

Redes de Computadores

2.º Trabalho Prático - Rede de Computadores

Gonçalo Teixeira e Gonçalo Alves

Mestrado Integrado em Engenharia
Informática e Computação



FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

23 de dezembro de 2020

Porto

Sumário

Este relatório foi desenvolvido no âmbito do segundo trabalho prático da Unidade Curricular de Redes de Computadores. O trabalho tem como objetivos a implementação de uma aplicação *download*, a configuração e o estudo de uma rede de computadores propriamente dita.

Apesar das limitações, devido à atual conjuntura, conseguiram-se concluir todos objetivos do trabalho, sobrando apenas o ponto 7 da parte II por concluir, que é opcional, portanto todos os objetivos obrigatórios do trabalho foram concluídos com sucesso, e as conclusões podem ser encontradas nas secções apropriadas.

Conteúdo

| | |
|--|----|
| Sumário | 1 |
| Introdução | 3 |
| Parte I - Aplicação Download | 4 |
| Arquitetura | 4 |
| Resultados | 4 |
| Parte II - Configuração e Análise de Rede | 5 |
| Experiência 1 - Configuração IP de Rede | 5 |
| Experiência 2 - Implementar duas LANs Virtuais no Switch | 6 |
| Experiência 3 - Configurar um Router em Linux | 6 |
| Experiência 4 - Configurar um Router comercial e implementar o NAT | 7 |
| Experiência 5 - DNS | 8 |
| Experiência 6 - Ligações TCP | 8 |
| Conclusão | 10 |
| Referências | 11 |
| Anexos | 12 |
| Imagens | 12 |
| Aplicação Download | 16 |

Introdução

Este trabalho é composto por duas partes, tal como foi mencionado no sumário, a primeira parte consiste no desenvolvimento de uma aplicação de *download*; a segunda parte consiste na configuração e estudo de uma rede de computadores.

A rede configurada deve ser capaz de permitir a execução da aplicação *download* com sucesso, além disso, deve contemplar duas vlans dentro de um Cisco Switch, a configuração de um Cisco Router e implementação de NAT e DNS.

Este relatório está também subdividido em duas partes principais, a exposição teórica do processo de desenvolvimento e teste da aplicação *download*, e para segunda parte, proceder-se-á a configuração da rede experiêcia a experiência, respondendo a cada pergunta colocada no guião, acompanhado de figuras e registos.

Depois da exposição teórica do assunto e do desenvolvimento, apresentar-se-ão conclusões acerca dos objetivos propostos e conhecimentos adquiridos.

Parte I - Aplicação Download

A primeira parte deste segundo trabalho foi o desenvolvimento de uma aplicação de download, que requer um link do tipo `ftp://[<user>:<password>@]<host>/<url-path>` (tal como descrito em RFC1738), através de um servidor FTP.

Arquitetura

Numa fase inicial é feito o *parsing* do link passado como argumento. A função *parseArg* recolhe os devidos campos (*user*, *password*, *host* e *url_path*), e guarda nas variáveis com o mesmo nome. No caso dos campos *user* e *password* não estarem presentes no link, serão dados os valores de *anonymous* e *1234*, respetivamente.

De seguida, através de código já fornecido pelos docentes, é obtido o *IP address*, usando a função *getIP(host)*.

Após estes passos, é iniciada a conexão entre o cliente e o servidor, através da abertura de um *socket* (*socket_connection*, adaptado do código disponibilizado pelos docentes). É necessário referir que a partir desta conexão, a comunicação entre cliente-servidor é muito semelhante para os diferentes comandos enviados: é feita com base no envio de pedidos (*ftp_write_socket*, retorna 0 se o envio foi bem sucedido, -1 em contrário) e leitura de respostas (*ftp_read_socket*, retorna o código de resposta do servidor). Como tal, quando for referido o envio de um comando, na realidade está-se a querer dizer o envio de um pedido e receção da resposta consequente. Assim, após a conexão inicial:

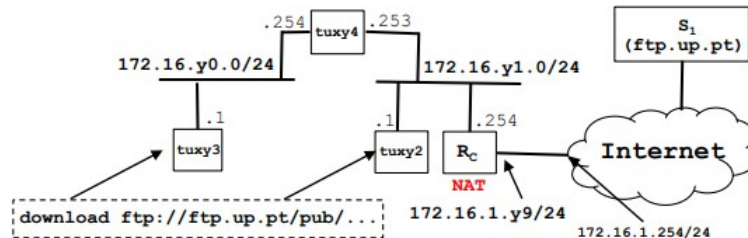
- É verificado se a conexão foi estabelecida (*ftp_init_connection*);
- São enviados os comandos **USER** *user*, **PASS** *password*, **TYPE I** de modo a fazer o login (*ftp_login*);
- É enviado o comando **PASV**, de modo a entrar no modo passivo (*ftp_passive*);
- É aberto um novo *socket*, para troca de dados, é enviado o comando **RETR** *url_path* (*ftp_retrieve*), de modo a fazer o pedido do ficheiro, é feito o download do ficheiro (*ftp_get_file*) e são fechados os *sockets* de dados e de comandos (*socket_exit*), terminando a aplicação (*ftp_download*).

Resultados

A nossa aplicação foi testada com: modo anónimo, modo autenticado, diferentes tipos de ficheiro e diferentes tamanhos de ficheiros. Sendo em todos os cenários bem sucedida. Durante a execução do programa são apresentadas as informações iniciais (*user*, *password*, *host*, *url_path* e *IP*) e as respostas do servidor, de modo a proporcionar ao utilizador o estado do download.

Parte II - Configuração e Análise de Rede

No final da atividade o objetivo é ter uma rede com esta configuração:



Experiência 1 - Configuração IP de Rede

O objetivo desta experiência é ligar dois tux através de um switch.

1. O que são os pacotes ARP e para que são usados?

O ARP (*Address Resolution Protocol*) é um protocolo de comunicação que serve para descobrir o endereço da camada de ligação associado ao endereço IP numa LAN (*Local Area Network*). O endereço da camada de ligação é também conhecido por Endereço MAC (*Media Access Control*).

