

# Rede de Computadores

## *2º Trabalho Laboratorial*

- 1. Aplicação de download**
- 2. Configuração e análise de uma rede**

Mestrado Integrado em Engenharia Informática e  
Computação

André Esteves

[up201606673@fe.up.pt](mailto:up201606673@fe.up.pt)

Francisco Friande

[up201508213@fe.up.pt](mailto:up201508213@fe.up.pt)

Luís Diogo Silva

[up201503730@fe.up.pt](mailto:up201503730@fe.up.pt)

# Índice

<b>Introdução.....</b>	<b>3</b>
<b>Parte 1 – Aplicação de download .....</b>	<b>4</b>
Arquitetura da aplicação .....	4
Exemplo de um download bem-sucedido .....	5
<b>Parte 2 – Configuração e análise de uma rede.....</b>	<b>6</b>
Experiência 1 – Configurar um IP de rede .....	6
Experiência 2 – Implementar duas LAN's virtuais num switch .....	7
Experiência 3 – Configurar um Router em Linux .....	8
Experiência 4 – Configurar um Router Comercial e Implementar NAT .....	9
Experiência 5 – DNS .....	11
Experiência 6 – Conexões TCP .....	12
<b>Conclusões.....</b>	<b>14</b>
<b>Referências.....</b>	<b>14</b>
<b>Anexos .....</b>	<b>15</b>
Imagens .....	15
Código.....	21
info.h .....	21
client.c .....	22

## Introdução

Este relatório foi elaborado no âmbito da unidade curricular de Rede de Computadores. O trabalho em causa consistia em duas partes. A primeira requer a elaboração de uma aplicação que faz o download de um ficheiro dado o seu URL e a segunda parte consiste na configuração e análise de uma rede no decorrer das aulas práticas sendo ao todo sete experiências.

O trabalho foi realizado no seu todo no ambiente disponibilizado, sendo concluído com sucesso em todos os aspetos, cumprindo os objetivos pedidos.

## Parte 1 – Aplicação de download

A primeira parte do trabalho consistia do desenvolvimento de uma aplicação de linha de comandos em C que fizesse um download de um ficheiro de um servidor FTP, como está descrito em RFC959. O programa recebe do utilizador um URL com o formato *ftp://[<user>:<password>@]<host>/<url-path>*, como está descrito em RFC1738.

### Arquitetura da aplicação

A aplicação começa por fazer interpretar o URL inserido pelo utilizador, fazendo uso da função *parseInfo*, colocando esta informação numa instância da *struct Info*:

```
struct
Info {
    char hostname[100];
    char path[150];
    char* filename;
    char user[50];
    char password[50];
};
```

Caso o utilizador tenha optado por não inserir o *username* e a *password* na linha de comandos, como está previsto em RFC1738, estes são obtidos através do *input* do utilizador, fazendo uso da função *getUserInfo*.

De seguida, é criada uma instância da *struct hostent* (definida em *netdb.h*), na qual é guardado o IP do servidor, fazendo uso da função *getHostInfo*. Esta informação é usada para criar o *socket* que vai ser utilizado para comunicar com o servidor. Isto é feito através da função *connectTCP*. Depois desta ligação, é lida a resposta do servidor (através da função *readResponseCode*), de forma a certificarmos-nos que esta foi feita com sucesso. De forma a concluir esta etapa inicial de conexão, é enviada ao servidor a informação de *login*, que está armazenada na *struct Info*, através da função *sendLoginInfo*.

Estando estabelecida a ligação, é pedido ao servidor para entrar em modo passivo. Isto é feito através da função *getServerPort*. Esta função retorna a porta que vai ser utilizada pelo servidor para o envio do ficheiro. Esta a informação é usada para criar um novo canal de comunicação com o servidor, na forma de um outro *socket*, fazendo uso, novamente, da função *connectTCP*.

Finalmente, o ficheiro é recebido por parte do utilizador após a chamada à função *retrieveFile*. Após esta chamada, os canais de comunicação são fechados e a aplicação termina.

## Exemplo de um download bem-sucedido

Na secção *Anexos*, estão documentados a compilação da aplicação (Figura 1) e dois downloads bem-sucedidos. No primeiro caso (Figura 2), o utilizador colocou o *username* e a *password* no argumento passado na linha de comandos. No segundo caso (Figuras 3 e 4), o utilizador optou por não inserir estes dados na linha de comandos. A figura 3 mostra o programa à espera que o utilizador insira o *username*. A figura 4 mostra o programa após a sua execução. Na figura 5, é possível ver o ficheiro transferido do diretório onde foi executada a aplicação.

## Parte 2 – Configuração e análise de uma rede

### Experiência 1 – Configurar um IP de rede

O objetivo desta experiência é ligar dois computadores (no caso, **tux1** e **tux4**) na mesma rede, fazendo uso de um *switch*. Isto foi realizado através dos seguintes comandos:

**Tux1** – `ifconfig eth0 172.16.60.1/24`

**Tux4** – `ifconfig eth0 172.16.60.254/24`

Isto fez com que ambos os computadores usassem como rota de comunicação a sua porta eth0 (que estavam ligadas ao *switch*), utilizando os endereços IP respetivos.

- 1) O que são os pacotes ARP? Para o que são usados?

Os pacotes ARP são pacotes que obedecem às definições do protocolo ARP (*Address Resolution Protocol*). Este protocolo serve para fazer o mapeamento de endereços de rede, como os endereços IP, com endereços físicos, como os endereços MAC.

- 2) O que são os endereços MAC e IP dos pacotes ARP e porquê?

Quando existe uma tentativa de comunicação entre dois computadores em rede, o computador recetor envia um pacote ao emissor no sentido de obter o endereço físico (endereço MAC) deste. Este pacote enviado pelo recetor é um pacote ARP, que contém simultaneamente o endereço IP e o endereço MAC deste, assim como o endereço IP do emissor (ou seja, o endereço IP que o recetor deseja corresponder com um endereço MAC). O emissor responde com um pacote ARP muito semelhante à pergunta enviada pelo recetor, exceto que contém também o seu endereço MAC (Figuras 6 e 7).

- 3) Quais os pacotes gerados pelo comando *ping*?

O comando *ping* gera pacotes ARP de forma a obter o endereço MAC do recetor (como foi explicado na pergunta anterior), e pacotes ICMP (*Internet Control Message Protocol*). Estes são usados para transferir mensagens de controlo entre endereços IP.

- 4) Quais são os endereços MAC e IP dos pacotes *ping*?

Os pacotes IP contêm os endereços MAC e IP tanto do emissor como do recetor (Figuras 8 e 9).

5) Como determinar se a trama Ethernet recebida é ARP, IP, ICMP?

Todas as tramas Ethernet contêm um Ethernet Header que contém informação sobre o tipo de trama. Caso este valor seja 0x0800, estamos perante uma trama IP. Estas tramas, por sua vez, contêm um IP Header, que identifica o tipo de protocolo nelas utilizado. Caso este tome o valor 0x0806, a trama utiliza o protocolo ARP (Figura 10). Se, pelo contrário, o valor for 1, a trama utiliza o protocolo ICMP (Figura 11).

6) Como determinar o comprimento de uma trama recebida?

O tamanho de tramas IP está presente no IP Header (Figura 12).

7) O que é a interface *loopback* e porque é importante?

A interface *loopback* é uma interface de rede virtual que é utilizada pelo computador para testar o estado do sistema. Esta envia pacotes do tipo LOOP em intervalos de 10 segundos ao próprio computador (Figura 13), verificando se este ainda se encontra em estado de atividade

## Experiência 2 – Implementar duas LAN's virtuais num switch

O objetivo desta experiência é ligar três computadores em duas redes diferentes (no caso, **tux1** e **tux4** em **vlan0** e **tux2** em **vlan1**), fazendo uso de um só *switch*. Isto foi realizado através dos seguintes comandos:

**Tux1** – `ifconfig eth0 172.16.60.1/24`

**Tux2** – `ifconfig eth0 172.16.61.1/24`

**Tux4** – `ifconfig eth0 172.16.60.254/24`

1) Como configurar **vlan0**?

A configuração da **vlan0** é realizada através da configuração do *switch*. Esta é feita a partir do terminal *GTKTerm* de um dos **tux**, cuja porta *S0* está ligada à *Switch Console*.

