

Rede de Computadores

Redes de Computadores

3ºAno MIEIC

23 de dezembro de 2020

Rafael Cristino, up201806680@fe.up.pt

Rita Peixoto, up201806257@fe.up.pt

Grupo 3, Turma 2

Índice

| | |
|---|----|
| Rede de Computadores..... | 1 |
| Sumário | 3 |
| Introdução..... | 3 |
| Parte 1 – Aplicação de download | 3 |
| Aplicação..... | 3 |
| Resultados..... | 4 |
| Parte 2 – Configuração e análise de uma rede de computadores..... | 5 |
| Experiência 1: Configuração do <i>IP</i> de uma rede | 5 |
| Experiência 2: Implementação de duas <i>LANs</i> virtuais num <i>switch</i> | 6 |
| Experiência 3: Configuração de um router em Linux | 6 |
| Experiência 4: Configuração de um <i>router</i> comercial e implementação da <i>NAT</i> | 7 |
| Experiência 5: DNS..... | 9 |
| Experiência 6: Conexões TCP..... | 9 |
| Conclusões..... | 11 |
| Referências | 11 |
| Anexos..... | 12 |
| Código da aplicação: | 12 |
| Experiência 1 | 23 |
| Experiência 2 | 24 |
| Experiência 3 | 25 |
| Experiência 4 | 28 |
| Experiência 5 | 29 |
| Experiência 6 | 30 |

Sumário

Este trabalho laboratorial, desenvolvido no âmbito da unidade curricular de Redes de Computadores, teve como objetivo a configuração e análise de uma rede de computadores, tendo sido utilizados, para tal, um *router* e um *switch* para interligar os três computadores (*tuxes*). Para além disso, foi desenvolvida uma aplicação para efetuar o download de um ficheiro de um servidor através do protocolo FTP.

Este relatório visa descrever as diferentes etapas e experiências efetuadas, bem como identificar as conclusões retidas. Os requisitos deste trabalho foram cumpridos, tendo sido possível concluir todas as etapas da configuração da rede e produzido um simples cliente FTP, capaz de efetuar o download de ficheiros.

Introdução

O trabalho integra duas partes distintas: o desenvolvimento de uma aplicação de download e a configuração de uma rede de computadores.

Na primeira parte a aplicação de download foi conseguida através da implementação de um simples cliente de *File Transfer Protocol* (FTP) e usando uma ligação do tipo *Transmission Control Protocol* (TCP).

Na segunda parte foram efetuadas um conjunto de experiências de modo a configurar uma rede de computadores.

Este relatório pretende examinar a implementação deste trabalho laboratorial, os seus diferentes componentes e como estes se complementam. O relatório organiza-se na seguinte estrutura:

Aplicação de download - Arquitetura da aplicação e resultados.

Configuração e análise de uma rede de computadores - Descrição e análise de cada experiência efetuada.

Conclusões - Síntese da informação apresentada nas secções anteriores e reflexão acerca dos objetivos de aprendizagem alcançados.

Anexos - Código da aplicação de download e *logs* capturados.

Parte 1 – Aplicação de download

Nesta primeira fase do projeto, foi desenvolvida uma aplicação à qual é fornecido um URL com o formato: ftp://[<user>:<password>@]<host>/<url-path>, que efetua o download do ficheiro especificado em **url-path** de um servidor FTP.

Aplicação

A aplicação foi desenvolvida ao longo de um conjunto de fases: processamento dos argumentos, início da conexão ao servidor, autenticação, download e fim da conexão.

Na primeira fase, é utilizada a função *parseUrl* para retirar do URL fornecido os diversos campos necessários para o funcionamento correto da aplicação: dados para autenticação (se especificados nos campos “user” e “password”), o servidor e o ficheiro a fazer a download.

De seguida, é iniciada a conexão ao servidor com a chamada à função *connectToIP*, que abre uma *socket* TCP e cria a conexão entre o servidor e essa *socket* e utiliza a função *getIP*, que recebe um *hostname* e retorna o correspondente endereço de IP público.

Após conectar com o servidor, de modo a permitir o envio de comandos para este servidor é necessária a devida autenticação. Esta autenticação pode ser genérica, utilizando como utilizador *anonymous* e como palavra-passe um valor qualquer, ou, se o utilizador e palavra-passe forem especificados no URL recebido como argumento, então vão ser estes os dados utilizados na autenticação. A autenticação é feita recorrendo à função *loginHost*.

Antes de iniciar o download do ficheiro, é, então, necessário pedir ao servidor de FTP para transferir os dados em modo passivo. Tal é conseguido com o envio do comando “*pasv*”, a que o servidor responde com uma série de valores que nos permite o cálculo da porta a abrir através do qual será transferido o ficheiro. Tanto o envio deste comando, como o cálculo desta porta são realizados na função *passiveMode*, sendo o que o cálculo recorre a uma função auxiliar *parsePsv*.

Por fim, é iniciada a conexão do servidor a um segundo *socket* na porta calculada na fase anterior. Certifica-se que a aplicação se encontra em modo binário, enviando o comando “*type I*” na função *binaryMode*, é enviado o comando “*retr*” seguido com o nome do ficheiro que se pretende fazer download, através da função *retrCommand*, e inicia-se a transferência do ficheiro, recorrendo à função *receiveFile*. Quando a transferência termina, o servidor envia uma mensagem “*transfer complete*”.

Ao receber esta mensagem, é terminada a conexão com o envio do comando “*QUIT*” são fechados os *sockets* bem como o ficheiro obtido, através da função *disconnect*.

Ao longo de todo o correr da aplicação são verificadas as respostas do servidor e confirmadas com os códigos de sucesso das ações. Caso, algum código de erro seja recebido, a aplicação termina.

Resultados

De seguida encontra-se um exemplo de um download bem-sucedido, onde é possível observar os comandos enviados e as respostas recebidas. Neste exemplo foi utilizado o servidor de FTP para teste disponibilizado no moodle:

```
./download ftp://rcom:rcom@netlab1.fe.up.pt/files/pic1.jpg
Retrieving 'files/pic1.jpg' from 'netlab1.fe.up.pt', using username 'rcom' and
password 'rcom'.
Host name: netlab1.fe.up.pt
IP Address: 192.168.109.136
220 Welcome to netlab-FTP server
User command: user rcom
331 Please specify the password.
Password command: pass rcom
230 Login successful.
227 Entering Passive Mode (192,168,109,136,177,191).
Binary command: type I

200 Switching to Binary mode.
RetrCommand: retr files/pic1.jpg
150 Opening BINARY mode data connection for files/pic1.jpg (340603 bytes).
226 Transfer complete.
QuitCommand: quit
221 Goodbye
```

Parte 2 – Configuração e análise de uma rede de computadores

Encontram-se nos anexos os *logs* do *wireshark* que sustentam a análise descrita na secção da experiência correspondente.

Experiência 1: Configuração do IP de uma rede

A primeira experiência consistiu em configurar uma rede IP, pelo que a interface eth0 do tux63 e do tux64 foram ligados ao switch.

De modo a configurar a rede IP, foi necessária a atribuição de um endereço IP a cada um dos computadores:

tux63 --> ifconfig eth0 172.16.60.1/24

tux64 --> ifconfig eth0 172.16.60.254/24

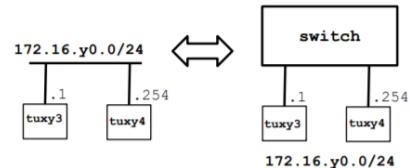


Figura 1: Topologia da rede Experiência 1

Foram então registados os endereços IP e MAC das interfaces de rede obtidos através dos pacotes ARP. Address Resolution Protocol (ARP) é um o protocolo de comunicação utilizado para obter o endereço MAC, endereço da camada de ligação, associado a um determinado endereço IP, endereço da camada de rede/internet. Os seus pacotes são usados para, sabendo o endereço IP de uma máquina, que serve para identificar uma máquina quando esta quer comunicar numa determinada rede, pedir o endereço MAC, que identifica a placa de rede.

No caso de o emissor do pacote não ter na sua tabela ARP uma entrada com o IP do recetor, então o pedido ARP é enviado para o canal de transmissão da sub-rede, com o campo de endereço MAC vazio (00:00:00:00:00:00) e o campo do endereço IP com o IP do recetor. O recetor identifica-se como o recetor correto, respondendo ao pedido com um pacote ARP com o endereço MAC correspondente.

Foi utilizado o comando *ping* para testar a conectividade entre os dois computadores. Primeiramente, este irá gerar pacotes ARP de forma a saber o endereço MAC do destinatário e depois gera pacotes do tipo ICMP (*Internet Control Message Protocol*) que, neste caso, possuem mensagens de ‘Echo Request’ ou ‘Echo Reply’.

No pacote enviado no comando ping pelo tux63 os endereços IP e MAC de origem são 172.16.60.1 e 00:21:5a:61:2d:df, ou seja, os do tux63 e os endereços de destino são 172.16.60.254 e 00:21:5a:61:79:97, os do tux64.

No caso exatamente oposto, os endereços de origem do pacote enviado pelo tux64 através do comando ping são os do mesmo e os de destino são os do tux63, como referidos acima.

O tipo da trama *Ethernet* recebida pode ser determinado através da análise do valor do cabeçalho Ethernet II que se for 0x0806, trata-se de um ARP e no caso de ser 0x0800 é do tipo IPv4. Apenas dentro do cabeçalho do IPv4 é que é possível determinar se se trata de ICMP, sendo que o campo *Protocol* terá o valor 0x01.

O comprimento de uma trama é possível obter através de análise dos registos da trama no *Wireshark*, mais especificamente no campo *Frame Length*.

A interface loopback é uma interface virtual, que está sempre a funcionar e que não está sujeita às mesmas fraquezas de uma interface física. É particularmente importante para verificar se a configuração da placa de rede foi efetuada corretamente. Uma máquina envia um pacote cujo endereço de destino é ele mesmo e se receber este pacote, então a placa está configurada da forma correta.

Experiência 2: Implementação de duas LANs virtuais num switch

Nesta Experiência, procedeu-se à implementação de duas LANs virtuais (VLANs) num switch, vlan60 e vlan61, em que os tux63 e tux64 foram associados à vlan60 e o tux62 à vlan61.

Tendo já previamente configurado os tux63 e tux64 na experiência anterior, foi necessário configurar o tux62, recorrendo, à semelhança dos outros, ao comando:

```
ifconfig eth0 172.16.61.1/24
```

Para proceder à configuração da vlan60, o eth0 do tux63 foi ligado à porta 5 do switch, o eth0 do tux64 foi ligado à porta 9 do switch e a porta série do tux63 foi ligada à porta *switch console*. A configuração da VLAN deu-se utilizando os seguintes comandos na consola *GKTerm* do tux63:

```
configure terminal  
vlan 60  
end
```

Após isso foram adicionadas as portas respetivas à VLAN, com os seguintes comandos:

```
Ex: associação do tux63 à vlan60  
configure terminal  
interface fastEthernet 0/ [nº da porta, neste caso, 5]  
switchport mode access  
switchport access vlan 60  
end
```

A configuração da vlan61 foi em tudo semelhante, bem como a adição da porta do tux62.

Nesta configuração há dois domínios de *broadcast*, um para cada subrede, vlan60 - 172.16.60.255, que contém o tux63 e o tux64 e vlan61 - 172.16.61.255, que contém o tux62.

