2º Trabalho Laboratorial

Rede de computadores

Redes de computadores

3º Ano - MIEIC - 2020/21

Realizado por:

- Davide Castro - up201806512 Turma 3 – Grupo 2
- Ricardo Fontão - up201806317

Índice

Sumário

Este relatório foi realizado no âmbito do segundo trabalho laboratorial da unidade curricular de Redes de Computadores, intitulado de 'Rede de Computadores' e com base no desenvolvimento de um cliente de transferência de dados **FTP** (*File Transfer Protocol*), assim como a configuração de uma rede de computadores para a sua utilização.

O desenvolvimento deste trabalho proporcionou um alargamento do conhecimento de redes e de protocolos de transferência de dados.

Introdução

Este projeto foi realizado em duas partes principais: o desenvolvimento da aplicação de transferência de dados e a configuração e estudo de uma rede de computadores.

A aplicação foi desenvolvida tendo em conta o *File Transfer Protocol* baseado no RFC959 e permite a transferência de um ficheiro dado um URL.

Quanto à segunda parte, foi efetuado um processo iterativo com o percorrer das aulas laboratoriais através de experiências, com a configuração de uma rede de computadores no laboratório e o estudo dos seus aspetos principais, como a configuração de endereços IP, configuração de router e switch com VLAN's, DNS, implementação de NAT e, usando a aplicação desenvolvida, efetuar conexões TCP através da rede.

O relatório para este trabalho está assim dividido nestas partes:

- **Aplicação** de transferência de dados;
- **Configuração e estudo** da rede de computadores, obtendo as respostas às questões impostas sobre esta e analisando os resultados obtidos no laboratório em cada experiência;
- **Conclusões**

Nota: Os *logs* usados para análise nas experiências 1, 4, 5 e 6 são da autoria do grupo 3 da turma 3.

Parte 1 – Aplicação de *Download*

Arquitetura

A aplicação de *download* neste trabalho foi desenvolvida de acordo com as normas do RFC959 para o protocolo da transferência e usando a sintaxe de URL segundo o RFC1738.

Para o funcionamento da transferência de ficheiros, foi primeiramente necessário implementar o *parsing* dos dados fornecidos pelo utilizador. Para isto usamos uma função **parse_input**, que identifica e guarda os dados das credenciais, no caso em que existam, o *hostname* e o URL do ficheiro pretendido. Caso não sejam fornecidas credenciais é assumido o modo de utilizador anónimo. Todos estes dados são armazenados numa estrutura de dados que denominamos de **connection**. De seguida é obtido o endereço IP do *host* com a função **get_ip** usando o *hostname* fornecido.

Com todos os dados obtidos, é possível efetuar a conexão ao *host* pretendido. Para isto, usamos um *socket*, efetuando uma conexão TCP através da **porta 21**, fazendo uso das funções **socket** e **connect** e, assim, é iniciada a conexão ao servidor.

Recebendo o código 220 por parte do servidor, a conexão assume-se estabelecida e são enviados os dados do utilizador. São para isto usadas as funções **read_message** e **send_message**, que recebem uma mensagem do servidor e enviam uma mensagem, usando as funções **send** e **recv**, respetivamente. Se o servidor responder positivamente, com código iniciado por '3', neste caso o código 331, a *password* é enviada ao servidor e espera-se a resposta 230, indicando que o utilizador foi corretamente *logged in* no servidor.

Seguidamente, o cliente envia um pedido de modo passivo (comando **pasv**) e é esperado o código de resposta 227, junto com os dados necessários para calcular a porta para a nova conexão. O *parsing* desta resposta é efetuado pela função **parse_pasv_response**. Com esta porta inicia-se a nova conexão num novo socket. Esta conexão será usada para a transferência dos dados e é estabelecida usando a função **make_data_connection**, similar à função make_connection, exceto que usa a porta fornecida pela resposta do servidor. Já no modo passivo é então efetuado o envio do comando **retr** juntamente com o URL do ficheiro pretendido. Se for obtida uma resposta começada por '5', houve um erro no pedido, como por exemplo, o ficheiro indicado não existe, e o programa é terminado. Caso contrário é chamada a a função **read_data**, que vai receber os dados enviados pelo servidor e escrevê-los num ficheiro, transferindo assim o ficheiro indicado pelo URL fornecido através da conexão estabelecida.

Resultado de um *download* bem sucedido

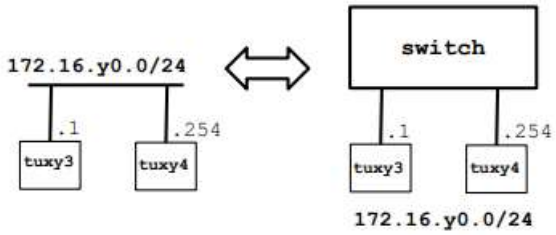
Após a execução de uma transferência com sucesso usando a aplicação desenvolvida, estes foram um dos resultados que obtemos:

```
./download ftp://ftp.up.pt/pub/debian/README
------------------
Connection Info:
Host name: ftp.up.pt
Host IP: 193.137.29.15
URL: pub/debian/README
Anonymous: TRUE
------------------
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
Sent 15 bytes: user anonymous

331 Please specify the password.
Sent 10 bytes: pass 1234

230 Login successful.
Sent 5 bytes: pasv

227 Entering Passive Mode (193,137,29,15,207,229)
Sent 23 bytes: retr pub/debian/README

150 Opening BINARY mode data connection for pub/debian/README (1190 bytes).
Finished reading file
```

Além deste exemplo, testamos a aplicação com diferentes *hosts*, ficheiros, com e sem credenciais, obtendo sempre um resultado positivo por parte da aplicação que efetuou sempre a transferência correta dos ficheiros.

Parte 2 – Configuração e análise da rede

Experiência 1



Objetivos

Ligar duas máquinas através de uma rede de computadores simples, configurando os endereços IP das mesmas e as rotas de forma a que consigam comunicar entre si.

