2º Projeto de RCOM

Rede de Computadores

Engenharia Informática e Computação

Regente: Manuel Alberto Pereira Ricardo

Turma 6

- **Eduardo Luís Pinheiro da Silva – up201603135 - up201603135@fe.up.pt**
- **João Pedro Viveiros Franco – up201605775 - up201605775@fe.up.pt**
- **Tomás Nuno Fernandes Novo – up201604503 - up201604503@fe.up.pt**

Sumário

O projeto "**Rede de Computadores**" consiste não só configuração de uma rede de computadores, usufruindo do ***Cisco Router*** e ***Switch***, como também no estudo do seu funcionamento. Para além do manipulamento da rede, foi também desenvolvida uma aplicação de *download* com recurso ao ***File Transfer Protocol (FTP)***. Com o fim de prestar apoio ao projeto, foi elaborado este relatório.

O relatório apresenta conceitos fundamentais para a compreensão do projeto, complementando-o e deixando claro o término com sucesso do mesmo, visto que os objetivos ambicionados foram cumpridos.

1. Introdução

No âmbito desta unidade curricular, foi-nos proposta como 2ª parte do trabalho prático a realização de um projeto cuja meta residia em configurar com sucesso uma rede de computadores, bem como a criação de uma aplicação na qual é realizada o *download* de um ficheiro.

Neste projeto objetivámos então a configuração com êxito de uma rede de computadores, a qual recorre a duas **VLAN's** residentes num ***Switch,*** com o objetivo de conceder a execução de uma aplicação.

Relativamente à aplicação desenvolvida, esta permite o download de um ficheiro consoante o ***File Transfer Protocol (FTP).*** Para tal, recorremos a ligações de acordo com o ***Transmission Control Protocol (TCP),*** nomeadamente a ***sockets***.

Como referido no sumário, o projeto consiste na implementação destas duas componentes.

A estrutura do relatório foi elaborada com o objetivo de auxiliar na interpretação do projeto concebido, sendo que esta se encontra dividida em:

1. **Introdução –** onde são referidos os objetivos do trabalho e do relatório bem como os tipos de informação que poderão ser encontrados na restante estrutura.
2. **Aplicação –** explicitação da implementação da aplicação de download.
3. **Rede de Computadores –** síntese das ilações retiradas da realização de cada experiência.
4. **Conclusões –** resumo de toda a informação e conclusões retiradas da realização do projeto.
5. **Anexos –** Estão aqui incluídas várias figuras representativas dos logs capturados durante as várias experiências, assim como os comandos de configuração usados e o código da aplicação desenvolvida.

2. Aplicação

A primeira parte do trabalho consistiu no desenvolvimento de um cliente ***File Transfer Protocol*** (FTP) usando a linguagem C e o RFC 959 como referência.

O programa é corrido da seguinte maneira: **./download ftp://[<user>:<password>@]<host>/<url-path>** sendo "<user>:<password>@" um argumento opcional. O programa começa por separar o URL em ***username*****, password,** ***hostname*** e ***filename***. De seguida, aloca espaço para dois objetos da estrutura *connection,* que contém um endereço IP, a porta e um apontador para *FILE*. Na primeira conexão é aberta a ligação na porta 21 ao servidor passado nos argumentos. A segunda conexão é reservada para a ligação ao servidor em modo passivo.

De seguida, é feito o *login* com *username* e *password* passado nos argumentos, ou, no caso de estes dois parâmetros não serem especificados, o *login* será efetuado com *username* a tomar o valor "*anonymous*" e a password o valor "none". Para isto é utilizado o comando **USER** e o comando **PASS**.

Depois de um login com sucesso, é chamada a função **receiveFile**, que começa por separar o caminho do ficheiro em caminho das pastas e nome do ficheiro. Se ao remover o nome do ficheiro do caminho este não ficar vazio, então é enviado o comando **CWD**, com o resto do caminho a mudar o "*working directory*" para essa pasta.

Posteriormente, é enviado ao servidor o comando **PASV** com o fim de abrir a ligação em modo passivo, e, de seguida, a mensagem do servidor é lida e dela é extraído o endereço IP e a porta proveniente da mensagem, abrindo a segunda ligação no servidor. Após estes acontecimentos, o programa muda o tipo de transferência para binário, utilizando o comando "**TYPE I**", e recebe o tamanho do ficheiro a transferir pelo comando "**SIZE <filename>**". Finalmente, o programa envia o comando "**RETR <filename>**" e entra num *loop* no qual vai ler da segunda conexão um certo número de bytes por ciclo, escrevendo-os num ficheiro com o mesmo nome.

Para auxiliar o programa, foram desenvolvidas funções como **sendCommand,** que envia uma *string* para o servidor acabada em "\r\n", como especificado pelo RFC 959. Outra função deste género é a função **receiveMessage,** que lê linha-a-linha a mensagem do servidor, guardando o código da mensagem e um número de três dígitos, no qual o primeiro dígito índica um erro se for 4 ou 5. Esta função acaba quando receber uma linha que contenha o código com um espaço de seguida, como especificado pelo RFC 959, ou então quando ocorrer um *timeout*.

O programa também permite o *upload* de ficheiros usando o argumento -u com o comando **STOR**.

3. Rede de Computadores

3.1. Experiência 1 – Configuração de um IP de rede

Esta experiência consiste na conexão entre o **tux1** e o **tux4** através do ***switch***. Para este efeito basta configurar os tux's com o comando ifconfig eth0 up, seguido do ip respetivo.

1- Pacotes ARP e sua finalidade

Os pacotes ARP baseiam-se no protocolo de comunicação *Adress Resolution Protocol*, cuja finalidade é encontrar o endereço da camada de ligação do endereço IPv4. Estes pacotes têm como fim o mapeamento do endereço de rede a um endereço físico (MAC).

2- Endereços MAC e IP dos pacotes ARP

Introduzindo o comando *ping* para o tux4 no tux1 (após efetuada a ligação destes), o tux4 envia um pacote ARP com o seu endereço IP e endereço MAC. O endereço IP do tux4 é 172.16.60.254 e o seu MAC é 00:21:5a:c5:61:bb. O pacote enviado funciona como uma pergunta ao tux1, visto que o tux4 inquere sobre qual o endereço MAC do tux que realizou o comando *ping*. Como o tux4 não sabe o MAC do tux1, esse endereço é registado como 00:00:00:00:00:00. O tux1 responde confirmando que foi ele que fez ping e envia o seu endereço MAC (Figura 1). Sendo assim:

- Tux1 – IP:172. 16.60.1 MAC: 00:0f:fe:8c:af:71.
- Tux4- IP: 172. 16.60.254 MAC: 00:21:5a:c5:61:bb.

3- Pacotes gerados pelo comando *ping*

O comando *ping,* quando introduzido na consola, gera não só pacotes ARP como também pacotes ICMP (Figura 2). Enquanto que os primeiros têm o objetivo de obter os endereços MAC, os segundos auxiliam na ocorrência de erros.

