Redes de Computadores

2º Trabalho Laboratorial

Desenvolvimento de uma Aplicação de Download & Configuração e Estudo de uma Rede

(22 de dezembro de 2023)

Artur Telo Luís **(up202104487)**
Xavier Maria Pimenta Santos **(up202108894)**

Sumário

Este relatório foi elaborado no âmbito da UC de Redes de Computadores, e apresenta compreensivamente o código desenvolvido para uma aplicação de transferência de ficheiros, bem como um guia através dos passos seguidos para configurar uma rede para testar a aplicação.

Introdução

Neste projeto desenvolvemos uma aplicação de transferência de ficheiros através do protocolo FTP, e configuramos uma rede de computadores (segundo o guião fornecido) para conseguir acesso à Internet e poder testar a nossa aplicação nessa rede. Testamos a aplicação com diversos hosts, ficheiros e credenciais de acesso, e tivemos sucesso em todos os testes.

Parte 1 – Aplicação de Download

A aplicação é separada em três ficheiros com código (.c) e os seus respetivos ficheiros header (.h).

getIP.c contém apenas a função getIP(const char* hostname, char* ip), que nos foi fornecida, que guarda no argumento 'ip' o endereço IP do host especificado no argumento 'hostname'.
clientTCP.c contém a função clientTCP(const char* ip, uint16_t port)', que, tal como getIP.c, nos foi fornecida; esta recebe como argumentos um endereço IP e uma porta, abre uma nova conexão ("file descriptor") para essa porta no IP fornecido, e retorna o valor associado a esse file descriptor.
main.c contém as principais funções:
- `checkResponse(const int socket, char* buf)`: verifica a resposta do servidor, e devolve o valor da mesma.
- `connectToServer(const int socket, const char* user, const char* password)`: com as credenciais fornecidas pelo utilizador (ou as *default*) tenta autenticar-se no servidor.
- `enterPassiveMode(const int socket, char* ip_address, int* port)`: coloca o servidor em modo passivo, e recebe em *ip_address* e *port* o endereço IP e *port*, respetivamente, para abrir nova conexão com o servidor, de onde irá transferir o ficheiro.
- `main(int argc, char* argv[])`: função principal, responsável por receber e filtrar o *input* do utilizador, e controlar o fluxo de execução do programa, chamando as funções auxiliares quando necessário.

Um dos testes à aplicação foi realizado na máquina tux43, na sala I320 da FEUP. Após compilar o código-fonte, corremos no terminal o comando `./download ftp://rcom:rcom@netlab1.fe.up.pt/pipe.txt`, para transferir o ficheiro pipe.txt, no *host* netlab1.fe.up.pt, com as credenciais `USER rcom` e `PASS rcom`. A comunicação entre tux43 e o servidor foi capturada no computador com o Wireshark; podemos ver essa comunicação nas imagens 1.1 e 1.2 (presentes na secção Anexos/Imagens).

Os pacotes 4 e 5 são pacotes DNS, usados para o cliente obter o endereço IP do *host* com o qual está a tentar comunicar.
Os pacotes 6 e 7 são do protocolo TCP, em que o cliente e o servidor estabelecem a ligação.
Pacotes 8 a 19, 22 a 27, e 29 a 35 alternam entre TCP e FTP, e contêm a troca de mensagens entre as duas máquinas.
Os pacotes 20 e 21 são, à semelhança dos 6 e 7, do protocolo TCP, e representam o segundo terminal do tux43 a ligar-se à porta do servidor obtida após o pedido 'pasv' no primeiro terminal.
Pacotes 28, 36, 37 e 38 são TCP, e representam o fecho das conexões.

Parte 2 - Configuração e Estudo de uma Rede

Experiência 1 - Configurar uma *IP network*

Nesta experiência, configuraram-se dois endereços IP, um para o tux43 (172.16.40.1/24) e outro para o tux44 (172.16.40.254/24), conectando ambos a um switch. O objetivo era analisar o funcionamento do protocolo ARP, que associa endereços IP a endereços MAC, essenciais para estabelecer conexões. Após usar o comando ping, observou-se a troca de pacotes ARP e ICMP entre os dois computadores (como se pode observar na imagem 2.1.1).

Ao eliminar entradas das tabelas ARP, notou-se uma nova troca de pacotes ARP no início da conexão para restaurar as associações entre endereços IP e MAC removidos. O protocolo ICMP foi utilizado para mensagens de controle, indicando sucesso ou erros na comunicação.

Perguntas:

1. Quais são os comandos necessários para configurar esta experiência?
Ver anexos “Experiência 1”.

2. O que são os pacotes ARP e para que são usados?
Os pacotes ARP (Address Resolution Protocol) são utilizados para mapear endereços IP (Internet Protocol) para endereços MAC (Media Access Control) em redes de computadores.

3. Quais são os endereços MAC e IP dos pacotes ARP e por quê?
Quando o tux43 com endereço IP 172.16.40.01, precisa de encontrar o endereço MAC associado a outro dispositivo na rede, como o tux44 com endereço IP 172.16.40.254, ele envia um pacote ARP em broadcast. Este pacote ARP contém o endereço IP da máquina de destino (tux44) e o endereço IP da máquina de origem (tux43) no mesmo pacote.

Em retorno a esta solicitação ARP, o tux44 envia um pacote ARP de volta ao tux43, fornece seu próprio endereço MAC. Essa troca de informações permite que os dispositivos da rede associem corretamente os endereços IP aos endereços MAC, facilitando a comunicação eficiente entre eles.

