.
7. O que é o **loopback interface** e qual a sua importância?

   A loopback interface é uma interface virtual de rede que permite ao computador comunicar com ele próprio com objectivo de realizar testes diagnósticos.

- **Experiência 2** - Implementar duas **VLANs** num **switch**

  1. Como configurar a vlan60?

     Através da porta de série ligada ao switch são enviados os seguintes comandos para criar a vlan:
     - conf t
     - vlan 60
     - exit

     Para adicionar portas a vlan são usados os seguintes comandos:
     - conf t
     - interface fastethernet 0/[PORTA]
     - switchport mode access
     - switchport access vlan60
     - end
  2. Quantos domínios de transmissão existem? O que se pode concluir das capturas?

     Existem dois domínios visto que o tux3 recebe resposta do tux4 mas não recebe resposta do tux2. O tux2 não recebe qualquer resposta. Logo existe um domínio com o tux3 e tux4, e outro com o tux2.

- **Experiência 3** - Configurar um **router** em **Linux**

  1. Que rotas existem nos **tuxes**? O que significam?

     O tux3 tem uma rota para o tux4 na porta eth0(vlan0), e o tux2 tem uma rota para o tux4 na porta eth1(vlan1). Estas rotas permitem o tux3 comunicar com o tux2 através do tux4.
  2. Que informação está contida em uma entrada da tabela de **forwarding**?

     Destino, gateway(rota), máscara de sub-rede, interface(Porta Ethernet), metric(custo), flags.

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

   É observado um pacote ARP a pedir o endereço MAC do eth0 do tux4 pelo tux3, e um pacote a pedir o endereço MAC do tux2 pelo eth1 do tux4. Isto porque o pacote é enviado do tux3 para o eth0 do tux4, e então é enviado para o tux2 via eth1 do tux4. Observado na figura 6.

4. Que pacotes **ICMP** são observados e porquê?

   São observados pacotes request e reply visto que tux2, tux3 e tux4 agora conseguem comunicar.

5. Que endereços **IP e MAC** estão associados aos pacotes **ICMP** e porquê?
   Para os requests são observados:
   - tux4 eth0
     Origem - IP e MAC do tux3
     Destino - IP do tux2 e MAC do eth0 do tux4
   - tux4 eth1
     Origem - IP do tux3 e MAC do eth1 do tux4
     Destino - IP e MAC do tux2

   O MAC de destino do pacote enviado pelo tux3 e o eth0 do tux4, visto que o pacote tem como destino o tux2 mas e encaminhado pelo tux4. No tux4 o pacote e encaminhado para o tux2 mas visto ser enviado via porta eth1 tem como MAC a mesma. Nos replies o processo é o mesmo mas do tux2 para o tux4, que envia para o tux3. Observado na figura

- **Experiência 4** - Configurar um **router** comercial e implementar NAT

1. Como configurar uma **rota estática** num **router comercial**?
   Através da porta de série ligada ao router são enviados os seguintes comandos para criar a rota estática:
   - conf t
   - ip route [rota de destino] [máscara] [gateway]
   - exit

2. Que caminhos seguiram os pacotes na experiências feitas e porquê?

   Os pacotes seguem a rota se existir, senão os pacotes vão para o router default, o router informa para enviar pelo router tux4.

3. Como configurar o **NAT** num **router comercial**?

   O NAT e configurado ao criar uma pool de nat e uma lista de acesso a essa pool. Depois são adicionadas as redes desejadas a essa lista de acesso e são criadas as rotas.

4. Qual é a funcionalidade do **NAT**?

   NAT é responsável pela tradução de endereços de rede, permitindo uma rede interna ligar a uma rede exterior, como a internet, precisando apenas de um endereço IP na rede exterior.

- **Experiência 5** - Configurar DNS

1. Como configurar um serviço **DNS** num **host**?
   O serviço DNS em Linux pode ser configurado editando o ficheiro /etc/resolv.conf com a informação:
   - search [hostname]
   - nameserver [IP do servidor]

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

   Um pacote é enviado do tux3 para o servidor DNS que, contém o hostname, a pedir o respectivo endereço de IP. O servidor de DNS responde com o IP do hostname pedido.

