Licenciatura em Engenharia Informática e Computação

Redes de Computadores

3LEIC025 - Turma 4 Grupo 9

João Pedro Rodrigues Coutinho up202108787

Miguel Jorge Medeiros Garrido up202108889

Sumário

Este trabalho realizado, proposto pela unidade curricular Redes de Computadores, tem como objetivo o desenvolvimento de uma aplicação que permita fazer download de ficheiros usando FTP, através de uma rede de computadores configurada pelos estudantes.

Introdução

Baseado num guião previamente disponibilizado, este trabalho consistiu no desenvolvimento e teste de um programa de download de ficheiros, utilizando o protocolo FTP, de modo a permitir a transferência de um ficheiro da internet para um computador específico, utilizado a suprarreferida rede de computadores. Este relatório tem como objetivo expor a implementação da aplicação e a configuração da rede anteriormente referidos, estando assim dividido em três secções diferentes:

1. **Aplicação de Download** - Arquitetura da aplicação; demonstração de um download bem-sucedido.

2. **Configuração e Análise da Rede** - Arquitetura da rede, objetivos, principais comandos de configuração e *logs* relevantes, bem como a respetiva análise, para todas as 6 experiências realizadas.

3. **Conclusões** - síntese da informação apresentada nas secções anteriores; reflexão sobre os objetivos de aprendizagem alcançados.

Aplicação de Download

Arquitetura

Esta primeira parte do projeto baseia-se no desenvolvimento de uma aplicação que permite o download de um ficheiro da internet utilizando o protocolo FTP.

Inicialmente, foram consultadas as normas RFC959 (implementação do protocolo de aplicação FTP) e RFC1738 (utilização de sintaxe de URL).

O URL, passado como argumento. é processado através da função *parse*, onde primeiro se verifica que tipo de URL é - com ou sem autenticação - e, posteriormente, são retirados valores como o *host*, o *resource* e o *IP* que é obtido através do primeiro. A cada chamada de função a resposta recebida é lida pelo programa, sendo reduzida aos três dígitos do código proveniente desta, cujo valor se espera que se encontre entre 2XX e 3XX; caso contrário, um erro será apresentado.

Após a chamada da função inicial, é criado um socket com o IP do *host* utilizando a função *socketInit*. Caso a conexão seja bem-sucedida, retornando o código 220, é feita a autenticação através do *login*, onde é enviado o *username* inicialmente e, em caso de sucesso (código 331), é enviada de seguida a *password*, ficando à espera de um código 230.

Se tudo correr como esperado, é então ativado o modo passivo do servidor através do envio de *pasv* na função *activatePassive*, onde se irá obter o endereço IP e a porta que serão utilizadas na criação de um novo *socket*. Para realizar o *download* do ficheiro, é feito um pedido na função *requestResource*, sendo o ficheiro posteriormente transferido na função *getResource*. Por fim, encerram-se as ligações e os *sockets* com a chamada da função *closeSocket*.

Resultados

Com o objetivo de testar a aplicação, foram realizadas experiências com múltiplos ficheiros de diversos tamanhos - todas com o resultado esperado (obtenção do ficheiro). Além das tentativas bem-sucedidas de transferência, foram efetuadas algumas em que o comportamento esperado era a deteção de erros como, por exemplo, no caso de tentarmos transferir ficheiros inexistentes ou de credenciais de autenticação erradas/inválidas.

No geral, o programa reagiu como esperado. O código da aplicação está disponível no **Anexo 1**, enquanto que os *logs* que demonstram os pacotes FTP e o ficheiro *pipe.txt* resultante de uma transferência bem sucedida estão disponíveis no **Anexo 3.6**.

Configuração e Análise da Rede

Experiência 1 - Configurar uma Rede IP

A primeira experiência consistiu essencialmente na configuração de dois endereços IP em dois computadores diferentes conectados a um *switch* - *Tux33* (172.16.30.1) e *Tux34* (172.16.30.254). Após a configuração dos endereços IP nos respetivos computadores, procedeu-se à análise das tabelas ARP (*Address Resolution Protocol* - relaciona o endereço IP de um computador ao respetivo endereço MAC) e do comportamento da ligação estabelecida entre o *Tux33* e o *Tux34* no caso da entrada da tabela ARP do *Tux33* ser eliminada.

Antes de executar os comandos de configuração, foi necessário ligar o E0 de cada um dos computadores ao *switch*. Os comandos necessários para a configuração desta experiência foram os seguintes:

- ***ifconfig eth0 up*** - ativa a interface eth0
- ***ifconfig eth0 <IP>*** - permite configurar eth0 com um endereço IP específico
- ***ping <IP>*** - testa se o computador no qual o comando é inserido consegue alcançar um determinado IP (computador) de destino
- ***route -n*** - lista as rotas existentes
- ***arp -a*** - lista a tabela ARP
- ***arp -d <IP>*** - permite a eliminação de uma entrada específica na tabela ARP

Após eliminar a entrada da tabela ARP do *Tux33* e correr novamente o comando *ping* (neste caso em específico, ***ping 172.16.30.254/24***), foi possível concluir que, no início da ligação, a associação entre o endereço IP e MAC foi reposta através da troca de pacotes ARP - pacotes cujo propósito é estabelecer a ligação entre um endereço IP e um endereço MAC.

Ao analisar os pacotes ARP, verificamos a presença de dois endereços IP e um endereço MAC. Isto deve-se ao facto de, inicialmente, tanto o endereço IP do computador de origem (*Tux33*, com endereço 172.16.30.1) como o de destino (*Tux34*, com endereço 172.16.30.254) serem enviados num único pacote ARP. Na resposta, o computador *Tux34* envia um pacote ARP com o seu endereço MAC.

Simultaneamente, podemos concluir que o comando *ping* gera pacotes ICMP (*Internet Control Message Protocol*, para diagnósticos de rede) somente se o computador de origem tiver conhecimento do endereço MAC do computador de destino; caso contrário, o comando gera pacotes ARP para associar o endereço IP ao endereço MAC respetivo. Assim, podemos concluir que os endereços IP e MAC utilizados nos pacotes ICMP são os endereços dos computadores pertencentes à rede: *Tux33* e *Tux34*.

Para determinar se uma trama Ethernet é do tipo ARP, IP ou ICMP, basta verificar o valor da coluna *Protocol* no log do Wireshark; do mesmo modo, para determinar o tamanho de uma trama, basta consultar o valor da coluna *Length* no log do Wireshark.

Também relevante para a experiência é a *loopback interface* - uma interface virtual que permite verificar se as ligações de uma rede estão corretamente configuradas (a interface é sempre possível de alcançar quando pelo menos uma das interfaces IP do *switch* estiver operacional).

Os comandos utilizados na configuração desta experiência estão disponíveis no **Anexo 2.1** e os *logs* desta experiência estão disponíveis no **Anexo 3.1**.

