Aplicação FTP + Configuração de uma Rede em Laboratório

Relatório do 2º Projeto de Redes de Computadores

Licenciatura em Engenharia Informática e Computação

Turma 7, Grupo 1

Ana Matilde Guedes Perez da Silva Barra (up20194795)
Sérgio Miguel Rosa Estêvão (up201905680)
Sérgio Rodrigues da Gama (up201906690)

24 de Janeiro 2021

Conteúdo

1 Sumário

No âmbito da disciplina de Redes de Computadores, do 3ºano da Licenciatura em Engenharia Informática e Computadores, foi-nos proposto o desafio de elaborar um conjunto de experiências em laboratório e uma aplicação para download de ficheiros a partir de um servidor, apoiada no **FTP - *File Transfer Protocol***.

A realização deste projeto permitiu uma melhor compreensão do lecionado nas aulas teóricas, dado que através da interação direta com o *hardware* observamos a sua influência no software e no sucesso da nossa aplicação.

2 Introdução

O objetivo deste trabalho era implementar uma rede de computadores em laboratório (no nosso caso, **sala I320**) que se ligava à internet através das configurações efetuadas nas aulas. Posteriormente, para verificar a sua eficiência, foi criada uma aplicação que utiliza o protocolo de transferência de ficheiros (**FTP**), desenvolvida autonomamente pelo nosso grupo.

Com este relatório pretendemos expor o trabalho desenvolvido ao longo das últimas semanas, numa perspetiva mais teórica, enfatizando os métodos utilizados para alcançar os objetivos propostos nos guiões de trabalho e também as conclusões alcançadas em cada um deles, tal como na globalidade.

3 Parte 1 - Aplicação de Download

A primeira parte deste projeto constituiu no desenvolvimento de uma aplicação de download de ficheiros através do protocolo *FTP* (*File Transfer Protocol*) com ligações *TCP* (*Transmission Control Protocol*) apartir de *sockets* e que consiga efetuar a transferência de ficheiros alojados em servidores *FTP*. A aplicação usa uma sintaxe de links do tipo:
`ftp://[<user>:<password>@]<host>/<url-path>`, onde estão presentes todos os dados necessários à execução da aplicação.

3.1 Arquitetura

O projeto é constituído por duas camadas, a camada de aplicação e a de abstração. A camada de aplicação é formada por todas as funções que fazem a comunicação projeto-servidor, já a camada de abstração funciona como suporte à de aplicação, gerindo o input e a conversão de dados.

3.1.1 Camada de Aplicação

Esta camada é responsável pela ligação e comunicação com o servidor. Inicialmente é estabelecida a ligação com o *socket* através do endereço *IP* obtido no *input* e a porta 21. Depois para esse socket são enviadas as credenciais e é ativado o modo passivo.

Em seguida o programa conecta-se ao *socket* de receção do ficheiro, utilizando o IP inicial e a porta retornada pela ativação do modo passivo, faz o pedido de *download* ao *socket* inicial e lê o ficheiro através do socket de receção. Após terminar o download, é encerrada a ligação com o socket inicial. É de notar que as respostas às mensagens enviadas ao servidor são verificadas para confirmar o seu sucesso. Esta camada está representada no ficheiro *download.c*, estando as funções deste declaradas no ficheiro *client.c*.

3.1.2 Camada de Abstração

Esta camada trata do processamento do *input*, sendo usada no princípio do programa para converter o *url* fornecido nos dados necessários à execução do programa. A conversão é feita através de uma *state machine* que divide o *url* e constrói um objeto da *struct url_data*.

As funções desta camada estão declaradas no ficheiro *input.c* e são usadas no *download.c*.

3.1.3 Estrutura de Dados

Para guarda a informação do *url* passado como argumento foi utilizada a estrutura de dados *url_data*, estando esta declarada no ficheiro **input.h**.

```
typedef struct{
    char url[5120];
    char user[1024];
    char password[1024];
    char host[1024];
    char url_path[1024];
}url_data;
```

3.2 Casos de Sucessos de Download

A compilação da aplicação pode ser feita a partir da execução do ficheiro **make**, para executar a aplicação utiliza-se o comando `./download ftp://[<user>:<password>@]<host>/<url-path>`. Os casos de teste estão representados nas figuras abaixo.

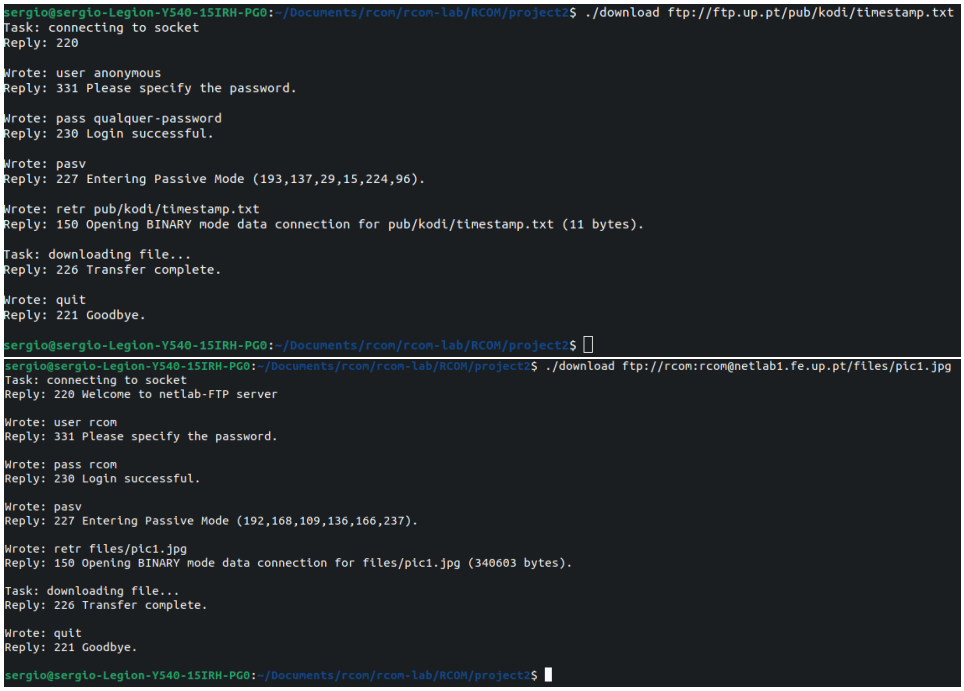

Fig. 1 e 2 - Download anónimo (1) e com credenciais (2)

4 Parte 2 - Configuração da Rede e Análise

Todas as experiências foram realizadas na Bancada 2 da sala I320 da FEUP. Com isto, foram editadas as imagens para apresentar os números Y e W corretos.