- **Experiência 6** - Conexões TCP

1. Quantas conexões **TCP** são abertas pela aplicação de **FTP**?

   A aplicação FTP abre duas conexões de TCP, uma para enviar comandos e outra para receber dados do servidor.

2. Em que conexão é transportada a informação de controlo de **FTP**?

   Na conexão aberta para a troca de comandos com o servidor.

3. Quais são as fases de uma conexão **TCP**?

   Estabelecimento de conexão, troca de dados, encerramento de conexão.

4. Como funciona o mecanismo **ARQ TCP**? Quais são os campos **TCP** relevantes? Que informação relevante pode ser observada nas capturas?

   Com o mecanismos ARQ, ao receber um pacote o receptor envia mensagens do tipo 'ACK' para confirmar a recepção, caso o transmissor não receber essa mensagem após um determinado limite de tempo o pacote é reenviado, servindo para controlar erros e o fluxo de transmissão. Os campos relevantes são o 'acknowledgment number', 'window size' e 'sequence number'. Esta informação pode ser vista na captura da figura 8.

5. Como funciona o mecanismo de controlo de congestão **TCP**? Como evolui o fluxo de dados ao longo do tempo? É de acordo com o mecanismo de congestão **TCP**?

   Para cada conexão TCP é mantida uma janela de congestão que determina o valor de bytes que pode ser enviado a cada momento. Para cada segmento recebido e confirmado a janela é aumentada. O fluxo aumenta até ser ligada uma segunda conexão, a partir desse momento a direção do fluxo varia, que é de acordo com o mecanismo de congestão.

6. O fluxo de dados de uma conexão **TCP** é perturbada pelo aparecimento de uma segunda conexão **TCP**? Como?
   Sim, o aparecimento de uma segunda conexão pode levar a quedas na taxa de transmissão.

Conclusões

O segundo trabalho laboratorial ajudou a ganhar conhecimentos sobre protocolos de comunicação e configurações de redes foi um trabalho com desafios, devido também ao regime COVID, mas no fim foi completo com sucesso.

Anexos

”utils.h”

```
#ifndef UTILS_H
#define UTILS_H


void abort_bad_url();
int chop_string(char *string, char **result, char delim);
void parseArguments(int argc, char *argv[], char **user, char**
     password, char **server, char **filepath, char **filename)
    ;
void cleanup(char *user, char* password, char *server, int
    socket_fd, int data_socket_fd);

#endif
```

”utils.c”

```
#ifndef UTILS_H
#define UTILS_H
#include "utils.h"

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

#define FTP_COMMAND_MIN_SIZE 12
#define FTP_PREFIX "ftp://"
#define FTP_PREFIX_SIZE 6

void abort_bad_url(){
    fprintf(stderr, "Invalid URL\n");
    exit(-1);
}

int chop_string(char *string, char **result, char delim){
  int size =0;
  const int string_length = strlen(string);

  bool delim_found = false;
  for (int i = 0; i < string_length; ++i){
    if (string[i] == delim){
      delim_found = true;
      size = i;
      break;
```

```
    }
  }
  if (!delim_found)
    return -1;

  *result = (char *)malloc((size + 1)*sizeof(char));
  memcpy(*result, string, size);
  result[size] = 0;
  return size;
}

void parseArguments(int argc, char *argv[], char **user, char**
    password, char **server, char **filepath, char **filename)
   {

  if (argc < 2){
    abort_bad_url();
  }

  char * ftp_command = argv[1];

  if (strlen(ftp_command) < FTP_COMMAND_MIN_SIZE)
    abort_bad_url();

  // Verify and advance ftp prefix
  if (strncmp(ftp_command, FTP_PREFIX, FTP_PREFIX_SIZE) != 0)
    abort_bad_url();
  ftp_command += FTP_PREFIX_SIZE * sizeof(char);

  // Read username
  int res = chop_string(ftp_command, user, ':');
  if (res == -1){
    *user = (char *)malloc(10 * sizeof(char));
    memcpy(*user, "anonymous", 9);
  }
  ftp_command += res + 1;

  // Read password
  res = chop_string(ftp_command, password, '@');
  if (res == -1){
    *password = (char *)malloc(2 * sizeof(char));
    memcpy(*password, "", 1);
  }
  ftp_command += res + 1;

  // Read server name
  res = chop_string(ftp_command, server, '/');
  if (res == -1)
    abort_bad_url();
```

```
  ftp_command += res + 1;

  //Read file name
  *filename = basename(ftp_command);
  *filepath = dirname(ftp_command);

}

void cleanup(char *user, char* password, char *server, int
    socket_fd, int data_socket_fd){
  close(socket_fd);
  close(data_socket_fd);
   if (server != NULL)
    free(server);
   if (password != NULL)
    free(password);
   if (user != NULL)
    free(user);
}
```