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

O objetivo da segunda experiência foi a criação de duas *Local Area Network* (LAN) distintas através da utilização de duas *bridges* no *switch*: uma contendo os computadores *Tux33* e *Tux34* e outra contendo somente o computador *Tux32*.

Aproveitando a configuração realizada na Experiência 1 para os *Tux33* e *Tux34*, apenas foi preciso configurar o *Tux32*, utilizando os mesmos comandos *ifconfig* e um endereço IP específico (172.16.31.1). Em seguida, foi necessário configurar o switch de modo a podermos proceder à criação das bridges; com esse objetivo, conectou-se a consola do switch ao *Tux33*.

Para a criação das duas *bridges* (*bridge30* e *bridge31*), utilizamos os seguintes comandos:

- ***/interface bridge add name=<Bridge>*** - permite a criação de *bridges* (neste caso, *bridge30* e *bridge31*)
- ***/interface bridge port remove [find interface=<Port>]*** - remove as portas às quais as interfaces conectadas a cada um dos computadores estavam ligadas por *default*
- ***/interface bridge port add bridge=<Bridge> interface=<Port>*** - adiciona as interfaces específicas de cada computador a uma *bridge* específica

Após concluirmos a configuração de ambas as *bridges*, utilizamos vários comandos *ping* no *Tux33* e no *Tux32* para analisar o comportamento da rede. Ao verificar os *logs*, foi possível concluir que existem dois domínios de *broadcast* diferentes, já que foram criadas e configuradas duas *bridges* diferentes.

No *broadcast* com origem no *Tux33* verificou-se uma resposta por parte do *Tux34* (observa-se no *log* a existência de pacotes do tipo *request* e *reply*), mas não por parte do *Tux32*; isto deve-se ao facto de tanto o *Tux33* como o *Tux34* se encontrarem na mesma rede. Por outro lado, no broadcast com origem no *Tux32* foi possível concluir que nenhum outro computador foi alcançado (apenas existem pacotes do tipo *request* no *log*), visto que o *Tux32* se encontrava numa rede isolada.

Os comandos utilizados na configuração desta experiência estão disponíveis no **Anexo 2.2** e os *logs* desta experiência estão disponíveis no **Anexo 3.2**.

Experiência 3 - Configurar um *router* em Linux

O objetivo da terceira experiência foi transformar o computador *Tux34* num *router* com capacidade de permitir que o *Tux33* e o *Tux32*, ainda que em redes diferentes, consigam comunicar entre si.

Aproveitando a configuração utilizada na Experiência 2, o primeiro passo consistiu em conectar o E1 do *Tux34* ao *switch* e configurá-lo com um endereço IP específico (172.16.31.253). Seguidamente, tal como na experiência anterior, removeu-se a porta à qual a interface conectada ao *Tux34* estava ligada por *default* e adicionou-se essa interface à *bridge31*.

Em seguida, correram-se os seguintes comandos no *Tux34*:

- ***sysctl net.ipv4.ip_forward=1*** - ativa *IP forwarding*

- ***sysctl net.ipv4.icmp_echo_ignore_broadcasts=0*** - desativa *ICMP echo-ignore-broadcast*

Por último, utilizou-se o comando *route add -net* tanto no *Tux32* como no *Tux33*, de modo a criar rotas em ambos os computadores com o endereço IP do *Tux34* acessível por cada rede (LAN) como *gateway* - 172.16.31.253 no caso do *Tux32* e 172.16.30.254 no caso do *Tux33*.

Existe, assim, uma rota tanto no *Tux32* como no *Tux33*, sendo que ambas utilizam o *Tux34* como *gateway* por ser o único computador comum às duas bridges existentes. É possível analisar as rotas ao observar a *forwarding table* (comando *route -n*) - verifica-se que cada entrada na tabela possui um endereço de destino e uma *gateway*.

Esta configuração permite que o *Tux34* aja como um *router* ao permitir a ligação entre duas *bridges* diferentes. É possível verificar isto ao utilizar o comando *ping*, com o *Tux32* como destino, no *Tux33* - todos os pacotes chegam ao *Tux32*. No entanto, apesar de o *ping* ocorrer entre o *Tux33* e o *Tux32*, ao observar os pacotes ARP verificamos que estes contêm somente os endereços MAC do *Tux33* e do *Tux34*. Isto deve-se ao facto de o *Tux33* não conhecer o endereço do *Tux32*, mas sim o endereço do *Tux34* (*gateway*) que lhe permite chegar ao *Tux32*.

Ao analisar os *logs*, podemos também verificar a presença de pacotes ICMP, o que significa que a rede está configurada corretamente; estes pacotes possuem dois endereços IP e um endereço MAC associados: o endereço IP de origem (*Tux33*), o endereço IP de destino (*Tux32*) e o endereço MAC do computador que permite a ligação entre as duas *bridges* (*Tux34*).

Os comandos utilizados na configuração desta experiência estão disponíveis no **Anexo 2.3** e os *logs* desta experiência estão disponíveis no **Anexo 3.3**.

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

O objetivo desta experiência foi configurar e adicionar um *Router* Comercial com NAT à *bridge31*, com a finalidade de obter acesso à internet.

O primeiro passo foi adicionar um cabo do *ether1* do *Router* até à rede do laboratório, que tem NAT como *default*, e um cabo do *ether2* ao *switch*. Seguidamente, teve de ser feita a configuração do *Router*, adicionando a sua interface à *bridge31* (tal como feito em experiências anteriores), seguindo-se da troca do cabo que ligava o *Tux34* à consola do switch para passar a ligar-se à consola do *Router* Comercial.

Através do *GTKterm*, configurou-se os endereços IP de cada interface (*/ip address add*), de modo a que o Router possuísse um endereço IP interno e outro externo; seguidamente, foram definidas rotas *default* para cada um dos computadores (*Router* como rota *default* para o *Tux32* e *Tux34* e *Tux34* como rota *default* para o *Tux33*).

No primeiro teste, como não existia conexão entre o *Tux32* e o *Tux34* nem ICMP *redirects* (que foram desativados utilizando comandos específicos para esse feito - **Anexo 2.4**), os pacotes de dados do *Tux32* enviados para o *Tux33* foram reencaminhados pelo *Router*, devido à existência de uma rota *default* para o *Tux32*.

Reativando as rotas e redirecionamentos, os pacotes recorrem à conexão mais direta disponível, através do *Tux34*. Podemos assim concluir que os pacotes ICMP responsabilizam-se pela escolha do melhor caminho possível.

A *Network Address Translation*, NAT, é utilizada na conversão de endereços de uma rede local em endereços públicos, ou vice-versa. Numa situação em que é enviado um pacote para uma rede externa, o endereço público é a origem deste, sendo a resposta enviada para este e

posteriormente redirecionada para o endereço local inicial. Este mecanismo é utilizado para precaver a excessiva utilização do IPv4, que resultaria em falta de endereços públicos caso a NAT não existisse. Para ativar/configurar a NAT, basta utilizar o comando */ip firewall nat enable 0*, após o qual passa a existir uma ligação à internet.