Comandos de configuração

ifconfig eth0 172.16.30.1/24
ifconfig eth0 172.16.30.254/24

Questões

1 – O que são pacotes ARP e para que servem? Os pacotes ARP são *broadcasted* por um computador numa rede de forma a receber o endereço MAC correspondente a um endereço de IP.

2 - Quais são os endereços MAC e IP dos pacotes ARP e porquê? O pacote ARP do broadcast inclui o endereço IP e MAC do computador que fez o pedido e também o endereço IP do qual queremos saber o endereço MAC. O pacote de resposta inclui o endereço IP e MAC tanto do emissor como do recetor. No caso da nossa experiência quando pingamos do tux3 (IP: 172.16.30.1) para o tux4 (IP: 172.16.30.254), o tux3 faz um *broadcast* para saber qual o endereço MAC do tux4. Desta forma o tux4 responde diretamente ao tux3 com o seu endereço MAC, 00:21:5a:5a:79:97, como podemos ver pelos logs da experiência 1 (exp1).

3 - Que pacotes gera o comando *ping*? O comando ping gera pacotes do tipo ICMP.

4 - Quais são os endereços MAC e IP dos pacotes de ping? O ping do tux3 para o tux4 gera pacotes com endereços IP:

- IP Emissor: 172.16.30.1 (tux3)
- IP Recetor: 172.16.30.254 (tux4)

O endereço de MAC do emissor é o do tux3 e o de destino é o de tux4 descoberto a partir de um broadcast ARP. No nosso caso:

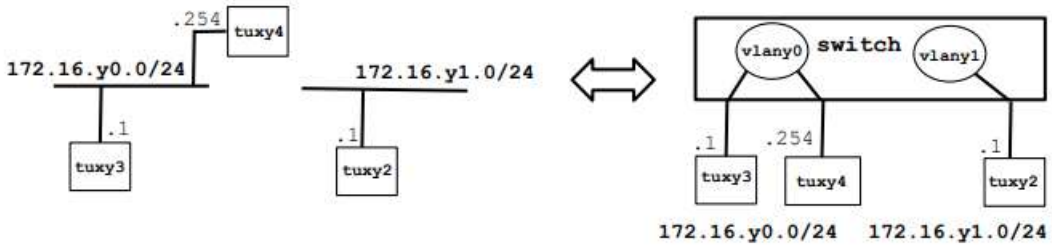
- MAC Emissor: 00:21:5a:61:2d:df (tux3)
- MAC Recetor: 00:21:5a:5a:79:97 (tux4)

Na reposta a este ping os endereços de emissor e recetor invertem-se. É possível observar estes dados na figura 1 dos anexos.

5 - Como determinar se um *frame* Ethernet recebido é ARP, IP ou ICMP?
Inspecionamos o *type* no cabeçalho do pacote. Se este valor for 0x0806 estamos perante um pacote ARP. No entanto, se o valor for 0x0800 estamos perante um pacote IP. Se este for o caso podemos inspecionar o cabelho do cabeçalho IP. Se o *protocol* estiver com o valor então estamos perante um pacote ICMP.

6- Como determinar o tamanho de um frame recebido? Verificando o campo EtherType no cabeçalho do pacote (Figura 2). Se neste estiver um valor inferior a 1500 este é o valor do tamanho em octetos. Se for maior que 1500 então indica o tipo do *frame* como foi referido na pergunta acima. Neste caso temos de inspecionar o cabeçalho do pacote encapsulado de forma a ver qual o seu tamanho. Por exemplo os pacotes Ipv4 contém um campo chamado *length*.

Experiência 2



Objetivos

Implementar duas VLANs virtuais num switch e configurá-las para os computadores usados

Comandos de configuração

configure terminal
vlan y0
end

configure terminal
vlan y1
end

configure terminal
interface fastethernet 0/3
switchport mode access
switchport por access vlan y0
end

configure terminal
interface fastethernet 0/4
switchport mode access
switchport por access vlan y0
end

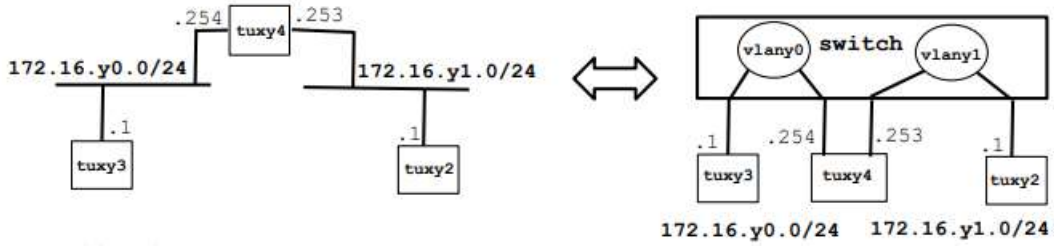
configure terminal
interface fastethernet 0/2
switchport mode access
switchport por access vlan y1
end

Questões

1 – Como configurar vlany0? Para configurar a vlany0 temos, em primeiro lugar, que criar a VLAN no *switch* (primeiras 3 linhas nos comandos acima). De seguida é necessário adicionar as portas 3 e 4 do *switch* à VLAN, que estarão ligadas às portas E0 do tuxy3 e tuxy4, respetivamente. Os comandos têm que ser inseridos no GTKTerm com a porta S0 do tux a usar e a porta do switch na régua 2 ligadas às portas 23 e 24 na régua 1 (RS232 – Cisco Adapter).

2- Quantos domínios de *broadcast* existem? Como concluí-lo pelos *logs*? Existem dois domínios de *broadcast*. Isto pode ser concluido nos logs pelo facto de ao fazer **ping** em *broadcast* a partir da máquina 3 só alcança a máquina 4, ou seja, as duas VLANs têm domínios diferentes, um com o tux3 e tux4 e outro com tux2. (Figuras 3, 4 e 5)

Experiência 3



Objetivos

Transformar uma máquina, neste caso o tux34,num *router* e configurá-lo de forma a que as máquinas na vlan30 consigam comunicar com a vlan31 e vice-versa.