"ftp_app.h"

```
#ifndef FTP_APP_H
#define FTP_APP_H

#include <stdbool.h>

int ftp_connect(char *server);
int ftp_connect_socket(char *ip, int port);
void ftp_login(int socket_fd, char * username, char* password);
void ftp_enter_passive_mode(int socket_fd, char *client_ip, int
     *client_port);
void ftp_retrieve_file(int socket_fd, char *filepath, char *
    filename);
void ftp_download(int data_fd, char * filename);

#endif
```

"ftp_app.c"

```
#include "ftp_app.h"

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

```
#include <string.h>
#include <unistd.h>

#define FTP_PORT_NUMBER 21
#define FTP_RETURN_CODE_SIZE 4
#define FTP_CODE_CONFIRM "2"
#define FTP_CODE_NEED_PASSWORD "331"
#define FTP_CODE_LOGIN_SUCCESS "230"
#define FTP_CODE_PASSIVE_MODE "227"
#define FTP_BUFFER_SIZE 30000

bool read_return_code(int socket_fd, char expected_return[]){

    char code[FTP_RETURN_CODE_SIZE];

    if (expected_return == NULL)
        return false;

    do {
        memset(&code, 0, FTP_RETURN_CODE_SIZE);
        read(socket_fd, &code, FTP_RETURN_CODE_SIZE);
    } while (code[3] != ' ' || code[0] < '1' || code[0] > '5');

    printf("received return code: %s\n", code);

    return !strncmp(code, expected_return, strlen(
    expected_return));
}

bool send_command(int socket_fd, char *command, char*
    expected_return_code){

    write(socket_fd, command, strlen(command));

    if (expected_return_code != NULL)
        return read_return_code(socket_fd, expected_return_code
    );

    return true;
}

int ftp_connect(char *server){

    struct hostent *h = gethostbyname(server);;
    if (h == NULL)
        exit(-1);

    /*get ip address*/
    printf("\nHost name  : %s\n", h->h_name);
```

```
58      char *ip_addr = inet_ntoa(*((struct in_addr *)h->h_addr));
59
60      int socket_fd = ftp_connect_socket(ip_addr, FTP_PORT_NUMBER
        );
61
62      if (!read_return_code(socket_fd, FTP_CODE_CONFIRM)){
63          fprintf(stderr, "unexpected FTP return code\n");
64          exit(-1);
65      }
66
67      return socket_fd;
68  }
69
70  int ftp_connect_socket(char *ip, int port){
71      printf("IP Address : %s\n", ip);
72      printf("connecting...\n");
73
74      /*server address handling*/
75      struct  sockaddr_in server_addr;
76      bzero((char*)&server_addr,sizeof(server_addr));
77      server_addr.sin_family = AF_INET;
78      server_addr.sin_addr.s_addr = inet_addr(ip);  /*32 bit
        Internet address network byte ordered*/
79      server_addr.sin_port = htons(port);   /*server TCP port
        must be network byte ordered */
80
81      /*open a TCP socket*/
82      int socketfd = socket(AF_INET,SOCK_STREAM,0);
83      if (socketfd < 0)
84          exit(-1);
85
86      /*connect to the server*/
87      if(connect(socketfd, (struct sockaddr *)&server_addr,
        sizeof(server_addr)) < 0){
88          perror("connect()");
89      exit(-1);
90    }
91
92      printf("connected.\n");
93
94      return socketfd;
95  }
96
97  void ftp_login(int socket_fd, char * username, char* password){
98
99  if (username == NULL || password == NULL)
100     exit(-1);
101
102 printf("\nloging in as user %s\n", username);
```

