

# 2º Trabalho Laboratorial: Rede de Computadores

Relatório



Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

**Turma 1 Grupo 2:**

André Cruz - 201503776  
Bruno Piedade - 201505668  
Edgar Carneiro - 201503748

Faculdade de Engenharia da Universidade do Porto  
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

22 de Dezembro de 2017

# Conteúdo

<b>1</b>	<b>Sumário</b>	<b>3</b>
<b>2</b>	<b>Introdução</b>	<b>3</b>
<b>3</b>	<b>Parte 1 - Aplicação de download</b>	<b>4</b>
3.1	Arquitetura . . . . .	4
3.2	Estrutura do Código & Aspetos Funcionais . . . . .	4
3.3	Casos de uso principais . . . . .	7
3.4	Validação . . . . .	8
<b>4</b>	<b>Parte 2 - Configuração e análise da rede</b>	<b>8</b>
4.1	Experiência 1 - Configurar uma rede IP . . . . .	8
4.1.1	O que são os pacotes ARP e para que são usados? . . . . .	8
4.1.2	Quais são os endereços IP e MAC dos pacotes ARP? . . . . .	9
4.1.3	Que pacotes são gerados aquando de um comando <i>ping</i> ? . . . . .	9
4.1.4	Quais são os endereços IP e MAC dos pacotes de <i>ping</i> ? . . . . .	9
4.1.5	Como é possível determinar se uma trama recebida é do tipo ARP, IP ou ICMP? . . . . .	9
4.1.6	Como é possível determinar o tamanho de uma trama recebida? . . . . .	9
4.1.7	O que é e qual a importância da interface de <i>loopback</i> ? . . . . .	9
4.2	Experiência 2 - Implementar duas <i>LAN</i> s virtuais no <i>switch</i> . . . . .	9
4.2.1	Configuração da <i>vlan50</i> . . . . .	10
4.2.2	Quantos <i>broadcast domains</i> há? . . . . .	10
4.3	Experiência 3 - Configurar um router em <i>Linux</i> . . . . .	10
4.3.1	Que rotas há nos <i>tuxes</i> ? E qual os seus significados? . . . . .	11
4.3.2	Que informação contém uma entrada na <i>forwarding table</i> ? . . . . .	12
4.3.3	Que mensagens ARP, e endereços MAC associados, são observados? . . . . .	12
4.3.4	Que pacotes ICMP são observados, e porquê? . . . . .	13
4.3.5	Quais são os endereços IP e MAC associados aos pacotes ICMP? . . . . .	13
4.4	Experiência 4 - Configurar um router comercial e implementar NAT . . . . .	13
4.5	Experiência 5 - DNS . . . . .	15
4.6	Experiência 6 -Ligações TCP . . . . .	15
<b>5</b>	<b>Conclusões</b>	<b>17</b>
<b>6</b>	<b>Anexo I</b>	<b>18</b>
<b>7</b>	<b>Anexo II</b>	<b>22</b>

# 1 Sumário

Este trabalho foi realizado no âmbito da cadeira de Redes de Computadores, e visou o estudo de uma rede de computadores, da sua configuração e posterior ligação a uma aplicação desenvolvida pelo grupo. Para tal, além de seguir as recomendações e instruções fornecidas no guião, o grupo teve de fazer pesquisas acerca do funcionamento do protocolo FTP e respectiva ligação a um servidor. A aplicação desenvolvida permite ao utilizador fazer a transferência de um ficheiro fornecido por um *URL*.

No final, é possível abordar as diversas conclusões obtidas estando estas relacionadas com os objetivos de cada uma das experiências, bem como com o desenvolvimento da aplicação.

# 2 Introdução

O projecto divide-se em duas grandes componentes: a configuração de uma rede de computadores; e o desenvolvimento de uma aplicação de download que permitisse o *download* de um ficheiro através do protocolo *FTP - File Transfer Protocol*.

O principal objectivo da configuração de rede é permitir a execução de uma aplicação, a partir de duas *VLANs* dentro de um *switch*. Numa das *VLAN* foi implementado o NAT, estando este activo, e na outra não, tendo esta última que conseguir ter ligação à *Internet* para a aplicação de download funcionar correctamente.

Quanto aos objectivos da aplicação de download, era essencial o grupo entender o que é um cliente, um servidor e as suas características TCP/IP, bem como o funcionamento do protocolo FTP. Com estes objectivos concluídos, o grupo poderia avançar para o desenvolvimento da aplicação, implementando um cliente FTP e uma ligação TCP a partir de *sockets*. Só então configuraríamos o servidor de DNS, para a conversão de um *URL* para um IP, permitindo a sua localização num *host* com domínio determinado.

O Relatório encontra-se dividido em diversas secções, nas quais se pode encontrar a seguinte informação:

- **Arquitetura**, onde são descritos os diferentes blocos funcionais e interfaces.
- **Estrutura do código**, apresentando as *API's*, principais estruturas de dados, principais funções e a sua relação com a arquitetura.
- **Casos de uso principais**, onde são identificados os principais casos de uso e as suas sequências de chamada de funções.
- **Aspetos Funcionais**, identificando os principais aspetos funcionais, bem como a descrição da estratégia de implementação.
- **Validação**, descrevendo os testes efetuados.
- **Experiências Laboratoriais**, onde é realizado um sumário de cada uma das experiências, bem como respondidas cada uma das perguntas relativas às experiências. De destacar que nas experiências quatro a seis, as respostas à pergunta já se encontram dentro do sumário da experiência para evitar grandes excertos de texto repetido.

- **Conclusões**, onde é feita uma tese da informação apresentada nas secções anteriores, bem como uma reflexão sobre os objetivos de aprendizagem alcançados.

## 3 Parte 1 - Aplicação de download

### 3.1 Arquitetura

#### Blocos Funcionais

É possível distinguir a existência de dois blocos funcionais bem definidos: o bloco responsável pela validação e *parsing* do *URL* e o bloco responsável pela transferência de informação do servidor para o cliente. Os ficheiros *URL.c* e *URL.h* representam o primeiro bloco funcional referido, enquanto os ficheiros *clientFTP.c* e *clientFTP.h* representam o segundo bloco funcional referido. Cada um destes blocos será mais detalhadamente analisado em secções posteriores.

#### Interface

Na interface da linha de comandos é permitido ao utilizador correr o programa, apenas tendo de especificar como parâmetro qual o *URL* do ficheiro a fazer *download*.

### 3.2 Estrutura do Código & Aspetos Funcionais

#### *URL Parsing*

Tal como já foi referido na subsecção **Blocos Funcionais**, é nos ficheiros *URL.c* e *URL.h* que se faz a análise e interpretação do *URL* que é passado como argumento na interface de comandos, aquando da execução do programa. Após a análise do *URL* é preenchida uma estrutura de dados que guarda: o *port* usado (será sempre o *port* 21), o endereço *IP* - 'Internet Protocol' - , o diretório do ficheiro, o nome do *host*, o nome do ficheiro, o nome utilizado para a autenticação do utilizador e a *password* para a autenticação do utilizador.

---

```
typedef struct url_t {
    int port;
    char ip[URL_STR_LEN];      // host's ip
    char path[URL_STR_LEN];    // file path
    char hostname[URL_STR_LEN]; // hostname
    char filename[URL_STR_LEN]; // filename
    char username[URL_STR_LEN]; // username
    char password[URL_STR_LEN]; // password
} URL;
```

---

As funções da **API**, que também são as principais funções deste bloco funcional, são:

---

```
URL * constructURL();
int parseURL(URL* url, const char* str);
void setIp(URL * url);
void printURL(URL * url);
void destructURL(URL * url);
```

---

Como o nome indicada, a função *constructURL* serve para criar a estrutura de dados. A função *parseURL* é a função responsável pelo *parsing* e população de 5 dos 7 elementos da estrutura de dados, sendo que apenas o *port* e o *ip* não são populados. Assim, esta função tem como argumentos: um apontador para a estrutura *URL* a ser populada, e a *string* a ser analisada. Para análise do *URL* é utilizado um *regex* por nós criado, sendo este:

---

```
#define URL_REGEX
"ftp://([a-zA-Z][^:]*):([^@]+)@?([a-z0-9]+[.]?+)/(^[^/]+[/])*([^/]+)$"
```