Experiência 3: Configuração de um router em Linux

A experiência começa por tornar o tux64 num router, de forma a estabelecer uma ligação entre as duas VLANs existentes e os respetivos computadores.

Para transformar o tux64 num router, foi necessário ligar a sua interface eth1 ao switch e foram efetuados os seguintes comandos:

```
ifconfig eth1 172.16.61.253/24  
echo 1 > /proc/sys/net/ipv4/ip_forward  
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
```

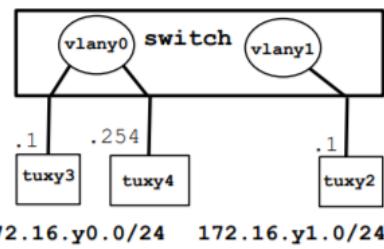


Figura 2: Topologia da rede Experiência 2

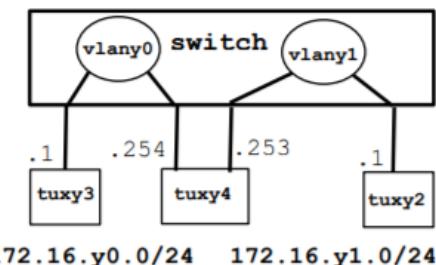


Figura 3: Topologia da rede Experiência 3

Para o tux63 e o tux62 poderem chegar um ao outro, foram adicionadas as rotas respetivas:

```

route add -net 172.16.61.0/24 gw 172.16.60.254 - adicionar a rota de tux63 para tux62 usando tux64
route add -net 172.16.60.0/24 gw 172.16.61.253 - adicionar a rota de tux62 para tux63 usando tux64

```

As rotas definem destinos intermédios dos pacotes quando o destino final não se encontra na sub-rede do emissor. Sendo assim as rotas existentes são:

- O tux63 tem uma rota para a vlan60 (172.16.60.0) pela gateway 172.16.60.1 e uma rota para a vlan61 (172.16.61.0) pela gateway 172.16.60.254.
- O tux64 tem uma rota para a vlan60 (172.16.60.0) pela gateway 172.16.60.254 e uma rota para a vlan61 (172.16.61.0) pela gateway 172.16.61.253.
- O tux62 tem uma rota para a vlan61(172.16.61.0) pela gateway 172.16.61.1 e uma rota para a vlan60 (172.16.60.0) pela gateway (172.16.61.253).

Após a adição destas rotas, é possível analisar a *forwarding table*, tabela onde se encontram as rotas para determinados destinos dentro da rede, com os seguintes campos: “*Destination*” (destino da rota), “*Gateway*” (endereço do próximo ponto para onde o pacote é enviado no caminho para o seu destino final), “*Genmask*” (permite juntamente com o “*Destination*” determinar o ID da rede), “*Flags*” (informações acerca das rotas), “*Metric*” (custo da rota, de modo a escolher a melhor rota), “*Ref*”(número de referencias para a rota), “*Use*”(contador de pesquisa sobre a rota), “*Iface*” (indica qual a interface, disponível para atingir o *gateway*, eth0 ou eth1, nestes casos).

Após configuração das rotas, as duas sub-redes encontram-se interligadas, pelo que os pedidos e respostas *ping* geram pacotes ICMP do tipo *echo requests* e *echo replies*, respetivamente, pois os tux62 e tux63 conseguem comunicar, caso contrário eram recebidos pacotes do tipo *host unreachable*. Então, ao executar o comando *ping* do tux63 para o tux62:

No caso do tux63 não ter na sua tabela ARP uma entrada com a correspondência IP – MAC do tux64, então o pedido ARP é enviado para o canal de *broadcast* da vlan0, ao qual o tux64 responde autoidentificando o seu endereço MAC, criando a entrada na tabela.

O pedido *ping* será então enviado ao tux64, que o reencaminha para o tux62. No caso de o tux63 não ter na sua tabela ARP uma entrada com a correspondência IP – MAC do tux 62, então este executa o mesmo processo de envio de pedido ARP para o canal de *broadcast* da vlan1, obtendo o endereço MAC do tux62, e reencaminhando o pedido *ping*.

Para a resposta do tux62 ao *ping* do tux63, é executado o mesmo processo, mas no sentido inverso (obtenção pelo tux62 do endereço MAC do tux64 e posteriormente pelo tux64 do endereço MAC do tux63).

Os endereços IP e MAC associados aos pacotes ICMP são os endereços do emissor e do recetor, alternadamente, do tux62 e do tux63.

Experiência 4: Configuração de um router comercial e implementação da NAT

Nesta fase, foi realizada a configuração de um router comercial por forma a ter acesso à rede de laboratório e a implementação da NAT para que os computadores da rede tenham acesso à internet. Para isso, foi necessário ligá-lo à rede do laboratório, através da porta fastethernet 0/1 e também à vlan61, ligando a porta 0/0 do router ao switch. Para introduzir os comandos de configuração do router, foi necessário ligar a porta de ligação série do tux63 à porta *router console*, sendo os comandos introduzidos através da consola GTKTerm.

Esta experiência teve também como objetivo determinar a influência do NAT na conexão entre os *tuxes* e a internet, bem como compreender o mecanismo de redireccionamento dos pacotes.

Para configurar uma rota estática num router comercial, tal como o usado nesta experiência, utiliza-se o comando *ip route*, com a seguinte estrutura:

```

ip route prefix mask {ip-address | interface-type interface-number [ip-address]}

```

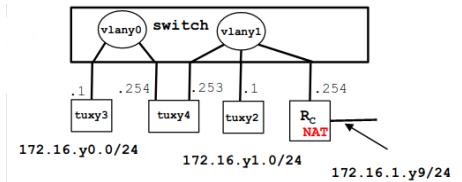


Figura 4: Topologia da rede Experiência 4

Exemplo de utilização, inserindo na consola do router através do GTKTerm:

Neste caso, é criada uma rota estática para o destino 172.16.60.0, tendo como máscara 255.255.255.0 e gateway 172.16.61.253.

```
conf t
ip route 172.16.60.0 255.255.255.0 172.16.61.253
end
```

No caso de um pacote tentar seguir uma rota, se a rota existir, o pacote segue essa rota, caso contrário, os pacotes são redirecionados para a rota *default*. Nesta experiência foram definidos o tux64 como rota *default* do tux63 e o Rc como rota *default* do tux62 e do tux64.

Quando são desativados os *redirects*, através dos comandos:

```
echo 0 > /proc/sys/net/ipv4/conf/eth0/accept_redirects
echo 0 > /proc/sys/net/ipv4/conf/all/accept_redirects
```

, o que acontece é que, por exemplo, sempre que o tux62 quiser comunicar com o tux63, e assumindo que não tem uma rota definida para esse computador, vai enviar para o router, que é a sua rota *default*, que depois redireciona para o tux64, devido à rota que tem definida (ip route 172.16.60.0 255.255.255.0 172.16.61.253).

No entanto, se os *redirects* estivessem ativos, isso apenas aconteceria da primeira vez, porque o tux62 guardaria na sua *forwarding table* o caminho até ao tux63, e então das próximas vezes não teria de passar pelo router, enviando o pedido logo para o tux64.

Para configurar a NAT no router comercial, foi inserida a seguinte sequência de comandos na consola do router, que se encontram no slide 46 do guião fornecido.

```
conf t
interface fastethernet 0/0
ip address 172.16.61.254 255.255.255.0
no shutdown
ip nat inside
exit
interface fastethernet 0/1
ip address 172.16.2.69 255.255.255.0
no shutdown
ip nat outside
exit
ip nat pool ovrlid 172.16.2.69 172.16.2.69 prefix 24
ip nat inside source list 1 pool ovrlid overload
access-list 1 permit 172.16.60.0 0.0.0.7
access-list 1 permit 172.16.61.0 0.0.0.7
ip route 0.0.0.0 0.0.0.0 172.16.2.254
ip route 172.16.60.0 255.255.255.0 172.16.61.253
end
```

A NAT (Network Address Translation) consiste no remapeamento de um endereço IP noutro endereço IP, de forma a ocultar o verdadeiro endereço do emissor/recetor dos pacotes enviados.

Este remapeamento tem o fim de assegurar a privacidade e segurança dos computadores ligados uma sub-rede local privada, mas a sua função é também necessária para os mesmos poderem comunicar na internet.

É no router que o NAT é executado, e é o router também o ponto de ligação entre a sub-rede privada e a rede pública. Quando um computador da sub-rede envia um pacote para um computador da rede pública, esse pacote é primeiro processado no router, que, utilizando NAT, mascara o endereço do computador com o seu próprio endereço. Assim, todos os pacotes enviados de qualquer computador da sub-rede, aos olhos dos computadores da rede pública, têm o mesmo endereço, que é o endereço do router. Quando o router recebe uma resposta a um pedido que foi feito por um dos computadores da sua sub-rede, este ‘desmascara’ então o endereço do computador recetor e reencaminha o pacote para o mesmo.

Se o endereço IP privado não fosse mascarado com o endereço IP do router, os outros computadores (da rede pública) não saberiam como responder ao pedido, porque para eles esse endereço não existe. É por esta razão que não é possível ter acesso à internet sem a NAT configurada.

Experiência 5: DNS

Esta experiência consiste em configurar o serviço DNS (*Domain Name System*) nos computadores da nossa rede, que traduz endereços IP em nomes de domínios.

Para fazer esta configuração num dado host, que no caso desta experiência é netlab.fe.up.pt, foi necessário garantir que o ficheiro *resolv.conf*, contém as seguintes linhas:

```
search netlab.fe.up.pt
nameserver 172.16.2.1
```

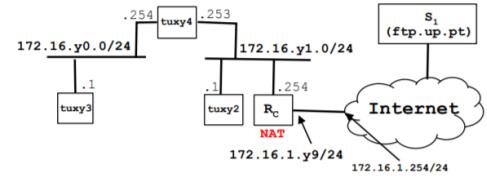


Figura 5: Topologia da rede Experiência 5

Definindo assim que pretendemos utilizar o servidor DNS *netlab.fe.up.pt* com o respetivo endereço IP 172.16.2.1.

Quando se faz um ping a um servidor externo, é enviado um pacote de DNS com o nome do host desejado e o servidor envia um pacote de resposta com o endereço IP que corresponde ao nome do pacote recebido.

Experiência 6: Conexões TCP

Com o objetivo de observar o modo de funcionamento do protocolo TCP, foi usada a aplicação de download desenvolvida na primeira parte do projeto.

Numa primeira fase, foram capturados os pacotes transmitidos (e recebidos), através do uso do *wireshark*, durante um simples download de um ficheiro. Numa segunda fase, capturaram-se os mesmos pacotes durante o download do ficheiro, mas num dado instante, iniciou-se outro download noutro computador ligado à mesma rede, de forma a avaliar a variação do *throughput* ao longo do tempo no primeiro, e a averiguar se o segundo download teve influência no *throughput* do primeiro.

É possível concluir que a aplicação de download abre duas conexões TCP. A primeira é utilizada para entrar em contacto com o servidor, e através da qual são enviados comandos de controlo FTP (e recebidas as respostas), de forma a iniciar a transferência do ficheiro. É nesta ligação que é transportado o controlo de informação. A segunda é utilizada para a própria transferência do ficheiro.

Uma ligação TCP divide-se em três fases: estabelecimento da ligação, transferência de dados e término da ligação.