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

Executando o comando *ping* do tux3 para o tux4, o tux3 envia uma pergunta para saber qual é o endereço MAC associado ao IP do tux4. A «pergunta» é feita através de um pacote ARP que contém o endereço IP e MAC do tux3 (172.16.30.1 e 00:21:5a:5a:7d:74) e o endereço IP do tux4 (172.16.1.254), uma vez que se quer descobrir o endereço MAC do tux4, o campo dedicado a esse efeito está a 00:00:00:00:00:00. De seguida é enviada uma resposta, também sob a forma de um pacote ARP, do tux4 para o tux3, indicando o seu endereço MAC (00:21:5a:5a:7d:74). Figura 1

3. Quais os pacotes gerados pelo comando ping?

Primeiro o comando *ping* gera pacotes ARP para fazer a relação entre endereços IP e MAC, de seguida gera pacotes ICMP (*Internet Control Message Protocol*).

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

Quando se executa o comando *ping* no tux3 para o tux4, os endereços (IP e MAC) vão ser os endereços dos tux. Podemos ver de seguida os endereços registados nos pacotes de pedido e resposta, respetivamente.

| | tux | MAC | IP |
|---------|-----|-------------------|---------------|
| Origem | 3 | 00:21:5a:61:24:92 | 172.16.30.1 |
| Destino | 4 | 00:21:5a:5a:7d:74 | 172.16.30.254 |

Tabela 1: Pacote de Pedido

Devem-se consultar as figuras 2 e 3 para referência.

| | tux | MAC | IP |
|---------|-----|-------------------|---------------|
| Origem | 4 | 00:21:5a:5a:7d:74 | 172.16.30.254 |
| Destino | 3 | 00:21:5a:61:24:92 | 172.16.30.1 |

Tabela 2: Pacote de Resposta

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

O *Ethernet Header* de um pacote contém a informação acerca do tipo da trama. Para as tramas IP, o valor do tipo será 0x0800, se o *IP Header* tiver o valor 1 então o tipo de protocolo é ICMP. Para as tramas ARP o valor do tipo será 0x0806.

Para referência devem-se consultar as figuras 4 e 5.

6. Como determinar o comprimento de uma trama recetora?

O comprimento de uma trama recetora pode ser determinado inspecionando a entrada no registo do *Wireshark*, tal como se pode observar na figura 6.

Experiência 2 - Implementar duas LANs Virtuais no Switch

O objetivo desta experiência é implementar duas VLANs num Cisco Switch, uma VLAN é uma rede virtual local.

1. Como configurar a VLANy0?

Primeiro é necessário ligar um cabo série do tux3 ao switch para aceder ao terminal de configuração (*configure terminal*) do switch. De seguida cria-se uma vlan, de ID y0, no caso, 31. Por fim resta atribuir as portas em questão a essa vlan que acabou de ser criada.

```

1 configure terminal
2 > vlan y0
3 > end
4 > configure terminal
5 > interface fastethernet 0/[n da porta]
6 > switchport mode access
7 > switchport access vlan y0
8 > end

```

2. Quantos domínios de broadcast existem? O que se pode concluir a partir dos registos?

O tux3 recebe resposta do tux4 quando faz *ping broadcast*, mas não recebe do tux2. O tux2 não recebe nenhuma resposta quando executa a instrução de *ping broadcast*. Desta forma pode-se concluir que existem dois domínios de broadcast, um que contem o tux3 e tux4, e outro que contém o tux2.

Experiência 3 - Configurar um Router em Linux

O objetivo desta experiência é configurar um tux para servir de router e transmitir dados da uma vlan para outra.

1. Que rotas existem nos tux? Qual o seu significado?

O destino das rotas é até onde o tux que está na origem da rota consegue chegar.

| tux | vlan | gateway |
|-----|-------------------|---------------|
| 2 | vlan0 172.16.y0.0 | 172.16.y1.253 |
| 2 | vlan1 172.16.y1.0 | 0.0.0.0 |
| 3 | vlan0 172.16.y0.0 | 0.0.0.0 |
| 3 | vlan1 172.16.y1.0 | 172.16.y1.254 |
| 4 | vlan0 172.16.y0.0 | 0.0.0.0 |
| 4 | vlan1 172.16.y1.0 | 0.0.0.0 |

Tabela 3: Rotas Existentes nos tux

2. Qual é a informação que uma entrada da tabela de *forwarding* contém?

Destination: o destino da rota;

Gateway: o IP do próximo ponto por onde passará a rota;

Netmask: usado para determinar o ID da rede a partir do endereço IP do destino;

Flags: informações sobre a rota;

Metric: o custo de cada rota;

Ref: número de referências para esta rota (não usado no kernel do Linux);

Use: contador de pesquisas pela rota, dependendo do uso de -F ou -C isto vai ser o número de falhas da cache (-F) ou o número de sucessos (-C);

Interface: qual a placa de rede responsável pela gateway (*eth0/eth1*).

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

Tal como referido nos pontos 1 e 2 da experiência 1, quando um tux faz *ping* para outro tux é preciso relacionar o IP do destino com um endereço MAC. Podem ser consultadas a figuras 7 (*eth1 tux4*) e 8 (*eth0 tux4*).

4. Que pacotes ICMP são observados e porquê?

São observados pacotes de pedido e resposta ICMP, uma vez que as rotas estão configuradas, caso contrário seriam enviados pacotes ICMP de *Host Unreachable*. Pode ser consultada a figura 9 para referência.

5. Quais são os endereços IP e MAC associados a um pacote ICMP e porquê?

Os endereços IP e MAC associados com os pacotes ICMP são os endereços IP e MAC dos tux de origem e destino. Quando é feito *ping* do tux3 para o tux4 os endereços de origem vão ser 172.16.y0.1 (IP) e 00:21:5a:61:24:92 (MAC) e o de destino 172.16.y1.253 (IP) e 00:21:5a:5a:7d:74 (MAC).

Experiência 4 - Configurar um Router comercial e implementar o NAT

O objetivo desta experiência é configurar um Router comercial e configurar o serviço NAT para acesso à Internet.

1. Como se configura um Router estático num Router comercial?

Para configurar o Router é necessário ligar um cabo série do tux ao router, depois executam-se os seguintes comandos no *GTKTerm* (router):


```
1  configure terminal
2  > ip route [destino] [mascara] [gateway]
3  > exit
```

2. Quais são as rotas seguidas pelos pacotes durante a experiência? Explique.

No caso de a rota existir, os pacotes utilizam essa rota, caso contrário, os pacotes passam pela rota default (router), são informados que o tux4 existe, e deverão ser enviados pelo mesmo.

