U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Redes de Computadores

2º Trabalho Laboratorial

Mestrado Integrado em Engenharia Informática e Computação

(20 de dezembro de 2018)

Bruno Sousa **up201604145**@fe.up.pt

João Gonçalves **up201604245**@fe.up.pt

Pedro Neto **up201604420**@fe.up.pt

Índice

Sumário

Este relatório serve de complemento ao segundo trabalho de Redes de Computadores. O trabalho consiste na configuração de uma rede de computadores e o desenvolvimento de uma aplicação que transfere um ficheiro com FTP ("File Transfer Protocol") usando para isso a rede configurada.

O trabalho foi concluído com sucesso, funcionando sem problemas a aplicação que executa a transferência de um ficheiro usando a rede previamente configurada.

Introdução

O trabalho pode ser dividido em dois pontos: a configuração de uma rede e o desenvolvimento de uma aplicação de download usando, para tal, a rede configurada.

Quanto ao relatório, vai ter a seguinte estrutura:

- **Parte 1: Aplicação de Download**

  Arquitetura da aplicação de download e respetivos resultados.

- **Parte 2: Configuração de Rede e Análise**

  Análise de cada experiência, bem como a resposta ás questões propostas.

- **Conclusões**

  Síntese da informação apresentada nas secções anteriores e reflexão sobre os objetivos de aprendizagem alcançados.

Parte 1: Aplicação de Download

A primeira parte deste trabalho foi desenvolver uma aplicação de transferência para qualquer tipo de ficheiros de um servidor com "File Transfer Protocol" com auxílio de um servidor da FEUP. Os argumentos para esta aplicação têm de apresentar a seguinte estrutura: **ftp://[<user>:<password>@]<host>/<url-path>**. Na omissão de user e password, a aplicação volta a pedir estes argumentos.

1.1 Arquitetura

A aplicação *download* começa por interpretar os argumentos (utilizando a função ***parseArguments***) com que é chamada, retornando um erro caso não estejam corretos.

Após isso, obtém o IP do hostname (utilizando a função ***getIPaddress***) e cria um socket para a comunicação com o servidor (utilizando a função ***createSocket***). Utilizando o socket criado, obtém a resposta do servidor (utilizando a função ***getServerReply***) e, se a resposta for a pretendida, envia ao servidor (utilizando a função ***sendCommand***) o username e a password (caso o utilizador não os tenha colocado como argumento, pede-os agora) verificando depois a resposta do servidor, para ver se conseguiu dar login.

Caso consiga, pede ao servidor para abrir um port para o envio do ficheiro, interpreta o número do port (utilizando a função ***getPort***) e cria um socket para comunicar com esse port. Em seguida pede ao servidor para enviar o ficheiro pretendido e lê-o do socket criado (utilizando a função ***downloadFile***).

Após isto tudo, verifica se o servidor lhe enviou a resposta de transferência completa, envia-lhe o comando ***quit*** e lê do servidor a resposta *goodbye*.

1.2 Resultados

Em "Anexos" pode-se verificar dois exemplos da nossa aplicação.

Na figura AP1 foram dados os argumentos com a estrutura "ftp://<user>:<password>@<host>/<url-path>".

Para o caso da figura AP2, foi omitido o user e password primeiramente, apresentando a estrutura: "ftp://<host>/<url-path>", pedindo ao utilizador que os forneça quando o servidor pedir os dados de login.

Na figura AP3, podemos ver que o ficheiro foi corretamente transferido.

Parte 2: Configuração de Rede e Análise

Experiência 1 – Configurar um IP de rede

O objetivo desta experiência foi conectar o tux1 ao tux4 utilizando switch.

1) O que são os pacotes ARP e para o que são usados?

ARP (*Address Resolution Protocol*) é um protocolo de comunicação usado para descobrir o endereço da camada de ligação (o endereço MAC, por exemplo) associado a um endereço de rede, tipicamente um endereço IPv4.

2) Quais são os endereços MAC e IP dos pacotes ARP e porquê?

Quando é efetuado um ping do tux1 para o tux4, é enviado (em *broadcast*) do tux1 um pacote ARP a perguntar qual o endereço MAC do endereço 172.16.30.254 (endereço IP do tux4) e para comunicar esse endereço ao endereço 172.16.30.1 (endereço IP do tux4). Em seguida, é recebido no tux1 um pacote ARP vindo do tux4 a dizer que o endereço MAC do endereço IP 172.16.30.254 é 00:21:5a:5a:7d:74 (endereço MAC do tux4).

Posteriormente são gerados os mesmos pacotes para o tux4, em que pergunta ao tux1 pelo seu endereço MAC (00:0f:fe:8b:e4:4d).

Consultar logs na figura EXP1.2.

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

O comando *ping* gera pacotes ARP para o tux1 obter o endereço MAC do tux4, pacotes ICMP (*Internet Control Message Protocol*) de request e reply e pacotes ARP para o tux4 obter o endereço MAC do tux1.

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

Dando *ping* ao tux4 desde o tux1 são gerados dois tipos de pacotes: *request* e *reply*.

Os pacotes *request* têm endereço MAC de origem 00:0f:fe:8b:e4:4d (tux1), endereço IP de origem 172.16.30.1 (tux1), endereço MAC de destino 00:21:5a:5a:7d:74 (tux4), endereço IP de destino 172.16.30.254 (tux4). Consultar logs na figura EXP1.4.1.

Os pacotes *reply* têm endereço MAC de origem 00:21:5a:5a:7d:74 (tux4), endereço IP de origem 172.16.30.254 (tux4), endereço MAC de destino 00:0f:fe:8b:e4:4d (tux1), endereço IP de destino 172.16.30.1 (tux1). Consultar logs na figura EXP1.4.2.

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

Ao examinar o *Ethernet Header* de um pacote conseguimos determinar o tipo de trama. Se o valor for 0x0806, é uma trama ARP. Caso o valor seja 0x0800, então será uma trama IP e se o *IP Header* for 1 então o protocolo é ICMP. Consultar logs figura EXP1.5.1 e EXP 1.5.2.

6) **Como determinar o comprimento de uma trama recetora?**

Podemos determina o comprimento de uma trama recetora inspecionando-a no **wireshark**. Consultar logs figura EXP1.5.1.