4- Endereços MAC e IP dos pacotes *ping*

Fazendo *ping* ao tux4 através do tux1, os endereços IP e MAC de origem e destino dos pacotes passam a ser os destes tux's.

Pacote de pedido:

- Endereço MAC origem: 00:0f:fe:8c:af:71 (tux1).
- Endereço IP origem: 172. 16.60.1 (tux1).
- Endereço MAC destino: 00:21:5a:c5:61:bb (tux4).
- Endereço IP destino: 172. 16.60.254 (tux4).

Pacote de resposta:

- Endereço MAC origem: 00:21:5a:c5:61:bb (tux4).
- Endereço IP origem: 172.16.60.254 (tux4).
- Endereço MAC destino: 00:0f:fe:8c:af:71 (tux1).
- Endereço IP destino: 172. 16.60.1 (tux1).

5- Determinação do tipo (ARP, IP, ICMP) da trama recetora Ethernet

O tipo da trama recetora Ethernet é descoberto através da inspeção do Ethernet Header de um pacote. A trama será do tipo IPv4 se o Ethernet Header possuir o valor 0x0800. Analisando o IP header, caso assuma valor 1, o protocolo é ICMP, caso assuma o valor 0x0806, o protocolo é ARP (Figura 3).

6- Determinação do comprimento de uma trama recetora

A determinação do comprimento da trama recetora é determinada recorrendo ao Wireshark, um programa analisador dos protocolos de internet.

7- Interface *loopback* e sua importância

A interface *loopback* trata-se de uma interface de rede virtual que permite que os tux's recebam respostas sobre eles próprios. A sua importância é fulcral no teste da configuração da carta de rede.

3.2. Experiência 2 – Implementação de duas LANs virtuais num switch

A segunda experiência efetuada traduz-se na criação de VLAN60 e VLAN61, duas LAN's virtuais. Os tux's 1 e 4 foram conectados à VLAN60. Por sua vez, o tux2 foi associado à outra VLAN concebida.

1- Configuração de VLAN60

Com o fim de configurar VLAN60, é necessário estabelecer ligações de cabos entre os tux's e o *switch,* sendo que a porta T4 da primeira régua tem que estar conectada à porta do *switch* da segunda régua. Por sua vez, a porta T3 da primeira régua precisa de estar ligada à porta S0 de um dos tux's.

A criação da *vlan* resulta da invocação no **GTKTerm** dos seguintes comandos:

- configure terminal
- vlan 60
- end

A adição das portas do tux1 e do tux4 é o resultado da introdução dos seguintes comandos:

- configure terminal

- interface fasethernet 0/nº da porta à qual o tux está ligado (por exemplo 1 para o tux1 e 2 para o tux4)
- switchport mode acess
- switchport acess vlan 60
- end

2- Domínios de transmissão e conclusões retiradas dos registos

Existem dois domínios de transmissão: um que contém tux1 e tux4 e outro que contém apenas tux2. Conclui-se a partir dos registos que tux1 consegue receber resposta do tux4, mas não do tux2, quando é feito *ping broadcast,* visto que tux2 não se encontra na mesma vlan (e estas se encontram, por agora, desconectadas).

3.3. Experiência 3 – Configuração de um router em LINUX

Por sua vez, a execução desta experiência residiu na configuração de tux4 como um *router*, com a finalidade de estabelecer uma ligação entre VLAN60 e VLAN61. Para tal foram adicionadas rotas extras aos tux's com o comando route add -net <ip de destino> gw <ip da gateway que se quer associar>.

1- Rotas dos tux's e seu significado

As rotas para associadas a cada tux são agora as seguintes:

- Tux1 – 172.16.61.0 gateway: 172.16.60.254 (eth0).
  172.16.60 gateway: 0.0.0.0 (eth0).
- Tux2 – 172.16.61.0 gateway: 0.0.0.0 (eth0).
  172.16.60.0 gateway: 172.16.61.253 (eth0).
- Tux4 – 172.16.60.0 gateway: 0.0.0.0 (eth0).
  172.16.61.0 gateway: 0.0.0.0 (eth1).
  172.16.60.0 gateway: 172.16.60.254 (eth0).
  172.16.61.0 gateway: 172.16.61.253 (eth1).

Isto significa que, para além do que já se sabia de experiências anteriores, há agora uma ligação estabelecida entre tux1 e tux2 através do tux4.

2- Informações fornecidas por uma entrada da tabela de *forwarding*

A tabela de *forwarding* contém a seguinte informação:

- Destination – destino da rota.
- Gateway – IP do próximo ponto da rota.
- Natemask – determina o ID da rede a partir do IP do destino.

- Flags – informações sobre as idiossincrasias da rota.
- Metric – custo das rotas.
- Ref – total de referências para a rota.
- Use – contador de pesquisas pela rota. Conta número de sucessos ou falhas da cache.
- Interface – indicação da placa de rede gateway (eth0 ou eth1).

3- Causa e observação de mensagens ARP e endereços MAC

No caso de um tux dar *ping* a outro tux que tenho recebido o *ping* mas não conheça o endereço MAC do que realizou o comando, o tux que recebeu o *ping* pergunta qual o MAC do tux com aquele IP através do envio de uma mensagem ARP. Aí, o tux na origem do ping envia outra mensagem ARP contendo o seu endereço MAC.

4- Causa e observação de pacotes ICMP

Os pacotes ICMP observados são de *request* e *reply,* visto que depois da adição das rotas os tux's têm a possibilidade de se visualizarem. No caso de esta visualização ser impossível, os pacotes ICMP enviados seriam de *Host Unreachable*.

5- Endereços IP e MAC associados a um pacote ICMP

Os endereços IP e MAC associados aos pacotes ICMP são os endereços IP e MAC dos tux's de origem e de destino.

3.4. Experiência 4 – Configuração de um router comercial e Implementação NAT

Esta experiência consistiu na configuração de um router comercial. Primeiro foi implementado com uma conexão com a rede do laboratório. Posteriormente, o router foi configurado com NAT, estabelecendo uma ligação entre os computadores da rede e a internet.

1- Configuração de um router estático num router comercial

Com o fim de configurar um router comercial, conectou-se a porta T4 da primeira régua à porta do router da segunda régua e a porta T3 também da primeira régua à porta S0 do tux4. De seguida invocam-se os seguintes comandos no terminal do **GTKTerm:**

- configure terminal
- interface gigabitethernet 0/0
- ip address 172.16.61.254 255.255.255.0
- no shutdown
- exit
- ip route 172.16.60.0 255.255.255.0 172.16.61.253

2- Rotas seguidas pelos pacotes no decorrer da experiência

Durante a experiência, os pacotes seguem a rota especificada consoante o IP de destino. No caso de não haver especificação para tal, os pacotes seguem a rota *default* associada à máquina de origem.

3- Configuração NAT num router comercial

A configuração do NAT num router comercial implica a configuração da interface interna através do **GTKTerm** com a inserção dos comandos presentes em anexo, secção 5.2.5.