3. Como se configura o NAT num Router comercial?

Para configurar o router, foi necessário configurar a interface interna no processo de NAT, o que foi feito recorrendo ao guião fornecido. Ligando ao router através da porta série, utilizamos os seguintes comandos que podem ser consultados em anexo, figura 10.

4. O que faz o NAT?

O *Network Address Translation* NAT foi concebido para a preservação de endereços IP. Permite que as redes IP privadas que usem endereços IP não registados a possibilidade de se ligarem à Internet. O NAT opera num router, normalmente ligando duas redes, e traduz os endereços da rede privada (que não são únicos à escala global) em endereços válidos, antes que os pacotes sejam transmitidos para outra rede. Adicionalmente, o NAT pode ser configurado para mostrar apenas um endereço correspondente à rede privada inteira para a rede exterior. Isto transmite segurança adicional uma vez que esconde de forma eficaz os endereços da rede que está por detrás daquele endereço público. Adicionalmente, o NAT oferece também funções de segurança e é implementado em ambientes de acesso remoto.

Resumidamente, o NAT permite que os computadores de uma rede interna tenham acesso ao exterior, sendo que, um único endereço IP é exigido para representar um grupo de computadores fora da sua própria rede.

Experiência 5 - DNS

O objetivo desta experiência é configurar um servidor de DNS para traduzir endereços URL em endereços IP.

1. Como configurar um serviço DNS num *host*?

Para configurar o serviço DNS, é necessário alterar o ficheiro `resolv.conf` no *host*. Esse ficheiro tem de conter as seguintes linhas:

```
1  search netlab.fe.up.pt
2  nameserver 172.16.1.1
```

Em que `netlab.fe.up.pt` é o nome do servidor DNS e `172.16.1.1` é o seu endereço IP. Após esta experiência é possível fazer *ping* a `google.com` com sucesso e, portanto, aceder à Internet nos tux.

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

São trocados pacotes de protocolo DNS, em que o *host* pede ao servidor de DNS o IP associado ao nome que indicou, por exemplo `ftp.up.pt`, e o servidor depois responde com o endereço IP associado. Pode ser consultada a figura 11 para referência.

Experiência 6 - Ligações TCP

O objetivo desta experiência é analisar como funcionam as ligações TCP e inspecionar o funcionamento da aplicação *download* desenvolvida.

1. Quantas ligações TCP foram abertas pela aplicação FTP?

A aplicação abriu duas ligações TCP, uma para enviar comandos e receber respostas do servidor e uma outra para receber dados do servidor e enviar as repostas do cliente.

2. Em que ligação é transportada a informação de controlo?

A informação de controlo é transportada na ligação de troca de comandos, ou seja, na primeira referida no ponto anterior.

3. Quais são as fases de uma ligação TCP?

Uma ligação TCP é constituída por 3 fases, uma fase de estabelecimento da ligação, uma fase de troca de dados e uma fase de encerramento da ligação.

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

O TCP utiliza o mecanismo ARQ (*Automatic Repeat Request*) com o método da janela deslizante, que consiste no controlo de erros na transmissão de dados. Os campos relevantes para o efeito são os *acknowledgement numbers*, que indicam se a mensagem foi recebida corretamente (recetor); o *window size* que indica a gama de pacotes que o emissor pode enviar e o *sequence number*, que indica o número de sequência do pacote a ser enviado.

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 é feito quando o TCP mantém uma janela de congestão que consiste numa estimativa do número de octetos que a rede consegue encaminhar, não enviando mais octetos do que o mínimo da janela definida pelo recetor e pela janela de congestão.

Ao iniciar a transferência no tux3 registou-se uma subida acentuada no gráfico de fluxo (taxa de transferência), e perto dos 4 segundos, registamos uma descida acentuada da curva do gráfico, seguida de uma estabilização, até terminar. Podemos concluir que ao iniciar a transferência no tux2, a taxa de transferência diminui, o que faz sentido uma vez que o fluxo de dados de ligação está de acordo com o mecanismo de controlo de congestão pois quando a rede estava mais congestionada tinha um bitrate menor. O gráfico em questão pode ser consultado na figura [12](#).

6. De que forma é afetada a ligação de dados TCP pelo aparecimento de uma segunda ligação TCP? Como?

Com o aparecimento de uma segunda ligação TCP, a existência de uma transferência de dados em simultâneo pode levar a uma queda na taxa de transmissão, uma vez que a taxa de transferência é distribuída de igualmente. Esta informação pode ser verificada novamente através da figura [12](#).

Conclusão

Para conclusão do relatório deste trabalho prático pode-se afirmar que os conceitos relacionados com o funcionamento de uma rede, um elemento presente no cotidiano da maioria das pessoas no mundo, foram explorados com sucesso, e portanto, foram interiorizados e consolidados. Acerca da aplicação *download* pode-se concluir que foram explorados os conceitos da implementação de um cliente TCP. Em suma, afirma-se que todos os objetivos foram alcançados, o que contribuiu para um aprofundamento dos conhecimentos teóricos e práticos da Unidade Curricular de Redes de Computadores.

Referências

- <https://www.cisco.com/c/en/us/support/docs/ip/network-address-translation-nat/26704-nat-faq-00.html>
- <https://www.cisco.com/c/en/us/support/docs/ip/domain-name-system-dns/12683-dns-descript.html>
- <https://www.cisco.com/c/en/us/support/docs/ip/routing-information-protocol-rip/13769-5.html>

Anexos

Imagens

| | | | | | | |
|----|-------------|-------------------|-------------------|------|----|---|
| 16 | 7.426228675 | HewlettP_61:24:92 | HewlettP_5a:7d:74 | ARP | 42 | Who has 172.16.30.254? Tell 172.16.30.1 |
| 17 | 7.426352225 | HewlettP_5a:7d:74 | HewlettP_61:24:92 | ARP | 60 | 172.16.30.254 is at 00:21:5a:5a:7d:74 |
| 18 | 7.426352225 | 172.16.30.1 | 172.16.30.254 | ICMP | 98 | Echo (ping) request id=0x1a46, seq=9/2304, ttl=64 |