A ligação TCP faz uso do mecanismo ARQ TCP. Este mecanismo funciona através do método *Sliding Window*. Os campos TCP relevantes para este mecanismo são o número de sequência (32bit *SeqNumber*), o número de *acknowledgment* (32bit *AckNumber*) e o *Window Size*.

De modo a controlar o congestionamento, para cada ligação, o TCP mantém uma *congestion window*, que limita a quantidade de tráfego em trânsito. Para isso, é usado um mecanismo de *slow start*. A *congestion window* inicia com um pequeno valor, múltiplo do MSS (*Maximum Segment Size*), e cada pacote que é ACKed faz com que ela aumente por 1 MSS. Quando a *congestion window* chega ao valor de *slow start threshold*, entra-se num novo estado, chamado *congestion avoidance*. Neste estado, a *congestion window* é incrementada por 1 MSS a cada RTT (*Round Trip Time* – o tempo desde que envia um pacote até que recebe a sua resposta), o que provoca um crescimento mais reduzido.

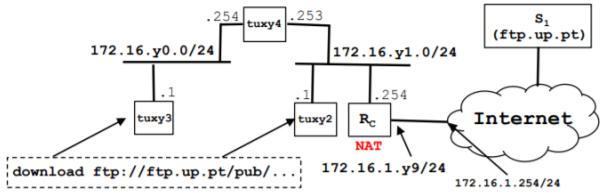


Figura 6: Topologia da rede Experiência 6

Ao ocorrer um timeout, ou seja, um pacote não foi *ACKed*, este mecanismo é reiniciado, voltando ao estado de *slow start*.

O valor da *congestion window* diminui com o aumento do congestionamento e aumenta com a diminuição do congestionamento.

Pode-se verificar no gráfico de *throughput* obtido que quando o primeiro download começou, o *throughput* aumentou rapidamente, eventualmente estabilizando, o que está de acordo com o previsto no mecanismo de controlo de congestão.

É também possível verificar que, ao iniciar um download no segundo computador enquanto o primeiro ainda está a acontecer, o *throughput* do primeiro reduz acentuadamente (estabilizando num valor mais baixo) enquanto o *throughput* do segundo aumenta, também acentuadamente, o que também está de acordo com o previsto anteriormente, sendo que o *throughput* é dividido igualmente pelas duas ligações.

Este fenómeno é observável na figura 8. Ao iniciar uma segunda ligação de dados TCP, por volta do instante 4, o *throughput* da primeira ligação diminui acentuadamente, quando comparado ao do *download* que foi realizado sem iniciar uma segunda ligação (figura 7), uma vez que a taxa de transferência de dados é distribuída igualmente pelas duas ligações.

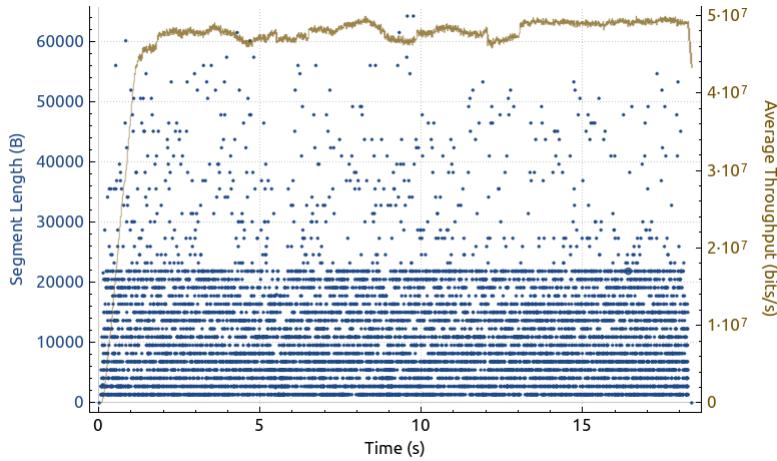


Figura 7: Gráfico da evolução do *throughput* da conexão TCP ao longo do tempo para a transferência de um ficheiro num computador

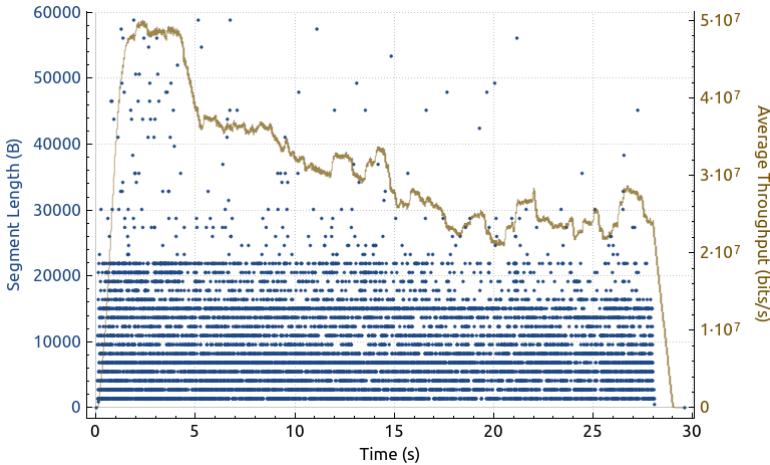


Figura 8: Gráfico da evolução do *throughput* da conexão TCP ao longo do tempo para a transferência simultânea de ficheiros em dois computadores na mesma rede

Conclusões

O propósito deste trabalho laboratorial centrou-se na análise e configuração de uma rede de computadores. Em acréscimo, foi proposto o desenvolvimento de uma aplicação de download de um ficheiro de um servidor utilizando os protocolos FTP e TCP. Foram então explorados dispositivos *switch* e *router*; técnicas como NAT e DNS, diferentes protocolos, FTP, TCP, ICMP, ARP e estruturas de dados, *ARP Tables* e *forwarding tables*.

Realçamos que foram cumpridos todos os objetivos do trabalho laboratorial, tanto o desenvolvimento da aplicação de download como a sequência de experiências que levaram à configuração da rede de computadores.

É de salientar que este trabalho teve algumas adversidades, nomeadamente o número de aulas e o tempo em laboratório que se mostrou ser pouco, não tendo havido oportunidade de realizar a experiência 6 e 7 no laboratório, sendo que a 6 teve de ser realizada em casa.

Por último, concluímos que este trabalho superou as nossas expectativas acerca do seu contributo para a compreensão dos conceitos lecionados na unidade curricular. Permitiu que a conexão de todos os conceitos teóricos e práticos fosse bastante mais simples, levando a uma aprendizagem de forma esclarecedora.

Referências

- Slides das aulas teóricas
- Guião do trabalho prático
- https://www.juniper.net/documentation/en_US/junos/topics/concept/interface-security-loopback-understanding.html
- <https://www.oreilly.com/library/view/cisco-ios-in/156592942X/ch05s03.html>
- <https://www.geeksforgeeks.org/network-address-translation-nat/>
- <https://www.cloudflare.com/pt-br/learning/dns/what-is-dns/>

Anexos

Código da aplicação

```
## main.c
#include "fileManager.h"

void printUsage(char ** argv) {
    fprintf(stderr, "Usage: %s ftp://[<user>:<password>@]<host>/<url-path>\n", argv[0]);
}

int main(int argc, char*argv[]){
    if (argc != 2) {
        printUsage(argv);
        exit(EXIT_FAILURE);
    }

    char *user = NULL;
    char *password = NULL;
    char *host = NULL;
    char *urlPath = NULL;
    char *filename = NULL;
    if (parseURL(argv[1], &user, &password, &host, &urlPath, &filename)){
        fprintf(stderr, "Error parsing the url!\n");
        printUsage(argv);
        exit(EXIT_FAILURE);
    }

    if (user != NULL) {
        printf("Retrieving '%s' from '%s', using username '%s' and password '%s'.\n",
        urlPath, host, user, password);
    } else {
        printf("Retrieving '%s' from '%s' anonymously.\n", urlPath, host);
    }

    char* ip = getIP(host);
    if (ip == NULL) exit(EXIT_FAILURE);
    int commandSocketFD = connectToIP(ip, FTP_COMMAND_PORT);

    unsigned reply = 0;
    if (readReply(commandSocketFD, &reply, NULL)) return EXIT_FAILURE;

    if (fileTransfer(commandSocketFD, user, password, host, urlPath, filename)) return
    EXIT_FAILURE; //download of the file

    free(user);
    free(password);
    free(host);
    free(urlPath);

    return 0;
}
```

```

## parser.c
#include "parser.h"

void copyBufferToVar(char * buffer, size_t bufferSize, char ** out) {
    *out = malloc((strlen(buffer)+1)*sizeof(char));
    memcpy(*out, buffer, (strlen(buffer)+1)*sizeof(char));
    memset(buffer, 0, bufferSize);
}

int parseURL(char *url, char **user, char **password, char **host, char **urlPath, char **filename) {
    if (strncmp(URL_PROTOCOL, url, 6) != 0) {
        fprintf(stderr, "The protocol isn't valid. Should be '%s'.\n", URL_PROTOCOL);
        return 1;
    }
    enum urlParsingState state = PROTOCOL_OK;

    size_t bufferSize = (strlen(url))*sizeof(char);
    char * buffer = malloc(bufferSize);

    for (unsigned i = 6; i < strlen(url)+1; i++) {
        char currentChar = url[i];
        enum urlParsingState prevState = state;
        switch (state) {
            case PROTOCOL_OK:
                switch (currentChar) {
                    case ':':
                        copyBufferToVar(buffer, bufferSize, user);
                        state = USERNAME_OK;
                        break;
                    case '/':
                        copyBufferToVar(buffer, bufferSize, host);
                        state = HOST_OK;
                        break;
                    case '@':
                        fprintf(stderr, "Invalid ftp url: Expecting username or host,
'@' in the wrong place\n");
                        return -1;
                    default: break;
                }
                break;
            case USERNAME_OK:
                switch(currentChar) {
                    case ':':
                        fprintf(stderr, "Invalid ftp url: Expecting password, ':' in the
wrong place\n");
                        return -1;
                    case '/':
                        fprintf(stderr, "Invalid ftp url: Expecting password, '/' in the
wrong place\n");
                        return -1;
                    case '@':
                        copyBufferToVar(buffer, bufferSize, password);
                        state = PASSWORD_OK;
                        break;
                    default: break;
                }
        }
    }
}

```

```

        }
        break;
    case PASSWORD_OK:
        switch(currentChar) {
            case '/':
                copyBufferToVar(buffer, bufferSize, host);
                state = HOST_OK;
                break;
            case '@':
                fprintf(stderr, "Invalid ftp url: Expecting host, '@' in the
wrong place\n");
                return -1;
            default: break;
        }
        break;
    case HOST_OK:
        if (i == (unsigned) strlen(url)) {
            copyBufferToVar(buffer, bufferSize, urlPath);
            char * tempUrlPath = malloc(strlen(*urlPath)*sizeof(char));
            strcpy(tempUrlPath, *urlPath);
            char * tok = strtok(tempUrlPath, "/");
            while (tok != NULL) {
                free(*filename);
                *filename = malloc(strlen(tok)*sizeof(char));
                strcpy(*filename, tok);

                tok = strtok(NULL, "/");
            }
            free(tempUrlPath);
            free(buffer);
            return 0;
        }
        break;
    default:
        fprintf(stderr, "Invalid urlParsingState.\n");
        break;
    }
    if (prevState == state) {
        buffer[strlen(buffer)] = currentChar;
        buffer[strlen(buffer)+1] = '\0';
    }
}
return -1;
}