4- Funcionalidades NAT

O **Network Adress Translation (**NAT) opera num router, no qual são conectadas duas redes e traduzidos endereços privados para endereços legais, objetivando a conservação de endereços IP, permitindo a conexão a uma rede pública ou à internet por parte de redes IP privadas que usem endereços IP não registados.

3.5. Experiência 5 – DNS

A experiência 5 consiste na configuração do *Domain Name System* (DNS) em todos os tux's. Um servidor de DNS possui uma base de dados que contém os endereços IP públicos, bem como os respetivos *hostnames.* O servidor DNS utilizado nesta experiência foi **services.netlab.fe.up.pt**(172.16.1.1).

1- Configuração do serviço DNS num *host*

Para a configuração do serviço DNS, é necessária a alteração em cada um dos tux's do ficheiro **resolv.conf**. O ficheiro alterado deve conter:

i. search netlab.fe.up.pt
ii. nameserver 172.16.1.1

2- Pacotes trocados pelo DNS e informação transportada

Através do DNS, é enviado um pacote que contém o *hostname* desejado do *host* para o *server,* pedindo o seu endereço IP. A resposta do servidor consiste no envio de um pacote que contém o IP do *hostname*.

3.6. Experiência 6 – Conexões TCP

Por fim, a última experiência que realizámos consistiu na observação do protocolo TCP através do uso da aplicação que concebemos e da rede criada nas experiências anteriores.

1- Conexões TCP abertas pela aplicação FTP

Foram abertas duas conexões TCP pela aplicação TCP. A primeira envia os comandos FTP ao *server* e recebe respostas. A segunda recebe dados enviados pelo servidor e recebe respostas do *client.*

2- Conexão de transporte do controlo de informação

É na conexão TCP responsável pela troca de comandos que é transportado o controlo de informação.

3- Fases da conexão TCP

A conexão TCP possui três fases: estabelecimento da ligação, troca de dados e encerramento da ligação.

4- Mecanismo AQR TCP, campos TCP relevantes, informação relevante visualizada nos logs

O Transmission Control Protocol (TCP) usa o mecanismo Automatic Repeat Request (ARP) com o método da janela deslizante, que consiste na transmissão de dados e controlo de erros. Os logs relativos a esta experiência encontram-se em anexo, secção 5.1.6.

5- Mecanismo de controlo de congestão TCP, evolução do fluxo de dados da conexão ao longo do tempo e comparação com mecanismo de controlo de congestão TCP

O mecanismo de controlo de congestão é realizado sempre que o TCP mantém uma janela de congestão que se baseia numa estimativa da quantidade de octetos que a rede encaminha, não enviando mais octetos do que o mínimo definido pelo recetor e janela de congestão. O fluxo de dados de conexão está de acordo com o mecanismo de controlo de congestão.

6- Alteração da conexão de dados TCP devido ao aparecimento de uma segunda conexão TCP

O aparecimento de uma segunda conexão TCP no decorrer de uma transferência de dados em simultâneo pode levar a um decréscimo da taxa de transmissão, visto que a taxa de transferência é distribuída igualmente para ambas as ligações.

4. Conclusões

Em suma, a implementação do cliente de *download* e a configuração de uma rede de computadores foram benéficas para os elementos constituintes do grupo devido à vasta aquisição de novos conhecimentos.

Enquanto que as experiências realizadas contribuíram para uma aprendizagem valiosa sobre conceitos importantes no estabelecimento de uma rede de computadores, a implementação do cliente de *download* trouxe-nos novas ilações sobre o funcionamento do *File Transfer Protocol.*

Os objetivos a atingir foram alcançados, visto que implementámos com êxito tanto a aplicação de *download* como a rede em si, conforme nos foi proposto.

5. Anexos

5.1. Logs capturados durante as experiências

5.1.1. Experiência 1

```
43 40.317012    HewlettP_c5:61:bb    G-ProCom_8c:af:71    ARP    60 Who has 172.16.60.1? Tell 172.16.60.254
44 40.317025    G-ProCom_8c:af:71    HewlettP_c5:61:bb    ARP    42 172.16.60.1 is at 00:0f:fe:8c:af:71
```

Figura 1 - Experiência 1, Ponto 10: Pacotes ARP

```
28 35.300953    172.16.60.1      172.16.60.254    ICMP    98 Echo (ping) request  id=0x0fbf, seq=1/256, ttl=64 (reply in 29)
29 35.301213    172.16.60.254    172.16.60.1      ICMP    98 Echo (ping) reply    id=0x0fbf, seq=1/256, ttl=64 (request in 28)
```

Figura 2 - Experiência 1, Ponto 10: Pacotes ICMP

```
Ethernet II, Src: G-ProCom_8c:af:71 (00:0f:fe:8c:af:71), Dst: HewlettP_c5:61:bb (00:21:5a:c5:61:bb)
  > Destination: HewlettP_c5:61:bb (00:21:5a:c5:61:bb)
  > Source: G-ProCom_8c:af:71 (00:0f:fe:8c:af:71)
    Type: IPv4 (0x0800)

Ethernet II, Src: G-ProCom_8c:af:71 (00:0f:fe:8c:af:71), Dst: HewlettP_c5:61:bb (00:21:5a:c5:61:bb)
  > Destination: HewlettP_c5:61:bb (00:21:5a:c5:61:bb)
  > Source: G-ProCom_8c:af:71 (00:0f:fe:8c:af:71)
    Type: ARP (0x0806)
```

Figura 3 - Experiência 1, Ponto 10: Tipos de trama recebidos

5.1.2. Experiência 2

```
20 12.519822    172.16.60.1      172.16.60.254    ICMP    98 Echo (ping) request  id=0x1524, seq=6/1536, ttl=64 (reply in 21)
21 12.520187    172.16.60.254    172.16.60.1      ICMP    98 Echo (ping) reply    id=0x1524, seq=6/1536, ttl=64 (request in 20)
22 12.523878    HewlettP_c5:61:bb    G-ProCom_8c:af:71    ARP    60 Who has 172.16.60.1? Tell 172.16.60.254
23 12.523889    G-ProCom_8c:af:71    HewlettP_c5:61:bb    ARP    42 172.16.60.1 is at 00:0f:fe:8c:af:71
```

Figura 4 - Experiência 2, Ponto 6 (resposta de tux4)

5.1.3. Experiência 3

6 4.045191	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request	id=0x20fb, seq=1/256, ttl=64 (reply in 7)
7 4.045401	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply	id=0x20fb, seq=1/256, ttl=64 (request in 6)

Figura 5 - Experiência 3, Ponto 7 (resposta de tux4, eth0)

21 12.757461	172.16.60.1	172.16.61.253	ICMP	98 Echo (ping) request	id=0x2102, seq=1/256, ttl=64 (reply in 22)
22 12.757820	172.16.61.253	172.16.60.1	ICMP	98 Echo (ping) reply	id=0x2102, seq=1/256, ttl=64 (request in 21)