Figura 1: Pacotes ARP

| | | | | | | | |
|---|----|--------------|---------------|---------------|------|----|---|
| → | 26 | 10.562260390 | 172.16.30.1 | 172.16.30.254 | ICMP | 98 | Echo (ping) request id=0x1a46, seq=9/2304, ttl=64 |
| ← | 27 | 10.562398118 | 172.16.30.254 | 172.16.30.1 | ICMP | 98 | Echo (ping) reply id=0x1a46, seq=9/2304, ttl=64 |
| < > | | | | | | | |
| > Frame 26: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0 | | | | | | | |
| > Ethernet II, Src: HewlettP_61:24:92 (00:21:5a:61:24:92), Dst: HewlettP_5a:7d:74 (00:21:5a:5a:7d:74) | | | | | | | |
| > Internet Protocol Version 4, Src: 172.16.30.1, Dst: 172.16.30.254 | | | | | | | |
| > Internet Control Message Protocol | | | | | | | |

Figura 2: Pacote de Pedido

| | | | | | | | |
|---|----|--------------|---------------|---------------|------|----|---|
| → | 26 | 10.562260390 | 172.16.30.1 | 172.16.30.254 | ICMP | 98 | Echo (ping) request id=0x1a46, seq=9/2304, ttl=64 |
| ← | 27 | 10.562398118 | 172.16.30.254 | 172.16.30.1 | ICMP | 98 | Echo (ping) reply id=0x1a46, seq=9/2304, ttl=64 |
| < > | | | | | | | |
| > Frame 27: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0 | | | | | | | |
| > Ethernet II, Src: HewlettP_5a:7d:74 (00:21:5a:5a:7d:74), Dst: HewlettP_61:24:92 (00:21:5a:61:24:92) | | | | | | | |
| > Internet Protocol Version 4, Src: 172.16.30.254, Dst: 172.16.30.1 | | | | | | | |
| > Internet Control Message Protocol | | | | | | | |

Figura 3: Pacote de Resposta

```

▼ Ethernet II, Src: HewlettP_61:24:92 (00:21:5a:61:24:92), Dst: HewlettP_5a:7d:74 (00:21:5a:5a:7d:74)
  > Destination: HewlettP_5a:7d:74 (00:21:5a:5a:7d:74)
  > Source: HewlettP_61:24:92 (00:21:5a:61:24:92)
  Type: IPv4 (0x0800)
▼ Internet Protocol Version 4, Src: 172.16.30.1, Dst: 172.16.30.254
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 84
  Identification: 0x9a46 (39494)
  > Flags: 0x40, Don't fragment
  Fragment Offset: 0
  Time to Live: 64
  Protocol: ICMP (1)

```

Figura 4: Campo Type Pacote ICMP

```

▼ Ethernet II, Src: HewlettP_61:24:92 (00:21:5a:61:24:92), Dst: HewlettP_5a:7d:74 (00:21:5a:5a:7d:74)
  > Destination: HewlettP_5a:7d:74 (00:21:5a:5a:7d:74)
  > Source: HewlettP_61:24:92 (00:21:5a:61:24:92)
  Type: ARP (0x0806)

```

Figura 5: Campo Type Pacote ARP

```

▼ Frame 9: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0
  > Interface id: 0 (eth0)
  Encapsulation type: Ethernet (1)
  Arrival Time: Nov 24, 2020 15:33:40.912061718 Hora padrão de GMT
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 1606232020.912061718 seconds
  [Time delta from previous captured frame: 0.408640354 seconds]
  [Time delta from previous displayed frame: 0.408640354 seconds]
  [Time since reference or first frame: 4.418256468 seconds]
  Frame Number: 9
  Frame Length: 98 bytes (784 bits)
  Capture Length: 98 bytes (784 bits)

```

Figura 6: Tamanho de uma trama Recetora

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|----------------|-------------------|-------------------|----------|--------|--|
| 74 | 117.4924495... | HewlettP_a6:a4:f1 | Broadcast | ARP | 42 | Who has 172.16.21.1? Tell 172.16.21.253 |
| 75 | 117.4925883... | HewlettP_61:2b:72 | HewlettP_a6:a4:f1 | ARP | 60 | 172.16.21.1 is at 00:21:5a:61:2b:72 |
| 76 | 117.4925931... | 172.16.20.1 | 172.16.21.1 | ICMP | 98 | Echo (ping) request id=0x3316, seq=1/256, ttl=63 (reply in 77) |
| 77 | 117.4927290... | 172.16.21.1 | 172.16.20.1 | ICMP | 98 | Echo (ping) reply id=0x3316, seq=1/256, ttl=64 (request in 76) |
| 91 | 122.6344745... | HewlettP_61:2b:72 | HewlettP_a6:a4:f1 | ARP | 60 | Who has 172.16.21.253? Tell 172.16.21.1 |
| 92 | 122.6344821... | HewlettP_a6:a4:f1 | HewlettP_61:2b:72 | ARP | 42 | 172.16.21.253 is at 00:22:64:a6:a4:f1 |

Figura 7: Pacotes ARP carta eth1 tux4

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|----------------|-------------------|-------------------|----------|--------|--|
| 75 | 119.6648582... | HewlettP_5a:7d:12 | Broadcast | ARP | 60 | Who has 172.16.20.254? Tell 172.16.20.1 |
| 76 | 119.6648850... | Netronix_50:3f:2c | HewlettP_5a:7d:12 | ARP | 42 | 172.16.20.254 is at 00:08:54:50:3f:2c |
| 77 | 119.6649841... | 172.16.20.1 | 172.16.21.1 | ICMP | 98 | Echo (ping) request id=0x3316, seq=1/256, ttl=64 (reply in 78) |
| 78 | 119.6652819... | 172.16.21.1 | 172.16.20.1 | ICMP | 98 | Echo (ping) reply id=0x3316, seq=1/256, ttl=63 (request in 77) |
| 92 | 124.9039676... | Netronix_50:3f:2c | HewlettP_5a:7d:12 | ARP | 42 | Who has 172.16.20.1? Tell 172.16.20.254 |
| 93 | 124.9040803... | HewlettP_5a:7d:12 | Netronix_50:3f:2c | ARP | 60 | 172.16.20.1 is at 00:21:5a:5a:7d:12 |