Os comandos utilizados na configuração desta experiência estão disponíveis no **Anexo 2.4** e os *logs* desta experiência estão disponíveis no **Anexo 3.4**.

Experiência 5 - DNS

Para completar esta experiência, foi necessário configurar o DNS (*Domain Name System*) para cada computador, utilizando a rede previamente configurada. Após este processo, será possível aceder a websites através do seu domínio.

Para configurar o DNS, foi necessário alterar o ficheiro */etc/resolv.conf* em todos os computadores, inserindo *nameserver 172.16.2.1* no ficheiro (endereço IP externo do router).

Após a realização de um *ping* para um website, verifica-se que os pacotes iniciais são pacotes DNS, pois o router tem de identificar e traduzir (o domínio para) o endereço IP de destino.

Os *logs* desta experiência estão disponíveis no **Anexo 3.5**.

Experiência 6 - Ligações TCP

Na última experiência, já com a rede totalmente configurada, utilizamos o nosso programa de *download* para avaliar a troca de pacotes com e sem congestionamento.

O primeiro passo foi verificar, utilizando a aplicação FTP desenvolvida, se conseguíamos realizar o *download* de ficheiros no *Tux33*; verificamos, após correr o programa, que a ordem de pacotes era a seguinte: DNS (onde ocorre a tradução do nome para um IP), FTP SYN/ACK (onde ocorre o estabelecimento da ligação), FTP Data (onde são passados os dados) e FTP FIN/ACK (termina a ligação). Estes tipos de pacotes representam, respetivamente, as diferentes fases da conexão TCP estabelecida.

Nesta aplicação, são estabelecidas duas conexões TCP - uma para enviar comandos de controlo ao servidor e outra para receber o ficheiro. A primeira conexão é responsável por transportar a informação de controlo FTP.

O mecanismo ARQ (*Automatic Repeat Request*) encontra-se nas conexões TCP não só para garantir a retransmissão numa rede congestionada mas também para controlo de erros, através de mensagens ACK (*acknowledge*) que indicam a receção correta do pacote e *timeouts* para determinar o tempo de receção. Quando um *timeout* é ultrapassado, o pacote é retransmitido.

O controlo de fluxo é utilizado na gestão da taxa de transmissão de dados entre os dois computadores. Isto permite que o emissor envie dados de forma a não sobrecarregar o recetor, evitando perda de pacotes. Este controlo é utilizado através de janelas, as quais possuem o tamanho que o transmissor pode enviar.

O controlo de congestionamento, tal como indica o nome, evita congestionamentos na rede; este é efetuado pelo emissor através do método *Selective Repeat* (envio de pacotes pela rede sem esperar pelas respetivas mensagens ACK). Quando existe perda de pacotes, é considerado que existe um congestionamento na rede; de modo a evitar que isto ocorra, o TCP adapta-se de forma a evitar perdas e diminuição do desempenho numa transmissão.

Para monitorizar se existe perda de pacotes na rede, o emissor transfere vários pacotes de uma única vez e utiliza um dos seguintes métodos: *Additive Increase* (na transferência seguinte envia um pacote a mais do que na transferência anterior) ou *Slow Start* (semelhante a *Additive Increase* mas os incrementos são exponenciais - base 2).

Por fim, realizou-se a experiência de começar um *download* no *Tux33*, seguido de um no *Tux32*. Foi possível verificar que a velocidade de transferência reduz-se significativamente quando outro download é iniciado - resultado do controlo de congestionamento por parte de ambos os

computadores, adaptando-se de modo a atingir valores estáveis durante a transferência mútua.

Os logs desta experiência estão disponíveis no **Anexo 3.6**.

Conclusão

Este projeto, através da configuração iterativa da rede, permitiu-nos perceber todos os mecanismos necessários para configurar e estudar uma rede, tal como a sua importância. Além disso, consolidamos os nossos conhecimentos dos protocolos utilizados na transferência de dados, bem como de outros conceitos envolvidos.

Referências

O guião do trabalho prático, bem como o guião relativo às experiências com protocolos de texto e os slides teóricos, foram fundamentais para a realização deste trabalho.

Do mesmo modo, o apoio prestado pelo professor Filipe Borges Teixeira, sob a forma de esclarecimento de dúvidas, foi também bastante importante para o desenvolvimento deste trabalho.