Figura 6 - Experiência 3, Ponto 7 (resposta de tux4, eth1)

37 26.805311	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request	id=0x2109, seq=1/256, ttl=64 (reply in 38)
38 26.805750	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply	id=0x2109, seq=1/256, ttl=64 (request in 37)

Figura 7 - Experiência 3, Ponto 7 (resposta de tux2)

8 11.992717	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request	id=0x22d1, seq=1/256, ttl=64 (reply in 11)
9 11.992791	Kye_04:20:8c	Broadcast	ARP	60 Who has 172.16.61.1? Tell 172.16.61.253	
10 11.992968	HewlettP_5a:7d:9c	Broadcast	ARP	60 Who has 172.16.60.1? Tell 172.16.61.1	
11 11.993125	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply	id=0x22d1, seq=1/256, ttl=64 (request in 8)

Figura 8 - Experiência 3, Ponto 11, eth0

7 9.987569	Kye_04:20:8c	Broadcast	ARP	42 Who has 172.16.61.1? Tell 172.16.61.253	
8 9.987679	HewlettP_5a:7d:9c	Kye_04:20:8c	ARP	60 172.16.61.1 is at 00:21:5a:5a:7d:9c	
9 9.987686	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request	id=0x22d1, seq=1/256, ttl=63 (reply in 11)
10 9.987781	HewlettP_5a:7d:9c	Broadcast	ARP	60 Who has 172.16.60.1? Tell 172.16.61.1	
11 9.987938	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply	id=0x22d1, seq=1/256, ttl=64 (request in 9)

Figura 9 - Experiência 3, Ponto 11, eth1

5.1.4. Experiência 4

29 15.339778	172.16.60.1	172.16.61.253	ICMP	98 Echo (ping) request	id=0x1d20, seq=2/512, ttl=64 (reply in 30)
30 15.340013	172.16.61.253	172.16.60.1	ICMP	98 Echo (ping) reply	id=0x1d20, seq=2/512, ttl=64 (request in 29)

Figura 10 - Experiência 4, Ponto 3 (resposta de tux4, eth1)

45 28.251234	172.16.60.1	172.16.61.1	ICMP	98 Echo (ping) request	id=0x1d2b, seq=1/256, ttl=64 (reply in 46)
46 28.251726	172.16.61.1	172.16.60.1	ICMP	98 Echo (ping) reply	id=0x1d2b, seq=1/256, ttl=63 (request in 45)

Figura 11 - Experiência 4, Ponto 3 (resposta de tux2)

63	41.027154	172.16.60.1	172.16.61.254	ICMP	98	Echo (ping) request id=0x1d35, seq=2/512, ttl=64 (reply in 64)
64	41.027824	172.16.61.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1d35, seq=2/512, ttl=254 (request in 63)

Figura 12 - Experiência 4, Ponto 3 (resposta do router)

15	6.668029	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) request id=0x51eb, seq=4/1024, ttl=64 (reply in 17)
16	6.668354	172.16.61.254	172.16.61.1	ICMP	70	Redirect (Redirect for host)
17	6.668750	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) reply id=0x51eb, seq=4/1024, ttl=63 (request in 15)
18	7.668031	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) request id=0x51eb, seq=5/1280, ttl=64 (reply in 20)
19	7.668373	172.16.61.254	172.16.61.1	ICMP	70	Redirect (Redirect for host)
20	7.668583	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) reply id=0x51eb, seq=5/1280, ttl=63 (request in 18)
21	8.019446	Cisco_3a:f1:04	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/61/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8004
22	8.676533	Kye_04:20:8c	HewlettP_5a:7d:9c	ARP	60	Who has 172.16.61.1? Tell 172.16.61.253
23	8.676553	HewlettP_5a:7d:9c	Kye_04:20:8c	ARP	42	172.16.61.1 is at 00:21:5a:5a:7d:9c

Figura 13 - Experiência 4, Ponto 3

5.1.5. Experiência 5

2	1.454294	172.16.60.1	172.16.1.1	DNS	73	Standard query 0x284a A www.google.pt
3	1.456018	172.16.1.1	172.16.60.1	DNS	347	Standard query response 0x284a A www.google.pt A 172.217.17.3 NS ns2.google.com NS ns1.google.com NS ns4.google.c…
4	1.456232	172.16.60.1	172.217.17.3	ICMP	98	Echo (ping) request id=0x331b, seq=1/256, ttl=64 (reply in 5)
5	1.471379	172.217.17.3	172.16.60.1	ICMP	98	Echo (ping) reply id=0x331b, seq=1/256, ttl=50 (request in 4)