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

A interface *loopback* é uma interface virtual de rede a um computador que permite enviar e receber respostas para/de si mesmo. Serve para testar se a rede do computador está configurada corretamente. Consultar logs figura EXP1.7.

Experiência 2 – Implementar duas LAN's virtuais no switch

Nesta experiência implementaram-se duas LAN's virtuais (VLANY0, VLANY1). Enquanto que o tux1 e o tux4 foram associados à mesma LAN (VLANY0) o tux2 já foi associado a outra (VLANY1).

1) **Como configurar VLANY0?**

Para configurar a VLANY0 é necessário aceder à consola do *switch* num tux. Para isso é necessário ligar a porta "Switch Console" à porta T4 e a porta T3 à porta S0 do tux pretendido. Posteriormente é necessário aceder ao **GTKTerm** do tux pretendido e dar login no switch com os seguintes comandos:

- enable
- Password: 8nortel

Posteriormente é necessário introduzir os seguintes comandos para adicionar a VLANY0:

- configure terminal
- vlan Y0
- end

Para adicionar as portas do tux1 e do tux4 são precisos os seguintes comandos:

- configure terminal
- interface fastethernet 0/[NºPorta]
- switchport mode access
- switchport access vlan Y0
- end

Por fim, é necessário conectar as portas introduzidas no switch com as portas E0 do tux1 e do tux4.

2) **Quantos domínios de transmissão existem? O que se pode concluir a partir dos registos?**

Existem dois domínios de transmissão. Podemos verificar através da execução de *ping* ao endereço de *broadcast* da **VLANY0** no **tux1**. Este só obtém resposta de um computador: o que está na mesma rede – **tux4**. O **tux2** não responde pois está noutra rede e por isso noutro domínio de transmissão.

Experiência 3 – Configurar um router em Linux

Nesta experiência foi configurado o tux4 como um router que depois foi usado para estabelecer a ligação entre as duas VLANs criadas anteriormente.

1) **Que rotas há nos tuxes? Qual o seu significado destas?**

Rotas do **tux1**:

Destination: 172.16.30.0 *Gateway*: 0.0.0.0 *Iface*: eth0

O tux1 está ligado diretamente ao domínio 172.16.30.0 pela porta eth0.

Destination: 172.16.31.0 *Gateway*: 172.16.30.254 *Iface*: eth0

O tux1 está ligado ao domínio 172.16.31.0 pelo endereço 172.16.30.254 (tux4) pela porta eth0.

Rotas do **tux2**:

Destination: 172.16.31.0 *Gateway*: 0.0.0.0 *Iface*: eth0

O tux2 está ligado diretamente ao domínio 172.16.31.0 pela porta eth0.

Destination: 172.16.30.0 *Gateway*: 172.16.31.253 *Iface*: eth0

O tux2 está ligado ao domínio 172.16.30.0 pelo endereço 172.16.31.253 (tux4) pela porta eth0.

Rotas do **tux4**:

Destination: 172.16.30.0 *Gateway*: 0.0.0.0 *Iface*: eth0

O tux4 está ligado diretamente ao domínio 172.16.30.0 pela porta eth0.

Destination: 172.16.31.0 *Gateway*: 0.0.0.0 *Iface*: eth2

O tux4 está ligado diretamente ao domínio 172.16.31.0 pela porta eth2.

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

Destination – Destino da rota.

Gateway – Próximo endereço da rota.

Genmask - Aplica a máscara ao *Destination* para determinar a rede de destino.

Flags – Informação sobre a rota.

Metric – Custo da rota

Ref – Número de referências da rota.

Use – Contador de pesquisas na rota.

Interface – Placa de rede responsável.

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

As mensagens ARP e os endereços MAC associados são semelhantes às da pergunta 2 da experiência 1. Consultar logs figura EXP3.3.1 e EXP3.3.2.

4) Que pacotes ICMP são observados e porquê? Quais são os endereços associados e porquê?

Os pacotes ICMP observados são pacotes *request* e pacotes *reply*. Os endereços de origem e de destino são os endereços do **tux1**(172.16.30.1) e do **tux2**(172.16.31.1) nos pacotes *request* e os endereços do **tux2** e do **tux1** nos pacotes reply, respetivamente. Consultar logs figura EXP3.4.

Experiência 4 – Configurar um router comercial e implementar o NAT

Nesta experiência foi configurado um router comercial com NAT permitindo assim o acesso à internet aos computadores da rede.

1) Como se configura um router estático num router comercial?

Para configurar o router é necessário executar o mesmo processo que para configurar o switch (experiência 2 pergunta 1) mas substituir a porta "Switch Console" por "Router Console" e introduzir os comandos para dar login:

- User: root
- Password: 8nortel

Depois os seguintes comandos para configurar o router:

- 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 durante a experiência? Explique.

tux1 → tux2: 172.16.30.1(tux1) → 172.16.30.254(tux4) → 172.16.31.253(tux4) → 172.16.31.1(tux2)

tux2 → tux1: 172.16.31.1(tux2) → 172.16.31.254(router) → 172.16.31.253(tux4) → 172.16.30.254(tux4) → 172.16.30.1(tux1)

tux1 → Router Lab: 172.16.30.1(tux1) → 172.16.30.254(tux4) → 172.16.31.253(tux4) → 172.16.31.1(router) → 172.16.1.39(router) → 172.16.1.254(router lab)

Router Lab → tux1: 172.16.1.254(router lab) → 172.16.1.39(router) → 172.16.31.1(router) → 172.16.31.253(tux4) → 172.16.30.254(tux4) → 172.16.30.1(tux1)

3) Como se configura o NAT num router comercial?

Para configurar o NAT num router comercial é necessário introduzir os seguintes comandos na consola do router:

- ip nat pool ovrld 172.16.1.Y9 172.16.1.Y9 prefix 24
- ip nat inside source list 1 pool ovrld 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
- end

4) O que faz o NAT?

O NAT é uma técnica que consiste em reescrever, usando uma hash table, os endereços IP de origem de um pacote que passam por um router de maneira a que um computador de uma rede interna tenha acesso ao exterior.

Experiência 5 - DNS

Nesta experiência foi necessário configurar o DNS (*Domain Name System*) para os tuxes em uso. O servidor de DNS "**services.netlab.fe.up.pt**" é usado para traduzir os hostnames para os seus respetivos endereços de IP.