```

```

int parsePsv(char *psvReply, char *ip, unsigned int *porta){ // 227 Entering Passive
Mode (193,136,28,12,19,91)
    unsigned int ip1, ip2, ip3, ip4, p1, p2;

    if(sscanf(psvReply, "227 Entering Passive Mode (%u, %u, %u, %u, %u, %u)",&ip1, &ip2,
&ip3, &ip4, &p1, &p2) != 6){
        fprintf(stderr, "Error in parsing pasv\n");
        return -1;
    }
}

```

```

        sprintf(ip, "%u.%u.%u.%u", ip1, ip2, ip3, ip4);
        *porta = p1*256 + p2;

    return 0;
}

## parser.h
#include "commandManager.h"

#define URL_PROTOCOL "ftp://"

enum urlParsingState {
    PROTOCOL_OK,
    USERNAME_OK,
    PASSWORD_OK,
    HOST_OK,
    PATH_OK
};

/***
 * Parses the received url into the necessary variables
 */
int parseURL(char *url, char **user, char **password, char **host, char **urlPath, char **filename);
/***
 * Parses the response from the server and calculates the port in which the server is
 * waiting for connection
 */
int parsePsv(char *psvReply, char *ip, unsigned int *porta);

void copyBufferToVar(char * buffer, size_t bufferSize, char ** out);

## fileManager.c
#include "fileManager.h"

int receiveFile(int dataSocketFD, char * filename) {
    unsigned prevFileBufferSize = REPLY_SIZE*sizeof(char);
    char * file = malloc(prevFileBufferSize);
    int n;

    while((n = read(dataSocketFD, file + prevFileBufferSize - (REPLY_SIZE*sizeof(char)), REPLY_SIZE)) != 0) {
        if (n == -1) {
            perror("receiveFile > read()");
            return 1;
        }
        file = realloc(file, prevFileBufferSize + n);
        prevFileBufferSize += n*sizeof(char);
    }

    FILE * output = fopen(filename, "w");

    fwrite(file, sizeof(char), prevFileBufferSize-REPLY_SIZE, output);
}

```

```

fclose(output);

close(dataSocketFD);

return 0;
}

int fileTransfer(int socketFD, char * user, char * password, char * host, char *
urlPath, char * filename){

if (loginHost(socketFD, user, password)) return EXIT_FAILURE;

unsigned int port;
char * ip = malloc(IP_SIZE*sizeof(char));
if (passiveMode(socketFD, ip, &port)) return EXIT_FAILURE;

int dataSocketFD = connectToIP(ip, port);
if (binaryMode(socketFD, ip, &port)) return EXIT_FAILURE;
if (retrCommand(socketFD, urlPath)) return EXIT_FAILURE;

if (receiveFile(dataSocketFD, filename)) return EXIT_FAILURE;

if (disconnect(socketFD)) return EXIT_FAILURE;

return 0;
}

## fileManager.h
#include "connection.h"

/*
 * Function that takes care of the download of the file
 */
int receiveFile(int dataSocketFD, char * filename);

/*
 * Function that calls all functions where the commands are executed
 */
int fileTransfer(int socketFD, char * user, char * password, char * host, char *
urlPath, char * filename);

## connection.c
#include "connection.h"

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

    if ((h = gethostbyname(hostname)) == NULL) {
        perror("Error getting host by name");
        return NULL;
    }
}

```

```

printf("Host 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));
}

int connectToIP(char * ip, uint16_t port) {
    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);           /*32 bit Internet address
network byte ordered*/
    server_addr.sin_port = htons(port);                  /*server TCP port must be
network byte ordered */

    int socketFD;
    /*open an TCP socket*/
    if ((socketFD = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("connectToIP > socket()");
        exit(EXIT_FAILURE);
    }
    /*connect to the server*/
    if(connect(socketFD, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0){
        perror("connectToIP > connect()");
        exit(EXIT_FAILURE);
    }

    return socketFD;
}

int loginHost(int socketFD, char *user, char *password){
    //check if there is a user
    int hasUser = user == NULL ? 0 : 1;

    //user command
    unsigned userCommandSize = hasUser ? strlen(user)+5 : 12;
    char *userCommand = malloc(sizeof(char)*userCommandSize);
    userCommand[0] = '\0';
    strcat(userCommand, "user ");

    if(hasUser) strcat(userCommand, user);
    else strcat(userCommand, "anonymous");

    unsigned passwordCommandSize = hasUser ? strlen(password)+5 : 11;
    char * passwordCommand = malloc(sizeof(char)*passwordCommandSize);
    passwordCommand[0] = '\0';
    strcat(passwordCommand, "pass ");

    if (hasUser) strcat(passwordCommand, password);
    else strcat(passwordCommand, "random");
}

```

```

//send user command
if(sendCommand(socketFD, userCommand) != 0){
    fprintf(stderr,"Error sending user command: %s\n", userCommand);
    free(userCommand);
    return EXIT_FAILURE;
}

printf("User command: %s\n", userCommand);
free(userCommand);

//read reply
unsigned replyCode = 0;
readReply(socketFD, &replyCode, NULL);
if(replyCode != USER_OK && replyCode != SUCESSFULL_LOGIN){
    fprintf(stderr,"Error reading user command reply\n");
    return EXIT_FAILURE;
}
if (replyCode != SUCESSFULL_LOGIN) {
    if(sendCommand(socketFD, passwordCommand) != 0){ //send password command
        fprintf(stderr,"Error sending password command: %s\n", passwordCommand);
        free(passwordCommand);
        return EXIT_FAILURE;
    }

    printf("Password command: %s \n", passwordCommand);
    free(passwordCommand);

    //read command answer
    replyCode = 0;
    readReply(socketFD, &replyCode, NULL);

    if (replyCode != SUCESSFULL_LOGIN){
        fprintf(stderr,"Error reading password command reply\n");
        return EXIT_FAILURE;
    }
}
return 0;
}

int passiveMode(int socketFD, char *ip, unsigned int *porta){
    char *pasvCommand = malloc(sizeof(char)* (strlen("pasv")+2));
    pasvCommand[0] = '\0';
    sprintf(pasvCommand, "pasv \n");

    //send pasv command
    if(sendCommand(socketFD,pasvCommand) != 0){
        fprintf(stderr,"Error sending pasv command\n");
        free(pasvCommand);
        return EXIT_FAILURE;
    }

    unsigned replyCode = 0;
    char * reply = malloc(REPLY_SIZE);

```

```

//read response < 227 Entering Passive Mode (193,136,28,12,19,91)
readReply(socketFD, &replyCode, reply);

if(replyCode != PASSIVE_MODE){
    fprintf(stderr,"Error reading pasv command reply\n");
    return EXIT_FAILURE;
}

if(parsePsv(reply, ip, porta) != 0){
    return EXIT_FAILURE;
}

return 0;
}

int binaryMode(int socketFD, char *ip, unsigned int *porta){
    char *bynaryCommand = malloc(sizeof(char)* (strlen("pasv")+2));
    bynaryCommand[0] = '\0';
    sprintf(bynaryCommand, "type I\n");

    //send pasv command
    if(sendCommand(socketFD,bynaryCommand) != 0){
        fprintf(stderr,"Error sending binary mode command\n");
        free(bynaryCommand);
        return EXIT_FAILURE;
    }
    printf("Binary command: %s \n", bynaryCommand);
    free(bynaryCommand);

    unsigned replyCode = 0;
    char * reply = malloc(REPLY_SIZE);
    readReply(socketFD, &replyCode, reply);

    if(replyCode != BINARY_MODE){
        fprintf(stderr,"Error reading binary command reply\n");
        return EXIT_FAILURE;
    }

    return 0;
}

int retrCommand(int socketFD, char*urlPath){
    char *retrCommand = malloc(sizeof(char)*(5+strlen(urlPath)));
    retrCommand[0] = '\0';
    strcat(retrCommand, "retr ");
    strcat(retrCommand,urlPath);

    if(sendCommand(socketFD, retrCommand) != 0){
        fprintf(stderr,"Error sending retr command: %s\n", retrCommand);
        free(retrCommand);
        return EXIT_FAILURE;
    }
}

```

```

printf("RetrCommand: %s\n", retrCommand);
free(retrCommand);

// Reads the 150 response
unsigned replyCode = 0;
if(readReply(socketFD, &replyCode, NULL) != 0){
    fprintf(stderr,"Error reading retr command reply\n");
    return EXIT_FAILURE;
} else if (replyCode == FAILED_OPEN_FILE) {
    fprintf(stderr, "The url path you inserted is not valid\n");
    return EXIT_FAILURE;
}

return 0;
}

int disconnect(int socketFD) {
char *quitCommand = malloc(sizeof(char)*(strlen("quit")+2));
quitCommand[0] = '\0';
sprintf(quitCommand, "quit \n");

// reads retr success
unsigned replyCode = 0;
if(readReply(socketFD, &replyCode, NULL) != 0){
    fprintf(stderr,"Error reading retr command reply\n");
    return EXIT_FAILURE;
}

if(sendCommand(socketFD, quitCommand) != 0){
    fprintf(stderr,"Error sending quit command: %s\n", quitCommand);
    free(quitCommand);
    return EXIT_FAILURE;
}
printf("QuitCommand: %s\n", quitCommand);
free(quitCommand);

replyCode = 0;
if(readReply(socketFD, &replyCode, NULL) != 0){
    fprintf(stderr,"Error reading quit command reply\n");
    return EXIT_FAILURE;
}

if(replyCode != QUIT_SUCESS){
    printf("%d\n", replyCode);
    fprintf(stderr,"Error reading quit command reply\n");
    return EXIT_FAILURE;
}

return 0;
}

## connection.h

#include "parser.h"

```

```

char *getIP(char * hostName);

int connectToIP(char * ip, uint16_t port);

int loginHost(int socketFD, char *user, char *password);

int passiveMode(int socketFD, char *ip, unsigned int *porta);

int retrCommand(int socketFD, char *urlPath);

int disconnect(int socketFD);

int binaryMode(int socketFD, char *ip, unsigned int *porta);

## commandManager.c
#include "commandManager.h"

int sendCommand(int socket, char* cmd){
    size_t cmdLength = strlen(cmd)+2;
    char * aux = calloc(cmdLength, sizeof(char));

    strcat(aux, cmd);
    strcat(aux, "\r\n"); // to add the endline in the end of the command
    if(write(socket,aux,cmdLength) != cmdLength){
        fprintf(stderr, "Error sending command %s\n", cmd);
        free(aux);
        return 1;
    }
    free(aux);

    return 0;
}

int readReply(int socketFD, unsigned * replyCode, char * reply) {
    FILE* socket = fdopen(socketFD, "r");

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

    size_t replySize = 0;
    char* buf = NULL;
    //printf("\n# Reading reply... \n");

    while(getline(&buf, &replySize, socket) > 0){
        printf("%s", buf);

        if(buf[3] == ' ') { // reply code has been read
            if (reply != NULL) strcpy(reply, buf);
            buf[3] = '\0';
            sscanf(buf,"%u", replyCode);
            break;
        }
    }
}