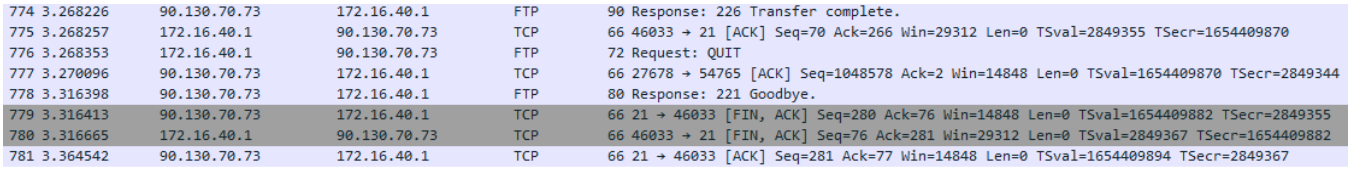
Figura 14 - Experiência 5, Ponto 3

5.1.6. Experiência 6[1]

9	1.233165	172.16.40.1	90.130.70.73	TCP	74	46033 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2848846 TSecr=0 WS=128
10	1.281827	90.130.70.73	172.16.40.1	TCP	74	21 → 46033 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PERM=1 TSval=1654409374 TSecr=2848846 WS=512
11	1.281872	172.16.40.1	90.130.70.73	TCP	66	46033 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=2848859 TSecr=1654409374
12	1.342323	90.130.70.73	172.16.40.1	FTP	86	Response: 220 (vsFTPd 2.3.5)
13	1.342358	172.16.40.1	90.130.70.73	TCP	66	46033 → 21 [ACK] Seq=1 Ack=21 Win=29312 Len=0 TSval=2848874 TSecr=1654409389
14	1.342425	172.16.40.1	90.130.70.73	FTP	82	Request: user anonymous
15	1.390428	90.130.70.73	172.16.40.1	TCP	66	21 → 46033 [ACK] Seq=21 Ack=17 Win=14848 Len=0 TSval=1654409401 TSecr=2848874
16	1.390446	90.130.70.73	172.16.40.1	FTP	100	Response: 331 Please specify the password.
17	1.390500	172.16.40.1	90.130.70.73	FTP	77	Request: pass none
18	1.477415	90.130.70.73	172.16.40.1	TCP	66	21 → 46033 [ACK] Seq=55 Ack=28 Win=14848 Len=0 TSval=1654409423 TSecr=2848886
19	1.538380	90.130.70.73	172.16.40.1	FTP	89	Response: 230 Login successful.
20	1.538426	172.16.40.1	90.130.70.73	FTP	72	Request: PASV
21	1.586338	90.130.70.73	172.16.40.1	TCP	66	21 → 46033 [ACK] Seq=78 Ack=34 Win=14848 Len=0 TSval=1654409450 TSecr=2848923
22	1.586825	90.130.70.73	172.16.40.1	FTP	116	Response: 227 Entering Passive Mode (90,130,70,73,108,30).
23	1.625316	172.16.40.1	90.130.70.73	TCP	66	46033 → 21 [ACK] Seq=34 Ack=128 Win=29312 Len=0 TSval=2848945 TSecr=1654409450
24	2.407250	Cisco_d4:1c:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
25	2.587009	172.16.40.1	90.130.70.73	TCP	74	54765 → 27678 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2849185 TSecr=0 WS=128
26	2.635050	90.130.70.73	172.16.40.1	TCP	74	27678 → 54765 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PERM=1 TSval=1654409711 TSecr=2849185 WS…
27	2.635083	172.16.40.1	90.130.70.73	TCP	66	54765 → 27678 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=2849197 TSecr=1654409711
28	2.635159	172.16.40.1	90.130.70.73	FTP	74	Request: TYPE I
29	2.683167	90.130.70.73	172.16.40.1	FTP	97	Response: 200 Switching to Binary mode.
30	2.683200	172.16.40.1	90.130.70.73	TCP	66	46033 → 21 [ACK] Seq=42 Ack=159 Win=29312 Len=0 TSval=2849209 TSecr=1654409724
31	2.683226	172.16.40.1	90.130.70.73	FTP	80	Request: SIZE 1MB.zip
32	2.731197	90.130.70.73	172.16.40.1	FTP	79	Response: 213 1048576
33	2.731292	172.16.40.1	90.130.70.73	FTP	80	Request: RETR 1MB.zip
34	2.779707	90.130.70.73	172.16.40.1	FTP	136	Response: 150 Opening BINARY mode data connection for 1MB.zip (1048576 bytes).
35	2.782537	90.130.70.73	172.16.40.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (TYPE I)
36	2.782582	172.16.40.1	90.130.70.73	TCP	66	54765 → 27678 [ACK] Seq=1 Ack=1449 Win=32128 Len=0 TSval=2849234 TSecr=1654409748

Figura 15 - Experiência 6, Ponto 3 (início da transferência)

[1] Notar que os logs desta experiência são referentes a uma bancada diferente (bancada 4) que a 6 devido a esta estar ocupada aquando da realização da experiência.



774	3.268226	90.130.70.73	172.16.40.1	FTP	90 Response: 226 Transfer complete.
775	3.268257	172.16.40.1	90.130.70.73	TCP	66 46033 → 21 [ACK] Seq=70 Ack=266 Win=29312 Len=0 TSval=2849355 TSecr=1654409870
776	3.268353	172.16.40.1	90.130.70.73	FTP	72 Request: QUIT
777	3.270096	90.130.70.73	172.16.40.1	TCP	66 27678 → 54765 [ACK] Seq=1048578 Ack=2 Win=14848 Len=0 TSval=1654409870 TSecr=2849344
778	3.316398	90.130.70.73	172.16.40.1	FTP	80 Response: 221 Goodbye.
779	3.316413	90.130.70.73	172.16.40.1	TCP	66 21 → 46033 [FIN, ACK] Seq=280 Ack=76 Win=14848 Len=0 TSval=1654409882 TSecr=2849355
780	3.316665	172.16.40.1	90.130.70.73	TCP	66 46033 → 21 [FIN, ACK] Seq=76 Ack=281 Win=29312 Len=0 TSval=2849367 TSecr=1654409882
781	3.364542	90.130.70.73	172.16.40.1	TCP	66 21 → 46033 [ACK] Seq=281 Ack=77 Win=14848 Len=0 TSval=1654409894 TSecr=2849367

Figura 16 - Experiência 6, Ponto 3 (fim da transferência)

5.2. Comandos de configuração usados

5.2.1. Tux61

- service networking restart
- ifconfig eth0 up 172.16.60.1/24
- route add default gw 172.16.60.254
- route add -net 172.16.61.0/24 gw 172.16.60.254
- printf "search netlab.fe.up.pt\nnameserver 172.16.1.1\nnameserver 172.16.2.1\n" > /etc/resolv.conf

5.2.2. Tux62

- service networking restart
- ifconfig eth0 up 172.16.61.1/24
- route add default gw 172.16.61.254
- route add -net 172.16.60.0/24 gw 172.16.61.253
- printf "search netlab.fe.up.pt\nnameserver 172.16.1.1\nnameserver 172.16.2.1\n" > /etc/resolv.conf

5.2.3. Tux64

- service networking restart
- ifconfig eth0 up 172.16.60.254/24
- ifconfig eth1 up 172.16.61.253/24
- route add default gw 172.16.61.254
- echo 1 > /proc/sys/net/ipv4/ip_forward
- echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
- printf "search netlab.fe.up.pt\nnameserver 172.16.1.1\nnameserver 172.16.2.1\n" > /etc/resolv.conf

5.2.4. Switch

```
-   configure terminal
-   vlan 60
-   end
-   configure terminal
-   interface fastethernet 0/1
-   switchport mode access
-   switchport access vlan 60
-   end
-   configure terminal
-   interface fastethernet 0/2
-   switchport mode access
-   switchport access vlan 60
-   end
-   configure terminal
-   vlan 61
-   end
-   configure terminal
-   interface fastethernet 0/3
-   switchport mode access
-   switchport access vlan 61
-   end
-   configure terminal
-   interface fastethernet 0/4
-   switchport mode access
-   switchport access vlan 61
-   end
-   configure terminal
-   interface fastethernet 0/5
-   switchport mode access
-   switchport access vlan 61
-   end
```

5.2.5. Router

```
-   conf t
-   interface gigabitethernet 0/0
-   ip address 172.16.61.254 255.255.255.0
-   no shutdown
-   ip nat inside
-   exit
-   interface gigabitethernet 0/1
-   ip address 172.16.1.69 255.255.255.0
-   no shutdown
-   ip nat outside
-   exit
-   ip nat pool ovrld 172.16.1.69 172.16.1.69 prefix 24
-   ip nat inside source list 1 pool ovrld overload
-   access-list 1 permit 172.16.60.0 0.0.0.7
-   access-list 1 permit 172.16.61.0 0.0.0.7
-   ip route 0.0.0.0 0.0.0.0 172.16.1.254
-   ip route 172.16.60.0 255.255.255.0 172.16.61.253
-   end
```