Para criar a **vlan0**, é necessário executar os seguintes comandos:

- » `configure terminal`
- » `vlan y0`
- » `end`

Para adicionar cada **tux** (no caso, **tux1** e **tux4**) à **vlan0**, é necessário executar os seguintes comandos:

- » configure terminal
- » interface fastethernet 0/[porta do *switch* correspondente]
- » switchport mode access
- » switchport access vlan y0
- » end

- 2) Quantos domínios *broadcast* existem? Como se pode tirar essa conclusão a partir dos *logs*?

Existem dois domínios *broadcast*. Isto pode ser identificado através de uma tentativa de *ping* ao endereço *broadcast* de **vlany0** por parte de **tux1**. Este só obtém resposta de um computador: o que se encontra na mesma rede – **tux4** (Figura 14). O **tux2** não responde porque se encontra noutra rede, ou seja, outro domínio *broadcast*.

### Experiência 3 – Configurar um Router em Linux

O objetivo desta experiência é estabelecer ligação entre dois computadores em duas redes diferentes (no caso, **tux1** em **vlany0** e **tux2** em **vlany1**), utilizando um computador que está ligado a ambas as redes como *router* (no caso, **tux4**). Isto foi realizado através dos seguintes comandos (y = 6):

**Tux1** – ifconfig eth0 172.16.60.1/24  
           route add -net 172.16.61.0/24 gw 172.16.60.254

**Tux2** – ifconfig eth0 172.16.61.1/24  
           route add -net 172.16.60.0/24 gw 172.16.61.253

**Tux4** – ifconfig eth0 172.16.60.254/24  
           ifconfig eth1 172.16.61.253/24

- 1) Que rotas existem nos tuxes? Qual é o seu significado?

Rotas de **tux1**:

- **vlany0** (172.16.y0.0) – **gw** 172.16.y0.1 (próprio endereço)
- **vlany1** (172.16.y1.0) – **gw** 172.16.y0.254 (**tux4**)

Rotas de **tux2**:

- **vlany0** (172.16.y0.0) – **gw** 172.16.y1.253 (**tux4**)
- **vlany1** (172.16.y1.0) – **gw** 172.16.y1.1 (próprio endereço)

Rotas de **tux4**:

- **vlany0** (172.16.y0.0) – **gw** 172.16.y0.254 (próprio endereço)
- **vlany1** (172.16.y1.0) – **gw** 172.16.y0.253 (próprio endereço)



2) Que informação é que uma entrada da tabela de *forwarding* contém?

- **(Network) Destination** – destino da rota
- **Netmask** – usado em combinação com *Network Destination* para determinar a rede de destino
- **Gateway** – indica o endereço do próximo ponto de passagem da rota
- **Interface** – interface local através da qual a *Gateway* pode ser acedida
- **Metric** – custo associado à rota

3) Quais mensagens ARP e endereços MAC associados são observados e porquê?

As mensagens ARP e endereços MAC associados observados são os normais que correspondem ao comando *ping* e, por isso, em tudo semelhantes aos visualizados na Experiência 1, pergunta 2 (Figuras 15 e 16).

4) Quais pacotes ICMP é que são observados e porquê?

Os pacotes ICMP observados são os de *request* e *reply* que seriam de esperar durante a comunicação bem-sucedida entre dois computadores, mesmo não estando estes na mesma rede (Figura 17).

5) Quais são os endereços IP e MAC associados aos pacotes ICMP e porquê?

Os endereços MAC e IP tanto do emissor como do recetor (os mesmos que foram identificados na Experiência 1, pergunta 4).

## Experiência 4 – Configurar um Router Comercial e Implementar NAT

O objetivo desta experiência é ligar os computadores já integrados na rede à Internet, fazendo uso de um *router* comercial com NAT conectado tanto à rede geral do laboratório como à **vlan**1, com o endereço 172.16.y1.254. Isto foi realizado através dos seguintes comandos (y = 6):

**Tux1** – `ifconfig eth0 172.16.60.1/24`

`route add -net 172.16.61.0/24 gw 172.16.60.254`

`route add default gw 172.16.60.254`

**Tux2** – `ifconfig eth0 172.16.61.1/24`

`route add -net 172.16.60.0/24 gw 172.16.61.253`

`route add default gw 172.16.61.254`

**Tux4** – ifconfig eth0 172.16.60.254/24  
ifconfig eth1 172.16.61.253/24  
route add default gw 172.16.61.254

1) Como configurar um router estático num router comercial?

A configuração do *router* é realizada através do terminal *GTKTerm* de um dos **tux**, cuja porta *S0* está ligada à *Router Console*.

Para realizar a configuração do *router*, é necessário executar os seguintes comandos:

```
» conf t
» interface gigabitethernet 0/0 *
» ip address 172.16.y1.254 255.255.255.0
» no shutdown
» ip nat inside
» exit

» interface gigabitethernet 0/1*
» ip address 172.16.1.y9 255.255.255.0
» no shutdown
» ip nat outside
» exit
» end
```

2) Quais são as rotas seguidas pelos pacotes nas experiências anteriormente seguidas e porquê?

Caso a rota esteja definida individualmente, essa é usada pelos pacotes. Caso contrário, é usada a rota *default*. No caso de estudo, a rota *default* do **tux1** leva os pacotes para o **tux4** (que funciona como *router* para a **vlan**1) e para o **tux2** e o **tux4** leva os pacotes para o *router* comercial ligado à rede de laboratório.

3) Como configurar o NAT num router comercial?

Para configurar o NAT num *router* comercial é necessário configurar a interface interna no processo de NAT, realizado seguindo o guião fornecido para a dada experiência. Foram usados os seguintes comandos:

```
» ip nat pool ovrlld 172.16.1.y9 172.16.1.y9 prefix 24
» ip nat inside source list 1 pool ovrlld overload

» access-list 1 permit 172.16.y0.0 0.0.0.7
» access-list 1 permit 172.16.y1.0 0.0.0.7
```

- » ip route 0.0.0.0 0.0.0.0 172.16.1.254
- » ip route 172.16.y0.0 255.255.255.0 172.16.y1.253

#### 4) O que faz o NAT?

O NAT tem como objetivo poupar o espaço de endereçamento público, recorrendo a IP's privados que estão não registados se conectem à Internet ou a uma rede pública. O NAT opera num *router*, onde conecta duas redes e traduz os endereços privados para endereços legais antes que os pacotes sejam encaminhados para outra rede, oferecendo também funções de segurança. Resumidamente, permite que os computadores de uma rede interna tenham acesso ao exterior.

## Experiência 5 – DNS

O objetivo desta experiência é configurar o DNS (*Domain Name System*) nos computadores em rede (no caso, **tux1**, **tux2** e **tux4**) usando um servidor de DNS, *services.netlab.fe.up.pt*, que contém uma base de dados dos endereços IP públicos e os seus respetivos *hostnames*. É usado para fazer o *match* entre os *hostnames* e os seus respetivos endereços IP.

#### 1) Como configurar o serviço DNS num *host*?

A configuração do serviço DNS num *host* é realizada através da alteração do ficheiro **resolv.conf** que se localiza em **/etc/resolv.conf** no *host* **tux**. O ficheiro tem de conter a seguinte informação:

- » search netlab.fe.up.pt
- » nameserver 172.16.1.1

em que **netlab.fe.up.pt** é o *hostname* do servidor DNS e 172.16.1.1 o seu endereço IP.

#### 2) Que pacotes são trocados pelo DNS e que informações são transportadas?

Inicialmente temos um pacote enviado do *Host* para o *Server* que contém *hostname* desejado, pedindo o seu endereço IP. O servidor responde com um pacote que contém o endereço IP do *hostname*.

## Experiência 6 – Conexões TCP

O objetivo desta experiência era usar a aplicação de *download* desenvolvida na primeira parte do trabalho na rede de computadores configurada nas experiências anteriores, observado o seu comportamento em diferentes circunstâncias.

### 1) Quantas conexões TCP são abertas pela aplicação FTP?

A aplicação FTP abre duas conexões TCP. Uma para comunicar com o servidor e outra para receber o ficheiro, após ter pedido ao servidor para este entrar em modo passivo.