Anexos

Anexo 1 - Aplicação de *Download*

1.1 - download.h

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

#define PORT 21

struct URL {
    char host[500];
    char resource[500];
    char file[500];
    char ip[500];
    char username[500];
    char password[500];
};
```

1.2 - download.c

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

int parse(const char *url, struct URL *res) {
    int ftp = (strstr(url, "@") != NULL);
    int ftp2 = (strstr(url, ":") != NULL);
    if (!ftp || !ftp2) {
        int len = sscanf(url, "ftp://%[^/]/%s", res->host,
res->resource);
        if (len != 2) {
            perror("The FTP URL is invalid");
            printf("\n");
            return 1;
        }

        strcpy(res->username, "anonymous");
        strcpy(res->password, "anonymous");
    }
    else {
        int len = sscanf(url, "ftp://%[^:]:%[^@]@%[^/]/%s",
res->username, res->password, res->host, res->resource);
        if (len != 4) {
            perror("The FTP URL is invalid");
            printf("\n");
            return 1;
        }
    }

    strcpy(res->file, strrchr(url, '/') + 1);
    struct hostent *h;
    if ((h = gethostbyname(res->host)) == NULL) {
        herror("gethostbyname()");
        exit(-1);
    }

    strcpy(res->ip, inet_ntoa(*((struct in_addr *) h->h_addr)));

    return 0;
}


void socketInit(int *sockfd, const char *ip, int port){
    struct sockaddr_in server_addr;

    bzero((char *) &server_addr, sizeof(server_addr));
```

```
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr(ip);
    server_addr.sin_port = htons(port);

    if ((*sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket()");
        exit(-1);
    }

    if (connect(*sockfd, (struct sockaddr *) &server_addr,
sizeof(server_addr)) < 0) {
        perror("connect()");
        exit(-1);
    }
}

void login(int sockfd, const char* username, const char* password) {
    int user_len = 6+strlen(username);
    char user[user_len];
    strcpy(user, "USER ");
    strcat(user, username);
    strcat(user, "\n");
    write(sockfd, user, user_len);

    char response[1024];

    if (read(sockfd, response, 1024) < 0) {
        perror("read()");
        exit(-1);
    }

    int code;
    sscanf(response, "%d", &code);
    printf("%d", code);
    if (code != 331){
        perror("User unknown");
        exit(-1);
    }


    int pass_len = 6+strlen(password);
    char pass[pass_len];
    strcpy(pass, "PASS ");
    strcat(pass, password);
    strcat(pass, "\n");
```

```
    write(sockfd, pass, pass_len);

    if (read(sockfd, response, 1024) < 0) {
        perror("read()");
        exit(-1);
    }

    sscanf(response, "%d", &code);
    if (code != 230){
        perror("Wrong password");
        exit(-1);
    }

}

void activatePassive(int sockfd, char *ip, int *port){
    write(sockfd, "pasv\n", 5);

    char response[1024];
    if (read(sockfd, response, 1024) < 0) {
        perror("read()");
        exit(-1);
    }

    int code;
        sscanf(response, "%d", &code);
    if (code != 227){
        perror("Passive mode inactive");
        exit(-1);
    }

    int byte1, byte2, byte3, byte4, byte5, byte6;
    sscanf(response, "%*[^(](%d,%d,%d,%d,%d,%d)%*[^\n$)]", &byte1,
&byte2, &byte3, &byte4, &byte5, &byte6);
    sprintf(ip, "%d.%d.%d.%d", byte1, byte2, byte3, byte4);
    *port = (byte5*256)+byte6;
}

void requestResource(const int sockfd, const char *resource) {
    int user_len = 6+strlen(resource);
    char file[user_len];
    strcpy(file, "retr ");
    strcat(file, resource);
    strcat(file, "\n");
    write(sockfd, file, user_len);
```

```
    char response[1024];
    if (read(sockfd, response, 1024) < 0) {
        perror("read()");
        exit(-1);
    }

    int code;
    sscanf(response, "%d", &code);
    if (code != 150){
        perror("Unknown resource");
        exit(-1);
    }
}

void resourceDownload(const int controlSocket, const int dataSocket,
char *filename) {
    char buf[1024];
    FILE *fptr;
    fptr = fopen(filename, "wb");

    if (fptr == NULL) {
        perror("This file wasn't found\n");
        exit(EXIT_FAILURE);
    }

    int bytes_to_read = read(dataSocket, buf, 1024);

    while (bytes_to_read > 0) {

        if (fwrite(buf, bytes_to_read, 1, fptr) < 0) {
            perror("write()");
            exit(-1);
        }

        bytes_to_read = read(dataSocket, buf, 1024);
    }

    fclose(fptr);

    char response[1024];
    if (read(controlSocket, response, 1024) < 0) {
        perror("read()");
        exit(-1);
    }
```

```
    int code;
    sscanf(response, "%d", &code);
    if (code != 226){
        perror("Error while downloading file");
        exit(-1);
    }
}

void closeSocket(const int controlSocket, const int dataSocket){
    write(controlSocket, "quit\n", 5);

    char response[1024];
    if (read(controlSocket, response, 1024) < 0) {
        perror("read()");
        exit(-1);
    }

    int code;
    sscanf(response, "%d", &code);
    if (code != 221){
        perror("Error while quitting");
        exit(-1);
    }
    close(controlSocket);
    close(dataSocket);
}

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

    if (argc != 2) {
        perror("Include a URL");
        exit(-1);
    }

    struct URL url;

    if (parse(argv[1], &url)){
        perror("Error while parsing the URL");
        exit(-1);
    }

    printf("Host: %s\n", url.host);
    printf("Resource: %s\n", url.resource);
    printf("Username: %s\n", url.username);
```

```
    printf("Password: %s\n", url.password);
    printf("File: %s\n", url.file);
    printf("Ip: %s\n", url.ip);

    int sockfd;
    socketInit(&sockfd, url.ip, PORT);
    sleep(1);
    char response[1024];

    if (read(sockfd, response, 1024) < 0) {
        perror("read()");
        exit(-1);
    }

    int code;
    sscanf(response, "%d", &code);
    printf("%d", code);
    if (code != 220){
        perror("Failed to connect to the service");
        exit(-1);
    }

    login(sockfd, url.username, url.password);

    char ip[500];
    int port = 0;
    activatePassive(sockfd, ip, &port);

    int sockfdB;
    socketInit(&sockfdB, ip, port);

    requestResource(sockfd, url.resource);

    resourceDownload(sockfd, sockfdB, url.file);

    closeSocket(sockfd, sockfdB);
}
```

Anexo 2 - Comandos de Configuração

2.1 - Experiência 1

```
# Tux34
$ ifconfig eth0 up
$ ifconfig eth0 172.16.30.254/24
```

```
# Tux33
$ ifconfig eth0 up
$ ifconfig eth0 172.16.30.1/24
```

2.2 - Experiência 2

```
# Tux32
$ ifconfig eth0 up
$ ifconfig eth0 172.16.31.1/24
```

```
# Switch
/interface bridge add name=bridge30
/interface bridge add name=bridge31

/interface bridge port remove [find interface=ether1]
/interface bridge port remove [find interface=ether2]
/interface bridge port remove [find interface=ether3]

/interface bridge port add bridge=bridge30 interface=ether1
/interface bridge port add bridge=bridge30 interface=ether2
/interface bridge port add bridge=bridge31 interface=ether3
```

2.3 - Experiência 3

```
# Tux34
$ ifconfig eth1 up
$ ifconfig eth1 172.16.31.253/24
$ sysctl net.ipv4.ip_forward=1
$ sysctl net.ipv4.icmp_echo_ignore_broadcasts=0
```

```
# Tux33
$ route add -net  172.16.31.0/24 gw 172.16.30.254
```

```
# Tux32
$ route add -net  172.16.30.0/24 gw 172.16.31.253
```

```
# Switch
/interface bridge port remove [find interface=ether4]
/interface bridge port add bridge=bridge31 interface=ether4
```

2.4 - Experiência 4

```
# Tux34
$ route add default gw 172.16.31.254
```

```
# Tux33
```

```
$ route add default gw 172.16.30.254
```

```
# Tux32
$ route add default gw 172.16.31.254
$ sysctl net.ipv4.conf.eth0.accept_redirects=0
$ sysctl net.ipv4.conf.all.accept_redirects=0
$ route del -net 172.16.30.0/24 gw 172.16.31.253
$ route add -net 172.16.30.0/24 gw 172.16.31.253
$ sysctl net.ipv4.conf.eth0.accept_redirects=1
$ sysctl net.ipv4.conf.all.accept_redirects=1
```

```
# Switch
/interface bridge port remove [find interface=ether5]
/interface bridge port add bridge=bridge31 interface=ether5
```

```
# Router
/ip address add address=172.16.2.39/24 interface=ether1
/ip address add address=172.16.31.254/24 interface=ether2
/ip route add dst-address=172.16.30.0/24 gateway=172.16.31.253
/ip route add dst-address=0.0.0.0/0 gateway=172.16.2.254
/ip firewall nat disable 0
/ip firewall nat enable 0
```