5.3. Código da aplicação

- **makefile**

```
CC          = gcc
CFLAGS      = -g
LDFLAGS     =
DEPS        = main.c connection.c connection.h utilities.c utilities.h makefile
OBJFILES    = main.o connection.o utilities.o
TARGET      = download

all: $(TARGET)

$(TARGET): $(OBJFILES) $(DEPS)
	$(CC) $(CFLAGS) -o $(TARGET) $(OBJFILES) $(LDFLAGS)

clean:
	rm -f $(OBJFILES) $(TARGET) *~
```

- **README.txt**

Usage:

./download ftp://[<user>:<password>@]<host>/<url-path>

Examples:

./download “ftp://ftp.fe.up.pt/welcome.msg”
./download “ftp://speedtest.tele2.net/50MB.zip”
./download “ftp://dlpuser@dlptest.com:e73jzTRTNqCN9PYAAjjn@ftp.dlptest.com/curl.txt”

- **constants.h**

```
#ifndef CONSTANTS_H
#define CONSTANTS_H
#define ANONYMOUS "anonymous"
#define SERVER_PORT 21
#define SERVER_HOSTNAME "ftp.fe.up.pt"
#define MAXCONNECTIONS 10
#define MAXATTEMPTS 3
#define TIMEOUT 3
#endif
```

- **connection.h**

```
#ifndef CONNECTION_H
#define CONNECTION_H
#include <stdio.h>
typedef struct
{
	char* IPAddress;
	int port;
	FILE* fp;
}
connection;

int initializeConnection(connection** conn);
int freeConnection(connection** conn);
#endif
```

- **connection.c**

```
#include "connection.h"
#include <stdlib.h>

int initializeConnection(connection** conn)
{
	*conn = malloc(sizeof(connection*));
	(*conn)->IPAddress = malloc(16);
	return 0;
}

int freeConnection(connection** conn)
{
	free((*conn)->IPAddress);
	free(*conn);
	return 0;
}
```

- **utilities.h**

```
#ifndef UTILITIES_H
#define UTILITIES_H

#include <stdio.h>
#include <fcntl.h>

int getFreeFilePointer(FILE** fpArray);
int closeSockets(FILE** fileArray);
void sigalrm_handler(int signal);
int setHandler();
int isNumber(char input);
void printUsage();
int findFirst(char* str, char target);
int findLast(char* str, char target);
int extractFromArgument(char* input, char* username, char* password, char* hostname, char*
filename);
void printPercentage(double percentage);
void printTransferRate(double rate);
void clearScreen();
int splitFilename(char* fullPath, char* path, char* filename);

#endif
```

- **utilities.c**

```
#include "utilities.h"
#include <string.h>
#include <stdlib.h>
#include <signal.h>
#include "constants.h"

int getFreeFilePointer(FILE** fpArray)
{
    int i;
    for (i = 0; i < MAXCONNECTIONS; i++)
    {
            if (fpArray[i] == NULL)
                    return i;
    }
    return -1;
}

int closeSockets(FILE** fileArray)
{
    int i;
    for (i = 0; fileArray[i] != NULL; ++i)
    {
            fclose(fileArray[i]);
    }

    free(fileArray);
    return 0;
}

void sigalrm_handler(int signal)
{
    printf("Server response timed out!\n");
}
```

```
int setHandler()
{
    struct sigaction sigalrm_action;
    sigalrm_action.sa_handler = sigalrm_handler;
    sigemptyset(&sigalrm_action.sa_mask);
    sigalrm_action.sa_flags = 0;

    if (sigaction(SIGALRM, &sigalrm_action, NULL) < 0)
    {
            fprintf(stderr,"Unable to install SIGINT handler\n");
            return 1;
    }
    return 0;
}


int isNumber(char input)
{
    return (input >= '0' && input <= '9');
}

void printUsage()
{
    printf("Usage: download [OPTIONS] [HOSTNAME] [FILE]\n");
}

int findFirst(char* str, char target)
{
    int i;
    for (i = 0; str[i] != 0; i++)
    {
            if (str[i] == target)
                    return i;
    }
    return -1;
}
```

```
int findLast(char* str, char target)
{
    int i, ret = -1;
    for (i = 0; str[i] != 0; i++)
    {
        if (str[i] == target)
            ret = i;
    }
    return ret;
}

int extractFromArgument(char* input, char* username, char* password, char* hostname, char*
filename)
{
    char ftp[7];
    memcpy(ftp, input, 6);
    ftp[6] = 0;

    username[0] = 0;
    password[0] = 0;
    hostname[0] = 0;
    filename[0] = 0;

    if (strcmp(ftp, "ftp://") == 0)
    {
        input += 6;
        int index = findLast(input, '@');

        if (index != -1) // Login present
        {
            int index2 = findFirst(input, ':');

            if (index2 != -1)
            {
                memcpy(username, input, index2+1);
                username[index2] = 0;

                memcpy(password, input+index2+1, index-index2-1);
                password[index-index2+1] = 0;
```

```
                input += index+1;
            }
            else
                return 1;
        }
        else // No login present
        {
            strcpy(username, ANONYMOUS);
            strcpy(password, "none");
        }

        int index2 = findFirst(input, '/');

        if (index2 != -1)
        {
            memcpy(hostname, input, index2);
            hostname[index2+1] = 0;

            strcpy(filename, input+index2+1);
        }
        else
            return 1;

        return 0;
    }

    return 1;
}
```

```
void printPercentage(double percentage)
{
    if (percentage >= 0 && percentage <= 1)
    {
            printf("<");
            int i, length = 15 /* length of the percentage bar */;
            for (i = 0; i < length; i++)
            {
                    if ((double)i/length < percentage)
                            printf("|");
                    else
                            printf(" ");
            }
            printf(">%.1f%%\n", percentage*100);
    }
}

void printTransferRate(double rate)
{
    if (rate > 0)
    { printf("Transfer rate : %.1f KB/s\n", rate/1024); }
}

void clearScreen()
{
    printf("\033[2J\033[1;1H");
    printf("\033[2J\033[1;1H");
}
```

```
int splitFilename(char* fullPath, char* path, char* filename)
{
    int i, index = -1;
    for (i = 0; fullPath[i] != 0; i++)
    {
            if (fullPath[i] == '/')
                    index = i;
    }

    if (index != -1)
    {
            memcpy(path, fullPath, index+1);
            path[index+1] = 0;
            strcpy(filename, fullPath+index+1);

            return 0;
    }
    else
    {
            path[0] = 0;
            strcpy(filename, fullPath);
            return -1;
    }
}
```

- **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 <netdb.h>
#include <string.h>
#include <fcntl.h>
#include <ctype.h>
#include <sys/time.h>
#include <sys/stat.h>