4.1 Experiência 1 - *IP Config*

4.1.1 Objetivos da experiência

Esta experiência assentava sobre a configuração de dois computadores, por via de um dispositivo intermédio, neste caso um ***Switch***. Os dois

computadores utilizados são intitulados, respetivamente, de **tux3** e **tux4**, omitiu-se o número de bancada de modo a simplificar o relatório (neste caso 2, e.g. tux23).

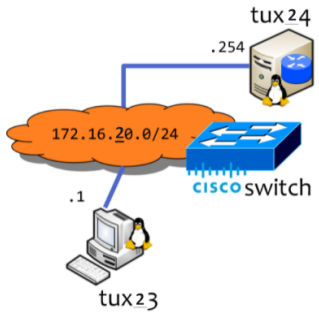

Fig. 3 - Aquitetura da *network* da Exp. 1

4.1.2 Configuração

Nesta secção encontram-se os comandos, cuja execução ordenada nos respetivos computadores leva a realização da experiência um. Acompanhada de cada um dos comandos está uma breve descrição da ação do mesmo. Para além disso, na coluna mais à esquerda revela-se o computador no(s) qual(ais) estes comandos são executados.

Comp.	**Descrição**	**Comando**
tux3 e 4	Desativar ambos os *network cards*	ifconfig eth0 down
tux3	Configurar o IP 172.16.20.1, com uma máscara de 24-bit	ifconfig eth0 up 172.16.20.1/24
tux4	Configurar o IP 172.16.20.254, com uma máscara de 24-bit	ifconfig eth0 172.16.20.254/24
tux4	Verificar a ligação com o tux3	ping 172.16.20.1
tux3 e 4	Verificar que a *network* foi configurada com sucesso	arp -a / route -n

4.1.3 Análise dos resultados

- O que são e para que são utilizados os pacotes **ARP**?

  O ARP é um protocolo utilizado para mapear dinamicamente endereços da **camada 3** (Camada de rede) para endereços da **camada 2** (Camada da ligação de dados), do modelo **OSI**. Deste modo, é usado para mapear endereços IP em endereços MAC.

- O que são os endereços **MAC** e **IP** dos pacotes **ARP**?

  Os pacotes ARP que são recebidos como resposta contêm ambos os endereços da camada 2 e 3 (OSI) dos dois intervenientes, tanto os do transmissor, como os do recetor (aqueles que se queriam efetivamente descobrir). Quando é enviado o pacote inicial, este obviamente ainda não contém o *placeholder* do endereço MAC do computador recetor preenchido.

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

  Este comando gera um pedido para um determinado host enviar pacotes do protocolo ICMP para o próprio transmissor.

- O que são os endereços **MAC** e **IP** dos pacotes de ping?

  Os pacotes ICMP têm ambos os endereços do transmissor e do recetor.

- Como determinar se o pacote *Ethernet* recebido é uma trama **ARP**, **IP**, ou **ICMP**?

  Todos os pacotes *Ethernet* têm um campo com um protocolo associado, sendo assim possível distinguir os diferentes tipos de pacotes.

- Como determinar o comprimento de uma trama recebida?

  Todas as tramas *Ethernet* têm um campo com o comprimento total.

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

  A interface de *loopback* é uma interface de rede virtual que permite que um cliente e um servidor no mesmo *host* comuniquem entre si. Esta é muito importante e útil para diagnóstico de problemas nos vários protocolos de rede, testes de software e hardware, conectividade e para conectar a servidores a rodar na própria máquina.

4.2 Experiência 2 - *Virtual LANs*

4.2.1 Objetivos da experiência

Os objetivos desta experiência eram configurar duas *Virtual LANs*:

- *VLAN* 20 - para estabelecer a comunicação entre os computadores *tux3* e *tux4*;

- *VLAN* 21 - conectada apenas ao *tux2*.

Como é possível observar na imagem seguinte, não existe qualquer conexão física entre as duas *VLANs*.

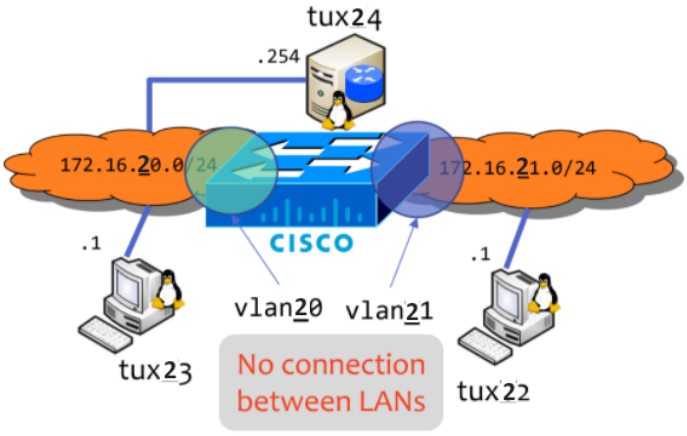

Fig. 4 - Arquitetura da *Network* da Exp. 2

4.2.2 Configuração

À semelhança da experiência anterior, apresentamos abaixo os comandos utilizados nesta experiência. Os comandos enviados para o ***Switch***, são feitos através do *tux3* (ligado a este atráves da porta SO). Estes comandos são escritos no *GTK Term*, sendo preciso fazer `enable` e inserir password, iniciando a ação '`configure terminal`' e terminando-a com '`end`'.

Comp.	Descrição	Comando
tux2	Configurar o IP 172.16.21.1, com máscara de 24-bit	ifconfig eth0 172.16.21.1/24
tux3	Criação da vlan 20	vlan 20
tux3	Adicionar a port 6 (ligação ao tux3) à Vlan20	interface fastethernet 0/6 switchport mode access switchport access vlan 20
tux3	Adicionar a port 8 (ligação ao tux4) à Vlan20	interface fastethernet 0/8 switchport mode access switchport access vlan 20
tux3	Verificar o resultado dos comandos executados	show vlan brief
tux3	Criação da vlan 21	vlan 21
tux3	Adicionar a port 4 (ligação ao tux2) à Vlan 21	interface fastethernet 0/4 switchport mode access switchport access vlan 21
tux3	Verificar se tudo está correto no final	show vlan brief

tux3 - representa comandos executados através do *GTK Term*.

O comando `show vlan brief` apresenta uma 'tabela' com as LANs existentes, onde se podem observar as criadas por nós (*vlan 20* e *vlan 21*).

4.2.3 Análise dos resultados

- Como configurar a **vlan20**?