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

Para configurar o serviço DNS num *host* é necessário alterar o ficheiro **resolv.conf** em **/etc** nesse host. O ficheiro deverá conter:

» search *hostname*

» nameserver *ipaddress*

Em que *ipaddress* e *hostname* são, respetivamente, o ip e o nome de um servidor de DNS. Para o nosso caso hostname = netlab.fe.up.pt e ipaddress = 172.16.1.1.

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

São trocados dois pacotes entre o *host* e o *server*. O primeiro é enviado pelo *host* contendo o hostname desejado e pedindo o seu endereço IP. O segundo é a resposta do servidor e contém o endereço IP do hostname.

Experiência 6 – Conexões TCP

Nesta experiência foi observado o comportamento do protocolo TCP utilizando para isso a aplicação desenvolvida para a transferência de um ficheiro.

1) **Quantas conexões TCP foram abertas pela aplicação FTP?**

A aplicação FTP abre duas conexões TCP. A primeira para comunicar com o servidor e a segunda para receber o ficheiro.

2) **Em que conexão é transportado o controlo de informação?**

O controlo da informação é transportado na primeira conexão TCP, responsável pela comunicação entre o cliente e o servidor e pelo envio/receção de comandos.

3) **Quais as fases da conexão TCP?**

- *connection establishment*
- *data transfer*
- *connection termination*

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

ARQ (Automatic Repeat Request) é um método de controlo de erro para a transmissão de dados. **TCP** (Transmission Control Protocol) utiliza o **ARQ** com o método de janela deslizante. Para isso, utiliza *acknowledgment numbers* enviados pelo recetor que indicam que a trama foi recebida corretamente, *window size* que indica a gama de pacotes que o emissor pode enviar e o *sequnce number*, número do pacote enviado. Quando o recetor recebe corretamente um pacote, envia um *aknowledgment* para informar o emissor. Caso este *aknowledgent* não chegue antes do *timeout*, o pacote será reenviado. Consultar logs figura EXP6.4.

5) **Como é que o mecanismo de controlo de congestão TCP funciona? Como é que o fluxo de dados da conexão evoluiu ao longo do tempo? Está de acordo com o mecanismo de controlo de congestão TCP?**

O mecanismo de controlo de congestão TCP estima quantos pacotes consegue transmitir e guarda essa estativa numa janela de congestão, tentando não sair dessa janela.

Consultando figura EXP6.5. Podemos verificar que quando acaba um download e começa outro (aos ~62 segundos) acontece um pico, procedido de uma tentativa de estabilizar o número de packets por segundo, reduzindo o rate de download.

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

Consultar figura EXP6.6. Foi iniciado um download no **tux1** e, dos 50 segundos aos 80 segundos foi feito um download dum ficheiro mais pequeno no **tux2**. Como é possível verificar, nesse intervalo de tempo a quantidade média de pacotes transmitidos por tick diminuiu, visto que a quantidade de pacotes transmitidos na rede foi dividida pelos dois downloads. Dado que a internet estava bastante instável aquando da experiência, a quantidade de pacotes transmitidos por tick varia bastante, mas verificando a quantidade média num intervalo de tempo maior, conseguimos verificar que desceu no intervalo de tempo em que estavam a ser executados os dois downloads.

Conclusões

O tema deste trabalho é a configuração de uma rede e a implementação de uma aplicação que com o auxílio da rede configurada e de um servidor da FEUP executa a transferência de um ficheiro fornecido.

Em suma, como o trabalho foi concluído com êxito, foram alcançados os objetivos de aprendizagem propostos.

Referências:

https://www.ietf.org/rfc/rfc959.txt

https://searchnetworking.techtarget.com/definition/Address-Resolution-Protocol-ARP

Anexos:

Imagens:

```
rcom-write@rcomwrite-VirtualBox:~/Desktop/RCOM/TP2$ ./download ftp://anonymous:a@ftp.up.pt/pub/sage/index.html
220-Welcome to the University of Porto's mirror archive (mirrors.up.pt)
220----------------------------------------------------------------------
220-
220-All connections and transfers are logged. The max number of connections is 200.
220-
220-For more information please visit our website: http://mirrors.up.pt/
220-Questions and comments can be sent to mirrors@uporto.pt
220-
220-
220
Connected to ftp.up.pt
331 Please specify the password.
230 Login successful.
227 Entering Passive Mode (193,137,29,15,222,5)
150 Opening BINARY mode data connection for pub/sage/index.html (6114 bytes).
226 Transfer complete.
221 Goodbye.
```

Figura AP1

```
rcom-write@rcomwrite-VirtualBox:~/Desktop/RCOM/TP2$ ./download ftp://ftp.up.pt/pub/sage/index.html
220-Welcome to the University of Porto's mirror archive (mirrors.up.pt)
220----------------------------------------------------------------------
220-
220-All connections and transfers are logged. The max number of connections is 200.
220-
220-For more information please visit our website: http://mirrors.up.pt/
220-Questions and comments can be sent to mirrors@uporto.pt
220-
220-
220
Connected to ftp.up.pt
User: anonymous
331 Please specify the password.
Password:
230 Login successful.
227 Entering Passive Mode (193,137,29,15,212,204)
150 Opening BINARY mode data connection for pub/sage/index.html (6114 bytes).
226 Transfer complete.
221 Goodbye.
```