Comandos de configuração

tuxy4
route add -net 172.16.y0.0/24 gw 172.16.y0.254 dev eth0 //CUIDADO ROTAS A MAIS
route add -net 172.16.y1.0/24 gw 172.16.y1.253 dev eth1
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

tuxy3
route add -net 172.16.y1.0/24 gw 172.16.y0.254 dev eth0

tuxy2
route add -net 172.16.y0.0/24 gw 172.16.y1.253 dev eth0

Questões

1 – Quais *routes* existem nos tuxes? Qual o significado delas?
- Tuxy3:
  - Uma route para a sua própria sub-rede (172.16.30.0) que lhe permite comunicar diretamente com qualquer máquina nessa sub-rede através do interface eth0.
  - Uma route para a sub-rede 172.16.31.0 com gateway 172.16.30.254 que lhe permite comunicar com os computadores da vlan 1 através do tux4 que atua como um router.
- Tuxy4:
  - Uma route para a sub-rede 172.16.30.0 (vlan0) com gateway 172.16.30.254 que lhe permite comunicar com todas as máquinas presentes nesta vlan através do interface eth0.
  - Uma route para a sub-rede 172.16.31.0 (vlan1) gw 172.16.31.253 dev eth1 que lhe permite comunicar com todas as máquinas presentes nesta vlan através do interface eth1.
-Tuxy2
  - Uma route para a sua própria sub-rede (172.16.31.0) que lhe permite comunicar diretamente com qualquer máquina nessa sub-rede através do interface eth0.

- Uma route para a sub-rede 172.16.30.0 com gateway 172.16.31.253 que lhe permite comunicar com os computadores da vlan 0 através do tux4 que atua como um router.

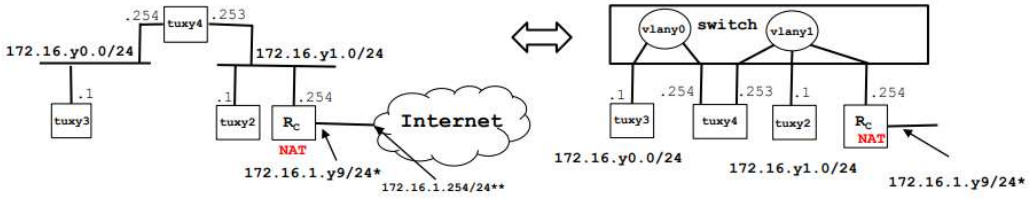
2- Que informação contém uma entrada na forwarding table? Destination: Tem formato networkAddress/mask; Gateway: Local para onde encaminhar o pacote; Interface: interface para comunicar com o gateway.

3 - Que mensagens ARP, e endereços MAC associados, são observados e porquê? Relativamente ao caso em que se faz um ping do tux3 para o tux2 e captura de logs no tux4: No eth0, o tux3 faz um pedido ARP pelo endereço MAC do tux4 que é respondido. De seguida começa o ping. No interface eth1, o tux4 faz um pedido ARP pelo endereço MAC do tux2. Este é respondido e o ping é "entregue" ao tux2. (Figuras 6 e 7)

4 - Que pacotes IMCP são observados e porquê? Os pacotes ICMP observados são os pacotes enviados do tux3 para o tux2 e os pacotes de resposta do tux2 para o tux3.

5 – Quais os endereços IP e MAC associados aos pacotes ICMP e porquê? Os endere;os MAC e IP de destino são os do tux 2 e os de origem são os de tux3 visto que são estes os computadores envolvidos no ping.

Experiência 4



Objetivos

Configurar um *router* comercial e implementar NAT para permitir o acesso à internet a partir de todos os computadores na rede.

Comandos de configuração

configure terminal
interface fastethernet 0/0
ip address 172.16.y1.254 255.255.255.0
no shutdown
ip nat inside
exit

interface fastethernet 0/1
ip address 172.16.2.y9 255.255.255.0
no shutdown
ip nat outside
exit

ip nat pool ovrld 172.16.2.y9 172.16.2.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 0.0.0.0 172.16.2.254
ip route 172.16.y0.0 255.255.255.0 172.16.y1.253

Os comandos devem ser inseridos no GTKTerm ligado ao router (o processo é idêntico a ligar ao switch, mas a porta 24 na régua 1 tem que ser ligada à porta do router na régua 2.

Questões

1 – Como configurar uma *route* estática num router comercial? Usando o comando ip route DestinationIP Mask GatewayIP. Isto é feito nesta experiência para definir a *route* para o exterior e interior no router nos comandos ip route 0.0.0.0 0.0.0.0 172.16.2.254 e ip route 172.16.y0.0 255.255.255.0 172.16.y1.253, sendo o primeiro para definir a rota default, neste caso para o exterior, e a rota com destino a 172.16.30.0, ou seja, para as máquinas da vlan30, com gateway pela máquina 4.

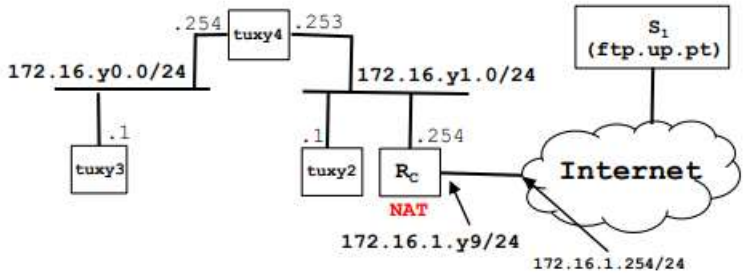
2 – Quais os caminhos percorridos pelos pacotes nas experiências e porquê?
Tux3 pinga Tux4 em 172.16.30.254:
172.16.30.1(Tux3) → 172.16.30.254(Tux4)
Tux3 pinga Tux4 em 172.16.31.253:
172.16.30.1(Tux3) → 172.16.30.254(Tux4)
Tux3 pinga Tux2 em 172.16.31.1:
172.16.30.1(Tux3) → 172.16.30.254(Tux4) → 172.16.31.1(Tux2)
Tux3 pinga router em 172.16.31.254:
172.16.30.1(Tux3) → 172.16.30.254(Tux4) → 172.16.31.254(router)
Tux2 pinga Tux3 sem default gateway para 172.16.30.0 e redirects desligados:
172.16.31.1(Tux2) → 172.16.31.254(router) → 172.16.31.253(Tux4) → 172.16.30.1(Tux3)
Tux2 pinga Tux3 sem default gateway para 172.16.30.0 e redirects ligados:
Primeiro pacote: 172.16.31.1(Tux2) → 172.16.31.254(router) → 172.16.31.253(Tux4) → 172.16.30.1(Tux3)
Pacotes seguintes: 172.16.31.1(Tux2) → 172.16.31.253(Tux4) → 172.16.30.1(Tux3)

Isto pode ser verificado nos logs: exp4_4_4, exp4_4_7 e exp4_step3.