---

É também feito uso de *capture groups* de forma a capturar quais os elementos que se encontram a ser *parsed* pelo *regex*, para posteriormente serem guardados no lugar correto, dentro da estrutura de dados *URL*. Assim à medida que o *parser* encontra um elemento que corresponda a um *capture group*, este é atualizado na estrutura de dados. Para tal efeito, usamos os seguintes importantes excertos de código:

```
while (i++ < N_MATCHES) {
    int j = 0;
    int nomatch = regexexec(r, p, N_MATCHES, m, 0);
    if (nomatch) {
        return i == 1 ? NO_MATCH : OK;
    }
    for (j = 0; j < N_MATCHES; j++) {
        int start = m[j].rm_so + (p - to_match);
        int finish = m[j].rm_eo + (p - to_match);

        setInUrl(url, j, to_match + start, finish - start);
    }
    p += m[0].rm_eo;
}
```

```
static int setInUrl(URL * url, int idx, const char * src, int size) {
    switch (idx) {
        case 0: // whole capture
        case 1: // identity:password
        case 5: // host's country
        case 7: // path termination
            break;
        case 2: // username
            if (0 == size) {
                strcpy(url->username, "anonymous");
            } else {
                strncpy(url->username, src, size);
                url->username[size] = 0;
            }
            break;
        case 3: // password
            if (0 == size) {
                strcpy(url->password, "a");
            } else {
                strncpy(url->password, src, size);
            }
    }
}
```

```

        url->password[size] = 0;
    }
    break;
case 4: // hostname
    strncpy(url->hostname, src, size);
    url->hostname[size] = 0;
    break;
case 6: // path
    strncpy(url->path, src, size);
    url->path[size] = 0;
    break;
case 8: // path
    strncpy(url->filename, src, size);
    url->filename[size] = 0;
    break;
default:
    break;
}

```

Continuando a análise das funções principais, a função *setIp* permite a alteração do valor do campo *ip* da estrutura de dados. A função *printURL* serve para imprimir os valores guardados na estrutura de dados para o ecrã. Por fim, a função *destructURL* serve para destruir a estrutura *URL*.

### ***Download*** do ficheiro

Nos ficheiros *clientFTP.c* e *clientFTP.h* é feita a chamada às funções que executam a transferência do ficheiro desejado. Nestes, existe uma estrutura que guarda os descritores dos ficheiros de controlo e de informação. O descritor do ficheiro de controlo é o local por onde é feita toda a comunicação com o servidor relativamente à configuração do *download*, enquanto o descritor do ficheiro de informação é o local por onde é transmitido o ficheiro que o utilizador pretende fazer *download*. Assim, faz sentido que toda a comunicação pelo canal de controlo seja feita antes da comunicação pelo canal de informação, de forma a garantir que todas as configurações se encontram corretas.

---

```

typedef struct ftp_t {
    int fdControl;
    int fdData;
} FTP;

```

---

As funções da **API** deste bloco funcional são:

---

```

int downloadFtpUrl(const char * str);

```

---

Esta função recebe como argumento a *string* correspondente ao *URL* do ficheiro a ser transferido e caso o *URL* seja válido, tenta executar essa transferência.

As principais funções deste bloco funcional são:

---

```

static int receiveCommand(int fd, char* responseCmd, char * expectedAnswer);

```

---

```
static int sendCommand(int fd, const char* msg, char* response, int readResponse,
    char * expectedAnswer);
static int connectSocket(const char* ip, int port);
static void retrieveFile(int fd);
static void sendUSER(int fd);
static void sendPASV(int fd);
static int download(int fd);
static void quitConnection();
```

---

As primeiras duas funções são as funções que merecem maior destaque pois é através destas que é feita toda a comunicação ‘cliente - servidor’.

Assim, e como o nome o indica, a função ***receiveCommand***, permite receber uma resposta do servidor, sendo que esta será enviada através do descritor de ficheiro *fd*. No argumento *expectedAnswer* é enviada a resposta que o cliente espera receber, sendo que se a resposta do servidor não corresponder à resposta que o cliente espera receber, é retornado erro. O argumento *responseCmd*, se diferente de nulo, retorna a resposta do servidor. Internamente, é verificado se as respostas do servidor não são respostas de erro, pela análise do primeiro dígito do código de retorno do servidor.

A função ***sendCommand*** permite ao cliente enviar uma mensagem *msg* ao servidor, através do descritor de ficheiro *fd*, sendo que este devolve a resposta do servidor - *response*. Verifica também se a resposta é semelhante à resposta esperada *expectedAnswer*.

A função *connectSocket* estabelece a comunicação entre o cliente e o servidor, pelo meio de *sockets*, usando o *ip* e o *port* passados como argumento para definir essa ligação. Esta função retorna o descritor de ficheiro que ficará associado às comunicações.

A função *retrieveFile* permite ao cliente pedir uma cópia do ficheiro do qual este pretende fazer o *download*. A comunicação é feita através do descritor de ficheiro de controlo *fd*.

A função *sendUSER* permite, através do descritor de ficheiro de controlo *fd*, que o conjunto utilizador - *password* seja autenticado pelo servidor.

A função *sendPASV* permite que a comunicação *FTP* com o servidor seja feita em modo passivo. A comunicação com o servidor é feita através do descritor de ficheiro de controlo *fd*.

A função *download* faz a transferência do ficheiro para o descritor de ficheiro de informação que lhe é passado como argumento.

A função *quitConnection* termina todas as conexões existentes entre o cliente e o servidor.

### 3.3 Casos de uso principais

A funcionalidade de cada uma das funções mencionadas nesta secção foi previamente explicada na secção anterior, secção **Estrutura do Código**.

Existe unicamente um caso de uso: correr a aplicação de forma a fazer *download* de um ficheiro, usando o protocolo de comunicação *FTP*. Para correr a aplicação deve-se respeitar a chamada ao programa, que é feita através de:

---

```
printf( "\nUsage: %s ftp://[<user>:<password>@]<host>/<url-path>\n", argv[0]);
```

---

A sequência de chamada de funções é:

- **downloadFtpUrl**, com argumento *argv[1]*;
  - **constructURL**;

- **parseURL**, com argumento *argv[1]*;
- **setIp**;
- **connectSocket**, para o descritor de ficheiro de controlo;
- **sendUSER**;
- **sendPASV**;
- **connectSocket**, para o descritor de ficheiro de informação;
- **retrieveFile**;
- **download**;
- **destructURL**;
- **quitConnection**;

### 3.4 Validação

Para validação da aplicação desenvolvida e para garantir que funcionava de acordo com o protocolo especificado, foram realizadas constantemente testes durante o desenvolvimento do programa e também na sua demonstração. Foi testado o *download* de ficheiros de diversos tamanhos, variando desde ficheiros de vários *Kbytes* até ficheiros de *Gbytes*, *download* em modo anónimo e mode autenticado e a robustez do *parser de URL* implementado.

## 4 Parte 2 - Configuração e análise da rede

### 4.1 Experiência 1 - Configurar uma rede IP

Esta experiência teve como objetivo a configuração de IP's em máquinas diferentes, de modo a que estas consigam comunicar entre si. Assim, após a configuração dos IP's das portas *eth0* de dois computadores e a adição das rotas necessárias à tabela de reencaminhamento, foi enviado um sinal *ping* de um computador para o outro, para verificar que, de facto, as máquinas tinham uma ligação entre si.

Para a configuração dos computadores foi utilizado o comando `<ifconfig [ip]>`, que atribui ao IP da interface o IP passado como argumento. Após esta configuração, executamos um ping de uma máquina para a outra com os IP's definidos, operação essa que foi executada com sucesso. Foi também possível ver os pedidos ARP, com os pings definidos e a resposta da máquina correspondente com o seu endereço MAC.

G-ProCom_8c:af:9d	Broadcast	ARP	42	Who has 172.16.1.24? Tell 172.16.1.21
HewlettP_a6:a4:f1	G-ProCom_8c:af:9d	ARP	60	172.16.1.24 is at 00:22:64:a6:a4:f1
172.16.1.21	172.16.1.24	ICMP	98	Echo (ping) request id=0x0b07, seq=1/256, ttl=64
172.16.1.24	172.16.1.21	ICMP	98	Echo (ping) reply id=0x0b07, seq=1/256, ttl=64

Figura 1: Características dos pacotes enviados quando é iniciado um comando ping.

#### 4.1.1 O que são os pacotes ARP e para que são usados?

Os pacotes ARP (Address Resolution Protocol) são usados para mapear um endereço de rede (endereço IP) para um endereço físico (endereço MAC). Pode ver um exemplo das características dos pacotes ARP na figura 1.



#### 4.1.2 Quais são os endereços IP e MAC dos pacotes ARP?

Os pacotes ARP têm como origem um IP / MAC já definido. No entanto, como destino têm ‘broadcast’ pois estes pacotes são emitidos para toda a rede local (LAN), tal como visível na figura 1. Funcionamento: questionam à rede quem tem o IP X e esperam que o IP X responda dizendo: o IP X está no MAC Y.

#### 4.1.3 Que pacotes são gerados aquando de um comando *ping*?

O comando ping gera pacotes de request (pedido de resposta) e reply (resposta) que seguem o protocolo ICMP (Internet Control Message Protocol) – ‘ICMP echo request packets’, tal como visível na figura 1.