Figura AP2

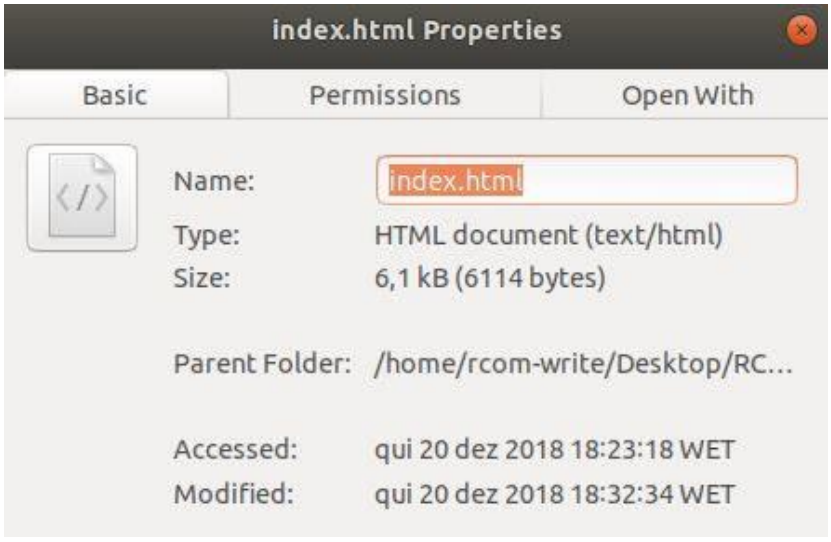
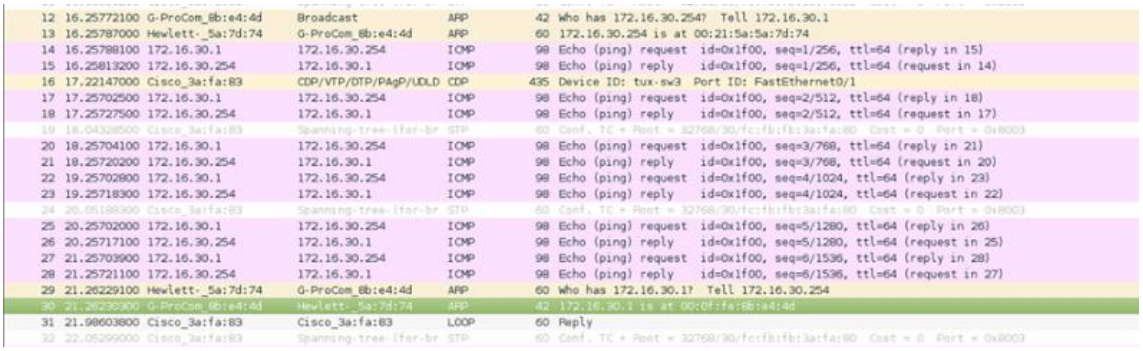
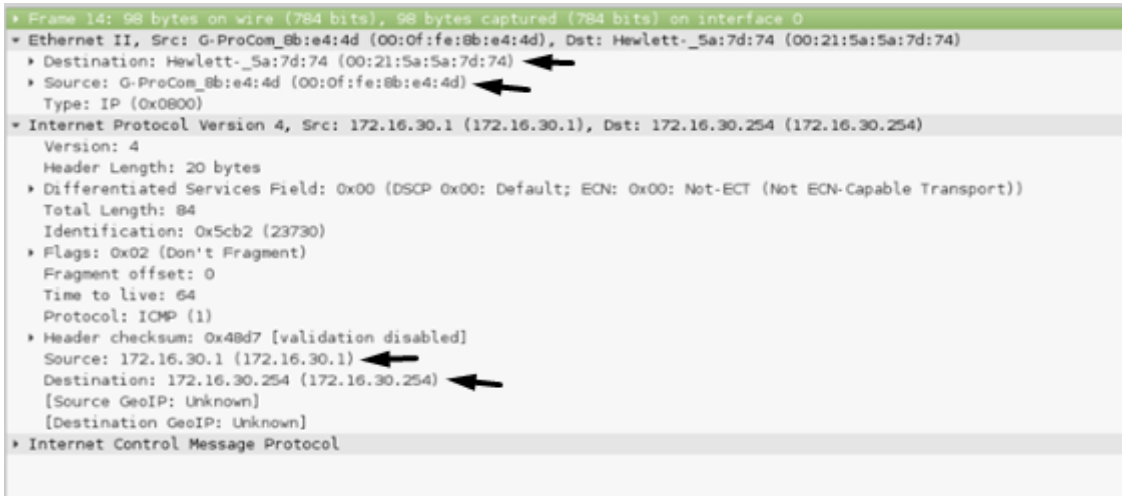


Figura AP3



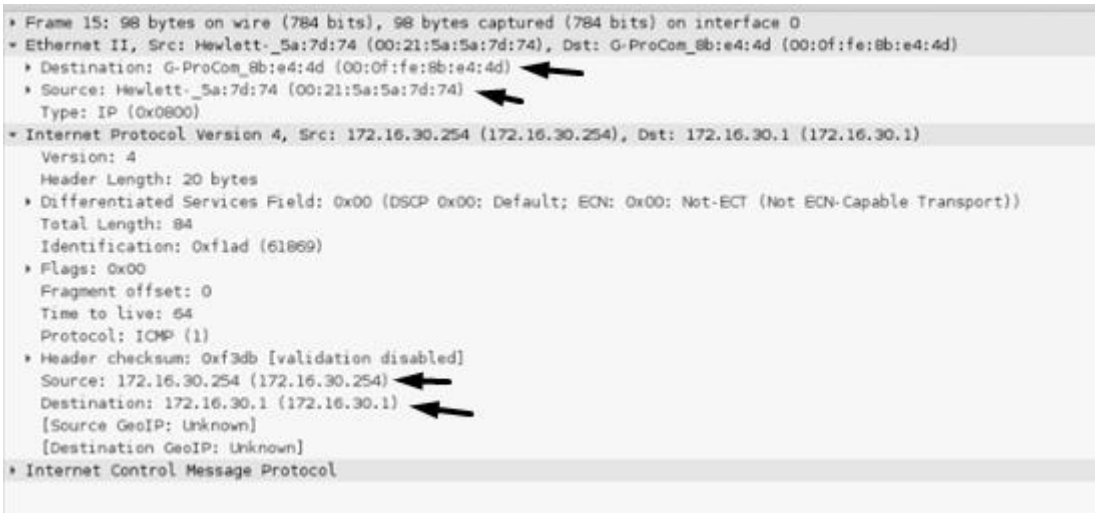
12	16.25772100	G-ProCom_8b:e4:4d	Broadcast	ARP	42 Who has 172.16.30.254? Tell 172.16.30.1
13	16.25787000	Hewlett-_5a:7d:74	G-ProCom_8b:e4:4d	ARP	60 172.16.30.254 is at 00:21:5a:5a:7d:74
14	16.25788100	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0x1f00, seq=1/256, ttl=64 (reply in 15)
15	16.25813200	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1f00, seq=1/256, ttl=64 (request in 14)
16	17.22147000	Cisco_3a:fa:83	CDP/VTP/DTP/PAgP/UDLD	CDP	435 Device ID: tux-sw3 Port ID: FastEthernet0/1
17	17.25702500	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0x1f00, seq=2/512, ttl=64 (reply in 18)
18	17.25727500	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1f00, seq=2/512, ttl=64 (request in 17)
19	18.04328500	Cisco_3a:fa:83	Spanning-tree-(for-br	STP	60 Conf. TC + Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
20	18.25704100	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0x1f00, seq=3/768, ttl=64 (reply in 21)
21	18.25720200	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1f00, seq=3/768, ttl=64 (request in 20)
22	19.25702800	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0x1f00, seq=4/1024, ttl=64 (reply in 23)
23	19.25718300	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1f00, seq=4/1024, ttl=64 (request in 22)
24	20.05188300	Cisco_3a:fa:83	Spanning-tree-(for-br	STP	60 Conf. TC + Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
25	20.25702000	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0x1f00, seq=5/1280, ttl=64 (reply in 26)
26	20.25717100	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1f00, seq=5/1280, ttl=64 (request in 25)
27	21.25703900	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0x1f00, seq=6/1536, ttl=64 (reply in 28)
28	21.25721100	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1f00, seq=6/1536, ttl=64 (request in 27)
29	21.26229100	Hewlett-_5a:7d:74	G-ProCom_8b:e4:4d	ARP	60 Who has 172.16.30.1? Tell 172.16.30.254
30	21.26230300	G-ProCom_8b:e4:4d	Hewlett-_5a:7d:74	ARP	42 172.16.30.1 is at 00:0f:fe:8b:e4:4d
31	21.90603800	Cisco_3a:fa:83	Cisco_3a:fa:83	LOOP	60 Reply
32	22.05299000	Cisco_3a:fa:83	Spanning-tree-(for-br	STP	60 Conf. TC + Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003