  Como se pode constatar pelos comandos acima, é necessário apenas configurar o *Switch* para adicionar uma nova *vlan* e referir quais as *ports* conectadas a esta.

- Quantos são os domínios de ***broadcast*** aqui? Como concluímos isso a partir dos logs?

  Existem dois domínios de *broadcast*, a *vlan 20* e a *vlan 21*. Isto acontece pois estão isoladas, não tendo contacto uma com a outra.

  Com um **ping broadcast** no *tux3*, apenas o mesmo e o *tux4* recebe pacotes *ICMP*, *tux2* não recebe qualquer pacote. Por outro lado, com **ping broadcast** no *tux2*, *tux3* e o *tux4* não recebem pacotes *ICMP* mas o próprio recebe-os.

  Os *logs* analisados estão disponíveis no anexo deste relatório.

4.3 Experiência 3 - *Router Configuration*

4.3.1 Objetivos da experiência

Esta experiência foi realizada remotamente e, por isso, o seu objetivo era a preparação para a Experiência 4, a ser realizada em laboratório.

Foi realizada a análise de um ficheiro de configuração de um router Cisco e experimentadas configurações *DNS* e comandos de *Linux Routing*.

4.3.2 Análise dos resultados

Tendo em conta o modo de funcionamento e carácter desta experiência, apresentamos seguidamente apenas a resposta às perguntas do enunciado.

- (Cisco) Como é configurada uma *static route* num router comercial?

  Através de um comando `ip route`, p.e.:

  `ip route 172.16.40.0 255.255.255.0 12.16.30.2`,

  que ao ser executado a partir do *GTK Term* adiciona essa nova rota ao Router. Posteriormente é possível verificar a conexão com essa rota fazendo *ping ip-address* (ip definido na criação da rota).

- (Cisco) Como é configurada a *NAT* num router comercial?

  A *NAT* é configurada através de comandos do tipo `ip nat`, que vão definir se um determinado *ip* tem o mesmo valor dentro e fora da *subnet* (se definido como *outside*) ou se é necessário ser adicionado à tabela de endereços que são mapeados para um endereço diferente fora da *subnet* (se definido como *nat inside*).

  Também é possível definir se o router usa *overloading* através do comando *ip nat pool*, como é possível visualizar nas linhas seguintes (retiradas do ficheiro de configuração fornecido):

```
ip nat pool ovrld 172.16.254.45 172.16.254.45 prefix-length 24
ip nat inside source list 1 pool ovrld overload
!
access-list 1 permit 172.16.40.0 0.0.0.7
access-list 1 permit 172.16.30.0 0.0.0.7
```

- (Cisco) O que faz a *NAT* (*Network Address Translation*)?

  A *NAT* procede à tradução de endereços privados de uma *subnet* num endereço público, de forma a permitir a comunicação para endereços fora da *subnet*.

- (DNS) Como configurar o serviço *DNS* (*Domain Name System*) num *host*?

  Para configurar um serviço *DNS* num *host* é necessário adicionar uma entrada nova no ficheiro */etc/hosts* em *Linux*. Especificando um *ip address* e qual o nome que vai estar associado a esse.

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

  São trocados pacotes *UDP* de modo a conseguir obter o endereço ip de um *host*, a partir do seu nome de "domínio".

- (Linux) Que pacotes *ICMP* são visíveis e porquê?

  Os pacotes *ICMP* capturados encontram-se relacionados com a execução do comando `traceroute`, dado que este vai enviando pacotes deste tipo com o *TTL* (*Time To Live*) sucessivamente maior, de modo a descobrir a rota entre os dois intervenientes.

- (Linux) Quais são os endereços *IP* e *MAC* associados aos *packets ICMP* e porquê?

  Os endereços IP associados são os das máquinas de origem e de destino. Já os endereços MAC vão variando de acordo com o *endpoint* que o pacote *ICMP* atinge em cada uma das "chamadas internas"ao *ping* do *traceroute*.

- (Linux) Que rotas estão presentes na nossa máquina de teste? O que significam?

  Tal como foi requerido, foi adicionada uma rota para o servidor *DNS*, sendo que é este que mapeia os *url's* para endereços *IP*. Todas as rotas presentes na máquina podem ser observadas na seguinte figura.

```
[01/04/22]seed@VM:~/Desktop$ sudo ip route add 9.9.9.9 via 10.0.2.1
[01/04/22]seed@VM:~/Desktop$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         10.0.2.1        0.0.0.0         UG    100    0        0 enp0s3
dns9.quad9.net  10.0.2.1        255.255.255.255 UGH   0      0        0 enp0s3
10.0.2.0        0.0.0.0         255.255.255.0   U     100    0        0 enp0s3
104.17.113.188  10.0.2.1        255.255.255.255 UGH   0      0        0 enp0s3
link-local      0.0.0.0         255.255.0.0     U     1000   0        0 enp0s3
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
```

Fig. 5 - Comandos utilizados e Output obtido

4.4 Experiência 4 - *Router Configuration (Lab)*

4.4.1 Objetivos da experiência

Os objetivos desta experiência passam por criar uma ligação entre as duas VLANs previamente criadas, através do *tux4* e posteriormente ligar a VLAN 21 ao router Cisco, para que a nossa *subnet* esteja ligada à *subnet* da FEUP, permitindo assim a ligação à *internet*.

Na imagem seguinte é possível observar o esquema das 'ligações' e rotas pretendidas:

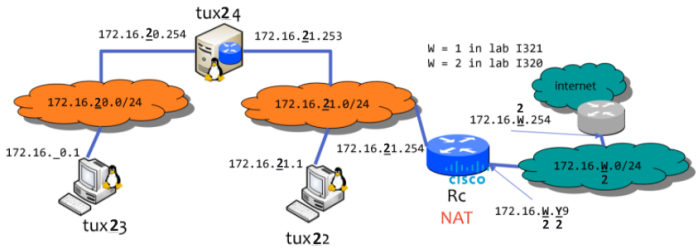

Fig. 6 - Arquitetura da *Network* final (Exp. 4)

4.4.2 Configuração

À semelhança do referido nas experiências 1 e 2, são apresentados abaixo os comandos executados para a realização da experiência, estando assinalados com o nome do computador sublinhado todos os comandos realizados no *GTK Term*, para configuração tanto do ***Switch***, como do **Router Cisco**. É necessário referir mais uma vez a necessidade de utilizar os comandos '`configure terminal`' e '`end`' para, respetivamente, entrar em modo de configuração e sair deste.