Anexo 3 - *Logs*

3.1 - Experiência 1 - *Ping* do *Tux33* para o *Tux34*

```
23 2023-11-13 12:0… HewlettP_5a:7d:b7  Broadcast           ARP    42 Who has 172.16.30.254? Tell 172.16.30.1
24 2023-11-13 12:0… HewlettP_5a:74:3e  HewlettP_5a:7d:b7  ARP    60 172.16.30.254 is at 00:21:5a:5a:74:3e
25 2023-11-13 12:0… 172.16.30.1        172.16.30.254      ICMP   98 Echo (ping) request  id=0x7e03, seq=1/256
26 2023-11-13 12:0… 172.16.30.254      172.16.30.1        ICMP   98 Echo (ping) reply    id=0x7e03, seq=1/256
27 2023-11-13 12:0… 172.16.30.1        172.16.30.254      ICMP   98 Echo (ping) request  id=0x7e03, seq=2/512
28 2023-11-13 12:0… 172.16.30.254      172.16.30.1        ICMP   98 Echo (ping) reply    id=0x7e03, seq=2/512
```

3.2 - Experiência 2

3.2.1 - *Ping* do *Tux33* para o *Tux34*

```
15 2023-11-20 12:2… 172.16.30.1        172.16.30.254      ICMP   98 Echo (ping) request  id=0x0e5e, seq=1/256
16 2023-11-20 12:2… 172.16.30.254      172.16.30.1        ICMP   98 Echo (ping) reply    id=0x0e5e, seq=1/256
17 2023-11-20 12:2… 172.16.30.1        172.16.30.254      ICMP   98 Echo (ping) request  id=0x0e5e, seq=2/512
18 2023-11-20 12:2… 172.16.30.254      172.16.30.1        ICMP   98 Echo (ping) reply    id=0x0e5e, seq=2/512
```

3.2.2 - *Ping* do *Tux33* para o *Tux32*

```
root@gnu33:~# ping 172.16.31.1
connect: Network is unreachable
root@gnu33:~#
```

3.2.3 - *Ping broadcast* desde o *Tux33*

```
36 2023-11-20 12:0… 172.16.30.1      172.16.30.255     ICMP      98 Echo (ping) request  id=0x0999, seq=2/512
37 2023-11-20 12:0… 172.16.30.1      172.16.30.255     ICMP      98 Echo (ping) request  id=0x0999, seq=3/768
38 2023-11-20 12:0… Routerbo_1c:a3:2a Spanning-tree-(for-… STP      60 RST. Root = 32768/0/c4:ad:34:1c:a3:2a  Co
39 2023-11-20 12:0… 172.16.30.1      172.16.30.255     ICMP      98 Echo (ping) request  id=0x0999, seq=4/102
40 2023-11-20 12:0… 172.16.30.1      172.16.30.255     ICMP      98 Echo (ping) request  id=0x0999, seq=5/128
```

3.3 - Experiência 3

3.3.1 - *Ping* do *Tux33* para o *Tux32*

```
76 2023-11-20 12:2… 172.16.30.1      172.16.31.1       ICMP      98 Echo (ping) request  id=0x0e79, seq=1/256
77 2023-11-20 12:2… 172.16.31.1      172.16.30.1       ICMP      98 Echo (ping) reply    id=0x0e79, seq=1/256
78 2023-11-20 12:2… 172.16.30.1      172.16.31.1       ICMP      98 Echo (ping) request  id=0x0e79, seq=2/512
79 2023-11-20 12:2… 172.16.31.1      172.16.30.1       ICMP      98 Echo (ping) reply    id=0x0e79, seq=2/512
```

3.3.2 - *Ping* do *Tux33* para o *Tux32* registado no *Tux34* (*eth0*)

```
154 2023-11-20 12:3… HewlettP_5a:7d:b7 Broadcast         ARP       60 Who has 172.16.30.254? Tell 172.16.30.1
155 2023-11-20 12:3… HewlettP_5a:74:3e HewlettP_5a:7d:b7 ARP       42 172.16.30.254 is at 00:21:5a:5a:74:3e
156 2023-11-20 12:3… 172.16.30.1       172.16.31.1       ICMP      98 Echo (ping) request  id=0x1031, seq=1/256
157 2023-11-20 12:3… 172.16.31.1       172.16.30.1       ICMP      98 Echo (ping) reply    id=0x1031, seq=1/256
158 2023-11-20 12:3… 172.16.30.1       172.16.31.1       ICMP      98 Echo (ping) request  id=0x1031, seq=2/512
159 2023-11-20 12:3… 172.16.31.1       172.16.30.1       ICMP      98 Echo (ping) reply    id=0x1031, seq=2/512
```

3.3.3 - *Ping* do *Tux33* para o *Tux32* registado no *Tux34* (*eth1*)

```
111 2023-11-20 12:3… EncoreNe_b4:b8:94 Broadcast         ARP       42 Who has 172.16.31.1? Tell 172.16.31.253
112 2023-11-20 12:3… HewlettP_61:24:01 EncoreNe_b4:b8:94 ARP       60 172.16.31.1 is at 00:21:5a:61:24:01
113 2023-11-20 12:3… 172.16.30.1       172.16.31.1       ICMP      98 Echo (ping) request  id=0x1031, seq=1/256
114 2023-11-20 12:3… 172.16.31.1       172.16.30.1       ICMP      98 Echo (ping) reply    id=0x1031, seq=1/256
115 2023-11-20 12:3… 172.16.30.1       172.16.31.1       ICMP      98 Echo (ping) request  id=0x1031, seq=2/512
116 2023-11-20 12:3… 172.16.31.1       172.16.30.1       ICMP      98 Echo (ping) reply    id=0x1031, seq=2/512
```

3.4 - Experiência 4

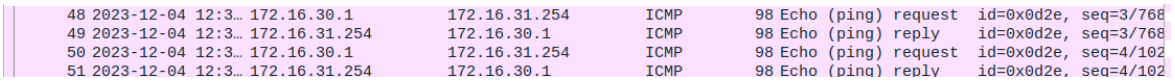
3.4.1 *Ping* do *Tux33* para o *Tux34*

```
30 2023-12-04 12:3… Routerbo_1c:a3:2c LLDP_Multicast       LLDP     110 MA/c4:ad:34:1c
31 2023-12-04 12:3… 172.16.30.1       172.16.30.254       ICMP      98 Echo (ping) req
32 2023-12-04 12:3… 172.16.30.254     172.16.30.1         ICMP      98 Echo (ping) rep
33 2023-12-04 12:3… Routerbo_1c:a3:2b Spanning-tree-(for-… STP      60 RST. Root = 32
34 2023-12-04 12:3… 172.16.30.1       172.16.30.254       ICMP      98 Echo (ping) req
35 2023-12-04 12:3… 172.16.30.254     172.16.30.1         ICMP      98 Echo (ping) rep
36 2023-12-04 12:3… 172.16.30.1       172.16.30.254       ICMP      98 Echo (ping) req
37 2023-12-04 12:3… 172.16.30.254     172.16.30.1         ICMP      98 Echo (ping) rep
38 2023-12-04 12:3… Routerbo_1c:a3:2b Spanning-tree-(for-… STP      60 RST. Root = 32
39 2023-12-04 12:3… 172.16.30.1       172.16.30.254       ICMP      98 Echo (ping) req
40 2023-12-04 12:3… 172.16.30.254     172.16.30.1         ICMP      98 Echo (ping) rep
```