Figura EXP1.2



```
▸ Frame 14: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
▾ Ethernet II, Src: G-ProCom_8b:e4:4d (00:0f:fe:8b:e4:4d), Dst: Hewlett-_5a:7d:74 (00:21:5a:5a:7d:74)
  ▸ Destination: Hewlett-_5a:7d:74 (00:21:5a:5a:7d:74)
  ▸ Source: G-ProCom_8b:e4:4d (00:0f:fe:8b:e4:4d)
    Type: IP (0x0800)
▾ Internet Protocol Version 4, Src: 172.16.30.1 (172.16.30.1), Dst: 172.16.30.254 (172.16.30.254)
    Version: 4
    Header Length: 20 bytes
  ▸ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 84
    Identification: 0x5cb2 (23730)
  ▸ Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)
  ▸ Header checksum: 0x48d7 [validation disabled]
    Source: 172.16.30.1 (172.16.30.1)
    Destination: 172.16.30.254 (172.16.30.254)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
▸ Internet Control Message Protocol
```

Figura EXP1.4.1



```
▸ Frame 15: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
▾ Ethernet II, Src: Hewlett-_5a:7d:74 (00:21:5a:5a:7d:74), Dst: G-ProCom_8b:e4:4d (00:0f:fe:8b:e4:4d)
  ▸ Destination: G-ProCom_8b:e4:4d (00:0f:fe:8b:e4:4d)
  ▸ Source: Hewlett-_5a:7d:74 (00:21:5a:5a:7d:74)
    Type: IP (0x0800)
▾ Internet Protocol Version 4, Src: 172.16.30.254 (172.16.30.254), Dst: 172.16.30.1 (172.16.30.1)
    Version: 4
    Header Length: 20 bytes
  ▸ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 84
    Identification: 0xf1ad (61869)
  ▸ Flags: 0x00
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)
  ▸ Header checksum: 0xf3db [validation disabled]
    Source: 172.16.30.254 (172.16.30.254)
    Destination: 172.16.30.1 (172.16.30.1)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
▸ Internet Control Message Protocol
```