3 - Como configurar NAT num router comercial? Para configurar a NAT é necessário definir os endereços para o interior e exterior, neste caso, 172.16.31.254 e 172.16.2.39, respetivamente, inserir também os comandos ip nat, também mostrados na secção acima.

4 - O que faz o NAT? A função do NAT é traduzir os endereços entre duas redes. Se fizermos um pedido de uma rede que esteja por trás de um router com NAT, esse router irá alterar esse pedido de forma a que o endereço de quem envia passe a ser um seu num certo port. Desta forma, a resposta será endereçada a si. Quando essa resposta chega, o router fará a tradução do endereço para o par IP-port da máquina original. Desta forma toda esta rede fica por trás de um mesmo endereço de IP.

Experiência 5



Objetivos

Configurar a DNS nas máquinas usadas para analisar o uso de um servidor DNS (Domain Name System), que faz a gestão dos *hostanames* de forma a obter um endereço.

Questões

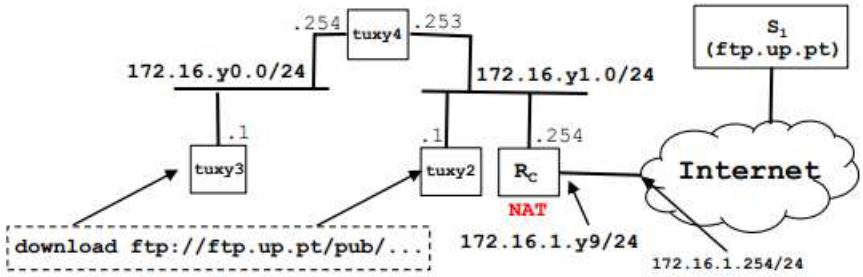
1 – Como configurar o serviço DNS num *host*? Para se configurar o DNS é necessário editar o ficheiro /etc/resolv.conf, alterando os campos ***nameserver*** e ***search*** para o nome e endereço do servidor pretendido. Como por exemplo, no caso desta experiência, o conteúdo do ficheiro nas máquinas era: search netlab.fe.up.pt
nameserver 172.16.2.1

2- Que pacotes são trocados pelo DNS e que informação é transportada? Para a experiência efetuou-se um *ping* a archlinux.org e observou-se que é enviado um pacote DNS contendo uma *query* pela DNS contendo o nome fornecido, o tipo (A e AAAA) e a *class*. De seguida é recebida uma resposta com a *query* juntamente com a *answer* contendo o endereço para o nome dado, neste caso, 138.201.81.199, o tipo, a *class* e também o *Time to Live*.

```
12 15.301861358  172.16.60.1   172.16.2.1    DNS   73 Standard query 0xc0f3 A archlinux.org
13 15.301870717  172.16.60.1   172.16.2.1    DNS   73 Standard query 0xe4fc AAAA archlinux.org
14 15.364709281  172.16.2.1    172.16.60.1   DNS  291 Standard query response 0xc0f3 A archlinux.org A 138.201.
15 15.367227612  172.16.2.1    172.16.60.1   DNS  331 Standard query response 0xe4fc AAAA archlinux.org AAAA 2a
```

(Mais detalhes nas figuras 8 e 9)

Experiência 6



Objetivos

Observar o comportamento da aplicação de *download* na rede configurada a partir da máquina 3, testando conexões TCP com o uso de vários servidores FTP para efetuar a transferência de ficheiros.

Questões

1 - Quantas conexões TCP são abertas pela sua aplicação FTP? São abertas 2 conexões. Uma ligação de controlo para a requisição do ficheiro e uma ligação de dados para a receção do ficheiro.

2 - Em que conexão é transportada a informação de controlo FTP? Na ligação de controlo.

3 - Quais são as fases da ligação TCP? As 3 fases de uma ligação TCP são a fase de estabelecimento de ligação, a fase de transferência de dados e a fase de terminação de ligação.

4 - Como funciona o mecanismo ARQ TCP? Quais são os campos TCP relevantes? Que informação relevante pode ser observada nos *logs*? O mecanismo ARQ TCP consiste no método da janela deslizante. Nos logs pode observar-se os pacotes TCP que contém os campos relevantes que são o *Sequence Number,* o *Acknowledgment Number* e o *Window Size*.

5 – Como funciona o mecanismo de controlo de congestão TCP? Quais os campos relevantes? Como evoluiu o *throughput* da conexão de dados ao longo do tempo? Segue o mecanismo de controlo de congestão TCP? O mecanismo de controlo de congestão TCP funciona usando uma janela de congestão que é ajustada baseada na quantidade de ACKs recebidos. O único campo relevante é o *Window Size.*

6 – O *throughput* de uma ligação TCP é alterado com o aparecimento de uma segunda ligação TCP? Como? Sim, com o aparecimento de uma segunda ligação, o *throughput é dividido igualmente pelas duas ligações.*

Conclusões

Com o desenvolvimento do segundo trabalho laboratorial, levando à elaboração deste relatório, é possível concluir que este foi bem sucedido e levou a uma mais vasta e aprofundada compreensão tanto dos contextos práticos como teóricos na base da unidade curricular de Redes de Computadores, em temas como os FTP, incluindo analisar RFCs, as ligações TCP e conceitos de rede/sub-rede, endereços IP e MAC, LAN, *switch*, configuração de routers e NAT. Consideramos que, em especial, as experiências realizadas no laboratório, contribuiram para este resultado positivo.