```
103
104 char usr_cmd[strlen(username) + 8];
105 sprintf(usr_cmd, "USER %s\r\n", username);
106
107 if (!send_command(socket_fd, usr_cmd, FTP_CODE_NEED_PASSWORD)){
108     fprintf(stderr, "login failed.\n");
109     exit(-1);
110 }
111
112 char pass_cmd[strlen(password) + 8];
113 sprintf(pass_cmd, "PASS %s\r\n", password);
114
115 if (!send_command(socket_fd, pass_cmd, FTP_CODE_LOGIN_SUCCESS))
    {
116     fprintf(stderr, "login failed.\n");
117     exit(-1);
118 }
119
120 printf("login sucessful\n");
121 }
122
123 void ftp_enter_passive_mode(int socket_fd, char *client_ip, int
     *client_port){
124     printf("\nentering passive mode\n");
125
126     if (client_ip == NULL){
127         fprintf(stderr, "failed to enter passive mode\n");
128         exit(-1);
129     }
130
131     send_command(socket_fd, "PASV\r\n", NULL);
132
133
134
135     char return_code[256];
136     do {
137         memset(&return_code, 0, 256);
138         read(socket_fd, &return_code, 256);
139     } while (return_code[3] != ' ' || return_code[0] < '1' ||
    return_code[0] > '5');
140
141
142     if (strncmp(return_code, FTP_CODE_PASSIVE_MODE, strlen(
    FTP_CODE_PASSIVE_MODE)) != 0){
143         fprintf(stderr, "failed to enter passive mode\n");
144         printf("received passive mode data: %s\n", return_code)
    ;
145         exit(-1);
146     }
```

```
147
148   char* client_info = strchr(return_code, '(');
149     int ip_values[6];
150   sscanf(client_info, "(%d, %d, %d, %d, %d, %d)", &ip_values
        [0],&ip_values[1],&ip_values[2],&ip_values[3],&ip_values
        [4],&ip_values[5]);
151   sprintf(client_ip, "%d.%d.%d.%d", ip_values[0],ip_values[1],
        ip_values[2],ip_values[3]);
152   *client_port = ip_values[4]*256+ip_values[5];
153
154     printf("entered passive mode: %s:%d\n", client_ip, *
        client_port);
155 }
156
157 void ftp_retrieve_file(int socket_fd, char *filepath, char *
        filename){
158
159     if (!send_command(socket_fd, "TYPE L 8\r\n",
        FTP_CODE_CONFIRM) || filepath == NULL
160             || filename == NULL){
161         fprintf(stderr, "failed to retrieve file\n");
162         exit(-1);
163     }
164
165     char cmd[8 + strlen(filename) + strlen(filepath)];
166     sprintf(cmd, "RETR %s/%s\r\n", filepath, filename);
167   send_command(socket_fd, cmd, NULL);
168 }
169
170 void ftp_download(int data_fd, char * filename){
171     FILE *fp = fopen(filename, "w");
172     if( fp == NULL ){
173         fprintf(stderr, "failed to open file\n");
174         exit(-1);
175     }
176
177     printf("\nStarting download\n");
178     char buffer[FTP_BUFFER_SIZE];
179     int prog = 0;
180     while(true){
181         ++prog;
182         int byte_count = read(data_fd, buffer, FTP_BUFFER_SIZE)
        ;
183         if (byte_count == 0)
184             break;
185         fwrite(buffer, byte_count, 1,  fp);
186         printf("\rdownloading");
187         for (int i = 0; i < prog % 5; ++i)
188             printf(".");
```

```
    }

    fclose(fp);
    printf("\ndownload finished\n\n");
}
```

"main.c"

```
#include "ftp_app/ftp_app.h"
#include "utils/utils.h"