### 2) Em qual conexão é transportado a informação de controlo FTP?

A comunicação principal entre cliente e servidor, assim como o controlo do fluxo de informação transmitida, são transportados na primeira conexão TCP referida na pergunta anterior.

### 3) Quais são as fases de uma conexão TCP?

Uma conexão TCP tem três fases:

- *connection establishment (handshake)*
- *data transfer*
- *connection termination*

### 4) Como é que o mecanismo ARQ TCP funciona? Quais são os campos TCP relevantes? Qual informação relevante pode ser observada nos *logs*?

O mecanismo ARQ TCP faz controlo da troca de informação usando o método da janela deslizante. Isto implica que o recetor dos pacotes envie um *acknowledgment* de cada vez que recebe um pacote com sucesso, informando o emissor desse mesmo facto. Isto tem que ser enviado antes que o *timeout* para esse pacote seja atingido, provocando o reenvio de informação. Exemplos destes pacotes podem ser vistos na figura 18.

### 5) Como é que o mecanismo de controlo de congestão TCP funciona? Quais são os campos relevantes? Como é que o fluxo da conexão de dados evoluiu ao longo do tempo? Está de acordo com o mecanismo de controlo de congestão TCP?

O mecanismo de controlo de congestão TCP mantém uma estimativa dos pacotes que consegue enviar na forma de uma janela de congestão, certificando-se de que nunca sai deste limite.

Como é possível ver na figura 19, na transição de um primeiro *download* para um segundo, podemos ver um pico (cerca de 22,3 segundos), quando existe uma tentativa de estabilizar o número de envios, alcançando posteriormente um *rate* mais ou menos estável até ao fim de transmissão, ainda que com um *bitrate* inferior ao inicial.

- 6) De que forma é afetada a conexão de dados TCP pelo aparecimento de uma segunda conexão TCP? Como?

O comportamento da conexão de dados TCP pode ser observado na figura 20. Enquanto a aplicação de *download* estava a ser executada, a transferir um ficheiro (0 – 125 segundos), foi inicializada uma segunda instância da mesma aplicação (35 - 50 segundos), que começou a transferir um ficheiro mais pequeno. Como é possível ver na figura, no intervalo de tempo em que ambas as aplicações estavam em execução, existiu uma quebra na taxa de transmissão (para cerca de metade) do primeiro *download*, visto que a taxa de transmissão é distribuída igualmente por ambos os *downloads*.

## Conclusões

O grupo foi capaz de compreender e interiorizar os conceitos relacionados tanto com os protocolos FTP, assim como os que dizem respeito à configuração de uma rede.

O desenvolvimento da aplicação de download permitiu-nos ter uma maior compreensão sobre o funcionamento de protocolos de comunicação em geral, assim como por em perspectiva o protocolo de comunicação desenvolvido pelo grupo no âmbito do primeiro trabalho desta unidade curricular. De modo semelhante, a configuração da rede em ambiente de laboratório permitiu ao grupo ter um maior entendimento sobre conceitos de rede, assim como compreender como é que o que foi feito em ambiente controlado poderia ser escalado a uma implementação mais alargada.

Após o término no desenvolvimento deste projeto, podemos dizer que foram cumpridos os objetivos que nos foram propostos.

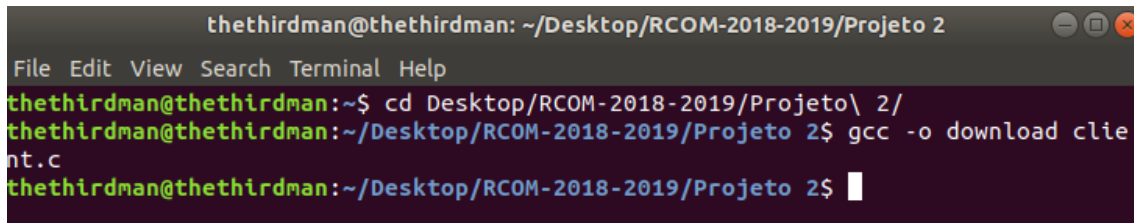
## Referências

<https://www.ietf.org/rfc/rfc959.txt> (consultado pela última vez em 2018/12/18)

<https://www.ietf.org/rfc/rfc1738.txt> (consultado pela última vez em 2018/12/18)

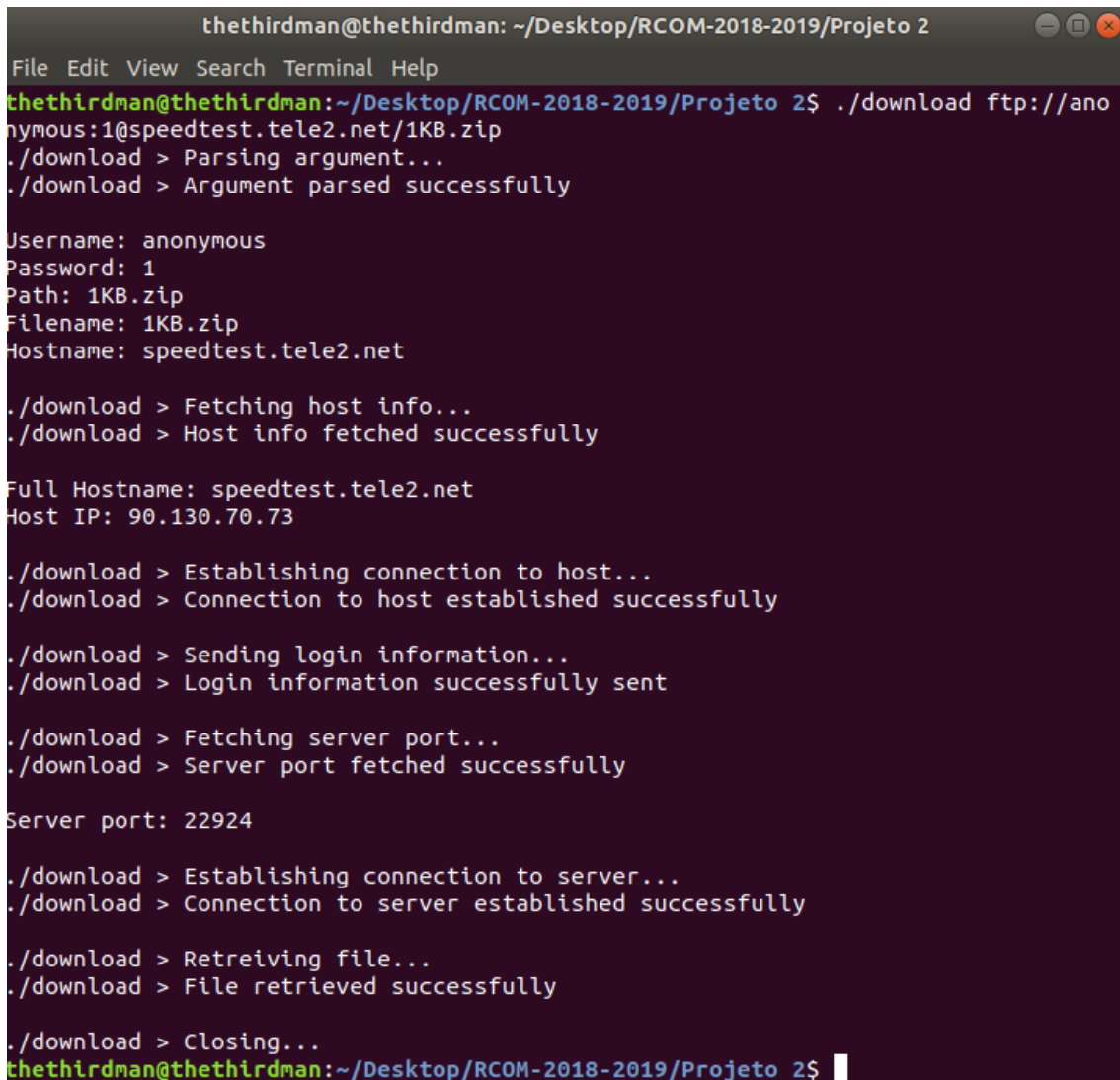
## Anexos

### Imagens



```
thethirdman@thethirdman: ~/Desktop/RCOM-2018-2019/Projeto 2
File Edit View Search Terminal Help
thethirdman@thethirdman:~$ cd Desktop/RCOM-2018-2019/Projeto\ 2/
thethirdman@thethirdman:~/Desktop/RCOM-2018-2019/Projeto 2$ gcc -o download client.c
thethirdman@thethirdman:~/Desktop/RCOM-2018-2019/Projeto 2$
```