```

```

    return 0;
}

## commandManager.h
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include "defines.h"

/**
 * Sending and reading commands
 */

/**
 * Sends the command
 */
int sendCommand(int socket, char* cmd);
/**
 * Reads the server reply
 */
int readReply(int socketFD, unsigned * replyCode, char * reply);

## defines.h
#define FTP_COMMAND_PORT 21
#define REPLY_SIZE 512
#define IP_SIZE 30

#define SUCESSFULL_LOGIN 230
#define USER_OK 331
#define PASSIVE_MODE 227
#define QUIT_SUCESS 221
#define BINARY_MODE 200
#define TRANSFER_COMPLETE 226
#define FAILED_OPEN_FILE 550

```

Experiência 1

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|--------------|-------------------|------------------------|----------|--------|--|
| 17 | 20.199218002 | Cisco_7b:ce:81 | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/0:00:24:50:92:b9:80 Cost = 4 F |
| 18 | 20.254788473 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) request id=0x0917, seq=1/256, ttl=64 (|
| 19 | 20.254821579 | HewlettP_61:2d:df | Broadcast | ARP | 42 | Who has 172.16.60.254? Tell 172.16.60.1 |
| 20 | 20.254942967 | HewlettP_5a:79:97 | HewlettP_61:2d:df | ARP | 60 | 172.16.60.254 is at 00:21:5a:5a:79:97 |
| 21 | 20.254948764 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) reply id=0x0917, seq=1/256, ttl=64 (|
| 22 | 20.743864686 | Cisco_7b:ce:81 | Cisco_7b:ce:81 | LOOP | 60 | Reply |
| 23 | 21.255259284 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) request id=0x0917, seq=2/512, ttl=64 (|
| 24 | 21.255288199 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) reply id=0x0917, seq=2/512, ttl=64 (|
| 25 | 22.212492645 | Cisco_7b:ce:81 | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/0:00:24:50:92:b9:80 Cost = 4 F |
| 26 | 22.279301895 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) request id=0x0917, seq=3/768, ttl=64 (|
| 27 | 22.279331159 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) reply id=0x0917, seq=3/768, ttl=64 (|
| 28 | 23.303325445 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) request id=0x0917, seq=4/1024, ttl=64 |
| 29 | 23.303357433 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) reply id=0x0917, seq=4/1024, ttl=64 |
| 30 | 24.246297706 | Cisco_7b:ce:81 | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/0:00:24:50:92:b9:80 Cost = 4 F |
| 31 | 24.327361573 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) request id=5/1280, seq=5/1280, ttl=64 |
| 32 | 24.327390767 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) reply id=0x0917, seq=5/1280, ttl=64 |
| 33 | 25.287360959 | HewlettP_5a:79:97 | HewlettP_61:2d:df | ARP | 60 | Who has 172.16.60.1? Tell 172.16.60.254 |
| 34 | 25.287379258 | HewlettP_61:2d:df | HewlettP_5a:79:97 | ARP | 42 | 172.16.60.1 is at 00:21:5a:61:2d:df |
| 35 | 25.351396800 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) request id=0x0917, seq=6/1536, ttl=64 |
| 36 | 25.351416356 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) reply id=0x0917, seq=6/1536, ttl=64 |

Pacotes do tipo ICMP e ARP

```

Frame 23: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0
  ▶ Ethernet II, Src: HewlettP_5a:79:97 (00:21:5a:5a:79:97), Dst: HewlettP_61:2d:df (00:21:5a:61:2d:df)
    ▶ Destination: HewlettP_61:2d:df (00:21:5a:61:2d:df)
    ▶ Source: HewlettP_5a:79:97 (00:21:5a:5a:79:97)
      Type: IPv4 (0x0800)
  ▶ Internet Protocol Version 4, Src: 172.16.60.254, Dst: 172.16.60.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 84
    Identification: 0x7e7a (32378)
  ▶ Flags: 0x4000, Don't fragment
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)
    Header checksum: 0xebb0e [validation disabled]
    [Header checksum status: Unverified]
    Source: 172.16.60.254

Frame 21: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0
  ▶ Interface id: 0 (eth0)
  ▶ Encapsulation type: Ethernet (1)
  ▶ Arrival Time: Nov 24, 2020 12:27:54.176508203 WET
    [Time shift for this packet: 0.000000000 seconds]
  ▶ Epoch Time: 1606220874.176508203 seconds
    [Time delta from previous captured frame: 0.000005797 seconds]
    [Time delta from previous displayed frame: 0.000005797 seconds]
    [Time since reference or first frame: 20.254948764 seconds]
  ▶ Frame Number: 21
  ▶ Frame Length: 98 bytes (784 bits)
  ▶ Capture Length: 98 bytes (784 bits)
  ▶ [Frame is marked: False]
  ▶ [Frame is ignored: False]
  ▶ [Protocols in frame: eth:ether:type:ip:icmp:data]
  ▶ [Coloring Rule Name: ICMP]
  ▶ [Coloring Rule String: icmp || icmpv6]
  ▶ 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: 172.16.60.254
  ▶ Internet Control Message Protocol

```

Campo frame length

Experiência 2

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|--------------|-------------------|------------------------|----------|--------|---|
| 22 | 31.421180733 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) request id=0x07a5, seq=1/256, t |
| 23 | 31.421356682 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x07a5, seq=1/256, t |
| 24 | 32.076173550 | Cisco_7b:ce:8b | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos |
| 25 | 32.428409022 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) request id=0x07a5, seq=2/512, t |
| 26 | 32.428567997 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x07a5, seq=2/512, t |
| 27 | 33.452408598 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) request id=0x07a5, seq=3/768, t |
| 28 | 33.452580566 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x07a5, seq=3/768, t |
| 29 | 34.081296462 | Cisco_7b:ce:8b | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos |
| 30 | 34.476395183 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) request id=0x07a5, seq=4/1024, |
| 31 | 34.476553390 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x07a5, seq=4/1024, |
| 32 | 35.500398671 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) request id=0x07a5, seq=5/1280, |
| 33 | 35.500552338 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x07a5, seq=5/1280, |
| 34 | 36.086507523 | Cisco_7b:ce:8b | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos |
| 35 | 36.460371627 | HewlettP_61:2d:df | HewlettP_5a:79:97 | ARP | 42 | Who has 172.16.60.254? Tell 172.16.60.1 |
| 36 | 36.460502384 | HewlettP_5a:79:97 | HewlettP_61:2d:df | ARP | 60 | 172.16.60.254 is at 00:21:5a:5a:79:97 |
| 37 | 36.516373057 | HewlettP_5a:79:97 | HewlettP_61:2d:df | ARP | 60 | Who has 172.16.60.1? Tell 172.16.60.254 |
| 38 | 36.516382487 | HewlettP_61:2d:df | HewlettP_5a:79:97 | ARP | 42 | 172.16.60.1 is at 00:21:5a:61:2d:df |
| 39 | 36.524393280 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) request id=0x07a5, seq=6/1536, |
| 40 | 36.524525302 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x07a5, seq=6/1536, |
| 41 | 36.577313765 | Cisco_7b:ce:8b | Cisco_7b:ce:8b | LOOP | 60 | Reply |
| 42 | 37.548404600 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) request id=0x07a5, seq=7/1792, |
| 43 | 37.548557778 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x07a5, seq=7/1792, |
| 44 | 38.091878327 | Cisco_7b:ce:8b | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos |
| 45 | 38.572397610 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) request id=0x07a5, seq=8/2048, |
| 46 | 38.572566574 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x07a5, seq=8/2048, |
| 47 | 39.596394393 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) request id=0x07a5, seq=9/2304, |
| 48 | 39.596547711 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x07a5, seq=9/2304, |
| 49 | 40.102125056 | Cisco_7b:ce:8b | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos |
| 50 | 40.620402072 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) request id=0x07a5, seq=10/2560, |
| 51 | 40.620551897 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x07a5, seq=10/2560, |
| 52 | 41.624410200 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) request id=0x07a5, seq=11/2816 |

Ping a partir do tux63 para o tux64

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|---------------|---------------------------------|------------------------|----------|--|--|
| 10 | 14.034224793 | Cisco_7b:ce:81 | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/61/00:1e:14:7b:ce:80 Cos |
| 11 | 16.039103184 | Cisco_7b:ce:81 | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/61/00:1e:14:7b:ce:80 Cos |
| 12 | 18.044001479 | Cisco_7b:ce:81 | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/61/00:1e:14:7b:ce:80 Cos |
| 13 | 20.048899496 | Cisco_7b:ce:81 | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/61/00:1e:14:7b:ce:80 Cos |
| 14 | 22.016073991 | Cisco_7b:ce:81 | Cisco_7b:ce:81 | LOOP | 60 | Reply |
| 15 | 22.05851999 | Cisco_7b:ce:81 | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/61/00:1e:14:7b:ce:80 Cos |
| 16 | 23.567951516 | 172.16.61.1 | 172.16.61.255 | ICMP | 98 | Echo (ping) request id=0x0c34, seq=1/256, t |
| 17 | 24.058722836 | Cisco_7b:ce:81 | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/61/00:1e:14:7b:ce:80 Cos |
| 18 | 24.575146773 | 172.16.61.1 | 172.16.61.255 | ICMP | 98 | Echo (ping) request id=0x0c34, seq=2/512, t |
| 19 | 25.599147484 | 172.16.61.1 | 172.16.61.255 | ICMP | 98 | Echo (ping) request id=0x0c34, seq=3/768, t |
| 20 | 26.063569519 | Cisco_7b:ce:81 | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/61/00:1e:14:7b:ce:80 Cos |
| 21 | 26.623153014 | 172.16.61.1 | 172.16.61.255 | ICMP | 98 | Echo (ping) request id=0x0c34, seq=4/1024, |
| 22 | 27.647151910 | 172.16.61.1 | 172.16.61.255 | ICMP | 98 | Echo (ping) request id=0x0c34, seq=5/1280, |
| 23 | 28.0668478989 | Cisco_7b:ce:81 | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/61/00:1e:14:7b:ce:80 Cos |
| 24 | 28.671143681 | 172.16.61.1 | 172.16.61.255 | ICMP | 98 | Echo (ping) request id=0x0c34, seq=6/1536, |
| 25 | 29.695142157 | 172.16.61.1 | 172.16.61.255 | ICMP | 98 | Echo (ping) request id=0x0c34, seq=7/1792, |
| 26 | 30.073393627 | Cisco_7b:ce:81 | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/61/00:1e:14:7b:ce:80 Cos |
| 27 | 30.719388989 | 172.16.61.1 | 172.16.61.255 | ICMP | 98 | Echo (ping) request id=0x0c34, seq=8/2048, |
| 28 | 31.743136944 | 172.16.61.1 | 172.16.61.255 | ICMP | 98 | Echo (ping) request id=0x0c34, seq=9/2304, |
| 29 | 32.019769345 | Cisco_7b:ce:81 | Cisco_7b:ce:81 | LOOP | 60 | Reply |
| 30 | 32.082947692 | Cisco_7b:ce:81 | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/61/00:1e:14:7b:ce:80 Cos |
| 31 | 32.767147573 | 172.16.61.1 | 172.16.61.255 | ICMP | 98 | Echo (ping) request id=0x0c34, seq=10/2560, |
| 32 | 33.791138227 | 172.16.61.1 | 172.16.61.255 | ICMP | 98 | Echo (ping) request id=0x0c34, seq=11/2816, |
| 33 | 34.083166123 | Cisco_7b:ce:81 | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/61/00:1e:14:7b:ce:80 Cos |
| 34 | 34.611141763 | fe80::222:64ff:fe19... ff02::fb | MDNS | 180 | Standard query 0x0000 PTR _ftp._tcp.local, " | |
| 35 | 34.611222499 | 172.16.61.1 | 224.0.0.251 | MDNS | 160 | Standard query 0x0000 PTR _ftp._tcp.local, " |
| 36 | 34.815122386 | 172.16.61.1 | 172.16.61.255 | ICMP | 98 | Echo (ping) request id=0x0c34, seq=12/3072, |
| 37 | 36.090098690 | Cisco_7b:ce:81 | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/61/00:1e:14:7b:ce:80 Cos |
| 38 | 38.084901756 | Cisco_7b:ce:81 | CDP/VTP/DTP/PAgP/UD... | CDP | 601 | Device ID: gnu-sw6 Port ID: FastEthernet0/1 |
| 39 | 38.093066359 | Cisco_7b:ce:81 | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/61/00:1e:14:7b:ce:80 Cos |