Comp.	**Descrição**	**Comando**
tux3 ***Switch***	Adicionar a port 9 (ligação ao tux4) à Vlan21	interface fastethernet 0/9 switchport mode access switchport access vlan 21
tux3 ***Switch***	Verificação da inserção da nova interface	show vlan brief
tux4	Configurar o IP 172.16.21.253, com uma máscara de 24-bit	ifconfig eth1 172.16.20.253/24
tux4	Ativar *IP forwarding*	`echo 1 > /proc/sys/net/ipv4/ip_forward`
tux4	Desativar *ICMP echo ignore broadcast*	`echo 0 > /proc/sys/net/ipv4/icmp` `_echo_ignore_broadcasts`
tux3	Adicionar rota do tux3 para o tux2 *via* tux4	ip route add 172.16.21.0/24 via 172.16.20.254
tux2	Adicionar rota do tux2 para o tux3 *via* tux4	ip route add 172.16.20.0/24 via 172.16.21.253

Para configuração do ***Cisco Router***, realizada no *tux3*, foi utilizado o ficheiro fornecido pelos professores, sendo de notar o código referido abaixo (com valores adaptados à nossa bancada)

```
configure terminal

interface fastethernet 0/0
ip address 172.16.21.254 255.255.255.0
no shutdown
ip nat inside
exit

interface fastethernet 0/1
ip address 172.16.1.29 255.255.255.0
no shutdown
ip nat outside //interface que liga ao 'exterior'
exit

ip nat pool ovrld 172.16.1.29 172.16.1.29 prefix 24
ip nat inside source list 1 pool ovrld overload
```

```
access-list 1 permit 172.16.20.0 0.0.0.7
access-list I permit 172.16.21.0 0.0.0.7

ip route 0.0.0.0 0.0.0.0 172.16.1.254
ip route 172.16.20.0 255.255.255.0 172.16.21.253

end
```

4.4.3 Análise dos resultados

- (Linux) Que rotas existem entre os *tuxes*? Qual é o seu significado?

  Existe uma rota entre o tux3 e o tux4.

  Existe uma rota entre o tux4 e o tux3, tux2 e o router cisco.

  Desta forma, com todas estas rotas, o tux3 apenas pode enviar e receber dados do tux4. O tux4 pode enviar e receber dados do tux3, tux2 e do router. Já o tux2 pode receber e enviar dados para o tux4 e para o router

- (Linux) Que informação está contida na *forwarding table*?

  As *forwarding table* encontram-se quer no router, quer no *switch*. Estas contêm os caminhos que os pacotes que chegam ao respetivo dispositivo têm de percorrer. Desta forma, contêm uma lista dos endereços *IPs* e as respetivas portas para onde os pacotes que vêm desse *IP* devem ser reencaminhados.

- (Linux) Que mensagens ARP e endereços MAC foram observados e porquê?

  Tanto na realização dos comandos *ping*, como na utilização da aplicação FTP desenvolvida por nós, este protocolo é observado e necessário, uma vez que sempre que há troca de informação de uma máquina para a outra temos de obter o endereço MAC (da camada 2), da máquina recetora, correspondente ao IP desta mesma. Este processo pode ser efeutado várias vezes numa só conexão quando as máquinas não pertencem à mesma *subnet*.

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

  No final da experiência, foram efetuados diversos comandos *ping* para e desde as demais máquinas, de modo a confirmar a correta configuração global da rede. Assim, os pacotes *ICMP* que foram observados

foram os relativos a estes comandos e os *IPs* da respetivas máquinas intevenientes.

- (Cisco) Quais são os caminhos utilizados pelos pacotes nas experiências realizadas e porquê?

  Foram feitas 3 experiências para testar conectividade, *ping* do Router Cisco para todos os *tuxes*, *ping* to Router Cisco para 172.16.**2**.254 (2 representa o W) e *ping* do *Cisco Router* para a *internet* (8.8.8.8 ou 104.17.113.188).

  Na 1a experiência, os pacotes sairam do Router, chegando ao *tux2* através de 172.16.21.1, ao *tux4* por 172.16.21.253 e ao *tux3* através da rota para o *tux4*, ligando depois ao *tux3 via* 172.16.20.1.

  Na 2a experiência, os pacotes 'viajaram' pela rota 172.16.2.29, passando depois pela *subnet* e percorrendo 172.16.2.254 para chegar à ligação com o resto da internet.

  Na última experiência, os pacotes seguiram pelo mesmo caminho que na experiência anterior e chegando ao router acessível através de 172.16.2.254, são distribuidos por rotas definidas por este, de modo a alcançarem o *endpoint* pretendido.

5 Conclusão

De facto, o nosso conhecimento prático acerca de como funcionavam os vários componentes de uma rede de computadores era reduzido. Tornou-se, portanto, num trabalho que nos levou bastantes horas de pesquisa e estudo, para além das tentativas e erros que efetuamos no laboratório. Desde de a parte da camada da aplicação, desenvolvendo o próprio software que utilizaria a nossa rede, até à própria troca de cabos *UTP*, foi o percurso completo.

Assim, penso que nos proporcionou uma ótima experiência global, dado que agora somos mais capazes de compreender as redes de computadores para além da teoria.

Por fim, também a forma como estavam divididas as várias etapas de aprendizagem, em experiências mais pequenas, contribuiu fortemente para uma melhor organização do trabalho em grupo e de uma boa distribuição das tarefas, uma vez que todos contribuíram igualmente para a realização do projeto.

6 Referências

Referências

[1] ICMP https://www.cloudflare.com/learning/ddos/glossary/internet-control-message-protocol-icmp/

[2] ICMP https://www.extrahop.com/resources/protocols/icmp/

[3] ARP https://www.fortinet.com/resources/cyberglossary/what-is-arp

[4] ARP https://blog.pantuza.com/artigos/o-protocolo-arp-address-resolution-protocol

[5] PING https://en.wikipedia.org/wiki/Ping_(networking_utility)

[6] DNS https://okiprinting-en-gb.custhelp.com/app/answers/detail/a_id/453/~/why-are-dns-packets-sent-in-tcp-and-not-udp\%3F

[7] Forwarding Table https://www.sciencedirect.com/topics/computer-science/forwarding-table