Figura 1 - Compilação da aplicação



```
thethirdman@thethirdman: ~/Desktop/RCOM-2018-2019/Projeto 2
File Edit View Search Terminal Help
thethirdman@thethirdman:~/Desktop/RCOM-2018-2019/Projeto 2$ ./download ftp://anonymous:1@speedtest.tele2.net/1KB.zip
./download > Parsing argument...
./download > Argument parsed successfully

Username: anonymous
Password: 1
Path: 1KB.zip
Filename: 1KB.zip
Hostname: speedtest.tele2.net

./download > Fetching host info...
./download > Host info fetched successfully

Full Hostname: speedtest.tele2.net
Host IP: 90.130.70.73

./download > Establishing connection to host...
./download > Connection to host established successfully

./download > Sending login information...
./download > Login information successfully sent

./download > Fetching server port...
./download > Server port fetched successfully

Server port: 22924

./download > Establishing connection to server...
./download > Connection to server established successfully

./download > Retrieving file...
./download > File retrieved successfully

./download > Closing...
thethirdman@thethirdman:~/Desktop/RCOM-2018-2019/Projeto 2$
```

Figura 2 - Exemplo de execução com username

```
thethirdman@thethirdman:~/Desktop/RCOM-2018-2019/Projeto 2$ ./download ftp://speedtest.tele2.net/1KB.zip
./download > Parsing argument...
Username: 
```

Figura 3 - Exemplo de execução sem username (1)

```
./download > Parsing argument...
Username: anonymous
Password: 1
./download > Argument parsed successfully
Username: anonymous
Password: 1
Path: 1KB.zip
Filename: 1KB.zip
Hostname: speedtest.tele2.net
./download > Fetching host info...
./download > Host info fetched successfully
Full Hostname: speedtest.tele2.net
Host IP: 90.130.70.73
./download > Establishing connection to host...
./download > Connection to host established successfully
./download > Sending login information...
./download > Login information successfully sent
./download > Fetching server port...
./download > Server port fetched successfully
Server port: 23614
./download > Establishing connection to server...
./download > Connection to server established successfully
./download > Retrieving file...
./download > File retrieved successfully
./download > Closing...
thethirdman@thethirdman:~/Desktop/RCOM-2018-2019/Projeto 2$ 
```

Figura 4 - Exemplo de execução sem username (2)



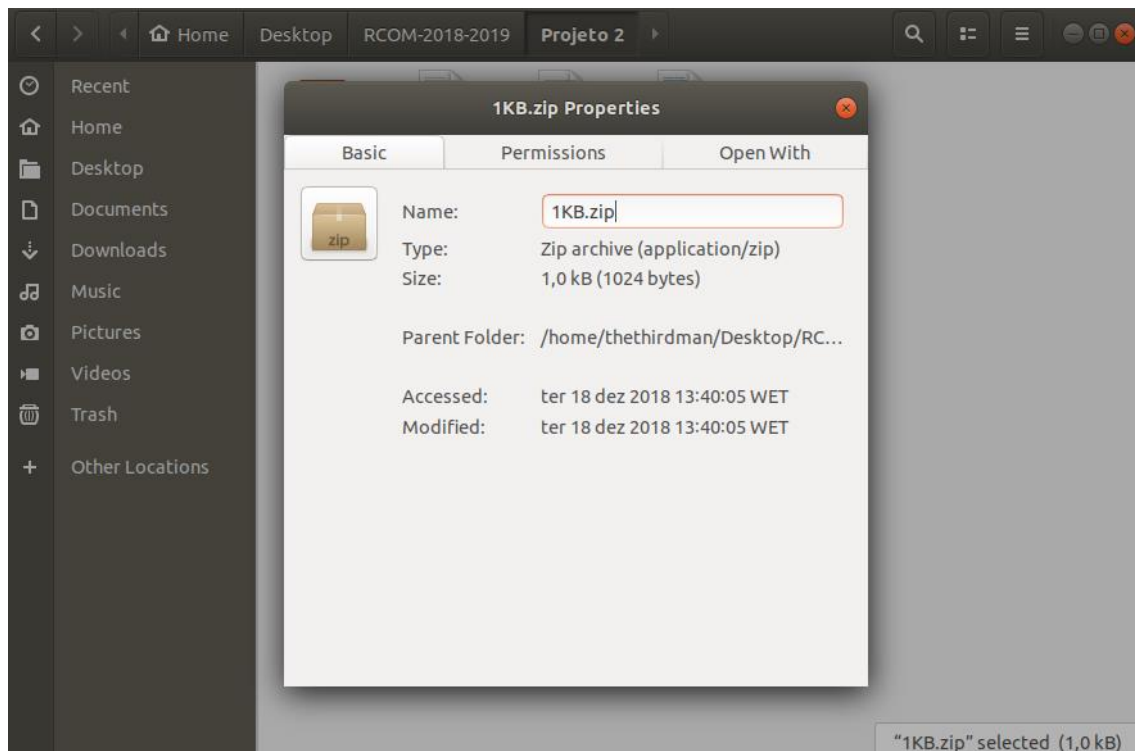


Figura 5 - Ficheiro recebido após execução da aplicação

No.	Time	Source	Destination	Protocol	Length	Info
19	21.63287400	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) request id=0x1af3, seq=3/768, ttl=64 (reply in 20)
20	21.63302900	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1af3, seq=3/768, ttl=64 (request in 19)
21	22.05376600	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:3a:f1:00 Cost = 0 Port = 0x8003
22	22.71338900	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Reply
23	24.05869100	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:3a:f1:00 Cost = 0 Port = 0x8003
24	24.63850900	Hewlett-_c5:61:bb	Hewlett-_61:2f:4e	ARP	60	Who has 172.16.60.1? Tell 172.16.60.254
25	24.63852700	Hewlett-_61:2f:4e	Hewlett-_c5:61:bb	ARP	42	172.16.60.1 is at 00:21:5a:61:2f:4e
26	26.06357900	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:3a:f1:00 Cost = 0 Port = 0x8003

▶ Frame 24: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0  
 ▶ Ethernet II, Src: Hewlett-\_c5:61:bb (00:21:5a:c5:61:bb), Dst: Hewlett-\_61:2f:4e (00:21:5a:61:2f:4e)  
 ▶ Address Resolution Protocol (request)  
   Hardware type: Ethernet (1)  
   Protocol type: IP (0x0800)  
   Hardware size: 6  
   Protocol size: 4  
   Opcode: request (1)  
   Sender MAC address: Hewlett-\_c5:61:bb (00:21:5a:c5:61:bb)  
   Sender IP address: 172.16.60.254 (172.16.60.254)  
   Target MAC address: 00:00:00\_00:00:00 (00:00:00:00:00:00)  
   Target IP address: 172.16.60.1 (172.16.60.1)

Figura 6 - Endereços IP e MAC de um pacote ARP (question)

No.	Time	Source	Destination	Protocol	Length	Info
19	21.63287400	172.16.60.1	172.16.60.254	ICMP	98	Echo (ping) request id=0x1af3, seq=3/768, ttl=64 (request in 20)
20	21.63302900	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1af3, seq=3/768, ttl=64 (request in 19)
21	22.05376600	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
22	22.71389000	Cisco_3a:f1:03	Cisco_3a:f1:03	LOOP	60	Reply
23	24.05869100	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
24	24.63850900	Hewlett-_c5:61:bb	Hewlett-_61:2f:4e	ARP	60	who has 172.16.60.1? Tell 172.16.60.254
25	24.63852700	Hewlett-_61:2f:4e	Hewlett-_c5:61:bb	ARP	42	172.16.60.1 is at 00:21:5a:61:2f:4e
26	26.06357900	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003