Anexos

```
15 17.023702174  172.16.60.1     172.16.60.254    ICMP   98 Echo (ping) request  id=0x0b58, seq=1/256, ttl=64 (reply in 16)
16 17.023837876  172.16.60.254    172.16.60.1      ICMP   98 Echo (ping) reply    id=0x0b58, seq=1/256, ttl=64 (request in 15)

Ethernet II, Src: HewlettP_61:2d:df (00:21:5a:61:2d:df), Dst: HewlettP_5a:79:97 (00:21:5a:5a:79:97)
  Destination: HewlettP_5a:79:97 (00:21:5a:5a:79:97)
  Source: HewlettP_61:2d:df (00:21:5a:61:2d:df)
  Type: IPv4 (0x0800)

Ethernet II, Src: HewlettP_61:2d:d
  Destination: HewlettP_5a:79:97 (
  Source: HewlettP_61:2d:df (00:21
  Type: IPv4 (0x0800)
```

Figura 1
(endereços IP e MAC do tuxy3 e tuxy4)

```
Ethernet II, Src: HewlettP_5a:79:97 (00:21:5a:5a:79:97), Dst: HewlettP_61:2d:df (00:21:5a:61:2d:df)
  Destination: HewlettP_61:2d:df (00:21:5a:61:2d:df)
  Source: HewlettP_5a:79:97 (00:21:5a:5a:79:97)
  Type: IPv4 (0x0800)
```

Figura 2
(campo EtherType)

```
 6 8.900331933   Cisco_b6:8c:03   Cisco_b6:8c:03   LOOP   60 Reply
 7 10.024420266  Cisco_b6:8c:03   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/31/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8003
 8 12.029327269  Cisco_b6:8c:03   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/31/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8003
 9 14.038218292  Cisco_b6:8c:03   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/31/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8003
10 16.039079116  Cisco_b6:8c:03   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/31/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8003
11 18.043989751  Cisco_b6:8c:03   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/31/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8003
12 18.903704792  Cisco_b6:8c:03   Cisco_b6:8c:03   LOOP   60 Reply
13 20.048930836  Cisco_b6:8c:03   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/31/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8003
14 22.053745788  Cisco_b6:8c:03   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/31/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8003
15 24.062630665  Cisco_b6:8c:03   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/31/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8003
16 26.063597439  Cisco_b6:8c:03   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/31/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8003
17 28.068424613  Cisco_b6:8c:03   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/31/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8003
18 28.907320769  Cisco_b6:8c:03   Cisco_b6:8c:03   LOOP   60 Reply
19 30.073287197  Cisco_b6:8c:03   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/31/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8003
20 32.078251819  Cisco_b6:8c:03   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/31/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8003
21 34.087208004  Cisco_b6:8c:03   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/31/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8003
22 36.087961831  Cisco_b6:8c:03   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/31/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8003
```

Figura 3 (tuxy2)

```
37 54.128424293  172.16.30.1      172.16.30.255    ICMP   98 Echo (ping) request  id=0x1715, seq=1/256, ttl=64 (no response found!)
38 54.245508317  Cisco_b6:8c:01   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/30/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8001
39 55.153977962  172.16.30.1      172.16.30.255    ICMP   98 Echo (ping) request  id=0x1715, seq=2/512, ttl=64 (no response found!)
40 56.177973052  172.16.30.1      172.16.30.255    ICMP   98 Echo (ping) request  id=0x1715, seq=3/768, ttl=64 (no response found!)
41 56.250702343  Cisco_b6:8c:01   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/30/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8001
42 57.201973311  172.16.30.1      172.16.30.255    ICMP   98 Echo (ping) request  id=0x1715, seq=4/1024, ttl=64 (no response found!)
43 58.225976364  172.16.30.1      172.16.30.255    ICMP   98 Echo (ping) request  id=0x1715, seq=5/1280, ttl=64 (no response found!)
44 58.259777869  Cisco_b6:8c:01   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/30/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8001
45 59.253978877  172.16.30.1      172.16.30.255    ICMP   98 Echo (ping) request  id=0x1715, seq=6/1536, ttl=64 (no response found!)
46 60.025429004  Cisco_b6:8c:01   Cisco_b6:8c:01   LOOP   60 Reply
47 60.260212281  Cisco_b6:8c:01   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/30/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8001
48 60.273961935  172.16.30.1      172.16.30.255    ICMP   98 Echo (ping) request  id=0x1715, seq=7/1792, ttl=64 (no response found!)
49 61.297977768  172.16.30.1      172.16.30.255    ICMP   98 Echo (ping) request  id=0x1715, seq=8/2048, ttl=64 (no response found!)
50 62.265094397  Cisco_b6:8c:01   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/30/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8001
```