7 Anexos

7.1 Código fonte da Aplicação de *Download*

client.c

```
#include "../include/client.h"


void write_cmd(int sockfd, char *cmd, char *arg){
        write(sockfd, cmd, strlen(cmd));
        write(sockfd, arg, strlen(arg));
        write(sockfd, "\n", 1);
        printf("Wrote: %s%s\n", cmd, arg);
}

char * read_reply(int sockfd){
        char *r = malloc(1024);

        size_t n = 0;
```

```
        ssize_t read;

        FILE* fp = fdopen(sockfd, "r");
        while((read = getline(&r, &n, fp)) != -1) {
                if(r[3] == ' ') break;
        }

        r[1023] = '\0';
        printf("Reply: %s\n", r);

        return r;
}

int give_credentials(url_data *data, int sockfd){
    char * r;
    write_cmd(sockfd, "user ", data->user);
        r = read_reply(sockfd);
        if(r[0] == '4' || r[0] == '5'){
                close(sockfd);
                return -1;
        }

        write_cmd(sockfd, "pass ", data->password);
        r = read_reply(sockfd);
        if(r[0] == '4' || r[0] == '5'){
                close(sockfd);
                return -1;
        }
    return 0;
}

int connect_socket(char * ip_addr, 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(ip_addr);
        server_addr.sin_port = htons(port);

        /*open an TCP socket*/
        if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
                fprintf(stderr, "Error opening socket!\n");
        return -1;
        }

        /*connect to the server*/
```

```
        if(connect(sockfd, (struct sockaddr *)&server_addr,
        sizeof(server_addr)) < 0) {
                fprintf(stderr, "Error connecting to server!\n");
        return -1;
        }

    return sockfd;
}

int activate_passive_mode(int sockfd){
        write_cmd(sockfd, "pasv", "");

        char *res = malloc(1024);
        res = read_reply(sockfd);
        if(res[0] == '3' || res[0] == '4' || res[0] == '5') {
        free(res);
        return -1;
    }


    strtok(res, "(");
    char * args = strtok(NULL, ")");


        int ip[4];
        int port[2];

        sscanf(args, "%d,%d,%d,%d,%d,%d", &ip[0],
        &ip[1], &ip[2], &ip[3], &port[0], &port[1]);

    free(res);

    return port[0] * 256 + port[1];
}

int download_file(int sockfd, char * url_path){
        printf("Task: downloading file...\n");
    char* filename = basename(url_path);

        FILE *f = fopen(filename, "wb+");

        if(f == NULL){
                perror("fopen()");
                exit(1);
        }

        char buffer[1024];
        int bytes_read;
```

```
        while((bytes_read = read(sockfd, buffer, 1024)) > 0){
                fwrite(buffer, bytes_read, 1, f);
        }

        fclose(f);

    return 0;
}
```

client.h

```
#ifndef CLIENT_H_
#define CLIENT_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>
#include <libgen.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>

#include "input.h"

#define SERVER_PORT 21

void write_cmd(int sockfd, char *cmd, char *arg);

char * read_reply(int sockfd);

int give_credentials(url_data *data, int sockfd);

int connect_socket(char * ip_addr, int port);

int activate_passive_mode(int sockfd);

int download_file(int sockfd, char * url_path);

#endif
```

download.c

```
#include "../include/download.h"
```

```
int main(int argc, char *argv[]){
    if (argc > 2){
        printf("Too many arguments.\n");
        return -1;
    }

    url_data *data = (url_data*)malloc(sizeof(url_data));

    if (inputCheck(argv[1], data) < 0){
        printf("Invalid url\n");
        return -1;
    }

    struct hostent *h;
    char * response;

    if (( h = gethostbyname(data->host)) == NULL){
        printf("Error getting IP\n");
        return -1;
    }

    char *ip_addr = inet_ntoa(*((struct in_addr *)h->h_addr));


    /*connect to server ------------------------
    ------------------------*/

    printf("Task: connecting to socket\n");
    int sockRequest = connect_socket(ip_addr, SERVER_PORT);


    response = read_reply(sockRequest);
    if((response[0]=='4') || response[0]=='5'){
        close(sockRequest);
        return -1;
    }


    /*login ------------------------------------------------*/

    if (give_credentials(data, sockRequest) < 0){
        printf("Error sending credentials \n");
        return -1;
    }

    /*enter passive mode ---------------------
    --------------------------*/
```

```
    int port = activate_passive_mode(sockRequest);

    if(port == -1){
        printf("Failed to activate passive mode\n");
        close(sockRequest);
        return -1;
    }


    /*connect to the server ----------------------
    -------------------------*/

    int sockReceive = connect_socket(ip_addr, port);



    /*save file ------------------------------------------------ */

    write_cmd(sockRequest, "retr ", data->url_path);
        response = read_reply(sockRequest);
    if((response[0]=='4') || response[0]=='5'){
        close(sockRequest);
        return -1;
    }

        download_file(sockReceive, data->url_path);
        response = read_reply(sockRequest);
    if((response[0]=='4') || response[0]=='5'){
        close(sockRequest);
        return -1;
    }

    /*exit ------------------------------------------------*/

        write_cmd(sockRequest, "quit", "");
        response = read_reply(sockRequest);
    if((response[0]=='4') || response[0]=='5'){
        close(sockRequest);
        return -1;
    }


    if (close(sockRequest)<0) {
        perror("close()");
        exit(-1);
    }
```

```
    if (close(sockReceive)<0) {
        perror("close()");
        exit(-1);
    }

    free(data);
    free(response);
    return 0;
}
```

download.h

```
#ifndef _DOWNLOAD_H_
#define _DOWNLOAD_H_

#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <libgen.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>


#include "client.h"

#endif // _DOWNLOAD_H_
```

input.c

```
#include "../include/input.h"

int inputCheck(char *url, url_data *data){
    if(strncmp("ftp://", url, 6) != 0){
        printf("Url header wrong \n");
        return -1;
    }


    return url_converter(url, data);
}

int url_converter(char *url, url_data *data){
    int state = 0;
    int counter = 0;
```

```
    long urlSize = strlen(url);

    if(strchr(url, '@') == NULL){
        state = 2;
        strcpy(data->user, "anonymous");
        strcpy(data->password, "qualquer-password");
    }

    for(int c = 6; c < urlSize; c++){
        char url_char = url[c];

        switch (state){
            case 0:
                if(url_char == ':'){
                    state = 1;
                    counter = 0;
                }
                else{
                    data->user[counter] = url_char;
                    counter++;
                }
                break;

            case 1:
                if(url_char == '@'){
                    state = 2;
                    counter = 0;
                }
                else{
                    data->password[counter] = url_char;
                    counter++;
                }
                break;

            case 2:
                if(url_char == '/'){
                    state = 3;
                    counter = 0;
                }
                else{
                    data->host[counter] = url_char;
                    counter++;
                }
                break;

            case 3:
                data->url_path[counter] = url_char;
                counter++;
```

```
                break;


        }

    }

    if(state != 3){
        printf("%d \n\n", state);
        printf("Wrong url structure\n");
        return -1;
    }

    strcpy(data->url, url);

    return 0;
}
```

input.h

```
#ifndef INPUT_H_
#define INPUT_H_

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

typedef struct{
    char url[5120];

    char user[1024];
    char password[1024];
    char host[1024];
    char url_path[1024];

}url_data;

int inputCheck(char *ur, url_data *data);

int url_converter(char *url, url_data *data);

#endif
```