▶ Frame 25: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0  
 ▶ Ethernet II, Src: Hewlett-\_61:2f:4e (00:21:5a:61:2f:4e), Dst: Hewlett-\_c5:61:bb (00:21:5a:c5:61:bb)  
 ▶ Address Resolution Protocol (reply)  
   Hardware type: Ethernet (1)  
   Protocol type: IP (0x0800)  
   Hardware size: 6  
   Protocol size: 4  
   Opcode: reply (2)  
   Sender MAC address: Hewlett-\_61:2f:4e (00:21:5a:61:2f:4e)  
   Sender IP address: 172.16.60.1 (172.16.60.1)  
   Target MAC address: Hewlett-\_c5:61:bb (00:21:5a:c5:61:bb)  
   Target IP address: 172.16.60.254 (172.16.60.254)

Figura 7 - Endereços IP e MAC de um pacote ARP (answer)

No.	Time	Source	Destination	Protocol	Length	Info
4	4.009737000	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
5	4.355493000	172.16.60.1	172.16.60.254	ICMP	98	Echo (ping) request id=0x1bf7, seq=1/256, ttl=64 (reply in 6)
6	4.355653000	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1bf7, seq=1/256, ttl=64 (request in 5)
7	5.354870000	172.16.60.1	172.16.60.254	ICMP	98	Echo (ping) request id=0x1bf7, seq=2/512, ttl=64 (reply in 8)
8	5.354999000	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1bf7, seq=2/512, ttl=64 (request in 7)
9	6.019677000	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
10	6.354870000	172.16.60.1	172.16.60.254	ICMP	98	Echo (ping) request id=0x1bf7, seq=3/768, ttl=64 (no response found!)
11	6.355006000	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1bf7, seq=3/768, ttl=64 (request in 10)

▶ Frame 5: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0  
 ▶ Ethernet II, Src: Hewlett-\_61:2f:4e (00:21:5a:61:2f:4e), Dst: Hewlett-\_c5:61:bb (00:21:5a:c5:61:bb)  
 ▶ Internet Protocol Version 4, Src: 172.16.60.1 (172.16.60.1), Dst: 172.16.60.254 (172.16.60.254)  
 ▶ Internet Control Message Protocol

Figura 8 - Endereços IP e MAC de pacote ping (request)

No.	Time	Source	Destination	Protocol	Length	Info
4	4.009737000	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
5	4.355493000	172.16.60.1	172.16.60.254	ICMP	98	Echo (ping) request id=0x1bf7, seq=1/256, ttl=64 (reply in 6)
6	4.355653000	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1bf7, seq=1/256, ttl=64 (request in 5)
7	5.354870000	172.16.60.1	172.16.60.254	ICMP	98	Echo (ping) request id=0x1bf7, seq=2/512, ttl=64 (reply in 8)
8	5.354999000	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1bf7, seq=2/512, ttl=64 (request in 7)
9	6.019677000	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
10	6.354870000	172.16.60.1	172.16.60.254	ICMP	98	Echo (ping) request id=0x1bf7, seq=3/768, ttl=64 (no response found!)
11	6.355006000	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1bf7, seq=3/768, ttl=64 (request in 10)

▶ Frame 6: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0  
 ▶ Ethernet II, Src: Hewlett-\_c5:61:bb (00:21:5a:c5:61:bb), Dst: Hewlett-\_61:2f:4e (00:21:5a:61:2f:4e)  
 ▶ Internet Protocol Version 4, Src: 172.16.60.254 (172.16.60.254), Dst: 172.16.60.1 (172.16.60.1)  
 ▶ Internet Control Message Protocol

Figura 9 - Endereços IP e MAC de pacote ping (reply)

▶ Ethernet II, Src: Hewlett-_61:2f:4e (00:21:5a:61:2f:4e), Dst: Hewlett-_c5:61:bb (00:21:5a:c5:61:bb) ▶ Destination: Hewlett-_c5:61:bb (00:21:5a:c5:61:bb) ▶ Source: Hewlett-_61:2f:4e (00:21:5a:61:2f:4e) Type: ARP (0x0806)
▶ Address Resolution Protocol (reply) Hardware type: Ethernet (1) Protocol type: IP (0x0800) Hardware size: 6 Protocol size: 4 Opcode: reply (2) Sender MAC address: Hewlett-_61:2f:4e (00:21:5a:61:2f:4e) Sender IP address: 172.16.60.1 (172.16.60.1) Target MAC address: Hewlett-_c5:61:bb (00:21:5a:c5:61:bb) Target IP address: 172.16.60.254 (172.16.60.254)

Figura 10 - Trama com protocolo ARP

▼ Ethernet II, Src: Hewlett-_61:2f:4e (00:21:5a:61:2f:4e), Dst: Hewlett-_c5:61:bb (00:21:5a:c5:61:bb)
<ul style="list-style-type: none"> <li>► Destination: Hewlett-_c5:61:bb (00:21:5a:c5:61:bb)</li> <li>► Source: Hewlett-_61:2f:4e (00:21:5a:61:2f:4e)</li> <li>Type: IP (0x0800)</li> </ul>
▼ Internet Protocol Version 4, Src: 172.16.60.1 (172.16.60.1), Dst: 172.16.60.254 (172.16.60.254)
Version: 4 Header Length: 20 bytes <ul style="list-style-type: none"> <li>► Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))</li> </ul> Total Length: 84 Identification: 0x64eb (25835) <ul style="list-style-type: none"> <li>► Flags: 0x02 (Don't Fragment)</li> </ul> Fragment offset: 0 Time to live: 64 Protocol: ICMP (1)

Figura 11 - Trama com protocolo ICMP

▼ Frame 8: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
Interface id: 0 (eth0) Encapsulation type: Ethernet (1) Arrival Time: Dec 19, 2018 13:26:29.490057000 WET [Time shift for this packet: 0.000000000 seconds] Epoch Time: 1545225989.490057000 seconds [Time delta from previous captured frame: 0.000129000 seconds] [Time delta from previous displayed frame: 0.000129000 seconds] [Time since reference or first frame: 5.354999000 seconds] Frame Number: 8 Frame Length: 98 bytes (784 bits) Capture Length: 98 bytes (784 bits) [Frame is marked: False] [Frame is ignored: False]

Figura 12 - Comprimento de uma trama

No.	Time	Source	Destination	Protocol	Length	Info
14	9.33930000	Hewlett-_61:2f:4e	Hewlett-_c5:61:bb	ARP	42	172.16.60.1 is at 00:21:5a:61:2f:4e
15	10.02445600	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
16	12.03430500	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
17	12.16377600	Cisco_3a:f1:03	Cisco_3a:f1:03	LOOP	60	Reply
18	14.03420400	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
19	16.03932100	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
20	18.04901100	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
21	19.04286000	Cisco_3a:f1:03	CDP/VTP/DTP/PagP/UDLD	CDP	435	Device ID: tux-sw6 Port ID: FastEthernet0/1
22	20.04890000	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
► Frame 17: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0 ► Ethernet II, Src: Cisco_3a:f1:03 (fc:fb:fb:3a:f1:03), Dst: Cisco_3a:f1:03 (fc:fb:fb:3a:f1:03) ► Configuration Test Protocol (loopback) skipCount: 0 Relevant function: Function: Reply (1) Receipt number: 0 ► Data (40 bytes)						

Figura 13 - Trama LOOP

No.	Time	Source	Destination	Protocol	Length	Info
2	2.004756000	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
3	4.009659000	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
4	6.014535000	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
5	6.794994000	Cisco_3a:f1:03	Cisco_3a:f1:03	LOOP	60	Reply
6	7.187984000	172.16.60.254	172.16.60.255	ICMP	98	Echo (ping) request id=0x1737, seq=1/256, ttl=64 (no response found!)
7	8.019411000	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
8	8.187880000	172.16.60.254	172.16.60.255	ICMP	98	Echo (ping) request id=0x1737, seq=2/512, ttl=64 (no response found!)
9	9.187907000	172.16.60.254	172.16.60.255	ICMP	98	Echo (ping) request id=0x1737, seq=3/768, ttl=64 (no response found!)
10	10.024334000	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
▶ Frame 6: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0 ▶ Ethernet II, Src: Hewlett-_c5:61:bb (00:21:5a:c5:61:bb), Dst: Broadcast (ff:ff:ff:ff:ff:ff) ▶ Internet Protocol Version 4, Src: 172.16.60.254 (172.16.60.254), Dst: 172.16.60.255 (172.16.60.255) ▶ Internet Control Message Protocol Type: 8 (Echo (ping) request) Code: 0 Checksum: 0xe5eb [correct] Identifier (BE): 5943 (0x1737) Identifier (LE): 14103 (0x3717) Sequence number (BE): 1 (0x0001) Sequence number (LE): 256 (0x0100) ▶ [No response seen]						