Figura 4 (tuxy3)

```
28 39.980393744  172.16.30.1      172.16.30.255    ICMP   98 Echo (ping) request  id=0x1715, seq=1/256, ttl=64 (no response found!)
29 40.097570669  Cisco_b6:8c:02   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/30/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8002
30 41.005938168  172.16.30.1      172.16.30.255    ICMP   98 Echo (ping) request  id=0x1715, seq=2/512, ttl=64 (no response found!)
31 42.029930428  172.16.30.1      172.16.30.255    ICMP   98 Echo (ping) request  id=0x1715, seq=3/768, ttl=64 (no response found!)
32 42.102880179  Cisco_b6:8c:02   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/30/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8002
33 43.053915355  172.16.30.1      172.16.30.255    ICMP   98 Echo (ping) request  id=0x1715, seq=4/1024, ttl=64 (no response found!)
34 44.077909221  172.16.30.1      172.16.30.255    ICMP   98 Echo (ping) request  id=0x1715, seq=5/1280, ttl=64 (no response found!)
35 44.111820624  Cisco_b6:8c:02   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/30/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8002
36 45.105904953  172.16.30.1      172.16.30.255    ICMP   98 Echo (ping) request  id=0x1715, seq=6/1536, ttl=64 (no response found!)
37 45.877403704  Cisco_b6:8c:02   Cisco_b6:8c:02   LOOP   60 Reply
38 46.112310142  Cisco_b6:8c:02   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/30/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8002
39 46.125873418  172.16.30.1      172.16.30.255    ICMP   98 Echo (ping) request  id=0x1715, seq=7/1792, ttl=64 (no response found!)
40 47.149891729  172.16.30.1      172.16.30.255    ICMP   98 Echo (ping) request  id=0x1715, seq=8/2048, ttl=64 (no response found!)
41 48.117160791  Cisco_b6:8c:02   Spanning-tree-(for-…  STP   60 Conf. Root = 32768/30/00:1e:14:b6:8c:00  Cost = 0  Port = 0x8002
42 48.173875398  172.16.30.1      172.16.30.255    ICMP   98 Echo (ping) request  id=0x1715, seq=9/2304, ttl=64 (no response found!)
43 49.197865563  172.16.30.1      172.16.30.255    ICMP   98 Echo (ping) request  id=0x1715, seq=10/2560, ttl=64 (no response found!)
```

Figura 5 (tuxy4)

```
23 12.923155014  HewlettP_5a:74:3e   HewlettP_5a:7d:b7   ARP   42 Who has 172.16.30.1? Tell 172.16.30.254
24 12.923283384  HewlettP_5a:7d:b7   HewlettP_5a:74:3e   ARP   60 172.16.30.1 is at 00:21:5a:5a:7d:b7

6 7.765654590    HewlettP_5a:7d:b7   Broadcast           ARP   60 Who has 172.16.30.254? Tell 172.16.30.1
7 7.765683295    HewlettP_5a:74:3e   HewlettP_5a:7d:b7   ARP   42 172.16.30.254 is at 00:21:5a:5a:74:3e
```

Figura 6 (tuxy4 – eth0)

```
12 14.757755823  EncoreNe_b4:b8:94   Broadcast           ARP   42 Who has 172.16.31.1? Tell 172.16.31.253
13 14.757870782  HewlettP_61:24:01   EncoreNe_b4:b8:94   ARP   60 172.16.31.1 is at 00:21:5a:61:24:01
26 19.806335251  HewlettP_61:24:01   EncoreNe_b4:b8:94   ARP   60 Who has 172.16.31.253? Tell 172.16.31.1
27 19.806355924  EncoreNe_b4:b8:94   HewlettP_61:24:01   ARP   42 172.16.31.253 is at 00:e0:7d:b4:b8:94
```

Figura 7 (tuxy3 – eth1)

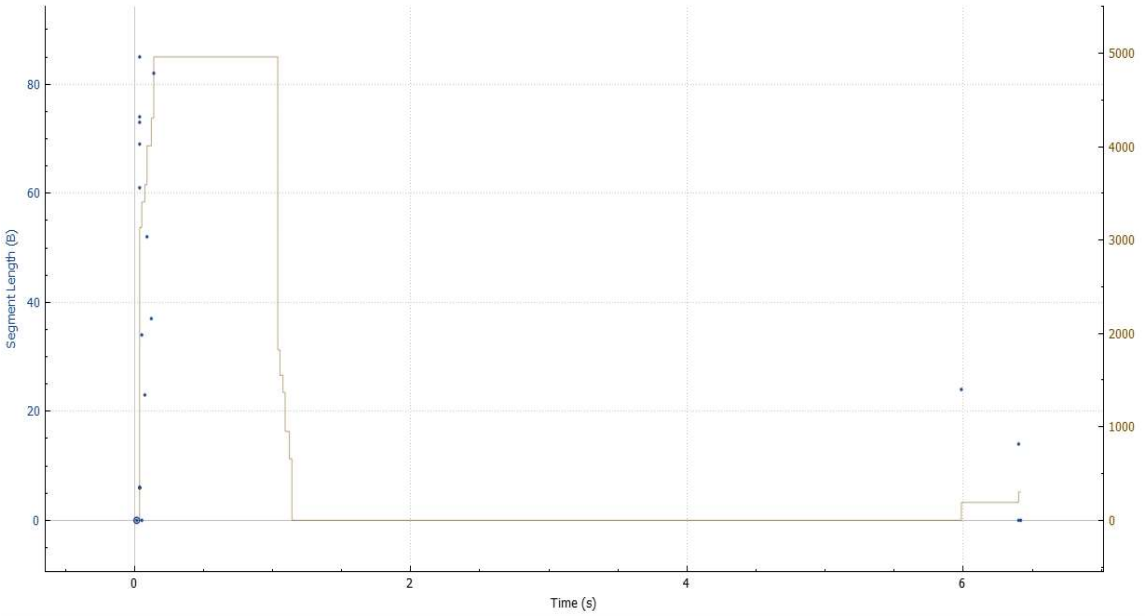
```
▾ Domain Name System (query)
    Transaction ID: 0xc0f3
  ▸ Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ▾ Queries
    ▸ archlinux.org: type A, class IN
    [Response In: 14]
```

Figura 8
(DNS query)

```
▾ Domain Name System (response)
    Transaction ID: 0xc0f3
  ▸ Flags: 0x8180 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 3
    Additional RRs: 5
  ▸ Queries
  ▾ Answers
    ▾ archlinux.org: type A, class IN, addr 138.201.81.199
        Name: archlinux.org
        Type: A (Host Address) (1)
        Class: IN (0x0001)
        Time to live: 600 (10 minutes)
        Data length: 4
        Address: 138.201.81.199
  ▸ Authoritative nameservers
  ▸ Additional records
    [Request In: 12]
    [Time: 0.062847923 seconds]
```

Figura 9
(DNS response)