#include "constants.h"
#include "connection.h"
#include "utilities.h"

connection* currentFp = NULL;
```

```
int attemptConnect(connection* conn, char* message)
{
    int sockfd, i;
    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(conn->IPAddress);       /*32 bit Internet address network byte
ordered*/
    server_addr.sin_port = htons(conn->port);                       /*server TCP port must be network byte ordered
*/

    for (i = 0; i < MAXATTEMPTS; i++)
    {
            sleep(1);
            /*open an TCP socket*/
            if ((sockfd = socket(AF_INET,SOCK_STREAM,0)) < 0)
            {
                    // fprintf(stderr, "socket() failed!\n");
                    continue;
            }
            /*connect to the server*/
            if (connect(sockfd,
                                    (struct sockaddr *)&server_addr,
                            sizeof(server_addr)) < 0)
            {
                    // fprintf(stderr, "connect() failed!\n");
                    continue;
            }

            break;
    }
    if (i == MAXATTEMPTS)
    {
            fprintf(stderr, "Connection timed out!\n");
            return 1;
    }
    conn->fp = fdopen(sockfd, "r+");
    return 0;
```

```
}
int receiveMessage(connection* conn, char* message)
{
    size_t* bufferSize = malloc(sizeof(size_t));
    *bufferSize = 1024;
    int bytes, i, flag = 1;
    char* buffer = malloc(*bufferSize);
    char code[4];
    message[0] = 0;
    buffer[0] = 0;

    for (i = 0; flag; i++)
    {
        alarm(TIMEOUT);
        bytes = getline(&buffer, bufferSize, conn->fp);
        alarm(0); // Cancel alarm

        if (bytes < 0)
        {
            fprintf(stderr, "Error on reading from server!\n");
            free(bufferSize);
            free(buffer);
            return -1;
        }
        buffer[bytes] = 0;
        if (i == 0)
        {
            memcpy(code, buffer, 3);
            code[3] = 0;
        }

        if (isNumber(buffer[0]) && isNumber(buffer[1]) && isNumber(buffer[2]))
        {
            if (memcmp(code, buffer, 3) == 0)
            {
                if (buffer[3] == ' ')
                    flag = 0;
            }
        }
        strcat(message, buffer);
    }
```

```
    free(bufferSize);
    free(buffer);

    return (code[0] == '4' || code[0] == '5');
}

int sendCommand(connection* conn, char* command)
{
    int length = strlen(command);
    char* buffer = malloc(length+2+1);
    buffer[0] = 0;

    strcpy(buffer, command);
    strcat(buffer, "\r\n");

    int bytes = fwrite(buffer, 1, length+2, conn->fp);

    if (bytes != length+2)
    {
            fprintf(stderr, "Error seding command!\n");
            return 1;
    }

    return 0;
}

int closeConnection(connection* conn, char* message)
{
    message[0] = 0;
    strcpy(message, "QUIT");

    if (sendCommand(conn, message) != 0)
            return 1;

    if (receiveMessage(conn, message) != 0)
            return 1;

    printf("%s\n", message);
    return 0;
}
```

```
int login(connection* conn, char* message, char* username, char* password)
{
    char buffer[100+6+1];
    buffer[0] = 0;

    strcat(buffer, "user ");
    strcat(buffer, username);

    sendCommand(conn, buffer);

    if (receiveMessage(conn, message) != 0)
            return 1;

    // printf("%s\n", message);

    buffer[0] = 0;

    strcat(buffer, "pass ");
    strcat(buffer, password);

    sendCommand(conn, buffer);

    if (receiveMessage(conn, message) != 0)
            return 1;

    // printf("%s\n", message);

    return 0;
}

int enterPassiveMode(connection** connections, char* message)
{
    int sockfd, i, j, k, start, IPAddressCounter, messageLength;

    for (k = 0; k < MAXCONNECTIONS; k++)
    {
            connections[1]->IPAddress[0] = 0;
            start = -1;
            IPAddressCounter = 0;
            connections[1]->port = 0;
```

```
sendCommand(connections[0], "PASV");

receiveMessage(connections[0], message);
messageLength = strlen(message);

printf("Changing server for passive mode\n");

for (i = 0; i < messageLength; ++i)
{
	if (message[i] == '(')
		start = i+1;

	if (start != -1 && (message[i] == ',' || message[i] == ')'))
	{
		if (IPAddressCounter < 4)
		{
			int offset = strlen(connections[1]->IPAddress);

			memcpy(&connections[1]->IPAddress[offset], &message[start], i-start);

			if (IPAddressCounter < 3)
			{
				connections[1]->IPAddress[offset + i-start] = '.';
				connections[1]->IPAddress[offset + i-start + 1] = 0;
			}
			else
				connections[1]->IPAddress[offset + i-start] = 0;
		}
		else
		{
			char portString[4];
			memcpy(portString, &message[start], i-start);
			portString[i-start] = 0;

			long int part = strtol(portString, NULL, 10);

			if (IPAddressCounter == 4)
			{
				connections[1]->port = 256*part;
			}
```

```
                                else if (IPAddressCounter == 5)
                                {
                                        connections[1]->port += part;
                                        break;
                                }

                        }

                        start = i+1;
                        IPAddressCounter++;

                }
        }

        // printf("IPAddress = %s\n", connections[1]->IPAddress);
        // printf("port = %i\n", connections[1]->port);

        if (attemptConnect(connections[1], message) == 0)
                break;
    }

    if (i == MAXCONNECTIONS)
        return 1;

    return 0;
}

int receiveFile(connection** connections, char* message, char* serverFilename)
{
    char path[200];
    char filename[100];
    splitFilename(serverFilename, path, filename);

    if (path[0] != 0) // Not empty string
    {
        message[0] = 0;

        strcat(message, "CWD ");
        strcat(message, path);

        sendCommand(connections[0], message);
```

```
        receiveMessage(connections[0], message);
        printf("%s\n", message);

}

if (enterPassiveMode(connections, message) != 0)
        return 1;

// Sets transfer type to binary
message[0] = 0;

strcat(message, "TYPE I");

sendCommand(connections[0], message);
if (receiveMessage(connections[0], message) != 0)
{
        printf("%s\n", message);
        return 1;
}


// Gets size of file
message[0] = 0;

strcat(message, "SIZE ");
strcat(message, filename);

sendCommand(connections[0], message);
if (receiveMessage(connections[0], message) != 0)
{
        printf("%s\n", message);
        return 1;
}

long long size = strtoll(&message[4], NULL, 10);
// printf("size = %lli\n", size);


message[0] = 0;
```

```
    strcat(message, "RETR ");
    strcat(message, filename);

    sendCommand(connections[0], message);

    receiveMessage(connections[0], message);
    // printf("%s\n", message);

    size_t bufferSize = 512;
    int i, sumBytes = 0, bytes = bufferSize, fd = open(filename, O_WRONLY | O_TRUNC | O_CREAT,
0777);
    char* buffer = malloc(bufferSize);

    struct timeval startTime, finishTime;
    double sumTime = 0;

    if (gettimeofday(&startTime, NULL) != 0)
                printf("Error getting time!\n");
    if (gettimeofday(&finishTime, NULL) != 0)
                printf("Error getting time!\n");

    int sumBytesAverage = 0;
    double sumTimeAverage = 0, repeatTime = 0.5, rate = 0;
    for (i = 0; sumBytes < size; i++)
    {
            // usleep(100 * 65); // Sleeps for 10 millisecond
            bytes = fread(buffer, 1, bufferSize, connections[1]->fp);

            if (bytes < 0)
            {
                    printf("Error getting file from!\n");
                    return 1;
            }

            bytes = write(fd, buffer, bytes);

            if (bytes < 0)
            {
                    printf("Error writing to file!\n");
                    return 1;
            }
```