Figura 14 - Broadcast Ping

No.	Time	Source	Destination	Protocol	Length	Info
15	17.411795000	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1f06, seq=2/512, ttl=63 (request in 14)
16	18.043987000	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
17	20.048850000	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
18	20.216819000	Cisco_3a:f1:03	Cisco_3a:f1:03	LOOP	60	Reply
19	21.426907000	Hewlett-_c5:61:bb	Hewlett-_61:2f:4e	ARP	60	Who has 172.16.60.1? Tell 172.16.60.254
20	21.426924000	Hewlett-_61:2f:4e	Hewlett-_c5:61:bb	ARP	42	172.16.60.1 is at 00:21:5a:61:2f:4e
21	22.053860000	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
22	24.058647000	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
▶ Frame 19: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0 ▶ Ethernet II, Src: Hewlett-_c5:61:bb (00:21:5a:c5:61:bb), Dst: Hewlett-_61:2f:4e (00:21:5a:61:2f:4e) ▶ Address Resolution Protocol (request) Hardware type: Ethernet (1) Protocol type: IP (0x0800) Hardware size: 6 Protocol size: 4 Opcode: request (1) Sender MAC address: Hewlett-_c5:61:bb (00:21:5a:c5:61:bb) Sender IP address: 172.16.60.254 (172.16.60.254) Target MAC address: 00:00:00:00:00:00 (00:00:00:00:00:00) Target IP address: 172.16.60.1 (172.16.60.1)						

Figura 15 - Endereços IP e MAC de um pacote ARP – redes diferentes (question)

No.	Time	Source	Destination	Protocol	Length	Info
15	17.411795000	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1f06, seq=2/512, ttl=63 (request in 14)
16	18.043987000	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
17	20.048850000	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
18	20.216819000	Cisco_3a:f1:03	Cisco_3a:f1:03	LOOP	60	Reply
19	21.426907000	Hewlett-_c5:61:bb	Hewlett-_61:2f:4e	ARP	60	Who has 172.16.60.1? Tell 172.16.60.254
20	21.426924000	Hewlett-_61:2f:4e	Hewlett-_c5:61:bb	ARP	42	172.16.60.1 is at 00:21:5a:61:2f:4e
21	22.053860000	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
22	24.058647000	Cisco_3a:f1:03	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
▶ Frame 20: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0 ▶ Ethernet II, Src: Hewlett-_61:2f:4e (00:21:5a:61:2f:4e), Dst: Hewlett-_c5:61:bb (00:21:5a:c5:61:bb) ▶ Address Resolution Protocol (reply) Hardware type: Ethernet (1) Protocol type: IP (0x0800) Hardware size: 6 Protocol size: 4 Opcode: reply (2) Sender MAC address: Hewlett-_61:2f:4e (00:21:5a:61:2f:4e) Sender IP address: 172.16.60.1 (172.16.60.1) Target MAC address: Hewlett-_c5:61:bb (00:21:5a:c5:61:bb) Target IP address: 172.16.60.254 (172.16.60.254)						

Figura 16 - Endereços IP e MAC de um pacote ARP – redes diferentes (answer)

14	17.411527000	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) request id=0x1f06, seq=2/512, ttl=64 (reply in 15)
15	17.411795000	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1f06, seq=2/512, ttl=63 (request in 14)

Figura 17 – Pacotes ICMP (request e reply)

33	52.13210300	Cisco_3a:fl:03	Spanning-tree-(for-br STP	60	Conf. Root = 32768/60/fc:fb:3a:fl:00 Cost = 0 Port = 0x8003
34	53.37871400	172.16.60.1	172.16.1.1	DNS	79 Standard query 0x5d6a A speedtest.tele2.net
35	53.38003700	172.16.1.1	172.16.60.1	DNS	233 Standard query response 0x5d6a A 90.130.70.73
36	53.38015100	172.16.60.1	90.130.70.73	TCP	74 33530->21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=4290945 TSecr=0 WS=128
37	53.42886100	90.130.70.73	172.16.60.1	TCP	74 21->33530 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PERM=1 TSval=1738224264
38	53.42886900	172.16.60.1	90.130.70.73	TCP	66 33530->21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=4290957 TSecr=1738224264
39	53.47935200	90.130.70.73	172.16.60.1	FTP	86 Response: 220 (vsFTPd 2.3.5)
40	53.47937000	172.16.60.1	90.130.70.73	TCP	66 33530->21 [ACK] Seq=1 Ack=21 Win=29312 Len=0 TSval=4290970 TSecr=1738224277
41	53.47941800	172.16.60.1	90.130.70.73	FTP	71 Request: user
42	53.52721600	90.130.70.73	172.16.60.1	TCP	66 21->33530 [ACK] Seq=21 Ack=6 Win=14848 Len=0 TSval=1738224289 TSecr=4290970
43	53.52723300	172.16.60.1	90.130.70.73	FTP	76 Request: anonymous
44	53.57524400	90.130.70.73	172.16.60.1	TCP	66 21->33530 [ACK] Seq=21 Ack=16 Win=14848 Len=0 TSval=1738224301 TSecr=4290982
45	53.57538000	90.130.70.73	172.16.60.1	FTP	100 Response: 331 Please specify the password.
46	53.57541800	172.16.60.1	90.130.70.73	FTP	71 Request: pass
47	53.66223300	90.130.70.73	172.16.60.1	TCP	66 21->33530 [ACK] Seq=55 Ack=21 Win=14848 Len=0 TSval=1738224323 TSecr=4290994
48	53.66224100	172.16.60.1	90.130.70.73	FTP	68 Request: l
49	53.71005300	90.130.70.73	172.16.60.1	TCP	66 21->33530 [ACK] Seq=55 Ack=23 Win=14848 Len=0 TSval=1738224334 TSecr=4291016
50	53.81535500	90.130.70.73	172.16.60.1	FTP	89 Response: 230 Login successful.
51	53.81543000	172.16.60.1	90.130.70.73	FTP	70 Request: pasv
52	53.86321000	90.130.70.73	172.16.60.1	TCP	66 21->33530 [ACK] Seq=78 Ack=27 Win=14848 Len=0 TSval=1738224373 TSecr=4291054
53	53.86322800	172.16.60.1	90.130.70.73	FTP	67 Request:
54	53.91098100	90.130.70.73	172.16.60.1	TCP	66 21->33530 [ACK] Seq=78 Ack=28 Win=14848 Len=0 TSval=1738224385 TSecr=4291066
55	53.91151200	90.130.70.73	172.16.60.1	FTP	116 Response: 227 Entering Passive Mode (90,130,70,73,104,42).
56	53.91159400	172.16.60.1	90.130.70.73	TCP	74 59735->26666 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=4291078 TSecr=0 WS=
57	53.94946200	172.16.60.1	90.130.70.73	TCP	66 33530->21 [ACK] Seq=28 Ack=128 Win=29312 Len=0 TSval=4291088 TSecr=1738224385
58	53.95715400	90.130.70.73	172.16.60.1	TCP	74 26666->59735 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PERM=1 TSval=173822439
59	53.95717300	172.16.60.1	90.130.70.73	TCP	66 59735->26666 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=4291089 TSecr=1738224397
60	53.95719600	172.16.60.1	90.130.70.73	FTP	71 Request: retr
61	54.04227200	90.130.70.73	172.16.60.1	TCP	66 21->33530 [ACK] Seq=128 Ack=33 Win=14848 Len=0 TSval=1738224418 TSecr=4291089
62	54.04228600	172.16.60.1	90.130.70.73	FTP	76 Request: 100MB.zip
63	54.08881800	90.130.70.73	172.16.60.1	FTP-DATA	1514 FTP Data: 1448 bytes
64	54.08882700	172.16.60.1	90.130.70.73	TCP	66 59735->26666 [ACK] Seq=1 Ack=1449 Win=32128 Len=0 TSval=4291122 TSecr=1738224430
65	54.08893800	90.130.70.73	172.16.60.1	FTP-DATA	1514 FTP Data: 1448 bytes
66	54.08894500	172.16.60.1	90.130.70.73	TCP	66 59735->26666 [ACK] Seq=1 Ack=2897 Win=35072 Len=0 TSval=4291122 TSecr=1738224430