7.2 Todos os *logs* capturados

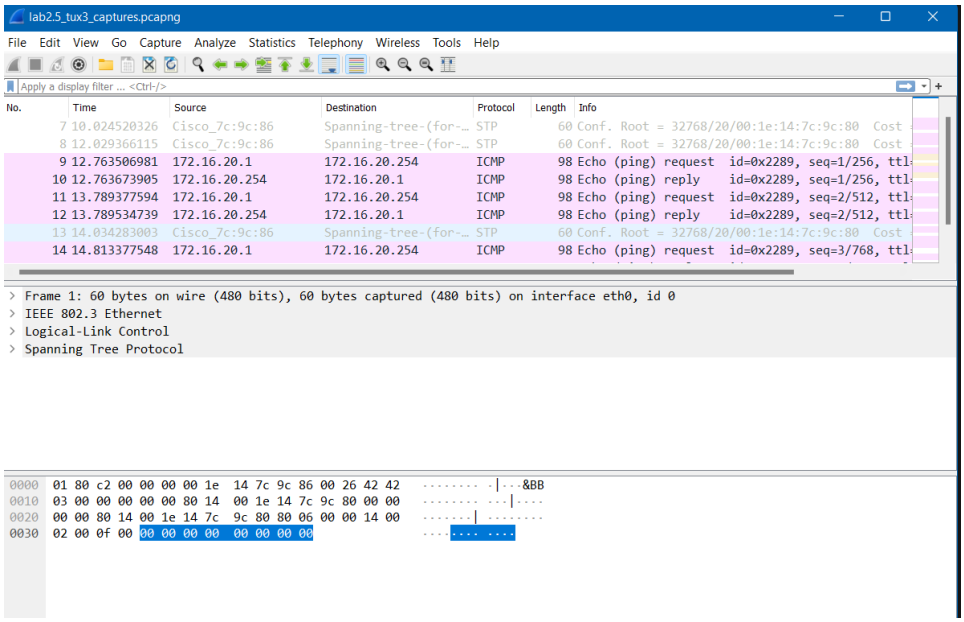

Fig. 7 - Lab 2, Step 5: tux3 pinging tux4

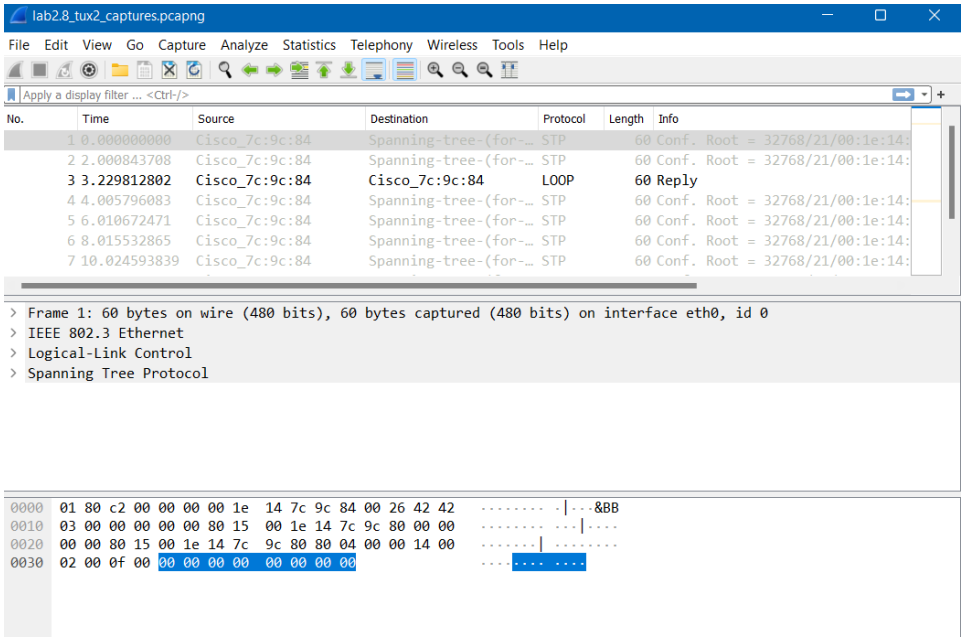

Fig. 8 - Lab 2, Step 8: tux3 broadcast pinging (captures in tux2 are none as tux2 is not in the same vlan as tux3)

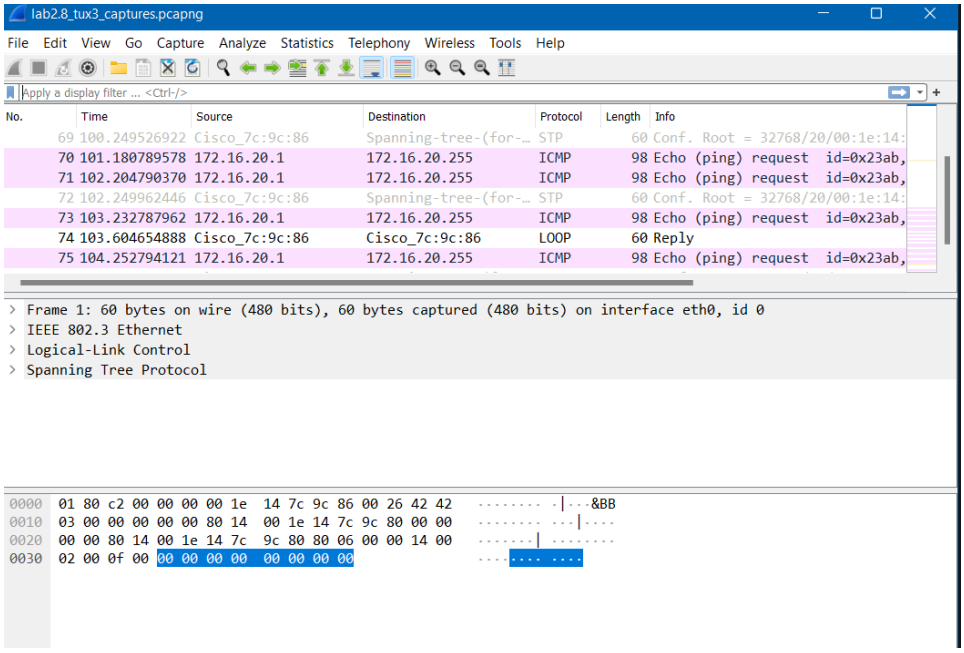

Fig. 9 - Lab 2, Step 8: tux3 broadcast pinging (captures in tux3 are not none as tux3 is in the same vlan as tux3)