Ping Broadcast (ping -b 172.16.61.255) a partir do tux62

Experiência 3

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|--------------|-------------------|---------------------------|--|--|------|
| 70 | 46.118525224 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | 60 Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos | | |
| 71 | 48.128119938 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | 60 Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos | | |
| 72 | 50.128836537 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | 60 Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos | | |
| 73 | 51.301926151 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 Echo (ping) request id=0x145e, seq=1/256, t | |
| 74 | 51.302220428 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 Echo (ping) reply id=0x145e, seq=1/256, t | |
| 75 | 52.133964917 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | 60 Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos | | |
| 76 | 52.327220923 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 Echo (ping) request id=0x145e, seq=2/512, t | |
| 77 | 52.327460298 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 Echo (ping) reply id=0x145e, seq=2/512, t | |
| 78 | 53.351229410 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 Echo (ping) request id=0x145e, seq=3/768, t | |
| 79 | 53.351470182 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 Echo (ping) reply id=0x145e, seq=3/768, t | |
| 80 | 54.042767603 | Cisco_7b:ce:85 | Cisco_7b:ce:85 | LOOP | 60 Reply | |
| 81 | 54.139129758 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | 60 Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos | | |
| 82 | 54.375226373 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 Echo (ping) request id=0x145e, seq=4/1024, | |
| 83 | 54.375483420 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 Echo (ping) reply id=0x145e, seq=4/1024, | |
| 84 | 55.399254767 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 Echo (ping) request id=0x145e, seq=5/1280, | |
| 85 | 55.399494771 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 Echo (ping) reply id=0x145e, seq=5/1280, | |
| 86 | 56.144270431 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | 60 Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos | | |
| 87 | 56.391196344 | HewlettP_61:2d:df | HewlettP_5a:79:97 | ARP | 42 Who has 172.16.60.254? Tell 172.16.60.1 | |
| 88 | 56.391305380 | HewlettP_5a:79:97 | HewlettP_61:2d:df | ARP | 60 172.16.60.254 is at 00:21:5a:5a:79:97 | |
| 89 | 56.423222113 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 Echo (ping) request id=0x145e, seq=6/1536, | |
| 90 | 56.423456739 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 Echo (ping) reply id=0x145e, seq=6/1536, | |
| 91 | 57.447245758 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 Echo (ping) request id=0x145e, seq=7/1792, | |
| 92 | 57.447485692 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 Echo (ping) reply id=0x145e, seq=7/1792, | |
| 93 | 58.153572963 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | 60 Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos | | |
| 94 | 58.428661464 | HewlettP_5a:79:97 | HewlettP_61:2d:df | ARP | 60 Who has 172.16.60.1? Tell 172.16.60.254 | |
| 95 | 58.428679486 | HewlettP_61:2d:df | HewlettP_5a:79:97 | ARP | 42 172.16.60.1 is at 00:21:5a:61:2d:df | |
| 96 | 60.154574549 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | 60 Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos | | |
| 97 | 62.159784933 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | 60 Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos | | |
| 98 | 64.047426492 | Cisco_7b:ce:85 | Cisco_7b:ce:85 | LOOP | 60 Reply | |
| 99 | 64.164866373 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | 60 Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos | | |
| 100 | 65.170007144 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | 60 Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos | | |

Ping de tux63 para tux62

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|---------------|-------------------|---------------------------|--|---|------|
| 62 | 100.056636480 | Cisco_7b:ce:89 | Cisco_7b:ce:89 | LOOP | 60 Reply | |
| 63 | 101.361057158 | Cisco_7b:ce:89 | Spanning-tree-(for... STP | 60 Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos | | |
| 64 | 103.369923907 | Cisco_7b:ce:89 | Spanning-tree-(for... STP | 60 Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos | | |
| 65 | 105.370825718 | Cisco_7b:ce:89 | Spanning-tree-(for... STP | 60 Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos | | |
| 66 | 106.604382660 | HewlettP_61:2d:df | Broadcast | ARP | 60 Who has 172.16.60.254? Tell 172.16.60.1 | |
| 67 | 106.604411853 | HewlettP_5a:79:97 | HewlettP_61:2d:df | ARP | 42 172.16.60.254 is at 00:21:5a:5a:79:97 | |
| 68 | 106.604539383 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 Echo (ping) request id=0x1632, seq=1/256, t | |
| 69 | 106.604790391 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 Echo (ping) reply id=0x1632, seq=1/256, t | |
| 70 | 106.730294657 | Cisco_7b:ce:89 | CDP/VTTP/DTTP/PAgP/UD... | CDP | 601 Device ID: gnu-sw6 Port ID: FastEthernet0/9 | |
| 71 | 107.375960937 | Cisco_7b:ce:89 | Spanning-tree-(for... STP | 60 Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos | | |
| 72 | 107.627408140 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 Echo (ping) request id=0x1632, seq=2/512, t | |
| 73 | 107.627576108 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 Echo (ping) reply id=0x1632, seq=2/512, t | |
| 74 | 108.651283375 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 Echo (ping) request id=0x1632, seq=3/768, t | |
| 75 | 108.651427527 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 Echo (ping) reply id=0x1632, seq=3/768, t | |
| 76 | 109.380773910 | Cisco_7b:ce:89 | Spanning-tree-(for... STP | 60 Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos | | |
| 77 | 109.675129417 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 Echo (ping) request id=0x1632, seq=4/1024, | |
| 78 | 109.675269937 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 Echo (ping) reply id=0x1632, seq=4/1024, | |
| 79 | 110.059822263 | Cisco_7b:ce:89 | Cisco_7b:ce:89 | LOOP | 60 Reply | |
| 80 | 110.699029096 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 Echo (ping) request id=0x1632, seq=5/1280, | |
| 81 | 110.699172549 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 Echo (ping) reply id=0x1632, seq=5/1280, | |
| 82 | 111.385564744 | Cisco_7b:ce:89 | Spanning-tree-(for... STP | 60 Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos | | |
| 83 | 111.722869690 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 Echo (ping) request id=0x1632, seq=6/1536, | |
| 84 | 111.723004622 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 Echo (ping) reply id=0x1632, seq=6/1536, | |
| 85 | 111.761532573 | HewlettP_5a:79:97 | HewlettP_61:2d:df | ARP | 42 Who has 172.16.60.1? Tell 172.16.60.254 | |
| 86 | 111.761658008 | HewlettP_61:2d:df | HewlettP_5a:79:97 | ARP | 60 172.16.60.1 is at 00:21:5a:61:2d:df | |
| 87 | 112.746753026 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 Echo (ping) request id=0x1632, seq=7/1792, | |
| 88 | 112.746920156 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 Echo (ping) reply id=0x1632, seq=7/1792, | |
| 89 | 113.394449302 | Cisco_7b:ce:89 | Spanning-tree-(for... STP | 60 Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos | | |
| 90 | 113.770622045 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 Echo (ping) request id=0x1632, seq=8/2048, | |
| 91 | 113.770763333 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 Echo (ping) reply id=0x1632, seq=8/2048, | |
| 92 | 114.784561617 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 Echo (ping) request id=0x1632, seq=9/2048, | |
| 93 | 114.784561617 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 Echo (ping) reply id=0x1632, seq=9/2048, | |

Ping de tux63 para tux62, visto a partir do tux64 eth0

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|---------------|-------------------|---------------------------|----------|--------|--|
| 64 | 102.249827445 | Cisco_7b:ce:91 | Spanning-tree-(for... STP | | 60 | Conf. Root = 32768/61/00:1e:14:7b:ce:80 Cos |
| 65 | 104.254771230 | Cisco_7b:ce:91 | Spanning-tree-(for... STP | | 60 | Conf. Root = 32768/61/00:1e:14:7b:ce:80 Cos |
| 66 | 106.259723046 | Cisco_7b:ce:91 | Spanning-tree-(for... STP | | 60 | Conf. Root = 32768/61/00:1e:14:7b:ce:80 Cos |
| 67 | 107.468288045 | Netronix_71:73:da | Broadcast | ARP | 42 | Who has 172.16.61.1? Tell 172.16.61.253 |
| 68 | 107.468466146 | HewlettP_19:02:ba | Netronix_71:73:da | ARP | 60 | 172.16.61.1 is at 00:22:64:19:02:ba |
| 69 | 107.468413060 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 | Echo (ping) request id=0x1632, seq=1/256, t |
| 70 | 107.468518659 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1632, seq=1/256, t |
| 71 | 107.594419133 | Cisco_7b:ce:91 | CDP/VTP/DTP/PAgP/UD... | CDP | 602 | Device ID: gnu-sw6 Port ID: FastEthernet0/1 |
| 72 | 108.264638616 | Cisco_7b:ce:91 | Spanning-tree-(for... STP | | 60 | Conf. Root = 32768/61/00:1e:14:7b:ce:80 Cos |
| 73 | 108.491163297 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 | Echo (ping) request id=0x1632, seq=2/512, t |
| 74 | 108.491302560 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1632, seq=2/512, t |
| 75 | 109.515036297 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 | Echo (ping) request id=0x1632, seq=3/768, t |
| 76 | 109.515153839 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1632, seq=3/768, t |
| 77 | 110.273859109 | Cisco_7b:ce:91 | Spanning-tree-(for... STP | | 60 | Conf. Root = 32768/61/00:1e:14:7b:ce:80 Cos |
| 78 | 110.538879964 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 | Echo (ping) request id=0x1632, seq=4/1024, t |
| 79 | 110.538996319 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1632, seq=4/1024, |
| 80 | 110.932056665 | Cisco_7b:ce:91 | Cisco_7b:ce:91 | LOOP | 60 | Reply |
| 81 | 111.562784183 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 | Echo (ping) request id=0x1632, seq=5/1280, t |
| 82 | 111.562898373 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1632, seq=5/1280, t |
| 83 | 112.274321272 | Cisco_7b:ce:91 | Spanning-tree-(for... STP | | 60 | Conf. Root = 32768/61/00:1e:14:7b:ce:80 Cos |
| 84 | 112.586621354 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 | Echo (ping) request id=0x1632, seq=6/1536, t |
| 85 | 112.586730306 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1632, seq=6/1536, t |
| 86 | 112.624942856 | HewlettP_19:02:ba | Netronix_71:73:da | ARP | 60 | Who has 172.16.61.253? Tell 172.16.61.1 |
| 87 | 112.624950748 | Netronix_71:73:da | HewlettP_19:02:ba | ARP | 42 | 172.16.61.253 is at 00:08:54:71:73:da |
| 88 | 113.610507345 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 | Echo (ping) request id=0x1632, seq=7/1792, t |
| 89 | 113.610647236 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1632, seq=7/1792, t |
| 90 | 114.279329660 | Cisco_7b:ce:91 | Spanning-tree-(for... STP | | 60 | Conf. Root = 32768/61/00:1e:14:7b:ce:80 Cos |
| 91 | 114.634375456 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 | Echo (ping) request id=0x1632, seq=8/2048, t |
| 92 | 114.634489506 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1632, seq=8/2048, t |
| 93 | 115.658287427 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 | Echo (ping) request id=0x1632, seq=9/2304, t |
| 94 | 115.658290022 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1632, seq=9/2304, t |