3.4.2 *Ping* do *Tux33* para o *Tux32*

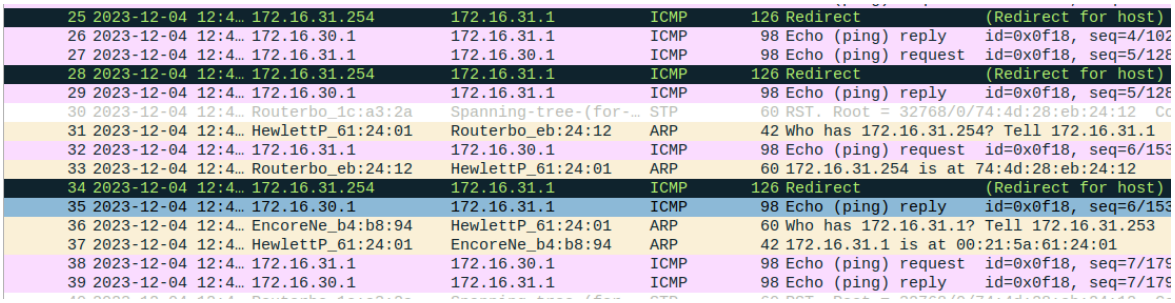
```
 4 2023-12-04 12:3… 172.16.30.1       172.16.31.1       ICMP      98 Echo (ping) req
 5 2023-12-04 12:3… 172.16.31.1       172.16.30.1       ICMP      98 Echo (ping) rep
 6 2023-12-04 12:3… 172.16.30.1       172.16.31.1       ICMP      98 Echo (ping) req
 7 2023-12-04 12:3… 172.16.31.1       172.16.30.1       ICMP      98 Echo (ping) rep
 8 2023-12-04 12:3… Routerbo_1c:a3:2b Spanning-tree-(for-… STP      60 RST. Root = 32
 9 2023-12-04 12:3… 172.16.30.1       172.16.31.1       ICMP      98 Echo (ping) req
10 2023-12-04 12:3… 172.16.31.1       172.16.30.1       ICMP      98 Echo (ping) rep
11 2023-12-04 12:3… 172.16.30.1       172.16.31.1       ICMP      98 Echo (ping) req
12 2023-12-04 12:3… 172.16.31.1       172.16.30.1       ICMP      98 Echo (ping) rep
13 2023-12-04 12:3… Routerbo_1c:a3:2b Spanning-tree-(for-… STP      60 RST. Root = 32
14 2023-12-04 12:3… 172.16.30.1       172.16.31.1       ICMP      98 Echo (ping) req
15 2023-12-04 12:3… 172.16.31.1       172.16.30.1       ICMP      98 Echo (ping) rep
16 2023-12-04 12:3… HewlettP_5a:7d:b7 HewlettP_5a:74:3e ARP       42 Who has 172.16.
17 2023-12-04 12:3… HewlettP_5a:74:3e HewlettP_5a:7d:b7 ARP       60 172.16.30.254 i
```

3.4.3 *Ping* do *Tux33* para o *Router*



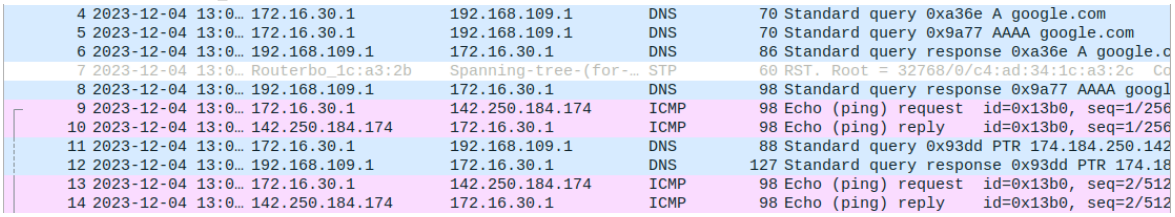
48	2023-12-04 12:3…	172.16.30.1	172.16.31.254	ICMP	98	Echo (ping) request id=0x0d2e, seq=3/768
49	2023-12-04 12:3…	172.16.31.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x0d2e, seq=3/768
50	2023-12-04 12:3…	172.16.30.1	172.16.31.254	ICMP	98	Echo (ping) request id=0x0d2e, seq=4/102
51	2023-12-04 12:3…	172.16.31.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x0d2e, seq=4/102

3.4.4 *Ping* do *Tux32* para o *Tux33* redirecionado pelo *Router*



25	2023-12-04 12:4…	172.16.31.254	172.16.31.1	ICMP	126	Redirect (Redirect for host)
26	2023-12-04 12:4…	172.16.30.1	172.16.31.1	ICMP	98	Echo (ping) reply id=0x0f18, seq=4/102
27	2023-12-04 12:4…	172.16.31.1	172.16.30.1	ICMP	98	Echo (ping) request id=0x0f18, seq=5/128
28	2023-12-04 12:4…	172.16.31.254	172.16.31.1	ICMP	126	Redirect (Redirect for host)
29	2023-12-04 12:4…	172.16.30.1	172.16.31.1	ICMP	98	Echo (ping) reply id=0x0f18, seq=5/128
30	2023-12-04 12:4…	Routerbo_1c:a3:2a	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/74:4d:28:eb:24:12 Co
31	2023-12-04 12:4…	HewlettP_61:24:01	Routerbo_eb:24:12	ARP	42	Who has 172.16.31.254? Tell 172.16.31.1
32	2023-12-04 12:4…	172.16.31.1	172.16.30.1	ICMP	98	Echo (ping) request id=0x0f18, seq=6/153
33	2023-12-04 12:4…	Routerbo_eb:24:12	HewlettP_61:24:01	ARP	60	172.16.31.254 is at 74:4d:28:eb:24:12
34	2023-12-04 12:4…	172.16.31.254	172.16.31.1	ICMP	126	Redirect (Redirect for host)
35	2023-12-04 12:4…	172.16.30.1	172.16.31.1	ICMP	98	Echo (ping) reply id=0x0f18, seq=6/153
36	2023-12-04 12:4…	EncoreNe_b4:b8:94	HewlettP_61:24:01	ARP	60	Who has 172.16.31.1? Tell 172.16.31.253
37	2023-12-04 12:4…	HewlettP_61:24:01	EncoreNe_b4:b8:94	ARP	42	172.16.31.1 is at 00:21:5a:61:24:01
38	2023-12-04 12:4…	172.16.31.1	172.16.30.1	ICMP	98	Echo (ping) request id=0x0f18, seq=7/179
39	2023-12-04 12:4…	172.16.30.1	172.16.31.1	ICMP	98	Echo (ping) reply id=0x0f18, seq=7/179