#### 4.1.4 Quais são os endereços IP e MAC dos pacotes de *ping*?

Os pacotes ping ficam com o endereço IP e MAC de origem do computador de onde são enviados, e com o IP e MAC de destino dos computadores para os quais o ping é enviado (serve de exemplo a figura 1).

#### 4.1.5 Como é possível determinar se uma trama recebida é do tipo ARP, IP ou ICMP?

Após o processo de *demultiplexing* é obtido o cabeçalho da trama. Este *header* tem uma correspondência única com cada protocolo, sendo, por exemplo, 0x0806 para as tramas do tipo ARP.

#### 4.1.6 Como é possível determinar o tamanho de uma trama recebida?

O tamanho de uma trama só é descoberto após a decodificação total da mesma, num comportamento semelhante a strings com terminação em NULL em C. Assim, o último *byte* de uma trama terá um valor especial que o sinaliza como tal.

#### 4.1.7 O que é e qual a importância da interface de *loopback*?

Uma interface de *loopback* é uma interface de rede virtual que permite que um cliente e um servidor no mesmo host/computador comuniquem entre si usando a pilha de protocolos TCP/IP. Por convenção é usado o endereço IP 127.0.0.1 para esta interface. Este mecanismo pode ser usado para testes de software ou de conectividade, permitindo verificar o funcionamento de parte da pilha TCP/IP num computador. Os pacotes enviados por esta interface nunca chegam à interface de rede física da máquina, o que permite testar software mesmo na ausência de tal interface de hardware.

### 4.2 Experiência 2 - Implementar duas *LANs* virtuais no *switch*

O objetivo desta experiência era a criação de duas *LANs* virtuais no *switch*: a primeira constituída pelas máquinas 1 e 4, e a segunda pela máquina 2. Com esta configuração, as máquinas 1 e 4 conseguem comunicar entre si, estando a máquina 2 isolada das restantes.

Foram efetuados comandos ping entre as máquina 1 e 4 e entre as máquinas 1 e 2, tendo este último falhado (como era de esperar), e o primeiro sido efetuado com sucesso, estando exposto na figura seguinte.

172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) request	id=0x666c, seq=2/512, ttl=64
172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply	id=0x666c, seq=2/512, ttl=64
172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) request	id=0x666c, seq=3/768, ttl=64
172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply	id=0x666c, seq=3/768, ttl=64
Cisco_3a:f6:03	Spanning-tree-(for...	STP	60	Conf. Root = 32768/50/fc:fb:fb:3a:f6:00	Cost = 0

Figura 2: Logs do comando ping com origem na máquina 1 e destino na máquina 4.

#### 4.2.1 Configuração da vlan50

Para a configuração de uma vlan, neste caso vlan50, tem primeiro de se criar a vlan. Para tal, tem de se criar ligações físicas entre os port do switch e os computadores que se prentede que façam parte da vlan . De seguida, no host que se encontra capaz de configurar o switch correm-se os seguintes comandos:

---

```
configure terminal
vlan 50
end
```

---

Agora que a a vlan já se encontra criada, é necessário criar as ligações virtuais correspondentes às ligações físicas porta-computador. Para tal, no host configurador do switch, inserem-se os seguintes comandos:

---

```
configure terminal
interface fastethernet 0/1
switchport mode access
switchport access vlan 50
end
```

---

Após a inserção desta configuração, passa a ser possível a comunicação entre os computadores configurados, através da vlan.

#### 4.2.2 Quantos *broadcast domains* há?

Existem dois *broadcast domains*, um por cada *vlan/subnet*, como é possível averiguar através dos logs obtidos durante a execução do comando **ping -b <ip>** (broadcast) com origem na máquina 1 e na máquina 2.

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

Esta experiência tinha como objetivo a configuração da máquina 4 como um router entre as duas sub-redes criadas na experiência anterior.

Para este efeito, foi necessário ligar a interface *eth1* - *ethernet 1* - da máquina 4 e configurá-la com um IP dentro da mesma gama da máquina 2; adicionando, de seguida, esta interface à sub-rede da máquina 2.

```
tux54:~# route -n
Kernel IP routing table
Destination    Gateway         Genmask         Flags Metric Ref    Use Iface
172.16.50.0    172.16.50.1    255.255.255.0   UG    0      0      0 eth0
172.16.50.0    0.0.0.0        255.255.255.0   U      0      0      0 eth0
172.16.51.0    0.0.0.0        255.255.255.0   U      0      0      0 eth1
```

Figura 3: Configuração da *routing table* da máquina 4.

Após este passo, adicionou-se uma rota à máquina 1 utilizando o comando `<route add -net 172.16.y1.0/24 gw 172.16.y0.254>`. O primeiro endereço identifica a gama de endereços para a qual se quer adicionar a rota; o segundo endereço identifica o IP para o qual se deve reencaminhar o pacote (neste caso o IP da máquina 4).

```
tux51:~# route -n
Kernel IP routing table
Destination    Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0        172.16.50.254  0.0.0.0         UG    0      0      0 eth0
172.16.50.0    172.16.50.254  255.255.255.0   UG    0      0      0 eth0
172.16.50.0    0.0.0.0        255.255.255.0   U      0      0      0 eth0
172.16.51.0    172.16.50.254  255.255.255.0   UG    0      0      0 eth0
```

Figura 4: Configuração da *routing table* da máquina 1.

De seguida, repetiu-se o procedimento para a máquina 2, mas utilizando os seguintes endereços: `<route add -net 172.16.y0.0/24 gw 172.16.y1.253>` (sendo o IP 172.16.y1.253 o endereço da máquina 4 nesta sub-rede).

```
tux52:~# route -n
Kernel IP routing table
Destination    Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0        172.16.51.253  0.0.0.0         UG    0      0      0 eth0
172.16.50.0    172.16.51.253  255.255.255.0   UG    0      0      0 eth0
172.16.51.0    0.0.0.0        255.255.255.0   U      0      0      0 eth0
```

Figura 5: Configuração da *routing table* da máquina 2.

Finalmente, foi possível enviar um comando *ping* da máquina 1 para a máquina 2 com sucesso. O pedido para o IP da máquina 2 (172.16.y1.1), é reencaminhado para a máquina 4 (172.16.y0.254); como a máquina 4 está ligada à sub-rede de ambas as máquinas, consegue aceder à máquina 2 através da sua interface eth1, que está nessa sub-rede, e à máquina 1 através da sua interface eth0. Na resposta, o processo é idêntico, sendo o pacote reencaminhado da máquina 2 para a máquina 1, através da máquina 4.

#### 4.3.1 Que rotas há nos *tuxes*? E qual os seus significados?

Em todos os *tuxes* existem redireccionamentos de rotas (ip forwarding). O redireccionamento de ip consiste no redireccionamento de pacotes através de um *gateway* que tem acesso à origem e ao destino do respetivo pacote -neste caso o *gateway* é o tux4. Assim, as rotas verificadas nos *tuxes* são:

- tuxy1: redireccionamento do endereço IP do tuxy2 (.1) para o IP do router na respetiva vlan, tuxy4 (.254) – quanto tenta comunicar com o tuxy2 a mensagem é ao invés enviada

172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) request	id=0x0e8d, seq=1/256, ttl=64
172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) reply	id=0x0e8d, seq=1/256, ttl=64
172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) request	id=0x0e8d, seq=2/512, ttl=64
172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) reply	id=0x0e8d, seq=2/512, ttl=64

Figura 6: *Logs* da interface *eth0* da máquina 4.

172.16.51.253	172.16.51.1	ICMP	98	Echo (ping) request	id=0x0e97, seq=2/512, ttl=64
172.16.51.1	172.16.51.253	ICMP	98	Echo (ping) reply	id=0x0e97, seq=2/512, ttl=64
172.16.51.253	172.16.51.1	ICMP	98	Echo (ping) request	id=0x0e97, seq=3/768, ttl=64
172.16.51.1	172.16.51.253	ICMP	98	Echo (ping) reply	id=0x0e97, seq=3/768, ttl=64

Figura 7: *Logs* da interface *eth1* da máquina 4.

para o tuxy4. Redirecionamento do endereço default IP para o endereço IP do router na respetiva vlan, tuxy4 (.254) – rota usada caso não seja encontrada mais nenhuma opção possível.

- tuxy2: redirecionamento do endereço IP do tuxy1 (.1) para o IP do router na respetiva vlan, tuxy4 (.253) - quanto tenta comunicar com o tuxy1 a mensagem é ao invés enviada para o tuxy4. Redirecionamento do endereço default IP para o endereço IP do router na respetiva vlan, tuxy4 (.253) – rota usada caso não seja encontrada mais nenhuma opção possível.

As referidas configurações de *routing tables* são visíveis nas figuras 3 (para o tux4), 4 (para o tux1) e 5 (para o tux2).