Ping de tux63 para tux62, visto a partir do tux64 eth1

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|--------------|-------------------|---------------------------|----------|--------|--|
| 9 | 14.010882663 | Cisco_7b:ce:85 | Cisco_7b:ce:85 | LOOP | 60 | Reply |
| 10 | 14.036483092 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | | 60 | Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos |
| 11 | 15.305961195 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) request id=0x1447, seq=1/256, t |
| 12 | 15.306127297 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1447, seq=1/256, t |
| 13 | 16.041294354 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | | 60 | Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos |
| 14 | 16.327242817 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) request id=0x1447, seq=2/512, t |
| 15 | 16.327374833 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1447, seq=2/512, t |
| 16 | 17.351243132 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) request id=0x1447, seq=3/768, t |
| 17 | 17.351373332 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1447, seq=3/768, t |
| 18 | 18.050430154 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | | 60 | Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos |
| 19 | 18.375239536 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) request id=0x1447, seq=4/1024, t |
| 20 | 18.375370155 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1447, seq=4/1024, t |
| 21 | 19.399233844 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) request id=0x1447, seq=5/1280, t |
| 22 | 19.399389330 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1447, seq=5/1280, t |
| 23 | 20.051520100 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | | 60 | Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos |
| 24 | 20.423244497 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) request id=0x1447, seq=6/1536, t |
| 25 | 20.423379656 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1447, seq=6/1536, t |
| 26 | 20.535908610 | HewlettP_5a:79:97 | HewlettP_61:2d:df | ARP | 60 | Who has 172.16.60.1? Tell 172.16.60.254 |
| 27 | 20.535928587 | HewlettP_61:2d:df | HewlettP_5a:79:97 | ARP | 42 | 172.16.60.1 is at 00:21:5a:61:2d:df |
| 28 | 20.551198444 | HewlettP_61:2d:df | HewlettP_5a:79:97 | ARP | 42 | Who has 172.16.60.254? Tell 172.16.60.1 |
| 29 | 20.551291833 | HewlettP_5a:79:97 | HewlettP_61:2d:df | ARP | 60 | 172.16.60.254 is at 00:21:5a:5a:79:97 |
| 30 | 21.447229585 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) request id=0x1447, seq=7/1792, t |
| 31 | 21.447358667 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1447, seq=7/1792, t |
| 32 | 22.056686268 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | | 60 | Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos |
| 33 | 22.471224452 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) request id=0x1447, seq=8/2048, t |
| 34 | 22.471357166 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1447, seq=8/2048, t |
| 35 | 23.495248376 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) request id=0x1447, seq=9/2304, t |
| 36 | 23.495379484 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1447, seq=9/2304, t |
| 37 | 24.019861341 | Cisco_7b:ce:85 | Cisco_7b:ce:85 | LOOP | 60 | Reply |
| 38 | 24.062028249 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | | 60 | Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cos |

Ping do tux63 para o tux64 (eth0)

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|--------------|----------------|---------------------------|----------|---|---|
| 42 | 32.082430358 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | 60 | Conf. Root = 32768/60/00:1e:14:7b:ce:80 | Cos |
| 43 | 34.020506314 | Cisco_7b:ce:85 | Cisco_7b:ce:85 | LOOP | 60 | Reply |
| 44 | 34.087571590 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | 60 | Conf. Root = 32768/60/00:1e:14:7b:ce:80 | Cos |
| 45 | 36.092744045 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | 60 | Conf. Root = 32768/60/00:1e:14:7b:ce:80 | Cos |
| 46 | 37.068428578 | 172.16.60.1 | 172.16.61.253 | ICMP | 98 | Echo (ping) request id=0x1454, seq=1/256, t |
| 47 | 37.068593074 | 172.16.61.253 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1454, seq=1/256, t |
| 48 | 38.087240662 | 172.16.60.1 | 172.16.61.253 | ICMP | 98 | Echo (ping) request id=0x1454, seq=2/512, t |
| 49 | 38.087374424 | 172.16.61.253 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1454, seq=2/512, t |
| 50 | 38.102541393 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | 60 | Conf. Root = 32768/60/00:1e:14:7b:ce:80 | Cos |
| 51 | 39.111242583 | 172.16.60.1 | 172.16.61.253 | ICMP | 98 | Echo (ping) request id=0x1454, seq=3/768, t |
| 52 | 39.111374879 | 172.16.61.253 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1454, seq=3/768, t |
| 53 | 40.103027138 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | 60 | Conf. Root = 32768/60/00:1e:14:7b:ce:80 | Cos |
| 54 | 40.135232561 | 172.16.60.1 | 172.16.61.253 | ICMP | 98 | Echo (ping) request id=0x1454, seq=4/1024, |
| 55 | 40.135382109 | 172.16.61.253 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1454, seq=4/1024, |
| 56 | 41.026252120 | Cisco_7b:ce:85 | CDP/FTP/DTP/PAgP/UD... | CDP | 601 | Device ID: gnu-sw6 Port ID: FastEthernet0/5 |
| 57 | 41.159226729 | 172.16.60.1 | 172.16.61.253 | ICMP | 98 | Echo (ping) request id=0x1454, seq=5/1280, |
| 58 | 41.159358326 | 172.16.61.253 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1454, seq=5/1280, |
| 59 | 42.108195681 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | 60 | Conf. Root = 32768/60/00:1e:14:7b:ce:80 | Cos |
| 60 | 42.183219570 | 172.16.60.1 | 172.16.61.253 | ICMP | 98 | Echo (ping) request id=0x1454, seq=6/1536, |
| 61 | 42.183350958 | 172.16.61.253 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1454, seq=6/1536, |
| 62 | 43.207229804 | 172.16.60.1 | 172.16.61.253 | ICMP | 98 | Echo (ping) request id=0x1454, seq=7/1792, |
| 63 | 43.207362100 | 172.16.61.253 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1454, seq=7/1792, |
| 64 | 44.029451464 | Cisco_7b:ce:85 | Cisco_7b:ce:85 | LOOP | 60 | Reply |
| 65 | 44.113642995 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | 60 | Conf. Root = 32768/60/00:1e:14:7b:ce:80 | Cos |
| 66 | 44.235233349 | 172.16.60.1 | 172.16.61.253 | ICMP | 98 | Echo (ping) request id=0x1454, seq=8/2048, |
| 67 | 44.235366273 | 172.16.61.253 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1454, seq=8/2048, |
| 68 | 45.255236372 | 172.16.60.1 | 172.16.61.253 | ICMP | 98 | Echo (ping) request id=0x1454, seq=9/2304, |
| 69 | 45.255387457 | 172.16.61.253 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x1454, seq=9/2304, |
| 70 | 46.118525224 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | 60 | Conf. Root = 32768/60/00:1e:14:7b:ce:80 | Cos |
| 71 | 48.128119938 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | 60 | Conf. Root = 32768/60/00:1e:14:7b:ce:80 | Cos |
| 72 | 50.128826527 | Cisco_7b:ce:85 | Spanning-tree-(for... STP | 60 | Conf. Root = 32768/60/00:1e:14:7b:ce:80 | Cos |

Ping do tux63 para o tux64 (eth1)

| Destination | Gateway | Genmask | Flags | Metric | Ref | Use | Iface |
|-------------|---------------|---------------|-------|--------|-----|-----|-------|
| 172.16.60.0 | 0.0.0.0 | 255.255.255.0 | U | 0 | 0 | 0 | eth0 |
| 172.16.61.0 | 172.16.60.254 | 255.255.255.0 | UG | 0 | 0 | 0 | eth0 |

Rotas a partir do tux63

Experiência 4

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|--------------|----------------|------------------------|----------|--------|------------------------------|
| 20 | 24.452747671 | Cisco_7b:ce:81 | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/61/00:1e: |
| 21 | 25.770333283 | Cisco_7b:ce:81 | CDP/VTP/DTP/PAgP/UD... | CDP | 601 | Device ID: gnu-sw6 Port ID: |
| 22 | 26.457648056 | Cisco_7b:ce:81 | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/61/00:1e: |
| 23 | 28.466651394 | Cisco_7b:ce:81 | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/61/00:1e: |
| 24 | 30.018755427 | Cisco_7b:ce:81 | Cisco_7b:ce:81 | LOOP | 60 | Reply |
| 25 | 30.467352097 | Cisco_7b:ce:81 | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/61/00:1e: |
| 26 | 30.880675197 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 | Echo (ping) request id=0x0f |
| 27 | 30.881421169 | 172.16.61.254 | 172.16.61.1 | ICMP | 70 | Redirect (Redirect |
| 28 | 30.881780711 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 | Echo (ping) reply id=0x0f |
| 29 | 31.881856063 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 | Echo (ping) request id=0x0f |
| 30 | 31.882606366 | 172.16.61.254 | 172.16.61.1 | ICMP | 70 | Redirect (Redirect |
| 31 | 31.882900466 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 | Echo (ping) reply id=0x0f |
| 32 | 32.472298717 | Cisco_7b:ce:81 | Spanning-tree-(for...) | STP | 60 | Conf. Root = 32768/61/00:1e: |
| 33 | 32.883004943 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 | Echo (ping) request id=0x0f |
| 34 | 32.883816705 | 172.16.61.254 | 172.16.61.1 | ICMP | 70 | Redirect (Redirect |
| 35 | 32.884108641 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 | Echo (ping) reply id=0x0f |
| 36 | 33.884181968 | 172.16.61.1 | 172.16.60.1 | ICMP | 98 | Echo (ping) request id=0x0f |
| 37 | 33.884929966 | 172.16.61.254 | 172.16.61.1 | ICMP | 70 | Redirect (Redirect |
| 38 | 33.885223577 | 172.16.60.1 | 172.16.61.1 | ICMP | 98 | Echo (ping) reply id=0x0f |
| 39 | 34.177121024 | Cisco_7b:ce:81 | Spanning tree /for | STD | 60 | Conf. Root = 32768/61/00:1e: |

Frame 27: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface eth0, id 0
 Ethernet II, Src: Cisco_96:eb:16 (00:1e:7a:96:eb:16), Dst: HewlettP_19:02:ba (00:22:64:19:02:ba)
 Internet Protocol Version 4, Src: 172.16.61.254, Dst: 172.16.61.1
 Internet Control Message Protocol
 Type: 5 (Redirect)
 Code: 1 (Redirect for host)
 Checksum: 0xf783 [correct]
 [Checksum Status: Good]
 Gateway address: 172.16.61.253
 Internet Protocol Version 4, Src: 172.16.61.1, Dst: 172.16.60.1

Ping do tux62 para o tux63 com *redirects* desativados

```
root@tux42:~/Desktop# traceroute 172.16.40.1
traceroute to 172.16.40.1 (172.16.40.1), 30 hops max, 60 byte packets
1  172.16.41.254 (172.16.41.254)  0.564 ms  0.624 ms  0.683 ms
2  172.16.41.253 (172.16.41.253)  0.820 ms  0.334 ms  0.330 ms
3  172.16.40.1 (172.16.40.1)  0.518 ms  0.507 ms  0.491 ms
```

Traceroute a partir do tuxy2 até ao tuxy3 sem rota para 172.16.y0.0/24 (*log* fornecido pelo grupo T2G8)

Devido a um lapso, não gravámos este log.

```
root@tux42:~/Desktop# traceroute 172.16.40.1
traceroute to 172.16.40.1 (172.16.40.1), 30 hops max, 60 byte packets
1  172.16.41.253 (172.16.41.253)  0.164 ms  0.144 ms  0.125 ms
2  172.16.40.1 (172.16.40.1)  0.337 ms  0.356 ms  0.339 ms
```

Traceroute a partir do tuxy2 até ao tuxy3 com rota para 172.16.y0.0/24 (*log* fornecido pelo grupo T2G8)

Devido a um lapso, não gravámos este log.