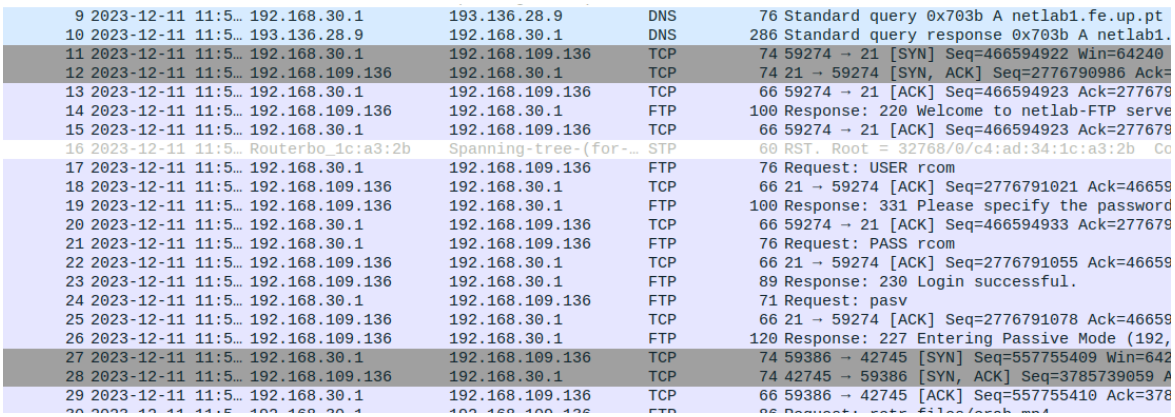
3.5 - Experiência 5 - DNS ao dar *ping* para google.com



4	2023-12-04 13:0…	172.16.30.1	192.168.109.1	DNS	70	Standard query 0xa36e A google.com
5	2023-12-04 13:0…	172.16.30.1	192.168.109.1	DNS	70	Standard query 0x9a77 AAAA google.com
6	2023-12-04 13:0…	192.168.109.1	172.16.30.1	DNS	86	Standard query response 0xa36e A google.c
7	2023-12-04 13:0…	Routerbo_1c:a3:2b	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:2c Co
8	2023-12-04 13:0…	192.168.109.1	172.16.30.1	DNS	98	Standard query response 0x9a77 AAAA googl
9	2023-12-04 13:0…	172.16.30.1	142.250.184.174	ICMP	98	Echo (ping) request id=0x13b0, seq=1/256
10	2023-12-04 13:0…	142.250.184.174	172.16.30.1	ICMP	98	Echo (ping) reply id=0x13b0, seq=1/256
11	2023-12-04 13:0…	172.16.30.1	192.168.109.1	DNS	88	Standard query 0x93dd PTR 174.184.250.142
12	2023-12-04 13:0…	192.168.109.1	172.16.30.1	DNS	127	Standard query response 0x93dd PTR 174.18
13	2023-12-04 13:0…	172.16.30.1	142.250.184.174	ICMP	98	Echo (ping) request id=0x13b0, seq=2/512
14	2023-12-04 13:0…	142.250.184.174	172.16.30.1	ICMP	98	Echo (ping) reply id=0x13b0, seq=2/512

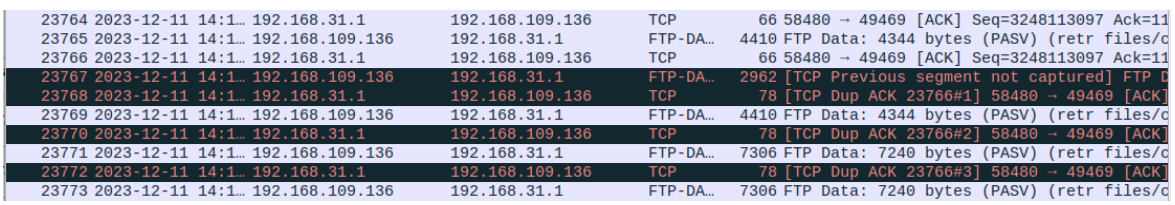
3.6 - Experiência 6

3.6.1 Transferência de ficheiro no *Tux33*



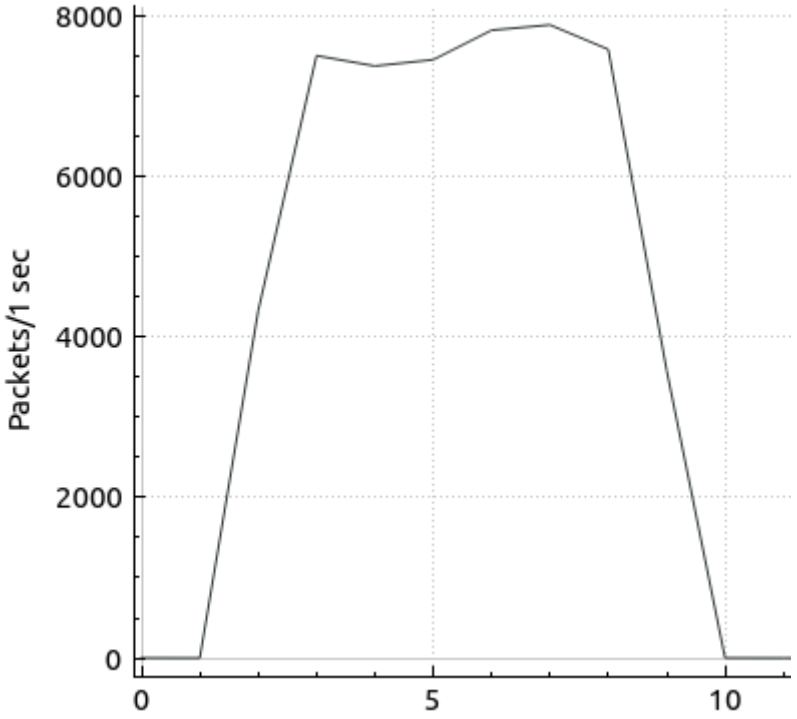
9	2023-12-11 11:5…	192.168.30.1	193.136.28.9	DNS	76	Standard query 0x703b A netlab1.fe.up.pt
10	2023-12-11 11:5…	193.136.28.9	192.168.30.1	DNS	286	Standard query response 0x703b A netlab1.
11	2023-12-11 11:5…	192.168.30.1	192.168.109.136	TCP	74	59274 → 21 [SYN] Seq=466594922 Win=64240
12	2023-12-11 11:5…	192.168.109.136	192.168.30.1	TCP	74	21 → 59274 [SYN, ACK] Seq=2776790986 Ack=
13	2023-12-11 11:5…	192.168.30.1	192.168.109.136	TCP	66	59274 → 21 [ACK] Seq=466594923 Ack=277679
14	2023-12-11 11:5…	192.168.109.136	192.168.30.1	FTP	100	Response: 220 Welcome to netlab-FTP serve
15	2023-12-11 11:5…	192.168.30.1	192.168.109.136	TCP	66	59274 → 21 [ACK] Seq=466594923 Ack=277679
16	2023-12-11 11:5…	Routerbo_1c:a3:2b	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:a3:2b Co
17	2023-12-11 11:5…	192.168.30.1	192.168.109.136	FTP	76	Request: USER rcom
18	2023-12-11 11:5…	192.168.109.136	192.168.30.1	TCP	66	21 → 59274 [ACK] Seq=2776791021 Ack=46659
19	2023-12-11 11:5…	192.168.109.136	192.168.30.1	FTP	100	Response: 331 Please specify the password
20	2023-12-11 11:5…	192.168.30.1	192.168.109.136	TCP	66	59274 → 21 [ACK] Seq=466594933 Ack=277679
21	2023-12-11 11:5…	192.168.30.1	192.168.109.136	FTP	76	Request: PASS rcom
22	2023-12-11 11:5…	192.168.109.136	192.168.30.1	TCP	66	21 → 59274 [ACK] Seq=2776791055 Ack=46659
23	2023-12-11 11:5…	192.168.109.136	192.168.30.1	FTP	89	Response: 230 Login successful.
24	2023-12-11 11:5…	192.168.30.1	192.168.109.136	FTP	71	Request: pasv
25	2023-12-11 11:5…	192.168.109.136	192.168.30.1	TCP	66	21 → 59274 [ACK] Seq=2776791078 Ack=46659
26	2023-12-11 11:5…	192.168.109.136	192.168.30.1	FTP	120	Response: 227 Entering Passive Mode (192,
27	2023-12-11 11:5…	192.168.30.1	192.168.109.136	TCP	74	59386 → 42745 [SYN] Seq=557755409 Win=642
28	2023-12-11 11:5…	192.168.109.136	192.168.30.1	TCP	74	42745 → 59386 [SYN, ACK] Seq=3785739059 A
29	2023-12-11 11:5…	192.168.30.1	192.168.109.136	TCP	66	59386 → 42745 [ACK] Seq=557755410 Ack=378