#### 4.3.2 Que informação contém uma entrada na *forwarding table*?

Uma linha da *forwarding table* contém informação relativamente a: destino da mensagem (para onde foi pedido que a mensagem fosse enviada); gateway, para onde a mensagem será verdadeiramente enviada/redirecionada; genmask, a mascara de rede para a rede de destino; iface, a interface para a qual o pacote será enviado; e outras informações não tão relevantes como: ref, metric e use.

#### 4.3.3 Que mensagens ARP, e endereços MAC associados, são observados?

Na tentativa do tuxy1 dar ping ao tuxy2, é possível verificar a mensagem ARP em que o tuxy4 pede o endereço MAC relativo ao IP do tuxy2. O tuxy1 envia a informação para o tuxy2, sendo esta redirecionada para o tuxy4 através da tabela de routing. Visto o tuxy4 não possuir na sua tabela ARP o endereço MAC do tuxy2 este necessita de transmitir (broadcast) um pacote ARP para descobrir esse endereço MAC.

HewlettP_5a:7c:e7	BbnBoltB_a0:ac:80	ARP	60	Who has 172.16.51.253? Tell 172.16.51.1
BbnBoltB_a0:ac:80	HewlettP_5a:7c:e7	ARP	42	172.16.51.253 is at 00:01:02:a0:ac:80

Figura 8: Mensagens ARP capturadas na máquina 4, interface eth1.

Na tentativa do tuxy2 dar ping ao tuxy1, irá acontecer um processo semelhante ao referido anteriormente, pelos mesmo motivos, sendo que neste caso o tuxy4 emitirá um pacote ARP para descobrir o endereço MAC do tuxy1.

HewlettP_5a:75:bb	HewlettP_19:09:5c	ARP	60	Who has 172.16.50.254? Tell 172.16.50.1
HewlettP_19:09:5c	HewlettP_5a:75:bb	ARP	42	172.16.50.254 is at 00:22:64:19:09:5c

Figura 9: Mensagens ARP capturadas na máquina 4, interface eth0.

No início da comunicação, quer o tuxy1, quer o tuxy2 não sabem qual o endereço MAC do respetivo tuxy4 e, portanto, também emitem um pacote ARP para descobrir esse endereço (no caso do tuxy1 o .254 e no caso de tuxy2 o .253).

HewlettP_19:09:5c	HewlettP_5a:75:bb	ARP	60	Who has 172.16.50.1? Tell 172.16.50.254
HewlettP_5a:75:bb	HewlettP_19:09:5c	ARP	42	172.16.50.1 is at 00:21:5a:5a:75:bb

Figura 10: Mensagens ARP capturadas na máquina 1.

#### 4.3.4 Que pacotes ICMP são observados, e porquê?

Na sequência da execução do comando ping no tux1, com o tux2 como destino, começa-se por verificar a existência de pacotes ICMP cujo endereço IP de origem é o tuxy1 e o endereço IP de destino é o tuxy2. No entanto, devido à possibilidade de *forwarding*, esses pings passam a ser do tuxy1 para o tuxy4 .254, e do tuxy1 para o tuxy4 .253 (ver figura 11).

172.16.50.1	172.16.51.1	ICMP	98	Echo (ping) request	id=0x157a, seq=12/3072, ttl=64
172.16.51.1	172.16.50.1	ICMP	98	Echo (ping) reply	id=0x157a, seq=12/3072, ttl=63
Cisco_7b:d5:01	Spanning-tree-(for-bridges)_00	STP	60	Conf. Root = 32768/50/00:1e:14:7b:d5:00	Cost = 0
Cisco_7b:d5:01	Cisco_7b:d5:01	LOOP	60	Reply	
Cisco_7b:d5:01	Spanning-tree-(for-bridges)_00	STP	60	Conf. Root = 32768/50/00:1e:14:7b:d5:00	Cost = 0
172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) request	id=0x1584, seq=1/256, ttl=64 (
172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply	id=0x1584, seq=1/256, ttl=64 (
172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) request	id=0x1584, seq=2/512, ttl=64 (
172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply	id=0x1584, seq=2/512, ttl=64 (

Figura 11: Pacotes ICMP capturados no tux1.

#### 4.3.5 Quais são os endereços IP e MAC associados aos pacotes ICMP?

Os endereços MAC associados aos pacotes observados são os das respetivas máquinas de destino e origem (tux1 e tux4, respetivamente). No entanto, devido ao *forwarding* de pacotes pelo tux4, os pacotes podem ter endereço IP de destino correspondente ao do tux4 (terminado em 50.254 ou 51.253). Este facto é visualizado nas várias figuras de logs anteriores (6 a 11).

### 4.4 Experiência 4 - Configurar um router comercial e implementar NAT

O objetivo desta experiência era adicionar à sub-rede a ligação ao router e configura-lo com funcionalidades NAT de forma a estabelecer ligação à Internet. Para este objetivo foi necessário configurar duas *interfaces* do router: interface *gigabitEthernet 0/0* ligada à *vlan1* e a interface *gigabitEthernet 0/1* ligada à rede exterior da sala.

A partir da consola de configuração para ambas *interfaces* foi definido o seu endereço IP e *mask* através do comando **ip address <ip><mask>** em que o IP era 172.16.51.254 para a interface *0/0* e 172.16.2.59 para a interface *0/1* com máscara de 24 bits. De seguida,

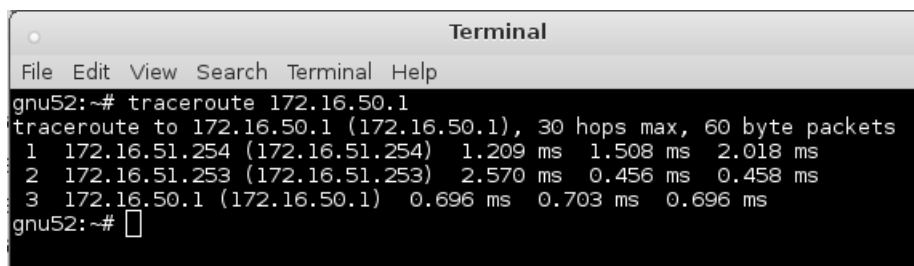
introduziu-se o comando **no shutdown** para que as configurações não fossem perdidas caso o router fosse desligado e finalmente foi definido o ponto de entrada e saída de NAT através do comando **nat inside/outside** sendo aplicados respectivamente à *interface 0/0* e *interface 0/1*.

Após as *interfaces* estarem configuradas, foi necessário executar os comandos **ip nat pool ovrld 172.16.2.59 172.16.2.59 prefix 24** e **ip nat inside source list 1 pool ovrld overload** para garantir a gama de endereços.

No seguimento, executou-se o comando **accesslist 1 permit ip <max>** sobre ambas sub-redes - 172.16.50.0 e 172.16.51.0 – com o max definido com 0.0.0.255 para criar uma lista de acessos e permissões de pacotes.

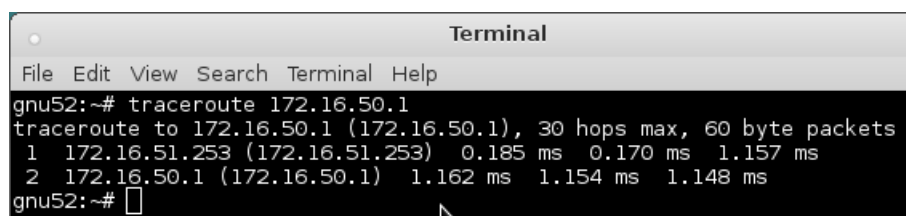
Finalmente, definiu-se as rotas através do comando **ip route <dest><mask><gw>** de forma a que os pacotes fossem redirecionados para o local correto. Foram adicionadas duas rotas: uma interna através do comando **ip route 0.0.0.0 0.0.0.0 172.16.1.254** para garantir que os pacotes eram enviados para a sub-rede 172.16.51.0 e uma rota externa através do comando **ip route 172.16.50.0 255.255.255.0 172.16.51.253** para garantir que os pacotes enviados para a sub-rede 172.16.50.0 eram redirecionados para o tuxy4 que estabelece ligação com esta sub-rede.