Figura 18 - Comunicação ARQ TCP

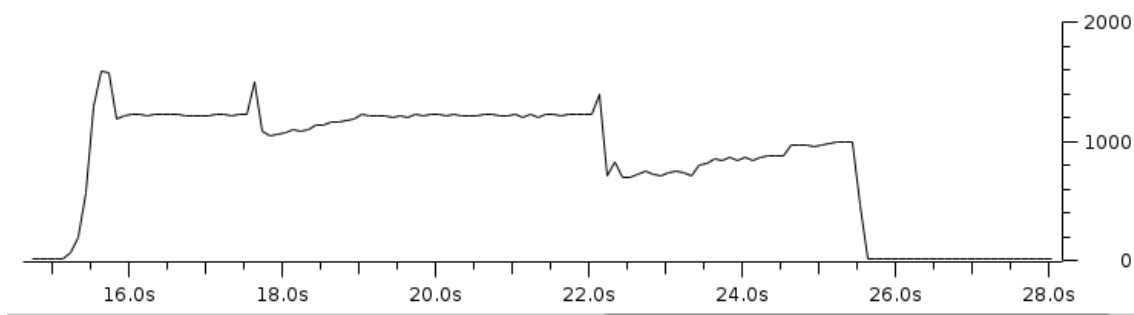


Figura 19 - Controlo de Congestão TCP

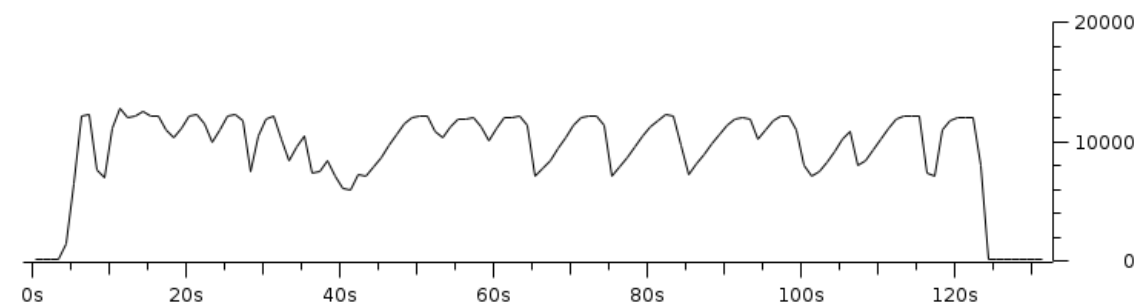


Figura 20 - Duas conexões TCP em simultâneo

## Código

info.h

```
/**
 * Compilation: gcc -o download client.c
 * Test: ./download ftp://anonymous:1@speedtest.tele2.net/1KB.zip
 */

#define BEGIN 0
```

```

#define END 99

#define USER 1
#define PASSWORD 2
#define HOST 3
#define PATH 4

#define CLEAR_LINE 1
#define MULTIPLE_LINE 2

#define PORT 21

#define CMD_USER "user "
#define CMD_PASS "pass "
#define CMD_PASSIVE "pasv"
#define CMD_RETRIEVE "retr "

/** @name Info Struct*/
/**@{
 *
 * Struct to store user information
 */
struct Info {
    char hostname[100];
    char path[150];
    char* filename;
    char user[50];
    char password[50];
};
/** @} end of Info Struct */

```

client.c

```

#include
<stdio.h>

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

```

```

#include <ctype.h>
#include "info.h"

/**
 * @brief Parses user information from argument input of user
 *
 * Implements a state machine to parse user information
 *
 * States:
 * - BEGIN - reads start section ftp://
 * - USER - reads user username (optional field)
 * - PASSWORD - reads user password (optional field)
 * - HOST - reads host name
 * - PATH - reads file saving path
 *
 * @param cmd string containing user input argument
 * @param info struct where user info will be saved on
 * @return 0 on success, non-zero otherwise
 */
int parseInfo(char* cmd, struct Info* info)
{
    int state = BEGIN;
    int i = 0;
    int aux = 0;
    char c;

    while(state != END)
    {
        c = cmd[i++];
        switch(state)
        {
            case BEGIN:
            {
                if(c == '\\0')
                    return -1;

                if(c != 'f')
                    return -2;
                c = cmd[i++];
                if(c != 't')
                    return -2;
                c = cmd[i++];
                if(c != 'p')
                    return -2;
                c = cmd[i++];
                if(c != ':')

```

```

        return -2;
    c = cmd[i++];
    if(c != '/')
        return -2;
    c = cmd[i++];
    if(c != '/')
        return -2;

    if(strrchr(cmd, '@') != NULL)
        state = USER;
    else
        state = HOST;
    break;
}
case USER:
{
    if(c == '\\0')
        return -1;

    if(c == ':')
    {
        (*info).user[aux] = 0;
        aux = 0;
        state = PASSWORD;
    }
    else
    {
        (*info).user[aux++] = c;
    }
    break;
}
case PASSWORD:
{
    if(c == '\\0')
        return -1;

    if(c == '@')
    {
        (*info).password[aux] = 0;
        aux = 0;
        state = HOST;
    }
    else
    {
        (*info).password[aux++] = c;
    }
}

```



```

        break;
    }
    case HOST:
    {
        if(c == '\\0')
            return -1;

        if(c == '/')
        {
            (*info).hostname[aux] = 0;
            aux = 0;
            state = PATH;
        }
        else
        {
            (*info).hostname[aux++] = c;
        }
        break;
    }
    case PATH:
    {
        if(c == '\\0')
        {
            (*info).path[aux] = 0;
            aux = 0;
            state = END;
        }
        else
        {
            (*info).path[aux++] = c;
        }
        break;
    }
    case END:
    {
        return 0;
        break;
    }
    default: return -1;
}

}

return 0;
}

```

/\*\*

\* @brief Gets username and password from user

```

*
* This function is only called if user did not fill the optional
* fields username and password in the command line argument
*
* @param info struct where user info will be saved on
*/
void getUserInfo(struct Info* info)
{
    char* buf = malloc(50 * sizeof(char));
    size_t size = 50;
    fflush(stdin);
    printf("\nUsername: ");
    getline(&buf, &size, stdin);
    strncpy(info->user, buf, strcspn(buf, "\n"));

    printf("Password: ");
    getline(&buf, &size, stdin);
    strncpy(info->password, buf, strcspn(buf, "\n"));
    printf("\n");
}

/**
 * @brief Gets filename from the file saving path
 *
 * Isolates the filename from the path so that it can be sent to the
server
 *
 * @param info struct where user info will be saved on
 */
void parseFilename(struct Info* info)
{
    char* filename = strrchr(info->path, '/');

    if(filename == NULL)
    {
        info->filename = info->path;
    }
    else
    {
        info->filename = (filename + 1);
    }
}

/**
 * @brief Gets host information using host name given by user
 *

```

```

    * Gets host information (particullary the IP address), so that a
connection
    * can be established with the server
    *
    * @param hostname string containing host name
    * @param h struct where host info will be saved on
    * @return 0 on success, non-zero otherwise
    */
int getHostInfo(char* hostname, struct hostent** h)
{
    if ((*h=gethostbyname(hostname)) == NULL)
        return -1;

    return 0;
}

/**
    * @brief Establishes a connection with the server
    *
    * Establishes a socket connecting to the server using its IP address
    *
    * @param addr string containing server ip address
    * @param port port used by server to send and receive information
    * @return socket file descriptor on success, negative otherwise
    */
int connectTCP(char* addr, int port)
{
    int sockfd;
    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(addr);
    server_addr.sin_port = htons(port);

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        return -1;

    if(connect(sockfd, (struct sockaddr *)&server_addr,
sizeof(server_addr)) < 0)
        return -2;

    return sockfd;
}

/**