Figura 8: Pacotes ARP carta eth0 tux4

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|--------------|-------------------|-------------------|----------|--------|--|
| 5 | 5.015726702 | 172.16.20.1 | 172.16.20.254 | ICMP | 98 | Echo (ping) request id=0x31ce, seq=1/256, ttl=64 (reply in 6) |
| 6 | 5.015866604 | 172.16.20.254 | 172.16.20.1 | ICMP | 98 | Echo (ping) reply id=0x31ce, seq=1/256, ttl=64 (request in 5) |
| 20 | 10.237511242 | Netronix_50:3f:2c | HewlettP_5a:7d:12 | ARP | 60 | Who has 172.16.20.1? Tell 172.16.20.254 |
| 21 | 10.237532545 | HewlettP_5a:7d:12 | Netronix_50:3f:2c | ARP | 42 | 172.16.20.1 is at 00:21:5a:5a:7d:12 |
| 40 | 25.016954288 | 172.16.20.1 | 172.16.21.253 | ICMP | 98 | Echo (ping) request id=0x31db, seq=1/256, ttl=64 (reply in 41) |
| 41 | 25.017093770 | 172.16.21.253 | 172.16.20.1 | ICMP | 98 | Echo (ping) reply id=0x31db, seq=1/256, ttl=64 (request in 40) |
| 53 | 30.109973528 | HewlettP_5a:7d:12 | Netronix_50:3f:2c | ARP | 42 | Who has 172.16.20.254? Tell 172.16.20.1 |
| 54 | 30.110059299 | Netronix_50:3f:2c | HewlettP_5a:7d:12 | ARP | 60 | 172.16.20.254 is at 00:08:54:50:3f:2c |
| 75 | 47.522478676 | 172.16.20.1 | 172.16.21.1 | ICMP | 98 | Echo (ping) request id=0x31eb, seq=1/256, ttl=64 (reply in 76) |
| 76 | 47.522769445 | 172.16.21.1 | 172.16.20.1 | ICMP | 98 | Echo (ping) reply id=0x31eb, seq=1/256, ttl=63 (request in 75) |
| 91 | 52.735893146 | Netronix_50:3f:2c | HewlettP_5a:7d:12 | ARP | 60 | Who has 172.16.20.1? Tell 172.16.20.254 |
| 92 | 52.735913471 | HewlettP_5a:7d:12 | Netronix_50:3f:2c | ARP | 42 | 172.16.20.1 is at 00:21:5a:5a:7d:12 |

Figura 9: Pacotes ICMP com rotas definidas

```

1  Cisco NAT
2  http://www.cisco.com/en/US/tech/tk648/tk361/technologies_tech_note09186a0080094e77.
   shtml
3
4  > conf t
5  > interface gigabitethernet 0/0 *
6  > ip address 172.16.y1.254 255.255.255.0
7  > no shutdown
8  > ip nat inside
9  > exit
10
11 > interface gigabitethernet 0/1*
12 > ip address 172.16.1.y9 255.255.255.0
13 > no shutdown
14 > ip nat outside
15 > exit
16
17 > ip nat pool ovrlld 172.16.1.y9 172.16.1.y9 prefix 24
18 > ip nat inside source list 1 pool ovrlld overload
19 > access-list 1 permit 172.16.y0.0 0.0.0.7
20 > access-list 1 permit 172.16.y1.0 0.0.0.7
21 > ip route 0.0.0.0 0.0.0.0 172.16.1.254
22 > ip route 172.16.y0.0 255.255.255.0 172.16.y1.253
23 > end
24

```

Figura 10: Configuração NAT router

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|--------------|-------------------|-------------------|----------|--------|---|
| 6 | 3.055558855 | HewlettP_5a:7d:12 | Netronix_50:3f:2c | ARP | 42 | Who has 172.16.20.254? Tell 172.16.20.1 |
| 7 | 3.055667461 | Netronix_50:3f:2c | HewlettP_5a:7d:12 | ARP | 60 | 172.16.20.254 is at 00:08:54:50:3f:2c |
| 8 | 3.188966172 | Netronix_50:3f:2c | HewlettP_5a:7d:12 | ARP | 60 | Who has 172.16.20.1? Tell 172.16.20.254 |
| 9 | 3.188986077 | HewlettP_5a:7d:12 | Netronix_50:3f:2c | ARP | 42 | 172.16.20.1 is at 00:21:5a:5a:7d:12 |
| 12 | 7.763124165 | 172.16.20.1 | 172.16.1.1 | DNS | 69 | Standard query 0xac8e A ftp.up.pt |
| 13 | 7.763134781 | 172.16.20.1 | 172.16.1.1 | DNS | 69 | Standard query 0x9b97 AAAA ftp.up.pt |
| 14 | 7.764788390 | 172.16.1.1 | 172.16.20.1 | DNS | 355 | Standard query response 0xac8e A ftp.up.pt CNAME mirrors.up.pt A 193.137.29.15 NS ns3.up.pt NS ns1.up.pt |
| 15 | 7.764852156 | 172.16.1.1 | 172.16.20.1 | DNS | 367 | Standard query response 0x9b97 AAAA ftp.up.pt CNAME mirrors.up.pt AAAA 2001:690:2200:1200::15 |
| 16 | 7.765204376 | 172.16.20.1 | 193.137.29.15 | ICMP | 98 | Echo (ping) request id=0x0d00, seq=1/256, ttl=64 (reply in 17) |
| 17 | 7.768218671 | 193.137.29.15 | 172.16.20.1 | ICMP | 98 | Echo (ping) reply id=0x0d00, seq=1/256, ttl=57 (request in 16) |
| 18 | 7.768323575 | 172.16.20.1 | 172.16.1.1 | DNS | 86 | Standard query 0x20ff PTR 15.29.137.193.in-addr.arpa |
| 19 | 7.769399580 | 172.16.1.1 | 172.16.20.1 | DNS | 387 | Standard query response 0x20ff PTR 15.29.137.193.in-addr.arpa PTR mirrors.up.pt NS ns3.up.pt NS ns1.up.pt |
| 42 | 16.778880106 | 172.16.20.1 | 193.137.29.15 | ICMP | 98 | Echo (ping) request id=0x0d00, seq=10/2560, ttl=64 (reply in 43) |
| 43 | 16.780613196 | 193.137.29.15 | 172.16.20.1 | ICMP | 98 | Echo (ping) reply id=0x0d00, seq=10/2560, ttl=57 (request in 42) |
| 49 | 26.099582512 | 172.16.20.1 | 172.16.1.1 | DNS | 70 | Standard query 0x5e87 A google.com |
| 50 | 26.099592570 | 172.16.20.1 | 172.16.1.1 | DNS | 70 | Standard query 0xcd90 AAAA google.com |
| 51 | 26.100966526 | 172.16.1.1 | 172.16.20.1 | DNS | 334 | Standard query response 0x5e87 A google.com A 172.217.17.14 NS ns1.google.com NS ns3.google.com NS ns2.google.com |
| 52 | 26.101025543 | 172.16.1.1 | 172.16.20.1 | DNS | 346 | Standard query response 0xcd90 AAAA google.com AAAA 2a00:1450:4003:802::200e NS ns1.google.com NS ns3.google.com |
| 53 | 26.101391941 | 172.16.20.1 | 172.217.17.14 | ICMP | 98 | Echo (ping) request id=0x0d0a, seq=1/256, ttl=64 (reply in 54) |
| 54 | 26.115220478 | 172.217.17.14 | 172.16.20.1 | ICMP | 98 | Echo (ping) reply id=0x0d0a, seq=1/256, ttl=112 (request in 53) |
| 55 | 26.115349199 | 172.16.20.1 | 172.16.1.1 | DNS | 86 | Standard query 0x7180 PTR 14.17.217.172.in-addr.arpa |
| 56 | 26.116673426 | 172.16.1.1 | 172.16.20.1 | DNS | 409 | Standard query response 0x7180 PTR 14.17.217.172.in-addr.arpa PTR mad07s09-in-f14.1e100.net NS ns1.google.com |
| 78 | 34.287561485 | HewlettP_5a:7d:12 | Netronix_50:3f:2c | ARP | 42 | Who has 172.16.20.254? Tell 172.16.20.1 |
| 79 | 34.287658637 | Netronix_50:3f:2c | HewlettP_5a:7d:12 | ARP | 60 | 172.16.20.254 is at 00:08:54:50:3f:2c |
| 81 | 35.113947340 | 172.16.20.1 | 172.217.17.14 | ICMP | 98 | Echo (ping) request id=0x0d0a, seq=10/2560, ttl=64 (reply in 82) |
| 82 | 35.127174594 | 172.217.17.14 | 172.16.20.1 | ICMP | 98 | Echo (ping) reply id=0x0d0a, seq=10/2560, ttl=112 (request in 81) |