Adicionalmente foram utilizados os comandos **route add** e **route del** para introduzir e remover, respetivamente, entradas nas listas de reencaminhamento e `traceroute [ip];` para analisar o percurso dos pacotes ao longo da rede.



```
Terminal
File Edit View Search Terminal Help
gnu52:~# traceroute 172.16.50.1
traceroute to 172.16.50.1 (172.16.50.1), 30 hops max, 60 byte packets
 1 172.16.51.254 (172.16.51.254) 1.209 ms 1.508 ms 2.018 ms
 2 172.16.51.253 (172.16.51.253) 2.570 ms 0.456 ms 0.458 ms
 3 172.16.50.1 (172.16.50.1) 0.696 ms 0.703 ms 0.696 ms
gnu52:~#
```

Figura 12: Traceroute sem rota para sub-rede 172.16.50.0



```
Terminal
File Edit View Search Terminal Help
gnu52:~# traceroute 172.16.50.1
traceroute to 172.16.50.1 (172.16.50.1), 30 hops max, 60 byte packets
 1 172.16.51.253 (172.16.51.253) 0.185 ms 0.170 ms 1.157 ms
 2 172.16.50.1 (172.16.50.1) 1.162 ms 1.154 ms 1.148 ms
gnu52:~#
```

Figura 13: Traceroute com rota para sub-rede 172.16.50.0

Através da análise dos resultados conclui-se que o router tem a capacidade de reencaminhar os pacotes para outras sub-redes quando no terminal não está definida a rota e que o a funcionalidade NAT é essencial à visibilidade da sub-rede para o exterior.

## 4.5 Experiência 5 - DNS

Nesta experiência pretendia-se configurar o servidor de DNS para permitir ligação à Internet através da procura de nomes de domínios. Domain Name System (DNS) é responsável por associar e traduzir diversa informação associada aos nomes dos domínios em particular o seu IP.

Para associar o DNS à sub-rede foi necessário modificar o ficheiro `/etc/resolv.conf` em todas as máquinas. Para tal, editou-se o seu conteúdo de acordo com o formato **search new-page-name <nameserver><IP-page>**, sendo que *new-page-name* representa o nome da página, e *IP-page* o endereço IP da página.

O conteúdo do ficheiro passou a ser:

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

Para testar esta funcionalidade foi feito um ping ao domínio `www.reddit.com`.

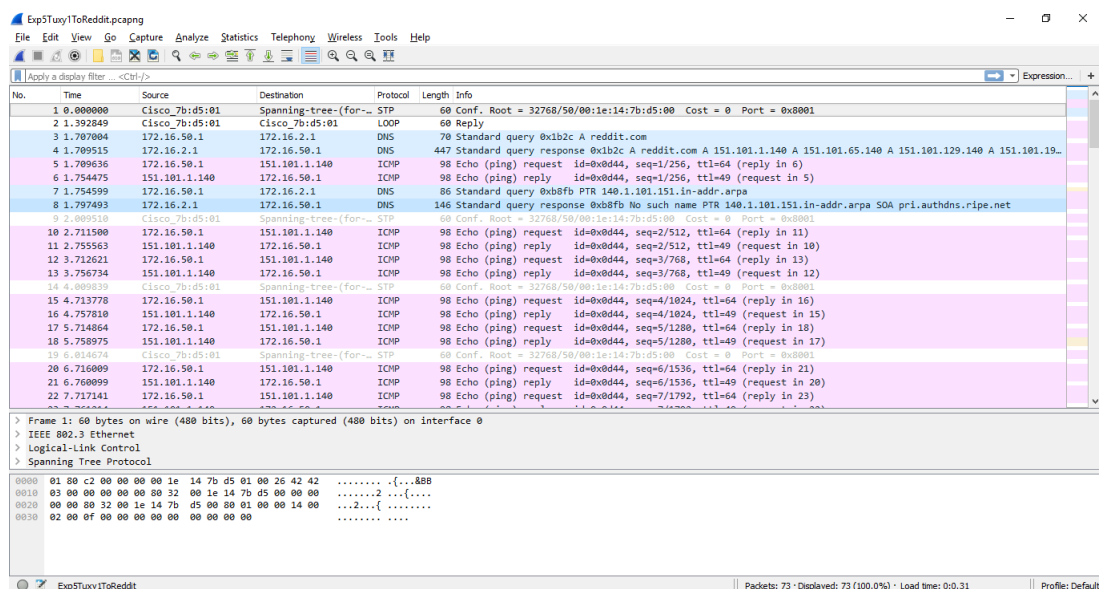


Figura 14: Pacotes registados no ping a de reddit.com

Através da análise dos resultados do *Wireshark* podemos observar que no início da comunicação, antes do envio/receção de pacotes ICMP (enviados pelo comando **ping <ip>**), foram enviados pacotes DNS para identificar o endereço IP do destino (neste caso `reddit.com`).

Foi pedido ao DNS que enviasse os atributos do domínio através do pacote 3. Como resposta o servidor enviou o pacote 4 que inclui entre outras informações o endereço IP do destino.

Posteriormente, foi também feito um pedido *reverse* DNS (rDNS) nos pacotes 7 e 8 que processa o pedido contrário, ou seja, obter o nome domínio a partir do endereço IP.

## 4.6 Experiência 6 -Ligações TCP

Por fim, o objetivo desta experiência era desenvolver uma aplicação de download com base no protocolo FTP e analisar as características da comunicação tais como as ligações

de controlo e dados e as suas fases, os dados transferidos através da ligação de controlo, o mecanismo ARQ e o mecanismo de controlo de congestão.

O mecanismo ARQ no TCP baseia-se numa variante de *Go-Back-N* utilizando uma janela deslizante, em que são enviados **ACKs** com um único número de sequência para informar o emissor que os pacotes até esse número foram recebidos com sucesso. Em caso de erro ou perda de pacotes são enviados **ACKs** duplicados até que o emissor reenvie os pacotes perdidos um de cada vez (assunção otimista). É também utilizado timeout para quando uma resposta não é recebida. Este timeout é adaptativo, calculado dinamicamente de acordo com RTT (*Round Trip Time*) medido ao longo da transmissão. Existe ainda a possibilidade de utilizar *selective acknowledgements* (**SACKs**). Para testar a aplicação foram feitas duas experiências. Em ambos os casos foi transferido um ficheiro com 758MB a partir do servidor ftp.up.pt.

Na primeira experiência a transferência foi feita a partir da maquina tuxy1.

Ao longo da execução da aplicação foram abertos dois canais de comunicação: um de controlo por onde são enviados vários comandos tais como **USER**, **PASV**, **RETR**, etc e um de dados por onde são recebidos os pacotes do ficheiro.

Podemos identificar 3 fases durante a comunicação: estabelecimento de conexão, transferência e terminação de conexão. A 1ª ocorre logo no início da transferência e é caracterizada por um handshake de 3 passos: envio de **SYN**, resposta de [**SYN**, **ACK**] e envio de **ACK**. Esta fase está visível na sequência dos pacotes 4-6 (Anexo I - figura 18).

A 2ª corresponde ao processo de transferência e é visível durante o envio e resposta de comandos FTP. Durante a transferência do ficheiro tornam-se evidentes várias características do TCP incluindo o mecanismo ARQ quando é pedido a retransmissão com pacotes duplicados (ex: pacote 62759 (Anexo I - figura 19)).

A 3ª ocorre no fim da transferência e é caracterizada por um handshake de 3 passos: envio de **FIN**, resposta de [**FIN**, **ACK**], envio de **ACK**. Esta fase está visível na sequência dos pacotes 875476-875478 (ver Anexo I - figura 15 & Anexo I - figura 20).

Através da análise do gráfico podemos concluir que o throughput inicialmente aumentou de forma linear até atingir um valor aproximado de 1300 pacotes/s e que se manteve em média por volta deste valor apenas com esporádicos decréscimos ao longo da transferência e por fim diminuiu novamente de forma linear.

O aspeto deste gráfico não está totalmente de acordo com nenhum dos algoritmos estudados, porém é possível detetar através do aumento linear rápido de throughput inicial que a janela de congestão estará a ser incrementada por um fator em cada RTT. É possível também analisar que deverá estar a ser utilizada a técnica de congestion avoidance já que após um decréscimo, são visíveis duas fases de incremento da janela de congestão: inicialmente mais rápida semelhante ao observado no início da transmissão e posteriormente mais lento, mas ainda linear.

Na segunda experiência foi feita a transferência a partir do tuxy1 e no decurso desta iniciou-se outra no tuxy2 (ver Anexo I - figura 16 & Anexo I - figura 17).

Através da análise dos gráficos podemos concluir que a transferência em simultâneo provocou um impacto no throughput da máquina 1, tendo em conta que se manteve em média por volta dos 1200 pacotes/s, ocorreram mais decréscimos e o tempo aumentou de 70s (re-



gistado na experiência anterior) para 74s.

Em relação à máquina 2, o resultado registado encontra-se mais próximo do teoricamente esperado. Porém, podemos analisar que o impacto da transferência da máquina 1 foi mínimo uma vez que a evolução do throughput manteve-se idêntica ao longo da transferência, mesmo após o tuxyl ter terminado.

## 5 Conclusões

No final, foi possível concluir que a aplicação implementada era robusta e permitia um *download* de ficheiros na sua íntegra e sem erros.