Figura EXP1.4.2

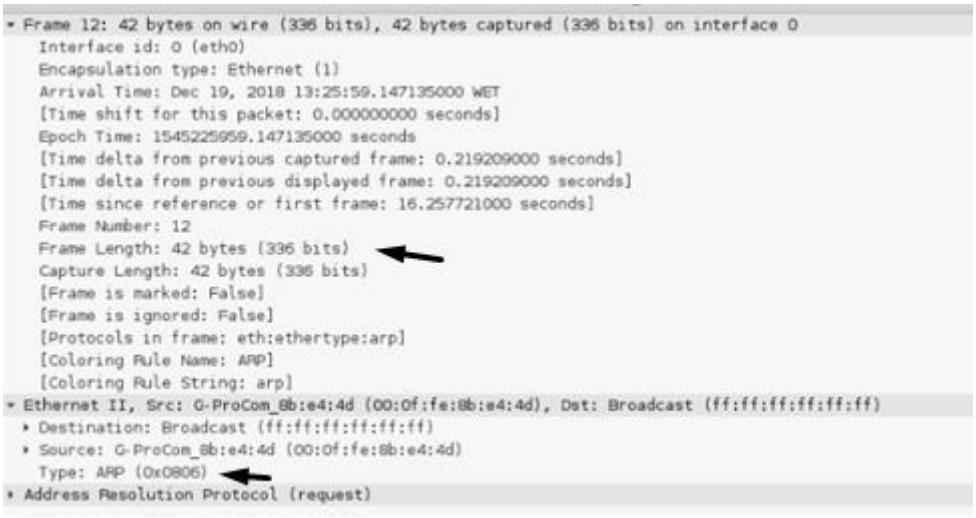


Figura EXP1.5.1

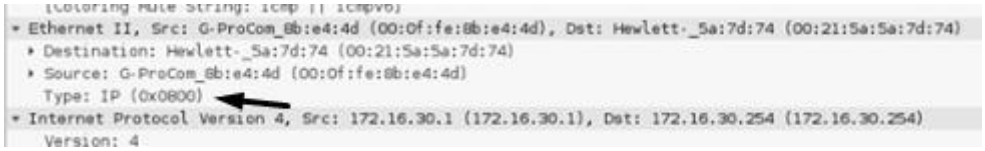


Figura EXP1.5.2

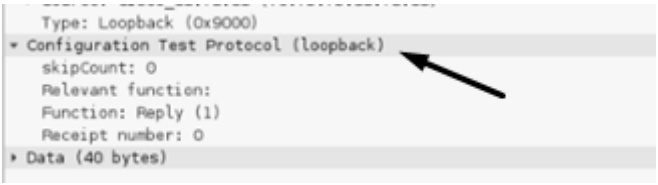


Figura EXP1.7

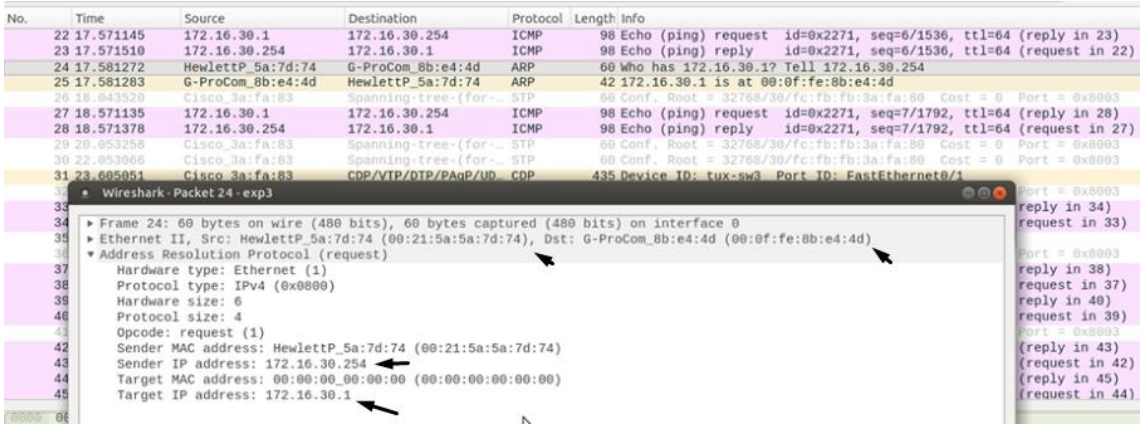


Figura EXP3.3.1

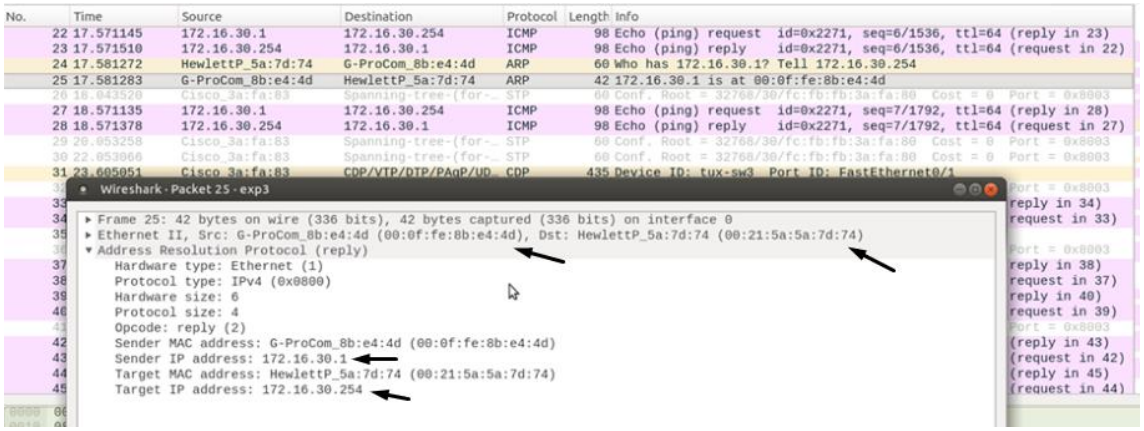


Figura EXP3.3.2

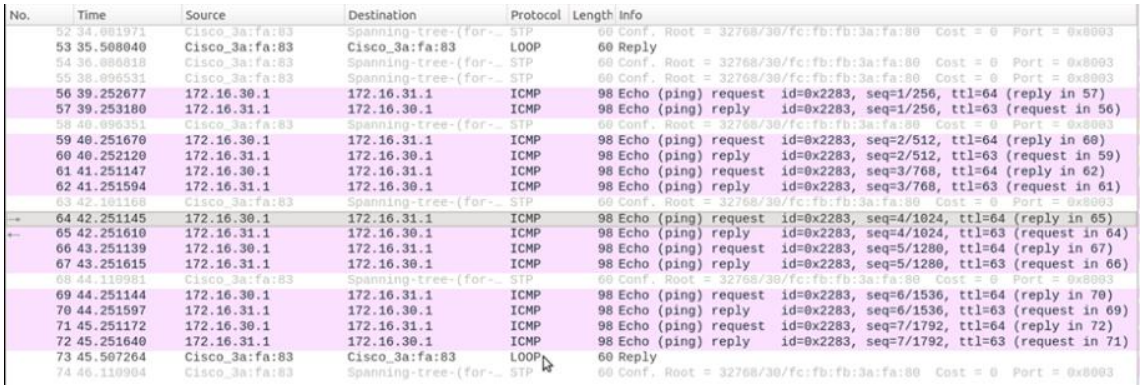


Figura EXP3.4

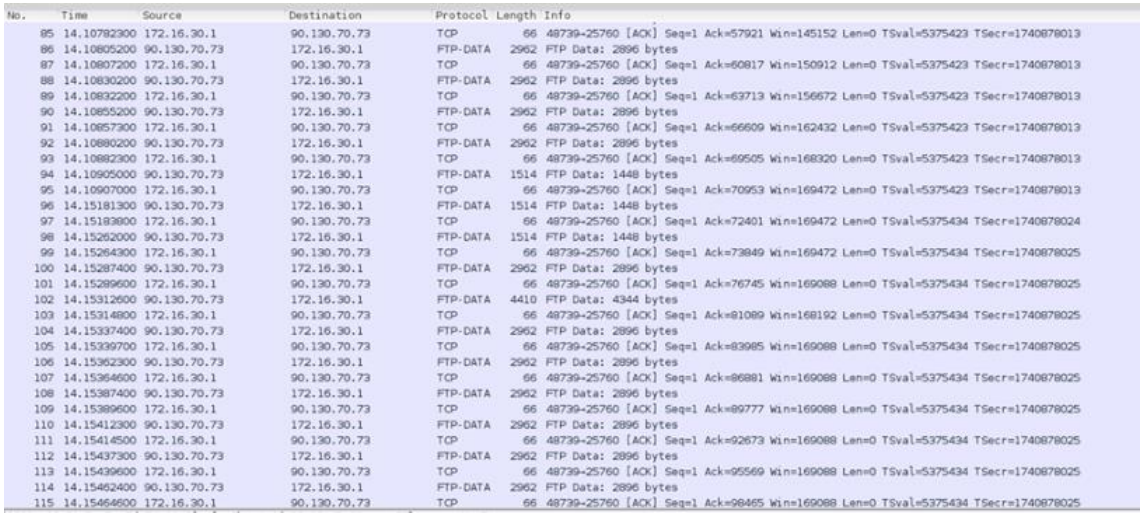


Figura EXP6.4

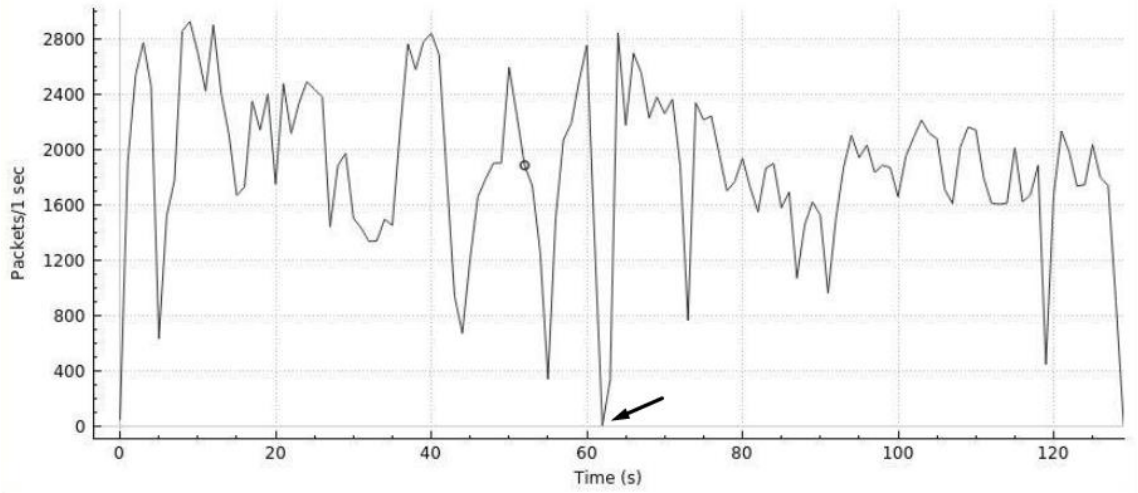


Figura EXP6.5

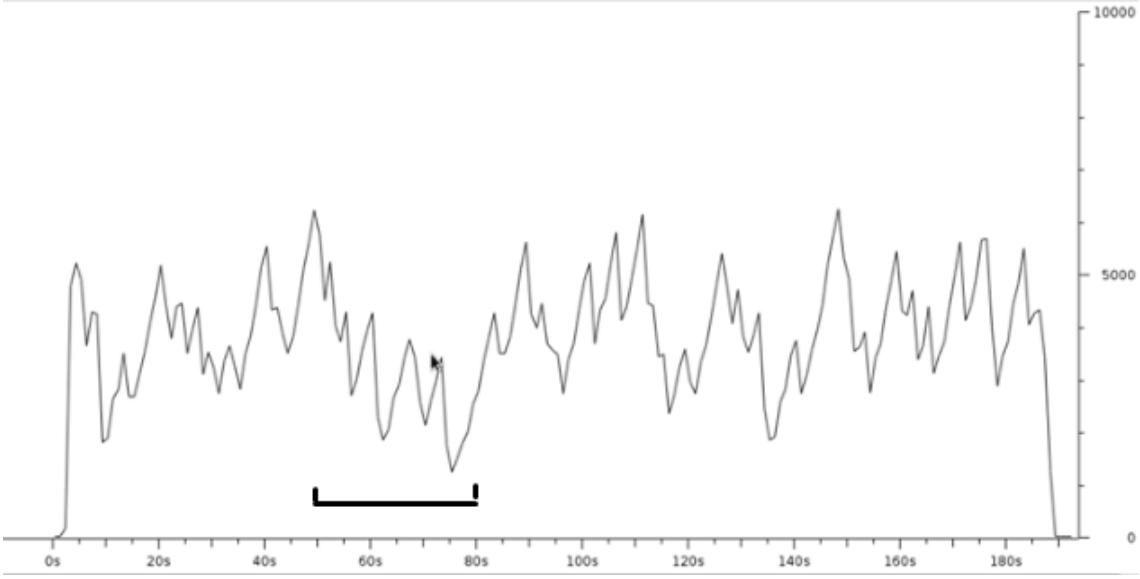


Figura EXP6.6

Código:

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

#define SERVER_PORT 21

#define CONNECTION_OK "220"
#define SEND_PASS "331"
#define LOGGED_IN "230"
#define START_TRANSFER "150"
#define TRANSFER_COMPLETE "226"
#define GOODBYE "221"

#define READ_START 0
#define READ_LINE 1
#define READ_LINES 2
#define READ_END 3
#define READ_IGNORE 4
#define READ_PORT 5

ssize_t getpassword(char **lineptr, size_t *n, FILE *stream)
{
    struct termios old, new;
    int nread;

    /* Turn echoing off and fail if we can't. */
    if (tcgetattr (fileno (stream), &old) != 0)
        return -1;
    new = old;
    new.c_lflag &= ~ECHO;
    if (tcsetattr (fileno (stream), TCSAFLUSH, &new) != 0)
        return -1;