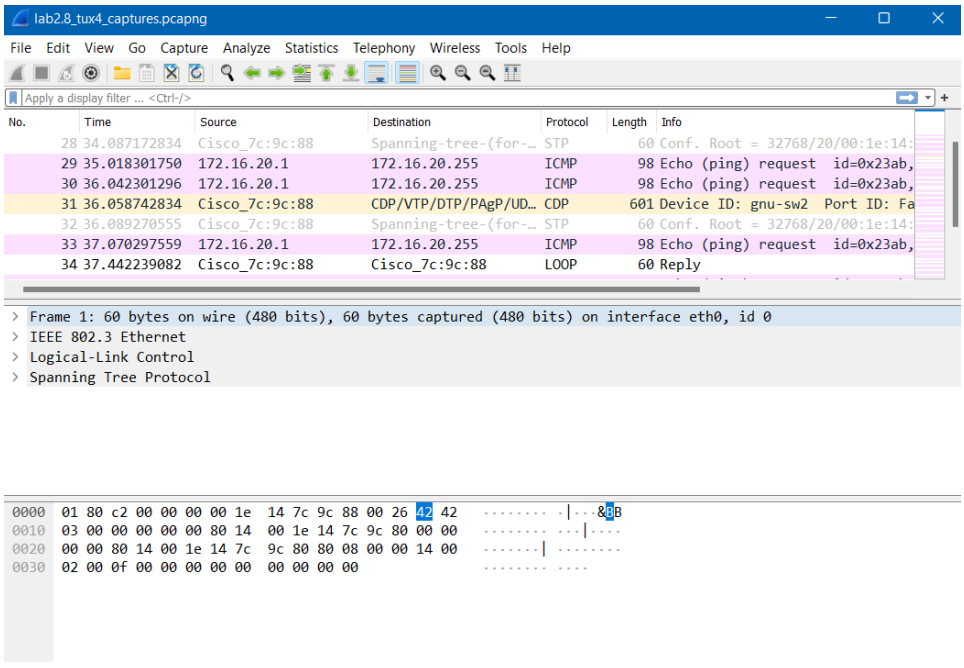

Fig. 10 - Lab 2, Step 8: tux3 broadcast pinging (captures in tux4 are not none as tux4 is in the same vlan as tux3)

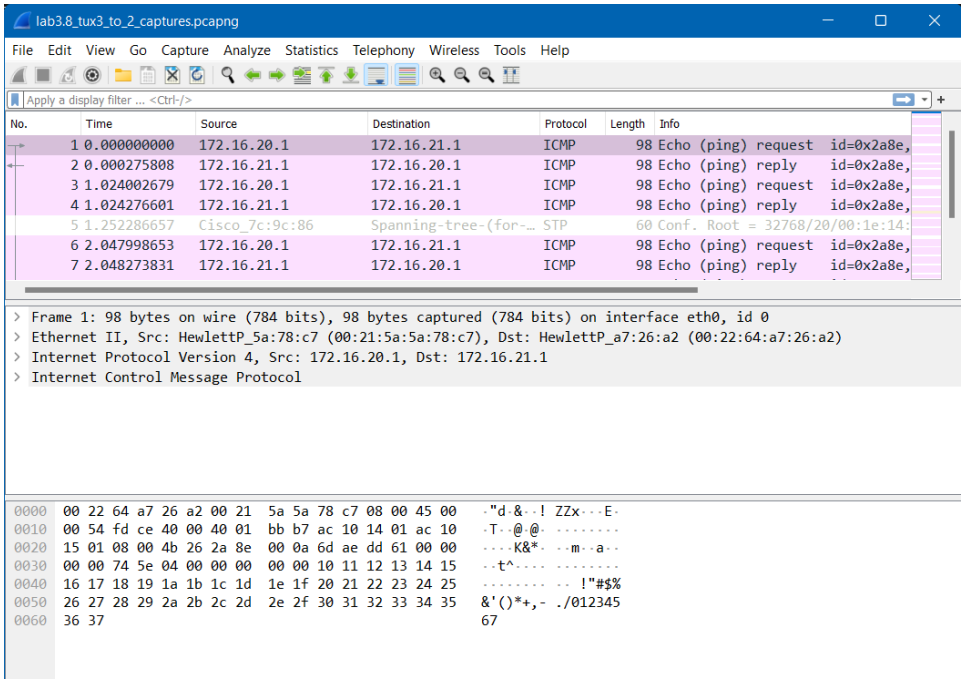

Fig. 11 - Lab 4, Step 8, : tux3 pinging tux2, captures on tux2

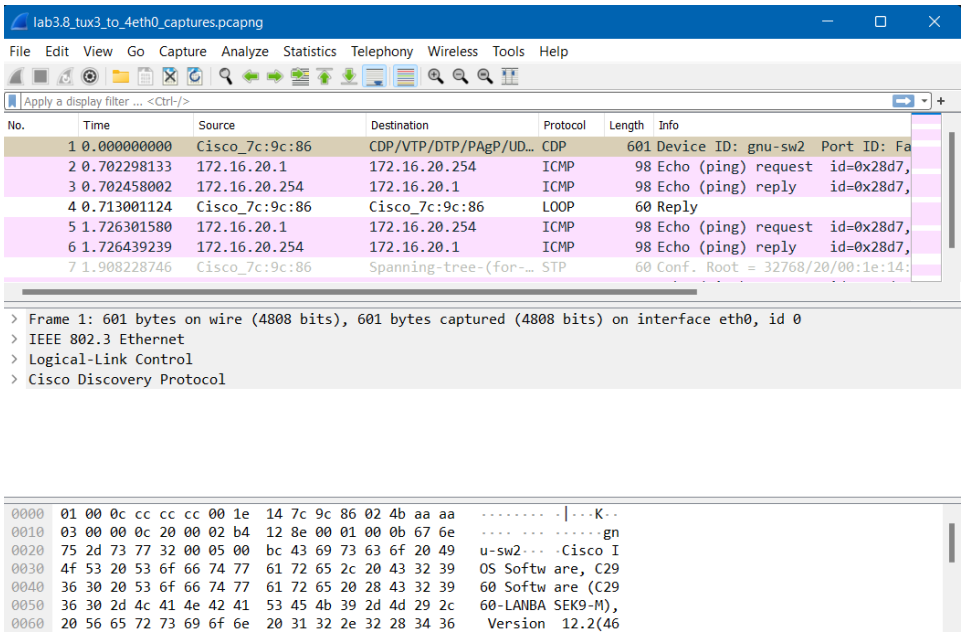

Fig. 12 - Lab 4, Step 8: tux3 pinging tux4, captures on tux4 (eth0)