O grupo considera também que a realização e compreensão exaustiva das experiências traduziu-se posteriormente na facilidade de desenvolvimento da aplicação e na compreensão de todos os conceitos relativos ao projeto. Assim, é conclusão unânime que as experiências se revelaram uma grande fonte de conhecimento e parte mais complexa e trabalhosa do projeto.

O grupo considera ter dominado os objetivos implícitos no trabalho tais como: compreensão do conceito de ‘cliente - servidor’, compreensão do protocolo de comunicação *TCP / IP*, compreensão do protocolo de comunicação *FTP*, compreensão do serviço *DNS*, compreensão do uso de *sockets* e compreensão de como localizar e ler *RFC*'s . O grupo considera que o desenvolvimento do protocolo consolidou os conhecimentos teóricos previamente adquiridos.

## 6 Anexo I

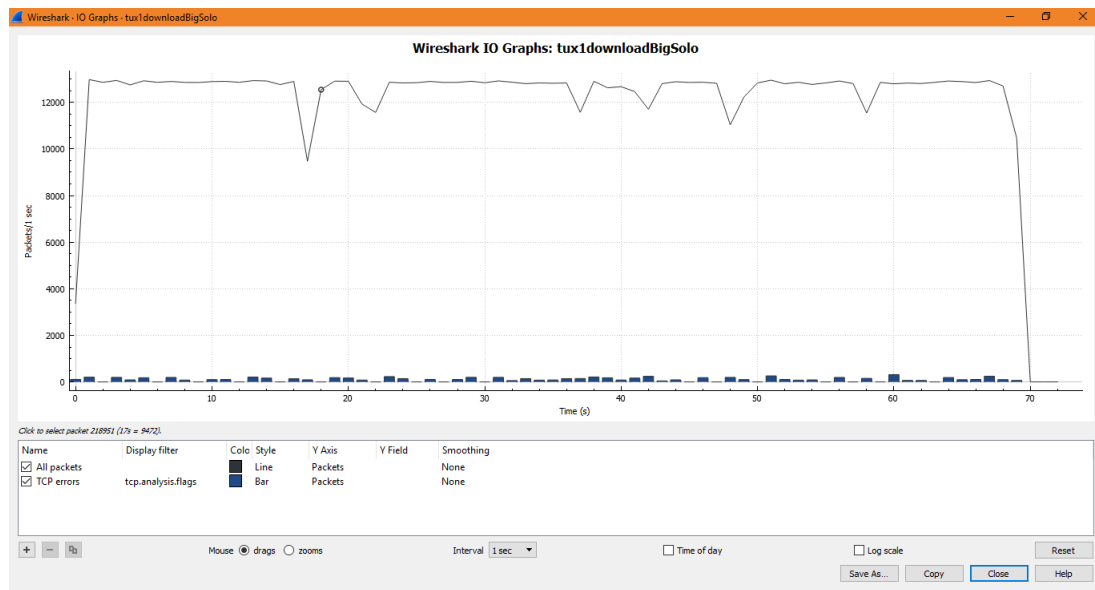


Figura 15: Variação de *throughput* de tuxy1

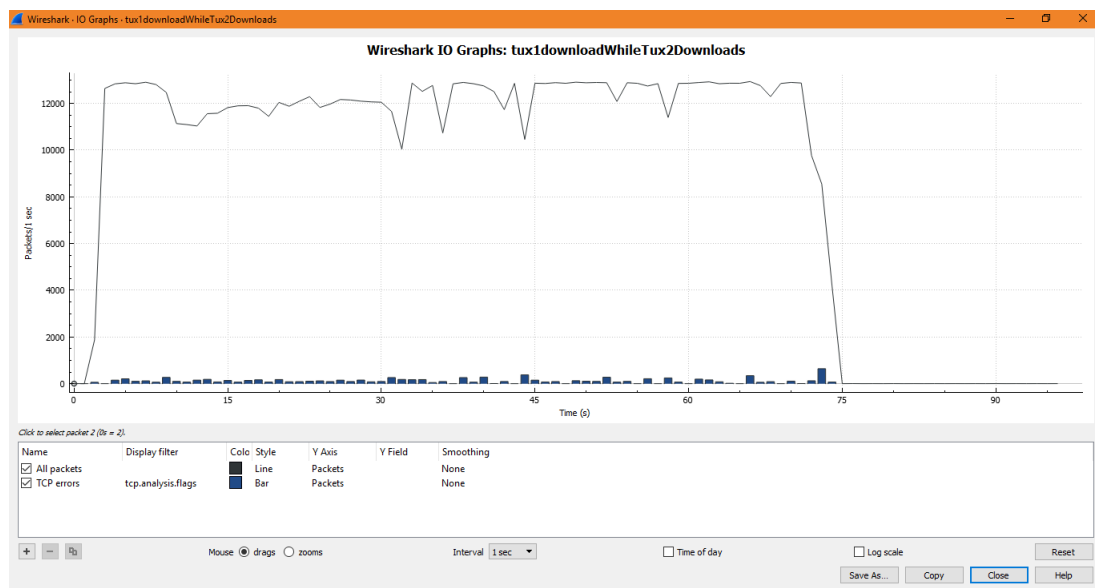


Figura 16: Variação de *throughput* de tuxy1 durante transferência em simultâneo

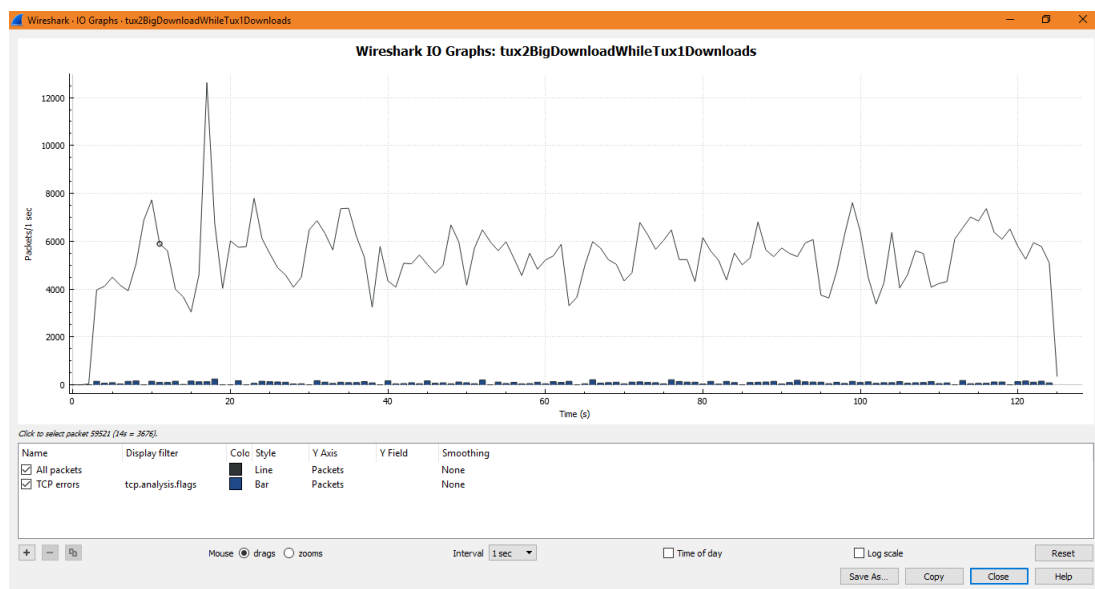


Figura 17: Variação de *throughput* de tuxy2 durante transferência em simultâneo



hwiidownload@igSolo.pcsang

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter: <Ctrl>

No.	Time	Source	Destination	Protocol	Length	Info
875455	69.813676	193.137.29.15	172.16.50.1	FTP-DA	1434	FTP Data: 1368 bytes
875456	69.813799	193.137.29.15	172.16.50.1	FTP-DA	1434	FTP Data: 1368 bytes
875457	69.813855	172.16.50.1	193.137.29.15	TCP	66	35129 → 54795 [ACK] Seq=1 Ack=794807425 Win=2386432 Len=0 TSval=42983568 TSecr=629169994
875458	69.813915	193.137.29.15	172.16.50.1	FTP-DA	1434	FTP Data: 1368 bytes
875459	69.814031	193.137.29.15	172.16.50.1	FTP-DA	1434	FTP Data: 1368 bytes
875460	69.814054	172.16.50.1	193.137.29.15	TCP	66	35129 → 54795 [ACK] Seq=1 Ack=794810161 Win=2386432 Len=0 TSval=42983568 TSecr=629169994
875461	69.814144	193.137.29.15	172.16.50.1	FTP-DA	1434	FTP Data: 1368 bytes
875462	69.814205	193.137.29.15	172.16.50.1	FTP-DA	1434	FTP Data: 1368 bytes
875463	69.814281	172.16.50.1	193.137.29.15	TCP	66	35129 → 54795 [ACK] Seq=1 Ack=794812897 Win=2386432 Len=0 TSval=42983568 TSecr=629169994
875464	69.814368	193.137.29.15	172.16.50.1	FTP-DA	1434	FTP Data: 1368 bytes
875465	69.814498	193.137.29.15	172.16.50.1	FTP-DA	1434	FTP Data: 1368 bytes
875466	69.814519	172.16.50.1	193.137.29.15	TCP	66	35129 → 54795 [ACK] Seq=1 Ack=794815633 Win=2386432 Len=0 TSval=42983568 TSecr=629169994
875467	69.814608	193.137.29.15	172.16.50.1	FTP-DA	1434	FTP Data: 1368 bytes
875468	69.814733	193.137.29.15	172.16.50.1	FTP-DA	1434	FTP Data: 1368 bytes
875469	69.814751	172.16.50.1	193.137.29.15	TCP	66	35129 → 54795 [ACK] Seq=1 Ack=794818369 Win=2386432 Len=0 TSval=42983568 TSecr=629169994
875470	69.814838	193.137.29.15	172.16.50.1	FTP-DA	1434	FTP Data: 1368 bytes
875471	69.814924	193.137.29.15	172.16.50.1	FTP-DA	938	FTP Data: 872 bytes
875472	69.814956	172.16.50.1	193.137.29.15	TCP	66	35129 → 54795 [ACK] Seq=1 Ack=794820610 Win=2386432 Len=0 TSval=42983568 TSecr=629169994
875473	69.817168	193.137.29.15	172.16.50.1	FTP	90	Response: 226 Transfer complete.
875474	69.817181	172.16.50.1	193.137.29.15	TCP	66	68312 → 21 [ACK] Seq=100 Ack=681 Win=29312 Len=0 TSval=42983561 TSecr=629169996
875475	69.879612	172.16.50.1	193.137.29.15	FTP	72	Request: QUIT
875476	69.879639	172.16.50.1	193.137.29.15	TCP	66	35129 → 54795 [FIN, ACK] Seq=1 Ack=794820610 Win=2386432 Len=0 TSval=42983601 TSecr=629169994
875477	69.879656	172.16.50.1	193.137.29.15	TCP	66	68312 → 21 [RST, ACK] Seq=114 Ack=681 Win=29312 Len=0 TSval=42983601 TSecr=629169996
875478	69.901501	193.137.29.15	172.16.50.1	TCP	66	54795 → 35129 [ACK] Seq=794820610 Ack=2 Win=29656 Len=0 TSval=629170038 TSecr=42983601
875479	70.175105	Cisco_70:05:01	Spanning-tree (for-br...	STP	90	Conf. Root = 32768/50/00:1e:14:70:05:00 Cost = 0 Port = 0x0001
875480	71.141030	HewlettP_5a:75:bb	HewlettP_19:09:5c	ARP	42	Who has 172.16.50.254? Tell 172.16.50.1
875481	71.141173	HewlettP_19:09:5c	HewlettP_5a:75:bb	ARP	60	172.16.50.254 is at 08:22:64:19:09:5c
875482	72.175024	Cisco_70:05:01	Spanning-tree (for-br...	STP	90	Conf. Root = 32768/50/00:1e:14:70:05:00 Cost = 0 Port = 0x0001

Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0

Ethernet II, Src: Intel 802.3 Ethernet

Logical-Link Control

Spanning Tree Protocol

Figura 20: Fim da transferência no tuxyl

## 7 Anexo II

### main.c

---

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

#include "URL.h"
#include "utils.h"
#include "clientFTP.h"

#define BUFFER_SIZE 1024

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

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

    if (argc != 2) {
        printUsage(argv);
        exit(1);
    }

    downloadFtpUrl(argv[1]);

    return 0;
}
```

---

# clientFTP.h

---

```
#ifndef __CLIENTFTP_H
#define __CLIENTFTP_H

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