    /* Read the passphrase */
    nread = getline (lineptr, n, stream);
```

```
    /* Restore terminal. */
    (void) tcsetattr (fileno (stream), TCSAFLUSH, &old);

    return nread;
}

void errorUsage()
{
    printf("Usage: download ftp://[<user>:<password>@]<host>/<url-
path>\n");
    exit(1);
}

void parseArguments(char *arg, char *user, char *password, char *host,
char *url_path, char *file_name)
{
    host[0] = '\0';
    url_path[0] = '\0';


    if(strncmp(arg, "ftp://", 6) != 0)
        errorUsage();

    char *start = arg + (6 * sizeof(char));

    char *div = strchr(start,'@');

    if(div == NULL)
    {
        user[0] = '\0';
        password[0] = '\0';

        div = strchr(start,'/');

        if(div == NULL)
            errorUsage();

        memcpy(host, start, (div - start) / sizeof(char));
        div++;

        memcpy(url_path, div, strlen(div));

        char *lastdiv;

        while(div != NULL)
        {
            div++;
            lastdiv = div;
```

```
            div = strchr(div,'/');
        }

        memcpy(file_name, lastdiv, strlen(lastdiv));
    }

    else
    {
        div = strchr(start,':');

        if(div == NULL)
            errorUsage();

        memcpy(user, start, (div - start) / sizeof(char));
        div++;
        start = div;

        div = strchr(start,'@');

        if(div == NULL)
            errorUsage();

        memcpy(password, start, (div - start) / sizeof(char));
        div++;
        start = div;

        div = strchr(start,'/');

        if(div == NULL)
            errorUsage();

        memcpy(host, start, (div - start) / sizeof(char));
        div++;

        memcpy(url_path, div, strlen(div));

        char *lastdiv;

        while(div != NULL)
        {
            div++;
            lastdiv = div;
            div = strchr(div,'/');
        }

        memcpy(file_name, lastdiv, strlen(lastdiv));
    }
}
```

```
char *getIPaddress(char *host)
{
    struct hostent *h;

    if ((h=gethostbyname(host)) == NULL)
    {
        herror("gethostbyname");
        exit(1);
    }

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

int createSocket(char *ip, int port)
{
    int sockfd;
    struct  sockaddr_in server_addr;

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

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

    return sockfd;
}

void getServerReply(int socket, char *server_reply)
{
    char c;
    int i = 0;
    int state = READ_START;
```

```
    while (state != READ_END)
    {
        if (read(socket, &c, 1) != 1)
            continue;

        printf("%c", c);

        switch(state)
        {
            case READ_START:
                switch(c)
                {
                    case ' ':
                        state = READ_LINE;
                        i = 0;
                        break;
                    case '-':
                        state = READ_LINES;
                        i = 0;
                        break;
                    default:
                        if(c >= '0' && c <= '9')
                        {
                            server_reply[i] = c;
                            i++;
                        }
                        break;
                }
                break;
            case READ_LINE:
                if (c == '\n')
                    state = READ_END;
                break;
            case READ_LINES:
                if (c == server_reply[i])
                    i++;
                else if (i == 3)
                {
                    if (c == ' ')
                        state = READ_LINE;

                    else if(c == '-')
                        i = 0;
                }
                break;
            default: break;
        }
    }
}
```

```
int getPort(int socket)
{
    char c;
    int port = 0;
    char port_temp[4];
    port_temp[3] = '\0';
    int i = 0;
    int separators = 0;
    int state = READ_START;

    while (state != READ_END)
    {
        if (read(socket, &c, 1) != 1)
            continue;

        printf("%c", c);

        switch(state)
        {
            case READ_START:
                if (c == ' ')
                {
                    if (i != 3)
                        exit(1);

                    i = 0;
                    state = READ_IGNORE;
                }
                else i++;
                break;
            case READ_IGNORE:
                if (c == ',')
                    separators++;
                if (separators == 4)
                    state = READ_PORT;
                break;
            case READ_PORT:
                if (c == ',')
                {
                    port += atoi(port_temp) * 256;
                    port_temp[0] = '\0';
                    port_temp[1] = '\0';
                    port_temp[2] = '\0';
                    i = 0;
                }
                else if (c == ')')
                {
                    port += atoi(port_temp);
```

```
                    state = READ_END;
                }
                else
                {
                    port_temp[i] = c;
                    i++;
                }
                break;
            default: break;
        }
    }

    //Read .
    read(socket, &c, 1);
    printf("%c", c);

    return port;
}

void sendCommand(int socket, char *cmd, char *arg, char *response, int
*port)
{
    write(socket, cmd, strlen(cmd));
    if(arg != NULL)
    {
        write(socket, " ", 1);
        write(socket, arg, strlen(arg));
    }
    write(socket, "\n", 1);

    if(strcmp("pasv", cmd) == 0)
        *port = getPort(socket);

    else getServerReply(socket, response);
}

void downloadFile(int port_file, char *file_name)
{
    int file = open(file_name, O_WRONLY | O_CREAT | O_TRUNC, 0644);

    char data[1024];
    int res;

    while ((res = read(port_file, data, 1024)) > 0)
    write(file, data, res);

    close(file);
}
```

```
int main(int argc, char **argv)
{
    if (argc != 2)
        errorUsage();

    char *user = (char*) malloc(100);
    char *password = (char*) malloc(100);
    char *host = (char*) malloc(100);
    char *url_path = (char*) malloc(100);
    char *file_name = (char*) malloc(100);

    parseArguments(argv[1], user, password, host, url_path, file_name);

    char *ip = getIPaddress(host);
    int socket = createSocket(ip, SERVER_PORT);

    char server_reply[4];
    server_reply[3] = '\0';

    getServerReply(socket, server_reply);

    if (strcmp(CONNECTION_OK,server_reply) == 0)
        printf("Connected to %s\n", host);

    else
    {
        printf("Could not connect to %s\n", host);
        return 1;
    }

    if(user[0] == '\0')
    {
        printf("User: ");
        scanf("%s", user);
        getchar();
    }

    sendCommand(socket,"user",user,server_reply,NULL);

    if(strcmp(SEND_PASS,server_reply) != 0)
        return 1;

    if(password[0] == '\0')
    {
        printf("Password: ");
        size_t size = 100;
        getpassword(&password, &size, stdin);
        password[strlen(password)-1] = '\0';
        printf("\n");
```

```
    }

    sendCommand(socket,"pass",password,server_reply,NULL);

    if(strcmp(LOGGED_IN, server_reply) != 0)
        return 1;

    int port;
    sendCommand(socket,"pasv",NULL,NULL,&port);

    int port_file = createSocket(ip, port);

    sendCommand(socket,"retr",url_path,server_reply,NULL);

    if(strcmp(START_TRANSFER, server_reply) != 0)
        return 1;

    downloadFile(port_file, file_name);

    getServerReply(socket,server_reply);

    if(strcmp(TRANSFER_COMPLETE, server_reply) != 0)
        return 1;

    sendCommand(socket,"quit",NULL,server_reply,NULL);

    if(strcmp(GOODBYE, server_reply) != 0)
        return 1;

    close(port_file);
    close(socket);
    free(user);
    free(password);
    free(host);
    free(url_path);
    free(file_name);

    return 0;
}
```