Figura 11: Pacotes DNS

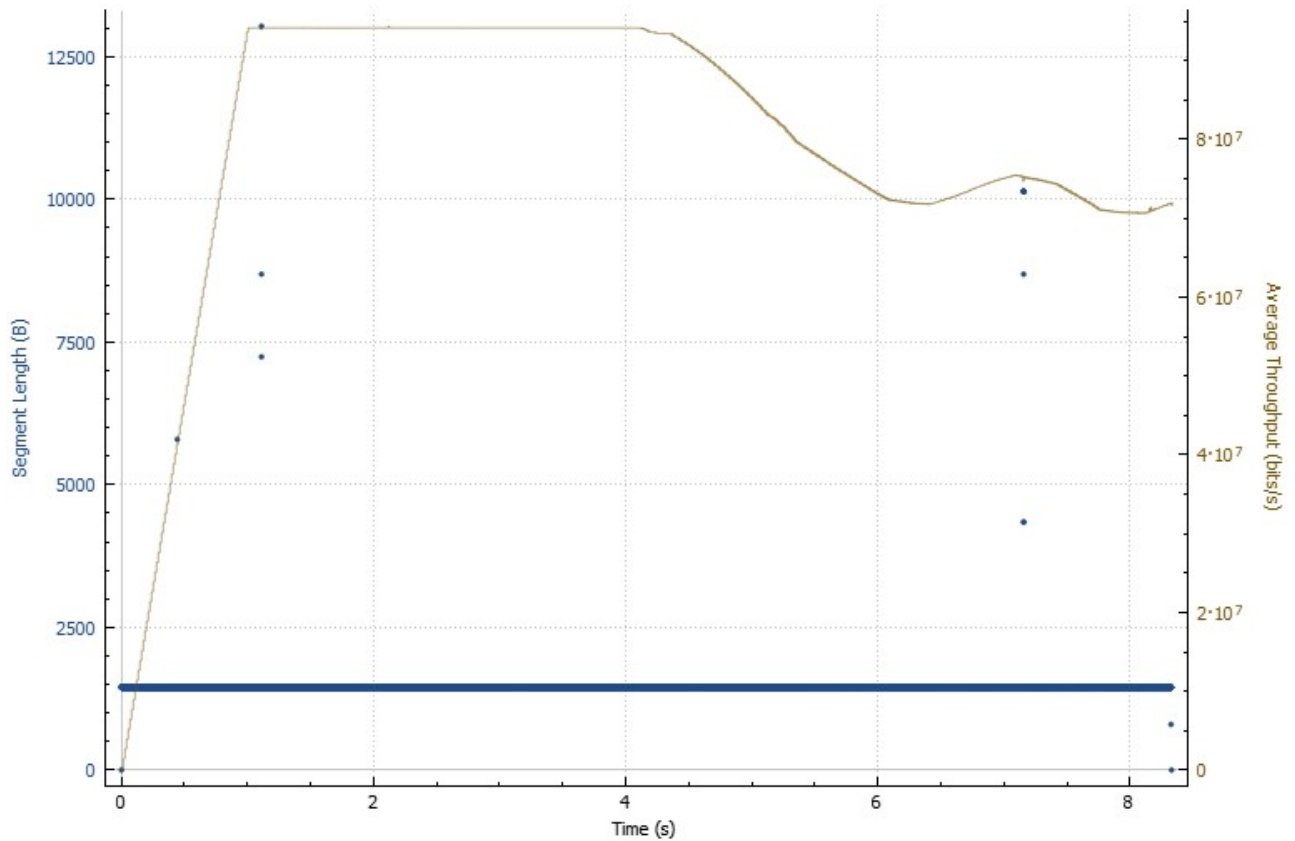


Figura 12: Throughput TCP

Aplicação Download

download.c

```
1  /*      (C)2000 FEUP      */
2
3  #include "connection.h"
4  #include "getip.h"
5
6  #define SERVER_PORT 6000
7  #define SERVER_ADDR "192.168.28.96"
8
9
10 int main(int argc, char **argv) {
11
12     if (argc != 2) {
13         printf("usage: download ftp://[<user>:<password>@]<host>/<url-path>\n");
14         return 1;
15     }
16
17     char user[1024], password[1024], host[1024], url_path[1024], *ip;
18
19     parseArg(argv[1], user, password, host, url_path);
20     printArg(user, password, host, url_path);
21     ip = getIP(host);
22     printf("IP - %s\n", ip);
23     if (ftp_init_connection(ip) == -1) return -1;
24     if (ftp_login(user, password) == -1) return -1;
25     if (ftp_download(url_path) == -1) return -1;
26
27     return 0;
28 }
29
30 void printArg(char *user, char *password, char *host, char *url_path) {
31     printf("User - %s\n", user);
32     printf("Password - %s\n", password);
33     printf("Host - %s\n", host);
34     printf("URL - %s\n", url_path);
35 }
36
37 // ./download ftp://user:1234@sftp.up.pt/pub/ficheiro.zip
38 void parseArg(char *arg, char *user, char *password, char *host, char *url_path) {
39
40
41     char *args = strtok(arg, "/");
42     args = strtok(NULL, "/*");
43     strcpy(user, args);
44
45     args = strtok(NULL, "@");
46     strcpy(password, args);
47
48     args = strtok(NULL, "/");
49     if (args == NULL) {
50         printf("No User\nSetting Default - anonymous\n");
51         strcpy(host, user);
52         strcpy(user, "anonymous");
53     } else {
54         strcpy(host, args);
55     }
56
57     args = strtok(NULL, "\\0");
58     if (args == NULL) {
```

```

59     printf("No Password\nSetting Default- 1234\n");
60     strcpy(url_path, password);
61     strcpy(password, "1234");
62 } else {
63     strcpy(url_path, args);
64 }
65
66 }

```

getip.h

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <errno.h>
4 #include <netdb.h>
5 #include <sys/types.h>
6 #include <netinet/in.h>
7 #include <arpa/inet.h>
8
9 char *getIP(char host[]);

```

getip.c

```

1
2 #include "getip.h"
3
4 char *getIP(char host[]) {
5     struct hostent *h;
6
7     /*
8     struct hostent {
9         char    *h_name;   Official name of the host.
10        char    **h_aliases; A NULL-terminated array of alternate names for the host.
11        int     h_addrtype; The type of address being returned; usually AF_INET.
12        int     h_length;   The length of the address in bytes.
13        char    **h_addr_list; A zero-terminated array of network addresses for the host.
14        Host addresses are in Network Byte Order.
15    };
16
17    #define h_addr h_addr_list[0] The first address in h_addr_list.
18    */
19    if ((h = gethostbyname(host)) == NULL) {
20        perror("gethostbyname");
21        exit(1);
22    }
23    char *IP = inet_ntoa(*((struct in_addr *) h->h_addr));
24    printf("Host name   : %s\n", h->h_name);
25    printf("IP Address  : %s\n", inet_ntoa(*((struct in_addr *) h->h_addr)));
26
27    return IP;
28 }

```

connection.h

```

1 #pragma once
2
3 #include <stdio.h>