#define DEBUG          0

#define SERVER_READY    "220"
#define TRANSFER_COMPLETE "226"
#define REQUIRED_PASSWORD "331"
#define SUCCESS_LOGIN   "230"
#define FINISHED        "150"
#define PASSIVE_MODE    "227"
#define DIRECTORY_OK    "250"
#define SET_TYPE        "200"

typedef struct ftp_t {
    int fdControl;
    int fdData;
} FTP;

int downloadFtpUrl(const char * str);

#endif /* __CLIENTFTP_H */
```

---

# clientFTP.c

---

```
#include "clientFTP.h"
#include "URL.h"
#include "utils.h"

static FTP * ftp;
static URL * url;

static void quitConnection();

static void forceQuit(char * errorMSg){
    int errorStatus = logError(errorMSg);
    quitConnection();
    exit(errorStatus);
}

static void printResponse(char * responseCmd){
    if(DEBUG)
        printf("%s", responseCmd);
    else {
        int msgLength = strlen(responseCmd)-RESPONSE_CODE_OFFSET;
        char * msgWithoutCode = (char *) malloc(msgLength);
        memcpy(msgWithoutCode, responseCmd+RESPONSE_CODE_OFFSET, msgLength+1);
        printf("%s", msgWithoutCode);
        free(msgWithoutCode);
    }
}

static int receiveCommand(int fd, char* responseCmd, char * expectedAnswer) {

    FILE* fp = fdopen(fd, "r");
    int allocated = 0;
    if(responseCmd == NULL) {
        responseCmd = (char*) malloc(sizeof(char) * MESSAGE_SIZE);
        allocated = 1;
    }
    do {
        memset(responseCmd, 0, MESSAGE_SIZE);
        responseCmd = fgets(responseCmd, MESSAGE_SIZE, fp);
        printResponse(responseCmd);
    } while (!('1' <= responseCmd[0] && responseCmd[0] <= '5') || responseCmd[3] !=
        ' ');

    char responseStatus = responseCmd[0];

    if(expectedAnswer != NULL && strncmp(expectedAnswer, responseCmd,
        strlen(expectedAnswer))) {
        if(allocated)
            free(responseCmd);
        return ERROR;
    }
}
```



```

    if(allocated)
        free(responseCmd);

    return (responseStatus > '4');
}

static int sendCommand(int fd, const char* msg, char* response, int readResponse,
    char * expectedAnswer) {
    int nBytes = write(fd, msg, strlen(msg));
    if (readResponse)
        return receiveCommand(fd, response, expectedAnswer);
    else return (nBytes == 0);
}

static int connectSocket(const char* ip, int port) {
    int sockfd;
    struct sockaddr_in server_addr;

    /*server address handling*/
    bzero((char*)&server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr(ip); /*32 bit Internet address network
        byte ordered*/
    server_addr.sin_port = htons(port); /*server TCP port must be network byte
        ordered */

    /*open an TCP socket*/
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket()");
        exit(0);
    }

    /*connect to the server*/
    if (connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0){
        perror("connect()");
        return -1;
    }

    return sockfd;
}

static void retrieveFile(int fd) {
    char userCommand[MESSAGE_SIZE + 1];

    sendCommand(fd, "TYPE L 8\r\n", NULL, TRUE, SET_TYPE); //Setting type of file
        to be transferred -> local file
    sprintf(userCommand, "RETR %s%s\r\n", url->path, url->filename);
    if(sendCommand(fd, userCommand, NULL, TRUE, NULL) != OK)
        forceQuit("Failed to retrieve file. Terminating Program.");
}

```

```

static void sendUSER(int fd) {

    char userCommand[MESSAGE_SIZE + 1];
    char passCommand[MESSAGE_SIZE + 1];

    receiveCommand(fd, NULL, NULL);

    if (strcmp(url->username, "anonymous") == OK)
        printf("Logging in: anonymous mode.\n");
    else
        printf("Logging in: authentication mode");

    sprintf(userCommand, "USER %s\r\n", url->username);
    if (sendCommand(fd, userCommand, NULL, TRUE, REQUIRED_PASSWORD) != OK)
        forceQuit("Failed to log in, wrong username?\nTerminating Program.");

    sprintf(passCommand, "PASS %s\r\n", url->password);
    if (sendCommand(fd, passCommand, NULL, TRUE, SUCCESS_LOGIN) != OK)
        forceQuit("Failed to log in, wrong password?\nTerminating Program.");
}

static void sendPASV(int fd) {

    char response[MESSAGE_SIZE + 1];

    if (sendCommand(fd, "PASV\r\n", response, TRUE, PASSIVE_MODE) != OK)
        forceQuit("Failed to enter passive mode. Terminating program.");

    int remoteIP[6];
    char* data = strchr(response, '(');
    sscanf(data, "(%d, %d, %d, %d, %d, %d)",
           &remoteIP[0], &remoteIP[1], &remoteIP[2], &remoteIP[3], &remoteIP[4], &remoteIP[5]);
    sprintf(url->ip, "%d.%d.%d.%d",
           remoteIP[0], remoteIP[1], remoteIP[2], remoteIP[3]);
    url->port = remoteIP[4]*256+remoteIP[5];
}

static int download(int fd) {
    FILE* outfile;
    if ( !(outfile = fopen(url->filename, "w")) )
        return logError("Unable to open file.");

    int dots = 0;
    printf("Downloading.");

    char buf[SOCKET_SIZE];
    int bytes;
    while ( (bytes = read(fd, buf, sizeof(buf))) != 0 ) {
        if (bytes < 0)

```