Throughput graph



Código fonte

download.c

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

#define CONNECTION_PORT 21
#define ANONYMOUS_PASSWORD "1234"

typedef struct {
   char* host_name;
   char* url;
   char* host_ip;
   char* username;
   char* password;
   int anonymous;
   int sockfd;
   int data_port;
   int datafd;
} connection;

void init_connection(connection *conn) {
   conn->host_name = NULL;
   conn->url = NULL;
   conn->host_ip = NULL;
   conn->username = NULL;
   conn->password = NULL;
   conn->anonymous = -1;
   conn->sockfd = -1;
   conn->data_port = -1;
   conn->datafd = -1;
}

void cleanup(connection *conn){
   if(conn->host_name != NULL)
      free(conn->host_name);
   if(conn->url != NULL)
      free(conn->url);
   if(conn->host_ip != NULL)
      free(conn->host_ip);
   if(conn->username != NULL)
      free(conn->username);
   if(conn->password != NULL)
      free(conn->password);
```

```
   if(conn->sockfd != -1)
      close(conn->sockfd);
   if(conn->datafd != -1)
      close(conn->datafd);
}

void print_connection(connection* conn){
   printf("------------------\n");
   printf("Connection Info:\n");
   printf("Host name: %s\n", conn->host_name);
   printf("Host IP: %s\n", conn->host_ip);
   printf("URL: %s\n", conn->url);

   if(conn->anonymous){
      printf("Anonymous: TRUE\n");
   } else {
      printf("Username: %s\n", conn->username);
      printf("Password: %s\n", conn->password);
   }
   printf("------------------\n");
}


int get_ip(connection *conn){

   struct hostent *h;

   if ((h = gethostbyname(conn->host_name)) == NULL) {
      return -1;
   }

   conn->host_ip = (char*)malloc(16);

   //Maybe improve later
   strcpy(conn->host_ip, inet_ntoa(*((struct in_addr *)h->h_addr)));
   return 0;
}

int parse_input(connection *conn, char *input) {
   const char s_login[2] = "@";
   const char s[2] = "/";
   const char s_url[2] = "";

   char* token;
   char* inputptr;
   token = strtok_r(input, s, &inputptr);
   if(token == NULL)
        return -1;
   if(strcmp(token, "ftp:") != 0){
      fprintf(stderr, "Not ftp protocol\n");
      return -1;
```

```
}

token = strtok_r(NULL, s, &inputptr);
if(token == NULL)
   return -1;

char* auxptr;
char* aux = (char*)malloc((sizeof(char) * strlen(token)) + 1);
strcpy(aux, token);
char* aux1 = strtok_r(aux, s_login, &auxptr);
if(strcmp(token, aux1) == 0){
   conn->anonymous = 1;
} else
   conn->anonymous = 0;

free(aux);

if(conn->anonymous == 0) {
   char* aux1 = (char*)malloc((sizeof(char) * strlen(token)) + 1);
   strcpy(aux1, token);

   char* infoptr;
   char* info = strtok_r(token, s_login, &infoptr);

   if(info == NULL) {
      free(aux1);
      return -1;
   }

   // printf("%s\n", info);
   // printf("%s\n", token);
   if(strcmp(info, aux1) == 0){
      free(aux1);
      return -1;
   }
   free(aux1);

   char* aux2 = (char*)malloc((sizeof(char) * strlen(info)) + 1);
   strcpy(aux2, info);

   char* loginptr;
   char* loginInfo = strtok_r(info, ":", &loginptr);
   if(loginInfo == NULL) {
      free(aux2);
      return -1;
   }

   if(strcmp(loginInfo, aux2) == 0){
      free(aux2);
      return -1;
```

```
      }
      free(aux2);

      conn->username = (char*)malloc((sizeof(char) * strlen(loginInfo)) + 1);
      strcpy(conn->username, loginInfo);

      loginInfo = strtok_r(NULL, "", &loginptr);
      if(loginInfo == NULL)
         return -1;
      conn->password = (char*)malloc((sizeof(char) * strlen(loginInfo)) + 1);
      strcpy(conn->password, loginInfo);

      token = strtok_r(NULL, s, &infoptr);
      if(token == NULL)
         return -1;
   }

   //Not anonymous and end of anonymous
   conn->host_name = (char*)malloc((sizeof(char) * strlen(token)) + 1);
   strcpy(conn->host_name, token);

   token = strtok_r(NULL, s_url, &inputptr);
   if(token == NULL)
      return -1;
   conn->url = (char*)malloc((sizeof(char) * strlen(token)) + 1);
   strcpy(conn->url, token);

   return 0;
}

int parse_pasv_response(connection *conn, char* message){
   char* messageptr;
   strtok_r(message, ",", &messageptr);
   strtok_r(NULL, ",", &messageptr);
   strtok_r(NULL, ",", &messageptr);
   strtok_r(NULL, ",", &messageptr);

   char* l1_str = strtok_r(NULL, ",", &messageptr);
   char* l2_str = strtok_r(NULL, ",", &messageptr);

   int L1 = atoi(l1_str);
   int L2 = atoi(l2_str);

   conn->data_port = L1 * 256 + L2;

   // printf("%d\n", conn->data_port);
   return 0;
}

int make_connection(connection *conn){
```

```
   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->host_ip); /*32 bit Internet address network
byte ordered*/
   server_addr.sin_port = htons(CONNECTION_PORT);          /*server TCP port must be network
byte ordered */

   /*open an TCP socket*/
   if ((conn->sockfd = socket(AF_INET,SOCK_STREAM,0)) < 0) {
      fprintf(stderr, "Failed to open socket\n");
      return -1;
   }
   /*connect to the server*/
   if(connect(conn->sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0){
      fprintf(stderr, "Failed to connect to server\n");
      return -1;
   }

   return 0;
}

int make_data_connection(connection *conn){

   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->host_ip); /*32 bit Internet address network
byte ordered*/
   server_addr.sin_port = htons(conn->data_port);          /*server TCP port must be network
byte ordered */