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|-------------|----------------|-------------------------|----------|--------|------------------------------------|
| 11 | 3.446468062 | 172.16.40.1 | 172.16.1.254 | ICMP | 98 | Echo (ping) request id: 0x00000000 |
| 12 | 3.447052346 | 172.16.1.254 | 172.16.40.1 | ICMP | 98 | Echo (ping) reply id: 0x00000000 |
| 13 | 3.777987516 | Cisco_d4:1c:03 | Cisco_d4:1c:03 | LOOP | 60 | Reply |
| 14 | 4.004447602 | Cisco_d4:1c:03 | Spanning-tree-(for-...) | STP | 60 | Conf. Root = 32768/40/30 |
| 15 | 4.470463302 | 172.16.40.1 | 172.16.1.254 | ICMP | 98 | Echo (ping) request id: 0x00000000 |
| 16 | 4.471055128 | 172.16.1.254 | 172.16.40.1 | ICMP | 98 | Echo (ping) reply id: 0x00000000 |
| 17 | 5.494464120 | 172.16.40.1 | 172.16.1.254 | ICMP | 98 | Echo (ping) request id: 0x00000000 |
| 18 | 5.495090028 | 172.16.1.254 | 172.16.40.1 | ICMP | 98 | Echo (ping) reply id: 0x00000000 |
| 19 | 6.009777571 | Cisco_d4:1c:03 | Spanning-tree-(for-...) | STP | 60 | Conf. Root = 32768/40/30 |
| 20 | 6.518468423 | 172.16.40.1 | 172.16.1.254 | ICMP | 98 | Echo (ping) request id: 0x00000000 |
| 21 | 6.519029729 | 172.16.1.254 | 172.16.40.1 | ICMP | 98 | Echo (ping) reply id: 0x00000000 |
| 22 | 6.554921170 | 172.16.40.1 | 172.16.1.1 | DNS | 86 | Standard query 0x33cb P |
| 23 | 6.556347935 | 172.16.1.1 | 172.16.40.1 | DNS | 422 | Standard query response |
| 24 | 7.542466363 | 172.16.40.1 | 172.16.1.254 | ICMP | 98 | Echo (ping) request id: 0x00000000 |
| 25 | 7.543080468 | 172.16.1.254 | 172.16.40.1 | ICMP | 98 | Echo (ping) reply id: 0x00000000 |
| 26 | 8.019623238 | Cisco_d4:1c:03 | Spanning-tree-(for-...) | STP | 60 | Conf. Root = 32768/40/30 |
| 27 | 8.566466808 | 172.16.40.1 | 172.16.1.254 | TCP | 98 | Echo (ping) request id: 0x00000000 |

```

> Frame 15: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0
> Ethernet II, Src: HewlettP_61:2f:d4 (00:21:5a:61:2f:d4), Dst: HewlettP_5a:7b:ea (00:21:5a:5a:7b:ea)
> Internet Protocol Version 4, Src: 172.16.40.1, Dst: 172.16.1.254
> Internet Control Message Protocol

```

Ping para o router do laboratório a partir do tuxy3 (*log fornecido pelo grupo T2G8*)

Devido a um lapso, não gravámos este log.

Experiência 5

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|--------------|-------------------|-------------------------|----------|--------|---|
| 1 | 0.0000000000 | HewlettP_61:2f:d4 | HewlettP_5a:7b:ea | ARP | 42 | Who has 172.16.40.254? Tell 172.16.40.1 |
| 2 | 0.000150645 | HewlettP_5a:7b:ea | HewlettP_61:2f:d4 | ARP | 60 | 172.16.40.254 is at 00:21:5a:5a:7b:ea |
| 3 | 0.017330267 | Cisco_d4:1c:03 | Spanning-tree-(for-...) | STP | 60 | Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 |
| 4 | 0.976350466 | 172.16.40.1 | 172.16.1.1 | DNS | 70 | Standard query 0xb667 A google.com |
| 5 | 0.976368694 | 172.16.40.1 | 172.16.1.1 | DNS | 70 | Standard query 0x747c AAAA google.com |
| 6 | 0.977824651 | 172.16.1.1 | 172.16.40.1 | DNS | 334 | Standard query response 0xb667 A google.com A 216 |
| 7 | 0.977875564 | 172.16.1.1 | 172.16.40.1 | DNS | 346 | Standard query response 0x747c AAAA google.com AA |
| 8 | 0.978244390 | 172.16.40.1 | 216.58.201.174 | ICMP | 98 | Echo (ping) request id=0x5076, seq=1/256, ttl=64 |
| 9 | 0.995135921 | 216.58.201.174 | 172.16.40.1 | ICMP | 98 | Echo (ping) reply id=0x5076, seq=1/256, ttl=11 |
| 10 | 0.995340692 | 172.16.40.1 | 172.16.1.1 | DNS | 87 | Standard query 0xdbd9 PTR 174.201.58.216.in-addr. |
| 11 | 0.996353025 | 172.16.1.1 | 172.16.40.1 | DNS | 442 | Standard query response 0xdbd9 PTR 174.201.58.216 |
| 12 | 1.004463837 | 172.16.40.1 | 172.16.1.1 | DNS | 88 | Standard query 0x1a18 PTR 174.184.250.142.in-addr. |
| 13 | 1.005402069 | 172.16.1.1 | 172.16.40.1 | DNS | 148 | Standard query response 0x1a18 No such name PTR 1 |
| 14 | 1.005552575 | 172.16.40.1 | 172.16.1.1 | DNS | 86 | Standard query 0x5bc2 PTR 26.114.82.140.in-addr.a |
| 15 | 1.006699210 | 172.16.1.1 | 172.16.40.1 | DNS | 422 | Standard query response 0x5bc2 PTR 26.114.82.140. |
| 16 | 1.539954910 | 172.16.40.1 | 140.82.114.26 | TCP | 66 | 34700 → 443 [ACK] Seq=1 Ack=1 Win=318 Len=0 TSval |
| 17 | 1.670942496 | 140.82.114.26 | 172.16.40.1 | TCP | 66 | 66 [TCP ACKed unseen segment] 443 → 34700 [ACK] Seq=1 |

```

Answer RRs: 1
Authority RRs: 4
Additional RRs: 8
Queries
  > google.com: type A, class IN
Answers
  > google.com: type A, class IN, addr 216.58.201.174
  > Authoritative nameservers
  > Additional records

```

Pedido de DNS (*log fornecido pelo grupo T2G8*)

Devido a um lapso, não gravámos este log.

Experiência 6

| No. | Time | Source | Destination | Protocol | Length | Info |
|-------|--------------|---------------|---------------|-----------|--------|-------------------------------------|
| 18322 | 17.695336977 | 192.168.1.20 | 193.137.29.15 | TCP | 66 | 36354 → 58686 [ACK] Seq=1 Ack=1041 |
| 18323 | 17.702403421 | 193.137.29.15 | 192.168.1.20 | FTP-DA... | 49314 | FTP Data: 49248 bytes (PASV) (type) |
| 18324 | 17.702463377 | 192.168.1.20 | 193.137.29.15 | TCP | 66 | 36354 → 58686 [ACK] Seq=1 Ack=1042 |
| 18325 | 17.704668866 | 193.137.29.15 | 192.168.1.20 | FTP-DA... | 2802 | FTP Data: 2736 bytes (PASV) (type) |
| 18326 | 17.704742865 | 192.168.1.20 | 193.137.29.15 | TCP | 66 | 36354 → 58686 [ACK] Seq=1 Ack=1042 |
| 18327 | 17.705634859 | 193.137.29.15 | 192.168.1.20 | FTP-DA... | 19218 | FTP Data: 19152 bytes (PASV) (type) |
| 18328 | 17.705660995 | 192.168.1.20 | 193.137.29.15 | TCP | 66 | 36354 → 58686 [ACK] Seq=1 Ack=1042 |
| 18329 | 17.708368916 | 193.137.29.15 | 192.168.1.20 | FTP-DA... | 4170 | FTP Data: 4104 bytes (PASV) (type) |
| 18330 | 17.708369102 | 193.137.29.15 | 192.168.1.20 | FTP-DA... | 4170 | FTP Data: 4104 bytes (PASV) (type) |
| 18331 | 17.708392869 | 192.168.1.20 | 193.137.29.15 | TCP | 66 | 36354 → 58686 [ACK] Seq=1 Ack=1042 |
| 18332 | 17.708415781 | 192.168.1.20 | 193.137.29.15 | TCP | 66 | 36354 → 58686 [ACK] Seq=1 Ack=1042 |
| 18333 | 17.709285182 | 193.137.29.15 | 192.168.1.20 | FTP-DA... | 13746 | FTP Data: 13680 bytes (PASV) (type) |
| 18334 | 17.709300669 | 192.168.1.20 | 193.137.29.15 | TCP | 66 | 36354 → 58686 [ACK] Seq=1 Ack=1042 |
| 18335 | 17.711877020 | 193.137.29.15 | 192.168.1.20 | FTP-DA... | 8274 | FTP Data: 8208 bytes (PASV) (type) |

```

> Frame 18328: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface wlp2s0, id 0
> Ethernet II, Src: IntelCor_6b:d4:fb (c0:b6:f9:6b:d4:fb), Dst: Sagemcom_fc:00:7e (2c:39:96:fc:00:7e)
> Internet Protocol Version 4, Src: 192.168.1.20, Dst: 193.137.29.15
> Transmission Control Protocol, Src Port: 36354, Dst Port: 58686, Seq: 1, Ack: 104248441, Len: 0
    Source Port: 36354
    Destination Port: 58686
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence number: 1      (relative sequence number)
    Sequence number (raw): 756148416
    [Next sequence number: 1      (relative sequence number)]
    Acknowledgment number: 104248441      (relative ack number)
    Acknowledgment number (raw): 1668477919
    1000 .... = Header Length: 32 bytes (8)
    Flags: 0x010 (ACK)
    Window size value: 12289
    [Calculated window size: 1572992]
    [Window size scaling factor: 128]
    Checksum: 0x2586 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
> Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
> [SEQ/ACK analysis]
> [Timestamps]

```

Pacotes FTP e TCP