```

        return logError("Empty data socket, nothing to receive.\n");

    if ((bytes = fwrite(buf, bytes, 1, outfile)) < 0) {
        return logError("Unable to write data in file.\n");
    }

    printDownloadProgress(&dots);
}

fclose(outfile);

printf("\rDownload finished with success.\n");

return OK;
}

static void quitConnection() {

    printf("Closing connection with the server.\n");
    if(sendCommand(ftp->fdControl, "QUIT\r\n", NULL, FALSE, NULL) != OK) {
        close(ftp->fdData);
        close(ftp->fdControl);
        exit(logError("Failed to exit connection. Forcing exit.\n"));
    }

    close(ftp->fdData);
    close(ftp->fdControl);
}

int downloadFtpUrl(const char * str) {

    ftp = (FTP *) malloc(sizeof(FTP));

    url = constructURL();
    if (parseURL(url, str))
        exit(logError("Error parsing url. Please provide a valid url.\n"));

    setIp(url);

    printURL(url);

    if((ftp->fdControl = connectSocket(url->ip, url->port)) == 0)
        exit(logError("Failed to open control conection. Terminating Program.\n"));

    sendUSER(ftp->fdControl);
    sendPASV(ftp->fdControl);

    if((ftp->fdData = connectSocket(url->ip, url->port)) == 0)
        exit(logError("Failed to open data connection. Terminating Program.\n"));

    retrieveFile(ftp->fdControl);
    download(ftp->fdData);
}

```

```
destructURL(url);  
quitConnection();  
free(ftp);  
  
printf("TERMINATING PROGRAM\n");  
  
return OK;  
}
```

---

# URL.h

---

```
#ifndef __URL_H
#define __URL_H

#define URL_STR_LEN 256

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

typedef struct url_t {
    int port;
    char ip[URL_STR_LEN];    // host's ip
    char path[URL_STR_LEN];  // file path
    char hostname[URL_STR_LEN]; // hostname
    char filename[URL_STR_LEN]; // filename
    char username[URL_STR_LEN]; // username
    char password[URL_STR_LEN]; // password
} URL;

/**
 * URL struct constructor
 */
URL * constructURL();

/**
 * Parses the url command to the given URL struct
 */
int parseURL(URL* url, const char* str);

/**
 * Sets the ip field from the URL struct
 */
void setIp(URL * url);

/**
 * Prints the contents of the URL struct to STDOUT
 */
void printURL(URL * url);

/**
 * URL struct destructor
 */
void destructURL(URL * url);
```

```
#endif /* __URL_H */
```

---

# URL.c

---

```
#include <string.h>
#include <regex.h>
#include "URL.h"
#include "utils.h"

#define URL_REGEX
    "^ftp://([a-zA-Z][^:]*):([~@]+)?([a-z0-9:~.~?~+]/([~/]~+)*)([~/]~+)$"

static int setInUrl(URL * url, int idx, const char * src, int size) {
    switch (idx) {
        case 0: // whole capture
        case 1: // identity:password
        case 5: // host's country
        case 7: // path termination
            break;
        case 2: // username
            if (0 == size) {
                strcpy(url->username, "anonymous");
            } else {
                strncpy(url->username, src, size);
                url->username[size] = 0;
            }
            break;
        case 3: // password
            if (0 == size) {
                strcpy(url->password, "a");
            } else {
                strncpy(url->password, src, size);
                url->password[size] = 0;
            }
            break;
        case 4: // hostname
            strncpy(url->hostname, src, size);
            url->hostname[size] = 0;
            break;
        case 6: // path
            strncpy(url->path, src, size);
            url->path[size] = 0;
            break;
        case 8: // path
            strncpy(url->filename, src, size);
            url->filename[size] = 0;
            break;
        default:
            break;
    }

    return OK;
}
```

```

/*
 * Match the string in "to_match" against the compiled regular
 * expression in "r", storing the values in the given URL*.
 */
static int match_url_regex(regex_t * r, const char * to_match, URL * url) {
    /* "p" is a pointer into the string which points to the end of the
       previous match. */
    const char * p = to_match;

    /* "N_MATCHES" is the maximum number of matches allowed. */
    const int N_MATCHES = 10;

    /* "m" contains the matches found. */
    regmatch_t m[N_MATCHES];

    int i = 0;
    while (i++ < N_MATCHES) {
        int j = 0;
        int nomatch = regexec(r, p, N_MATCHES, m, 0);
        if (nomatch) {
            return i == 1 ? NO_MATCH : OK;
        }
        for (j = 0; j < N_MATCHES; j++) {
            int start = m[j].rm_so + (p - to_match);
            int finish = m[j].rm_eo + (p - to_match);

            setInUrl(url, j, to_match + start, finish - start);
        }
        p += m[0].rm_eo;
    }

    return 0;
}

int parseURL(URL * url, const char* str) {
    regex_t r;
    const char * regex_text = URL_REGEX;
    int ret = 0;

    ret |= compile_regex(&r, regex_text);
    ret |= match_url_regex(&r, str, url);
    regfree(&r);

    return ret;
}

void printURL(URL * url) {
    printf("\nContents of URL struct:\n");
    printf("ip: %s\n", url->ip);
    printf("path: %s\n", url->path);
    printf("hostname: %s\n", url->hostname);
    printf("filename: %s\n", url->filename);
}

```



```

    printf("username: %s\n", url->username);
    printf("password: %s\n\n", url->password);
}

URL * constructURL() {
    URL * url = malloc(sizeof(URL));

    memset(url->ip, 0, URL_STR_LEN);
    memset(url->path, 0, URL_STR_LEN);
    memset(url->hostname, 0, URL_STR_LEN);
    memset(url->filename, 0, URL_STR_LEN);
    memset(url->username, 0, URL_STR_LEN);
    memset(url->password, 0, URL_STR_LEN);
    url->port = 21;

    return url;
}

void setIp(URL * url){
    memset(url->ip, 0, URL_STR_LEN);
    strcpy(url->ip, getIp(url->hostname));
}

void destructURL(URL * url) {
    free(url);
}

```

---

# utils.h

---

```
#ifndef __UTILS_H
#define __UTILS_H

#include <sys/types.h>
#include <regex.h>

#define ERROR      1
#define OK         0
#define NO_MATCH   2

#define TRUE       1
#define FALSE      0

#define SOCKET_SIZE      32768
#define MESSAGE_SIZE     1024
#define PATH_MAX         255
#define DOWNLOAD_PROGRESS_RESET 1000
#define RESPONSE_CODE_OFFSET 4

char * getIp(char * domain);

int compile_regex(regex_t * r, const char * regex_text);

void insertCharAt(char * dest, const char * src, char c, int size, int idx);

int logError(char * msg);

void printDownloadProgress(int * dots);

#endif /* __UTILS_H */
```

---

# utils.c

---

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "utils.h"

#define MAX_ERROR_MSG 0x1000

/*
 * Compile the regular expression described by "regex_text" into "r".
 */
int compile_regex(regex_t * r, const char * regex_text) {
    int status = regcomp(r, regex_text, REG_EXTENDED|REG_NEWLINE);
    if (status != 0) {
        char error_message[MAX_ERROR_MSG];
        regerror(status, r, error_message, MAX_ERROR_MSG);
        printf("Regex error compiling '%s': %s\n",
            regex_text, error_message);
        return 1;
    }
    return 0;
}

void insertCharAt(char * dest, const char * src, char c, int size, int idx) {
    if (idx >= size) return;
    strncpy(dest, src, size);
    memmove(dest + idx + 1, src + idx, size - idx);
    dest[idx] = c;
}

char * getIp(char * domain) {
    struct hostent *h;
    if ((h=gethostbyname(domain)) == NULL) {
        perror("gethostbyname");
        exit(1);
    }

    return inet_ntoa(*(struct in_addr *)h->h_addr));
}

int logError(char * errorMsg) {
    fprintf(stderr, "Error: %s\n", errorMsg);
    return ERROR;
}

void printDownloadProgress(int * dots) {
    (*dots)++;
}
```

```
if(*dots == 3*DOWNLOAD_PROGRESS_RESET){
    *dots = 0;
    printf("\r          ");
    printf("\rDownloading.");
}
else if((( *dots)%DOWNLOAD_PROGRESS_RESET) == 0)
    printf(".");

fflush(stdout);
}
```

---