   /*open an TCP socket*/
   if ((conn->datafd = socket(AF_INET,SOCK_STREAM,0)) < 0) {
      fprintf(stderr, "Failed to open socket\n");
      return -1;
   }
   /*connect to the server*/
   if(connect(conn->datafd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0){
      fprintf(stderr, "Failed to connect to server\n");
      return -1;
   }

   return 0;
}

int send_message(connection *conn, char* message){
   if(send(conn->sockfd, message, strlen(message), 0) < 0){
```

```
      return -1;
   }
   printf("Sent %ld bytes: %s\n", strlen(message), message);
   return 0;
}

int read_message(connection *conn, char* message){

   //TODO REFACTOR GETS STUCK ON FIRST MESSAGE SOMETIMES
   int read_bytes = -1;
   do {
      read_bytes = recv(conn->sockfd, message, 4096, 0);
      message[read_bytes] = '\0';
      if(read_bytes < 0){
         return -1;
      }
      // printf("Received %ld bytes: %s\n", strlen(message), message);
      // printf("Char: %c\n", message[3]);
      printf("%s", message);
   } while (read_bytes > 3 && message[3] == '-');

   return read_bytes;
}

int read_data(connection *conn){

   //Used to get the name of the file without its path
   char* urlptr;
   char* token = strtok_r(conn->url, "/", &urlptr);
   char* result;

   do {
      result = token;
      token = strtok_r(NULL, "/", &urlptr);
   } while(token != NULL);

   FILE* fd;
   if((fd = fopen(result, "w")) == NULL){
      fprintf(stderr, "Error opening file\n");
      return -1;
   }

   int i = 0;
   char c;
   int read_bytes = -1;
   char* message = (char*)malloc(sizeof(char) * 200);
   int cur_size = 200;
   while(1) {
      read_bytes = recv(conn->datafd, &c, 1, 0);
      if(read_bytes < 0)
         return -1;
```

```
      if(read_bytes == 0)
         break;
      if(i == cur_size){
         message = (char*)realloc(message, cur_size * 2);
         cur_size *= 2;
      }
      message[i] = c;
      i += read_bytes;
   }
   if(fwrite(message, 1, i, fd) < 0){
      perror("Error writing to file\n");
      return -1;
   }

   free(message);
   fclose(fd);
   printf("Finished reading file\n");
   return 0;

}

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

   if(argc != 2) {
      fprintf(stderr, "usage: download ftp://[<user>:<password>@]<host>/<url-path>\n");
      exit(-1);
   }

   connection conn;
   init_connection(&conn);

   if(parse_input(&conn, argv[1]) < 0){
      fprintf(stderr, "Error parsing input\n");
      fprintf(stderr, "usage: download ftp://[<user>:<password>@]<host>/<url-path>\n");
      cleanup(&conn);
      exit(-1);
   }

   if(get_ip(&conn) < 0){
      fprintf(stderr, "Failed to get host ip\n");
      cleanup(&conn);
      exit(-1);
   }

   print_connection(&conn);

   if(make_connection(&conn) < 0){
      fprintf(stderr, "Error making connection\n");
      cleanup(&conn);
      exit(-1);
   }
```

```
//Does not work everytime
//Greeting
char message[4096];
if(read_message(&conn, message) < 0 && message[0] != '2'){
   fprintf(stderr, "Error reading greeting\n");
   cleanup(&conn);
   exit(-1);
}

char user_msg[256];
strcpy(user_msg, "user ");
if(conn.anonymous == 1)
   strcat(user_msg, "anonymous\n");
else {
   strcat(user_msg, conn.username);
   strcat(user_msg, "\n");
}

//Send username
if(send_message(&conn, user_msg) < 0){
   fprintf(stderr, "Failed sending username\n");
   cleanup(&conn);
   exit(-1);
}

//Read password prompt
if(read_message(&conn, message) < 0 && message[0] != '3'){
   fprintf(stderr, "Error reading\n");
   cleanup(&conn);
   exit(-1);
}

if(message[0] == '5'){
   fprintf(stderr, "Error: server is anonymous only\n");
   cleanup(&conn);
   exit(-1);
}


char password_msg[256];
strcpy(password_msg, "pass ");
if(conn.anonymous == 1){
   strcat(password_msg, ANONYMOUS_PASSWORD);
   strcat(password_msg, "\n");
} else {
   strcat(password_msg, conn.password);
   strcat(password_msg, "\n");
}
```

```
//Send username
if(send_message(&conn, password_msg) < 0){
   fprintf(stderr, "Failed sending password\n");
   cleanup(&conn);
   exit(-1);
}

//Password reponse
if(read_message(&conn, message) < 0 && message[0] != '2'){
   fprintf(stderr, "Error reading\n");
   cleanup(&conn);
   exit(-1);
}

//Enter passive
if(send_message(&conn, "pasv\n") < 0){
   fprintf(stderr, "Failed sending pasv command\n");
   cleanup(&conn);
   exit(-1);
}

//Read pasv response
if(read_message(&conn, message) < 0 && message[0] != '2'){
   fprintf(stderr, "Error reading\n");
   cleanup(&conn);
   exit(-1);
}

parse_pasv_response(&conn, message);

if(make_data_connection(&conn) < 0){
   fprintf(stderr, "Error making connection\n");
   cleanup(&conn);
   exit(-1);
}


//Enter passive
char retrieve_msg[256];
char retr_response[256];
strcpy(retrieve_msg, "retr ");
strcat(retrieve_msg, conn.url);
strcat(retrieve_msg, "\n");
if(send_message(&conn, retrieve_msg) < 0){
   fprintf(stderr, "Failed sending pasv command\n");
   cleanup(&conn);
   exit(-1);
}

if(read_message(&conn, retr_response) > 0) {
```

```
        if(retr_response[0] == '5') {
            cleanup(&conn);
            exit(-1);
        }
    } else {
        fprintf(stderr, "No response for retr command\n");
        cleanup(&conn);
        exit(-1);
    }

    if(read_data(&conn) < 0){
        fprintf(stderr, "Error reading\n");
        cleanup(&conn);
        exit(-1);
    }


    cleanup(&conn);

    return 0;
}
```