```
            if (gettimeofday(&finishTime, NULL) != 0)
                printf("Error getting time!\n");

            double deltaTime = (double)(finishTime.tv_sec - startTime.tv_sec) +
(double)(finishTime.tv_usec - startTime.tv_usec)/1000/1000; // In seconds

            if (gettimeofday(&startTime, NULL) != 0)
                printf("Error getting time!\n");


            if (sumTimeAverage > repeatTime)
            {
                clearScreen();

                rate = (double)sumBytesAverage / sumTimeAverage;

                sumBytesAverage = 0;
                sumTimeAverage = 0;

                printPercentage((double)sumBytes / size);
                printTransferRate(rate);
            }

            sumBytes += bytes;
            sumTime += deltaTime;

            sumBytesAverage += bytes;
            sumTimeAverage += deltaTime;
    }

    clearScreen();
    printPercentage((double)sumBytes / size);
    printTransferRate(rate);

    free(buffer);
    close(fd);

    fclose(connections[1]->fp);
```

```
    receiveMessage(connections[0], message);
    printf("%s\n", message);
    printf("File downloaded with average %.1f KB/s\n", (double)sumBytes/sumTime/1024);

    return 0;
}

int sendFile(connection** connections, char* message, char* filepath)
{
    char path[200];
    char filename[100];
    splitFilename(filepath, path, filename);

    if (enterPassiveMode(connections, message) != 0)
            return 1;


    // Sets transfer type to binary
    message[0] = 0;

    strcat(message, "TYPE I");

    sendCommand(connections[0], message);

    receiveMessage(connections[0], message);
    printf("%s\n", message);

    message[0] = 0;

    strcat(message, "STOR ");
    strcat(message, filename);

    sendCommand(connections[0], message);

    receiveMessage(connections[0], message);
    printf("%s\n", message);

    size_t bufferSize = 512;
    int i, sumBytes = 0, bytes = bufferSize, fd = open(filepath, O_RDONLY);
    char* buffer = malloc(bufferSize);
```

```
struct stat st;

if (stat(filename, &st) != 0)
{
	printf("Failed to get file size!\n");
	return 1;
}

int size = st.st_size;

struct timeval startTime, finishTime;
double sumTime = 0;

if (gettimeofday(&startTime, NULL) != 0)
		printf("Error getting time!\n");
if (gettimeofday(&finishTime, NULL) != 0)
		printf("Error getting time!\n");

int sumBytesAverage = 0;
double sumTimeAverage = 0, repeatTime = 0.5, rate = 0;
for (i = 0; sumBytes < size; i++)
{
	bytes = read(fd, buffer, bytes);

	if (bytes < 0)
	{
		printf("Error reading from file!\n");
		return 1;
	}

	bytes = fwrite(buffer, 1, bufferSize, connections[1]->fp);

	if (bytes < 0)
	{
		printf("Error sending file to server!\n");
		return 1;
	}


	if (gettimeofday(&finishTime, NULL) != 0)
		printf("Error getting time!\n");
```

```
            double deltaTime = (double)(finishTime.tv_sec - startTime.tv_sec) +
(double)(finishTime.tv_usec - startTime.tv_usec)/1000/1000; // In seconds

            if (gettimeofday(&startTime, NULL) != 0)
                printf("Error getting time!\n");

            if (sumTimeAverage > repeatTime)
            {
                clearScreen();

                rate = (double)sumBytesAverage / sumTimeAverage;

                sumBytesAverage = 0;
                sumTimeAverage = 0;

                printPercentage((double)sumBytes / size);
                printTransferRate(rate);
            }

            sumBytes += bytes;
            sumTime += deltaTime;

            sumBytesAverage += bytes;
            sumTimeAverage += deltaTime;
        }

        clearScreen();
        printPercentage((double)sumBytes / size);
        printTransferRate(rate);

        free(buffer);
        close(fd);

        fclose(connections[1]->fp);

        receiveMessage(connections[0], message);
        printf("%s\n", message);

        printf("File uploaded with average %.1f KB/s\n", (double)sumBytes/sumTime/1024);
```

```
    return 0; }
int main(int argc, char** argv)
{
    if (argc < 2 || argc > 3)
    {
        printUsage();
        return 1;
    }

    char *username = malloc(100), *password = malloc(100), *hostname = malloc(100), *filename =
malloc(100);

    if (extractFromArgument(argv[argc-1], username, password, hostname, filename) != 0)
    {
        printUsage();
        return 1;
    }

    char* options = "hu";
    int i, opterr = 0, uflag = 0;
    char c;

    while ((c = getopt (argc-1, argv, options)) != -1)
    {
        if (c == 'h')
        {
            printUsage();
            return 0;
        }
        if (c == 'u')
        {
            uflag = 1;
        }
        else if (c == '?')
        {
            int flag = 1;
            for (i = 0; options[i] != 0; i++)
            {
                if (options[i] == ':' && optopt == options[i-1])
                {
                    fprintf (stderr, "Option -%c requires an argument.\n", optopt);
```

```
                                flag = 0;
                        }

                }

                if (flag)
                {
                        if (isprint (optopt))
                                fprintf (stderr, "Unknown option `-%c'.\n", optopt);
                        else
                                fprintf (stderr, "Unknown option character `\\x%x'.\n", optopt);
                }
        }
        else
                return 1;
}

for (i = optind; i < argc-1; i++)
        printf ("Non-option argument %s\n", argv[i]);


char message[4096];

setHandler();

struct hostent * ent = gethostbyname(hostname);
char IPAddress[16];
inet_ntop(AF_INET, ent->h_addr_list[0], IPAddress, INET_ADDRSTRLEN);

connection* connections[2];
initializeConnection(&connections[0]);
initializeConnection(&connections[1]);

strcpy(connections[0]->IPAddress, IPAddress);
connections[0]->port = SERVER_PORT;


if (attemptConnect(connections[0], message) != 0)
        return 1;

receiveMessage(connections[0], message);
```

```
    printf("%s\n", message);
    if (login(connections[0], message, username, password) != 0)
    {
            fprintf(stderr, "Failed to login!\n");
            return 1;
    }

    printf("Login sucessful!\n");

    if (uflag)
    {
            if (sendFile(connections, message, filename))
                    return 1;
    }
    else
    {
            if (receiveFile(connections, message, filename))
                    return 1;
    }

    if (closeConnection(connections[0], message) != 0)
            return 1;

    freeConnection(&connections[0]);
    freeConnection(&connections[1]);

    return 0;
}
```