3.6.2 Transferência simultânea de um ficheiro no *Tux33* e *Tux32*

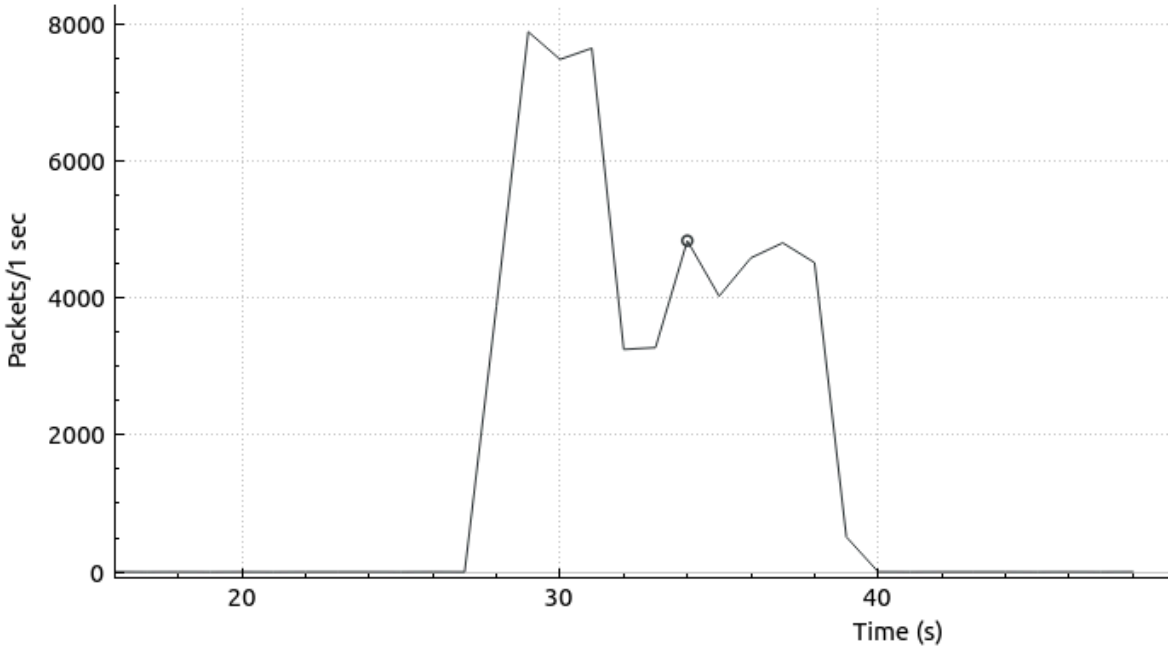


23764	2023-12-11 14:1…	192.168.31.1	192.168.109.136	TCP	66	58480 → 49469 [ACK] Seq=3248113097 Ack=11
23765	2023-12-11 14:1…	192.168.109.136	192.168.31.1	FTP-DA…	4410	FTP Data: 4344 bytes (PASV) (retr files/c
23766	2023-12-11 14:1…	192.168.31.1	192.168.109.136	TCP	66	58480 → 49469 [ACK] Seq=3248113097 Ack=11
23767	2023-12-11 14:1…	192.168.109.136	192.168.31.1	FTP-DA…	2962	[TCP Previous segment not captured] FTP D
23768	2023-12-11 14:1…	192.168.31.1	192.168.109.136	TCP	78	[TCP Dup ACK 23766#1] 58480 → 49469 [ACK]
23769	2023-12-11 14:1…	192.168.109.136	192.168.31.1	FTP-DA…	4410	FTP Data: 4344 bytes (PASV) (retr files/c
23770	2023-12-11 14:1…	192.168.31.1	192.168.109.136	TCP	78	[TCP Dup ACK 23766#2] 58480 → 49469 [ACK]
23771	2023-12-11 14:1…	192.168.109.136	192.168.31.1	FTP-DA…	7306	FTP Data: 7240 bytes (PASV) (retr files/c
23772	2023-12-11 14:1…	192.168.31.1	192.168.109.136	TCP	78	[TCP Dup ACK 23766#3] 58480 → 49469 [ACK]
23773	2023-12-11 14:1…	192.168.109.136	192.168.31.1	FTP-DA…	7306	FTP Data: 7240 bytes (PASV) (retr files/c

3.6.3 Velocidade de transferência de um ficheiro no *Tux33*



3.6.4 Velocidade de transferência de um ficheiro no *Tux33*, sendo outra transferência começada posteriormente no *Tux32*



3.6.5 Ficheiro *pipe.txt* transferido com sucesso