4 #include <sys/types.h>
5 #include <sys/socket.h>
6 #include <sys/stat.h>
7 #include <fcntl.h>
8 #include <netinet/in.h>
9 #include <arpa/inet.h>

```

```

10 #include <stdlib.h>
11 #include <unistd.h>
12 #include <signal.h>
13 #include <netdb.h>
14 #include <strings.h>
15 #include <string.h>
16
17 #define PORT_FTP 21
18 #define SERV_READY 220
19 #define USER_LOGIN 331
20 #define PASS_LOGIN 230
21 #define BIN_READY 200
22 #define PASV_READY 227
23 #define RETRV_READY 150
24 #define FILE_READY 226
25
26
27 int socket_connection(char *ip, int port);
28
29 int socket_exit();
30
31 int ftp_init_connection(char *ip);
32
33 int ftp_read_socket(int sockfd);
34
35 int ftp_write_socket(int sockfd, char *msg);
36
37 int ftp_enter_pasv(int sockfd, char *ip, int *port);
38
39 int ftp_login(char *username, char *password);
40
41 int ftp_passive(char *ip, int *port);
42
43 int ftp_retrieve(char *url_path);
44
45 int ftp_get_file(char *url_path);
46
47 int ftp_download(char *url_path);

```

connection.c

```

1 #include "connection.h"
2
3 static char cmd_send[1024];
4 static int cmd_socket;
5 static int data_socket;
6
7 int socket_connection(char *ip, int port) {
8     int sockfd;
9     struct sockaddr_in server_addr;
10    bzero((char *) &server_addr, sizeof(server_addr));
11    server_addr.sin_family = AF_INET;
12    server_addr.sin_addr.s_addr = inet_addr(ip);    /*32 bit Internet address network
byte ordered*/
13    server_addr.sin_port = htons(port);            /*server TCP port must be network byte
ordered */
14
15    /*open an TCP socket*/
16    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
17        perror("socket()");
18        exit(0);
19    }

```

```

20  /*connect to the server*/
21  if (connect(sockfd, (struct sockaddr *) &server_addr, sizeof(server_addr)) < 0) {
22      perror("connect()");
23      exit(0);
24  }
25  return sockfd;
26 }
27
28 int socket_exit() {
29     int ret;
30     sprintf(cmd_send, "QUIT \r\n");
31     ret = ftp_write_socket(cmd_socket, cmd_send);
32     if (ret == -1) {
33         printf("Couldn't send retrieve command");
34         close(cmd_socket);
35         return -1;
36     }
37     close(cmd_socket);
38     return 0;
39 }
40
41 int ftp_read_socket(int sockfd) {
42     FILE *f = fdopen(sockfd, "r");
43
44     char *buf;
45     size_t bRead = 0;
46     int response;
47
48     while (getline(&buf, &bRead, f) > 0) {
49         printf("%s", buf);
50         if (buf[3] == ' ') {
51             sscanf(buf, "%d", &response);
52             break;
53         }
54     }
55     return response;
56 }
57
58 int ftp_write_socket(int sockfd, char *msg) {
59     int b, len = strlen(msg);
60
61     if ((b = write(sockfd, msg, len)) != len) {
62         printf("Couldn't write to socket");
63         return -1;
64     }
65     return 0;
66 }
67
68 int ftp_enter_pasv(int sockfd, char *ip, int *port) {
69     int ret;
70     char *find, *buf;
71     int a, b, c, d, pa, pb;
72     size_t bRead = 0;
73     FILE *f = fdopen(sockfd, "r");
74
75     while (getline(&buf, &bRead, f) > 0) {
76         printf("%s", buf);
77         if (buf[3] == ' ') {
78             sscanf(buf, "%d", &ret);
79             break;
80         }
81     }

```