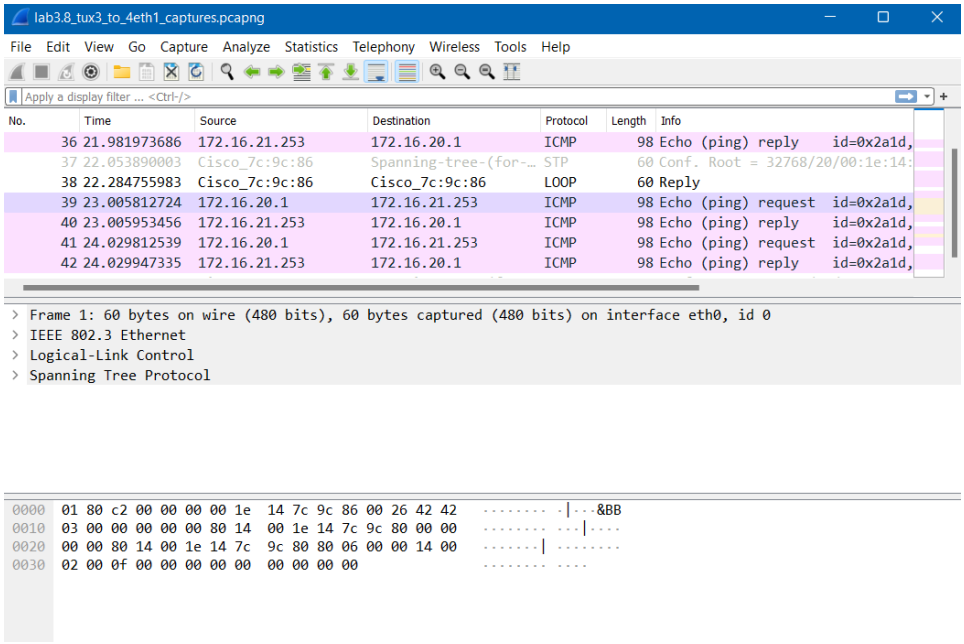

Fig. 13 - Lab 4, Step 8: tux3 pinging tux4, captures on tux4 (eth1)

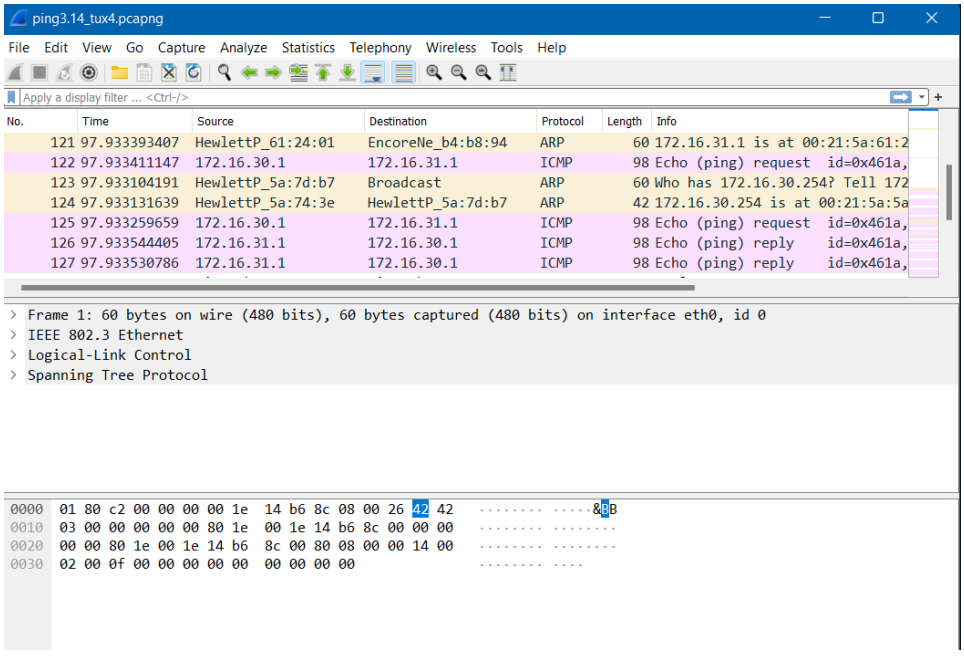

Fig. 14 - Lab 4, Step 14

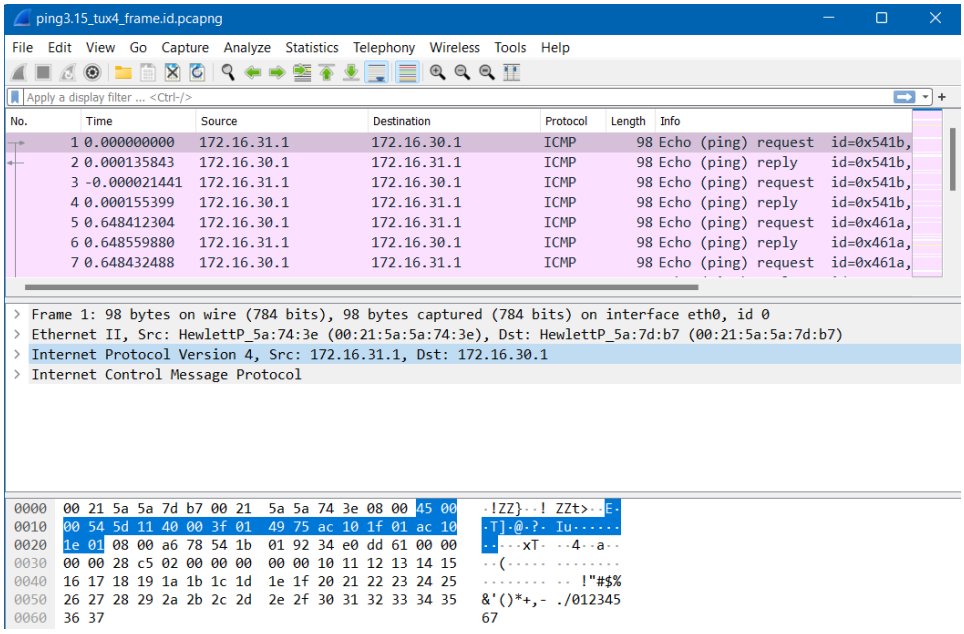

Fig. 15 - Lab 4, Step 15