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

  char *username, *password, *server, *filepath, *filename;
  parseArguments(argc, argv, &username, &password, &server, &
    filepath, &filename);

  int socket_fd = ftp_connect(server);

  ftp_login(socket_fd, username, password);

  char client_ip[16];
  int client_port;
  ftp_enter_passive_mode(socket_fd, client_ip, &client_port);

  int data_socket_fd = ftp_connect_socket(client_ip,
    client_port);
  ftp_retrieve_file(socket_fd, filepath, filename);
  ftp_download(data_socket_fd, filename);

  cleanup(username, password, server, socket_fd, data_socket_fd
    );
  return 0;
}
```

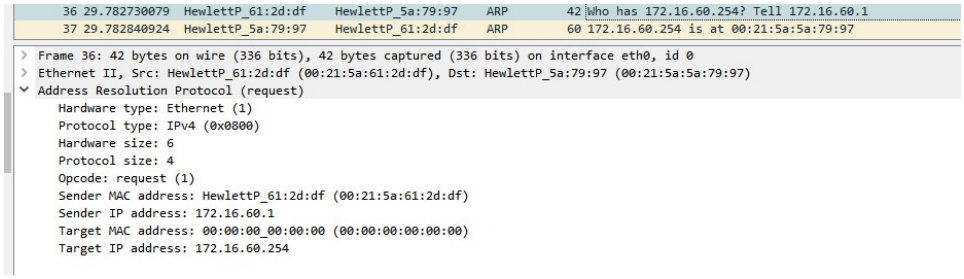

figura 1

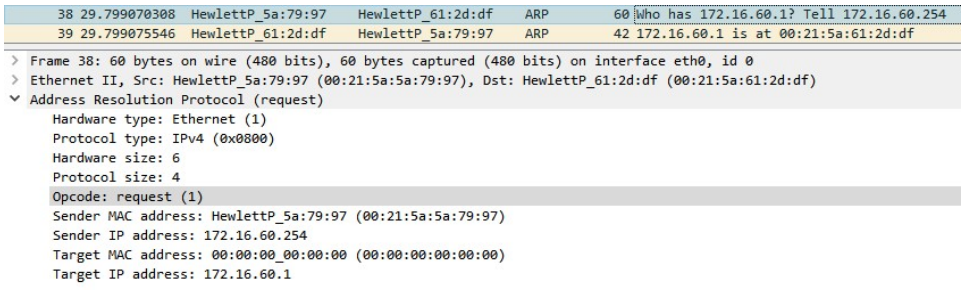

figura 2

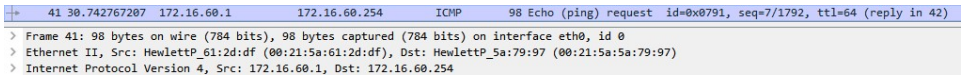

figura 3

figura 4

```
Type: ARP (0x0806)   Type: IPv4 (0x0800)
```

```
Internet Protocol Version 4, Src: 172.16.60.1, Dst: 172.16.60.254
   0100 .... = Version: 4
   .... 0101 = Header Length: 20 bytes (5)
 > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
   Total Length: 84
   Identification: 0xb2f1 (45809)
 > Flags: 0x40, Don't fragment
   Fragment Offset: 0
   Time to Live: 64
   Protocol: ICMP (1)
```

```
Frame Length: 98 bytes (784 bits)
```

figura 5

tux4 - eth0

71	92.649902004	HewlettP_61:2d:df	HewlettP_5a:79:97	ARP	60 Who has 172.16.60.254? Tell 172.16.60.1
72	92.649919674	HewlettP 5a:79:97	HewlettP 61:2d:df	ARP	42 172.16.60.254 is at 00:21:5a:5a:79:97
75	92.825746271	HewlettP_5a:79:97	HewlettP_61:2d:df	ARP	42 Who has 172.16.60.1? Tell 172.16.60.254
76	92.825866188	HewlettP_61:2d:df	HewlettP_5a:79:97	ARP	60 172.16.60.1 is at 00:21:5a:61:2d:df

tux4 - eth1

106	90.568788139	Netronix_71:73:da	HewlettP_19:02:ba	ARP	42 Who has 172.16.61.1? Tell 172.16.61.253
107	90.568877606	HewlettP_19:02:ba	Netronix_71:73:da	ARP	60 172.16.61.1 is at 00:22:64:19:02:ba
108	90.572288502	HewlettP_19:02:ba	Netronix_71:73:da	ARP	60 Who has 172.16.61.253? Tell 172.16.61.1
109	90.572295206	Netronix_71:73:da	HewlettP_19:02:ba	ARP	42 172.16.61.253 is at 00:08:54:71:73:da

figura 6

78	93.737868177	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request id=0x10e7, seq=7/1792, ttl=64 (reply in 79)
79	93.738016170	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply id=0x10e7, seq=7/1792, ttl=63 (request in 78)