```

82
83     if (ret != PASV_READY) return -1;
84     find = strrchr(buf, '(');
85     sscanf(find, "(%d,%d,%d,%d,%d,%d)", &a, &b, &c, &d, &pa, &pb);
86     sprintf(ip, "%d.%d.%d.%d", a, b, c, d);
87     *port = pa * 256 + pb;
88     return 0;
89 }
90
91 int ftp_init_connection(char *ip) {
92     int ret;
93     if ((cmd_socket = socket_connection(ip, PORT_FTP)) < 0) {
94         printf("Error establishing connection\n");
95         return -1;
96     }
97     ret = ftp_read_socket(cmd_socket);
98     if (ret != SERV_READY) {
99         printf("Received Bad Response\n");
100         close(cmd_socket);
101         return -1;
102     }
103     return 0;
104 }
105
106 int ftp_login(char *username, char *password) {
107     int ret;
108     sprintf(cmd_send, "USER %s\r\n", username);
109     ret = ftp_write_socket(cmd_socket, cmd_send);
110     if (ret == -1) {
111         printf("Couldn't send user command");
112         close(cmd_socket);
113         return -1;
114     }
115     /* Receive Login message*/
116     ret = ftp_read_socket(cmd_socket);
117     if (ret != USER_LOGIN) {
118         printf("Couldn't login user\n");
119         close(cmd_socket);
120         return -1;
121     }
122
123     sprintf(cmd_send, "PASS %s\r\n", password);
124     ret = ftp_write_socket(cmd_socket, cmd_send);
125     if (ret == -1) {
126         printf("Couldn't send password command");
127         close(cmd_socket);
128         return -1;
129     }
130     /* Receive Login message*/
131     ret = ftp_read_socket(cmd_socket);
132     if (ret != PASS_LOGIN) {
133         printf("Couldn't login\n");
134         close(cmd_socket);
135         return -1;
136     }
137
138     sprintf(cmd_send, "TYPE I\r\n");
139     ret = ftp_write_socket(cmd_socket, cmd_send);
140     if (ret == -1) {
141         printf("Couldn't send binary command");
142         close(cmd_socket);
143         return -1;

```

```

144     }
145     /* Receive Login message*/
146     ret = ftp_read_socket(cmd_socket);
147     if (ret != BIN_READY) {
148         printf("Couldn't login\n");
149         close(cmd_socket);
150         return -1;
151     }
152
153     return 0;
154 }
155
156 int ftp_passive(char *ip, int *port) {
157     int ret;
158
159     sprintf(cmd_send, "PASV\r\n");
160     ret = ftp_write_socket(cmd_socket, cmd_send);
161     if (ret == -1) {
162         printf("Couldn't send passive command");
163         close(cmd_socket);
164         return -1;
165     }
166
167     ret = ftp_enter_pasv(cmd_socket, ip, port);
168     if (ret != 0) {
169         printf("Couldn't set passive\n");
170         close(cmd_socket);
171         return -1;
172     }
173     return 0;
174 }
175
176 int ftp_retrieve(char *url_path) {
177     int ret;
178
179     sprintf(cmd_send, "RETR %s\r\n", url_path);
180     ret = ftp_write_socket(cmd_socket, cmd_send);
181     if (ret == -1) {
182         printf("Couldn't send retrieve command");
183         close(cmd_socket);
184         return -1;
185     }
186     ret = ftp_read_socket(cmd_socket);
187     if (ret != RETRV_READY) {
188         printf("Couldn't retrieve file\n");
189         close(cmd_socket);
190         return -1;
191     }
192     return 0;
193 }
194
195 int ftp_get_file(char *url_path) {
196     int fd;
197
198     char pathcpy[1024], *filename;
199     strcpy(pathcpy, url_path);
200
201     char *token = strtok(pathcpy, "/");
202     while (token != NULL) {
203         filename = token;
204         token = strtok(NULL, "/");
205     }

```

```

206     if ((fd = open(filename, O_WRONLY | O_CREAT, 0777)) < 0) {
207         printf("Error opening data file!\n");
208         return -1;
209     }
210
211     char buf[1024];
212     int bRead;
213
214     while ((bRead = read(data_socket, buf, 1024)) > 0) {
215         if (write(fd, buf, bRead) < 0) {
216             printf("Error writing data to file!\n");
217             return -1;
218         }
219     }
220
221     if (close(fd) < 0) {
222         printf("Error closing file!\n");
223         return -1;
224     }
225
226     return 0;
227 }
228
229 int ftp_download(char *url_path) {
230     char ip[64];
231     int port, ret;
232
233     if (ftp_passive(ip, &port) == -1) return -1;
234
235     if ((data_socket = socket_connection(ip, port)) < 0) {
236         close(cmd_socket);
237         return -1;
238     }
239     if (ftp_retrieve(url_path) == -1) {
240         close(cmd_socket);
241         return -1;
242     }
243
244     if (ftp_get_file(url_path) == -1) {
245         close(cmd_socket);
246         return -1;
247     }
248
249     close(data_socket);
250     ret = ftp_read_socket(cmd_socket);
251     if (ret != FILE_READY) {
252         printf("Couldn't retrieve file\n");
253         close(cmd_socket);
254         return -1;
255     }
256     return socket_exit();
257 }

```

Makefile

```

1 CC = gcc
2 CFLAGS = -Wall -pthread -Wno-pointer-sign -g
3 DEPS = connection.h getip.h
4 OBJ = connection.o getip.o
5 TARGETS = download
6
7 all: download

```

```
8
9 %.o: %.c $(DEPS)
10    $$ (CC) $(CFLAGS) -c -o $@ $<
11    @echo $@
12
13 download: $(OBJ)
14    $$ (CC) $(CFLAGS) -o $@ $@.c $(OBJ) -lm
15    @echo $@
16
17 clean:
18    @rm *.o $(TARGETS)
```