```

```

* @brief Reads response code from server
*
* Implements a state machine to reads response code followed by a
user command
* Must be called after each command is sent
*
* States:
* - BEGIN - reads response code
* - CLEAR_LINE - clears line after code is read
* - MULTIPLE_LINES - reads additional lines if necessary
*
* @param sockfd socket file descriptor
* @param responseCode string where the response code will be saved
* @return 0 on success, non-zero otherwise
*/
int readResponseCode(int sockfd, char *responseCode)
{
    int state = BEGIN;
    int index = 0;
    char c;

    while (state != END)
    {
        read(sockfd, &c, 1);
        switch(state)
        {
            case BEGIN:
            {
                if(c == ' ')
                {
                    if (index != 3)
                    {
                        return -1;
                    }
                    index = 0;
                    state = CLEAR_LINE;
                }
                else if(c == '-')
                {
                    state = MULTIPLE_LINE;
                    index = 0;
                }
                else if(isdigit(c))
                {
                    responseCode[index++] = c;
                }
            }
        }
    }
}

```

```

        break;
    }
    case CLEAR_LINE:
    {
        if (c == '\n')
        {
            state = END;
        }
        break;
    }
    case MULTIPLE_LINE:
    {
        if(c == responseCode[index])
        {
            index++;
        }
        else if(index == 3 && c == ' ')
        {
            state = CLEAR_LINE;
        }
        else if(index ==3 && c == '-')
        {
            index = 0;
        }
        break;
    }
    case END:
    {
        return 0;
        break;
    }
    default: return -1;
}

}

return 0;
}

/**
 * @brief Writes command to server
 *
 * @param fd socket file descriptor
 * @param cmd command to be sent
 * @param info command arguments
 */
void writeCmd(int fd, char* cmd, char* info)
{

```

```

        write(fd, cmd, strlen(cmd));
        write(fd, info, strlen(info));
        write(fd, "\n", 1);
    }

/**
 * @brief Sends command to user and awaits response
 *
 * Sends command to user and acts accordingly to server response
 *
 * Answer values:
 * - 1 - sending another response (except for retr command)
 * - 2 - success
 * - 3 - server needs additional information to proceed
 * - 4 - resend command
 * - 5 - error
 *
 * @param cmd string containing user input argument
 * @param info struct where user info will be saved on
 * @return Success values:
 * @return 0 - Command accepted
 * @return 1 - Command accepted - needs additional information
 * @return 2 - Command accepted - file ready for retrieval
 * @return Negative codes represent error
 */
int writeCommand(int fd, char* cmd, char* info)
{
    char code[3];
    writeCmd(fd, cmd, info);
    readResponseCode(fd, code);
    int answer = code[0] - '0';

    while(1)
    {
        switch(answer)
        {
            case 1:
            {
                if(strcmp(cmd, "retr ")==0)
                    return 2;
                readResponseCode(fd, code);
                break;
            }
            case 2:
                return 0;
            case 3:

```

```

        return 1;
    case 4:
    {
        writeCmd(fd, cmd, info);
        break;
    }
    case 5:
        return -1;
    }
}

/**
 * @brief Sends login info to server
 *
 * @param info struct containing user info
 * @param sockfd socket file descriptor
 * @return 0 on success, non-zero otherwise
 */
int sendLoginInfo(struct Info* info, int sockfd)
{
    if(writeCommand(sockfd, CMD_USER, info->user) != 1)
        return -1;

    if(writeCommand(sockfd, CMD_PASS, info->password) != 0)
        return -1;

    return 0;
}

/**
 * @brief Gets server port
 *
 * Gets port in which server will communicate using passive mode
 * Reads both bytes of the port and merges them
 *
 * @param sockfd socket file descriptor
 * @return server port on success, negative value otherwise
 */
int getServerPort(int sockfd)
{
    writeCmd(sockfd, CMD_PASSIVE, "");

    int state = 0;
    int index = 0;
    char msg1[4];

```

```

memset(msg1, 0, 4);
char msg2[4];
memset(msg2, 0, 4);

char c;

while (state != 7)
{
    read(sockfd, &c, 1);
    switch (state)
    {
        case 0:
        {
            if (c == ' ')
            {
                if (index != 3)
                    return -1;
                index = 0;
                state = 1;
            }
            else
            {
                index++;
            }
            break;
        }
        case 5:
        {
            if (c == ',')
            {
                index = 0;
                state++;
            }
            else
            {
                msg1[index] = c;
                index++;
            }
            break;
        }
        case 6:
        {
            if (c == ')')
            {
                state++;
            }
        }
    }
}

```



```

        else
        {
            msg2[index] = c;
            index++;
        }
        break;
    }

    default:
    {
        if (c == ',')
        {
            state++;
        }
        break;
    }
}

int b1 = atoi(msg1);
int b2 = atoi(msg2);
return (b1 * 256 + b2);
}

/**
 * @brief Gets file from server
 *
 * Gets file information from server and saves it in the specified
path
 *
 * @param info struct containing user info
 * @param sockfd socket file descriptor
 * @param serverfd passive server file descriptor
 * @return 0 on success, non-zero otherwise
 */
int retrieveFile(struct Info* info, int sockfd, int serverfd)
{
    if(writeCommand(sockfd, CMD_RETRIEVE, info->path) != 2)
        return -1;

    FILE *file = fopen(info->filename, "wb+");

    char buffer[1024];
    int bytes;

    bytes = read(serverfd, buffer, 1024);

```

```

        while (bytes > 0) {
            bytes = fwrite(buffer, bytes, 1, file);
            bytes = read(serverfd, buffer, 1024);
        }

        fclose(file);

        return 0;
    }

    int main(int argc, char** argv)
    {
        if(argc != 2)
        {
            printf("Usage: %s ftp://[<user>:<password>@]<host>/<url-  
path>\n", argv[0]);
            return -1;
        }

        printf("%s > Parsing argument...\n", argv[0]);

        struct Info info;

        if(parseInfo(argv[1], &info) != 0)
        {
            printf("Error parsing client info: %s\n", argv[1]);
            return -2;
        }

        parseFilename(&info);

        if(info.user[0] == 0)
            getUserInfo(&info);

        printf("%s > Argument parsed successfully\n\n", argv[0]);
        printf("Username: %s\n", info.user);
        printf("Password: %s\n", info.password);
        printf("Path: %s\n", info.path);
        printf("Filename: %s\n", info.filename);
        printf("Hostname: %s\n\n", info.hostname);

        printf("%s > Fetching host info...\n", argv[0]);

        struct hostent *host;

        if(getHostInfo(info.hostname, &host) != 0)

```

```

{
    printf("Error getting host IP address: %s\n", info.hostname);
    return -3;
}

char* addr_str = inet_ntoa*((struct in_addr *)host->h_addr));

printf("%s > Host info fetched successfully\n\n", argv[0]);
printf("Full Hostname: %s\n", host->h_name);
printf("Host IP: %s\n\n", addr_str);

printf("%s > Establishing connection to host...\n", argv[0]);

int sockfd = connectTCP(addr_str, PORT);

if(sockfd < 0)
{
    printf("Error connecting to IP %s using port %d\n", addr_str,
PORT);
    return -4;
}

char code[3];

if(readResponseCode(sockfd, code) != 0)
{
    printf("Error reading server response code\n");
    return -5;
}

if(code[0] != '2')
{
    printf("Error: server responded with code %s\n", code);
    return -6;
}

printf("%s > Connection to host established successfully\n\n",
argv[0]);

printf("%s > Sending login information...\n", argv[0]);

if(sendLoginInfo(&info, sockfd) != 0)
{
    printf("Error sending login information\n");
    return -7;
}

```

```

printf("%s > Login information successfully sent\n\n", argv[0]);

printf("%s > Fetching server port...\n", argv[0]);

int server_port = getServerPort(sockfd);

if(server_port < 0)
{
    printf("Error fetching server port\n");
    return -8;
}

printf("%s > Server port fetched successfully\n\n", argv[0]);
printf("Server port: %d\n\n", server_port);

printf("%s > Establishing connection to server...\n", argv[0]);

int serverfd = connectTCP(addr_str, server_port);

if(serverfd < 0)
{
    printf("Error connecting to IP %s using port %d\n", addr_str,
PORT);
    return -4;
}

printf("%s > Connection to server established successfully\n\n",
argv[0]);

printf("%s > Retrieving file...\n", argv[0]);

if(retrieveFile(&info, sockfd, serverfd) < 0)
{
    printf("Error retrieving file: %s\n", info.filename);
    return -9;
}

printf("%s > File retrieved successfully\n\n", argv[0]);

printf("%s > Closing...\n", argv[0]);

close(sockfd);
close(serverfd);

return 0;

```