```
> Frame 78: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0
> Ethernet II, Src: HewlettP_61:2d:df (00:21:5a:61:2d:df), Dst: HewlettP_5a:79:97 (00:21:5a:5a:79:97)
> Internet Protocol Version 4, Src: 172.16.60.1, Dst: 172.16.61.1
> Internet Control Message Protocol
```

118	94.552759850	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request id=0x10e7, seq=10/2560, ttl=63 (reply in 119)
119	94.552899742	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply id=0x10e7, seq=10/2560, ttl=64 (request in 118)

```
> Frame 99: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth1, id 0
> Ethernet II, Src: Netronix_71:73:da (00:08:54:71:73:da), Dst: HewlettP_19:02:ba (00:22:64:19:02:ba)
> Internet Protocol Version 4, Src: 172.16.60.1, Dst: 172.16.61.1
> Internet Control Message Protocol
```

figura 7

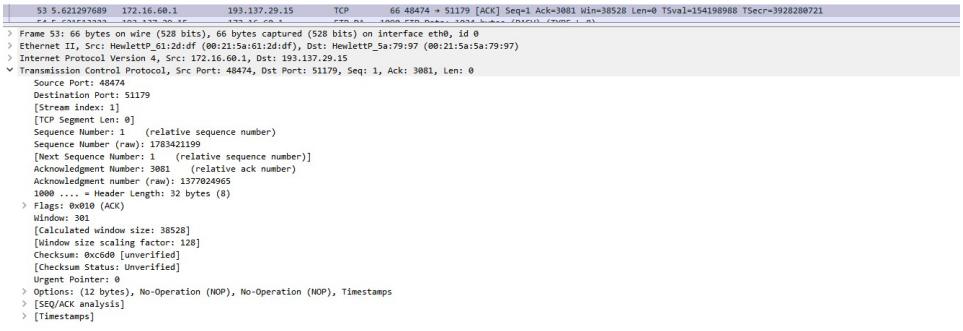

figura 8

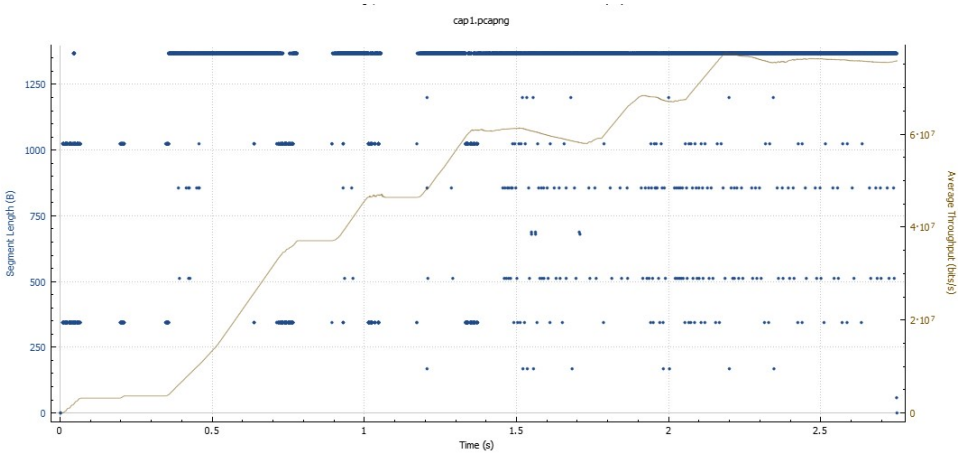

figura 9