4. Que pacotes o comando ping gera?
Antes de enviar pacotes ICMP para a máquina de destino, o comando 'ping' busca o endereço MAC correspondente ao endereço IP alvo através de pacotes ARP. Uma vez obtido o endereço MAC, são gerados pacotes ICMP para iniciar a comunicação e troca de informações com a máquina de destino.

5. Quais são os endereços MAC e IP dos pacotes de ping?
Os pacotes ICMP utilizam os endereços IP e MAC das máquinas envolvidas na comunicação, ou seja, no contexto mencionado, os endereços IP e MAC envolvem as máquinas tux43 e tux44.

6. Como determinar se um quadro Ethernet receptor é ARP, IP, ICMP?
A identificação do protocolo de cada frame pode ser encontrada na coluna "Protocol" da captura no Wireshark.

7. Como determinar o comprimento de um quadro receptor?
No Wireshark, há uma coluna que exibe o tamanho das frames em bytes no cabeçalho de cada frame.

8. O que é a interface de loopback e porque é importante?
É uma interface virtual que permanece acessível, desde que pelo menos uma das interfaces IP do switch esteja em funcionamento, possibilitando a verificação regular da configuração correta das conexões de rede.

Experiência 2 - Implementar duas *bridges* num *switch*

Nesta experiência, foram criadas duas redes locais (LANs) utilizando um switch com duas bridges, uma com os tux 43 e 44 e outra apenas com o tux 42. A configuração envolveu a repetição das configurações de rede da experiência anterior e a atribuição do endereço IP 172.16.41.1/24 ao tux42. As bridges 40 e 41 foram criadas no switch, e as portas padrão das interfaces conectadas aos tuxes foram removidas e realocadas às bridges correspondentes.

Durante os testes, observou-se que o tux43 conseguia alcançar o tux44, mas não o tux42, devido à separação em duas LANs (ver imagem 2.2.1). O tux42, por sua vez, não conseguia alcançar nenhum outro computador, evidenciando a existência de dois domínios de broadcast. Essas conclusões foram baseadas na análise dos pacotes ICMP capturados nos computadores envolvidos.

Perguntas:

1. Como configurar a *bridge40*?
A *bridge40* foi configurada para conectar os computadores tux43 e tux44, formando uma sub-rede. As configurações padrão do switch foram removidas, e novas portas foram configuradas para cada computador. O objetivo era personalizar a bridge de acordo com os requisitos da rede.

2. Quantos domínios de transmissão existem? Como pode concluir isso a partir dos logs?
Na configuração atual, há dois domínios de broadcast devido à presença de duas bridges distintas. Ao analisar os logs, observou-se que o tux43 recebeu resposta ao fazer ping para o tux44, mas não obteve resposta do tux42. Isso ocorre porque esses dois computadores estão conectados a bridges diferentes, resultando em domínios de broadcast separados e, consequentemente, limitando a comunicação entre eles.

Experiência 3 - Configurar um router em Linux

Nesta experiência, partindo de uma configuração anterior com as máquinas tux43, tux44 e tux42 conectadas a bridges diferentes, o objetivo foi transformar o tux44 num router para permitir a comunicação entre tux43 e tux42, que estavam em LANs distintas. Isso foi alcançado conectando o tux44 ao switch, configurando seu IP como 172.16.41.253/24 e adicionando a interface à *bridge41*. Foram feitas configurações adicionais, como desativar o icmp_echo_ignore_broadcasts, ativar o IP forwarding e criar rotas nos tuxes 42 e 43, usando o IP do tux44 como gateway. Isso transformou o tux44 em num router efetivo, permitindo que os pacotes entre tux43 e tux42 passassem por ele. O redirecionamento foi evidenciado nas tabelas de encaminhamento, garantindo que cada IP de destino tivesse um gateway apropriado para o roteamento da informação.

Perguntas:

1. Quais são os comandos necessários para configurar esta experiência?
Ver anexos “Experiência 3”.

2. Que rotas existem nos tuxes? Qual é o seu significado?
No tux42 e no tux43, foram estabelecidas rotas, e ambos têm o tux44 como gateway. Essa configuração reflete o papel do tux44 como router comum às duas bridges, facilitando a comunicação entre as LANs conectadas a essas bridges.

3. Que informações contém a entrada da tabela de encaminhamento?
Tanto o tux42 como o tux43 possuem entradas de rota que especificam o endereço de destino e o gateway correspondente.

4. Que mensagens ARP e endereços MAC associados são observados e porquê?
Os pacotes ARP envolvidos contêm apenas os endereços MAC do tux43 e do tux44, sem incluir o endereço MAC do destino final (tux42), ao fazer o ping. Essa ausência deve-se à implementação da rota, em que o tux43 conhece apenas o endereço da gateway (tux44)

para alcançar o tux42. O redirecionamento efetivo dos pacotes é realizado pelo tux44, atuando como router entre as duas redes, garantindo a comunicação entre o tux43 e o tux42.

5. Que pacotes ICMP são observados e porquê?
A presença consistente dos endereços de origem e destino nos pacotes ICMP indica que a configuração da rede está correta, permitindo uma comunicação adequada entre o tux43 e o tux42.

6. Quais são os endereços IP e MAC associados aos pacotes ICMP e porquê?
Ao realizar o ping do tux43 para o tux42, os pacotes ICMP indicam o endereço IP de origem (tux43), o endereço IP de destino (tux42) e apresentam o endereço MAC do tux44, que serve como ponte entre as duas bridges.

Experiência 4 - Configurar um Router comercial e implementar NAT

Começamos esta experiência por conectar a porta ether1 do *router* MicroTik à porta P4.1, e a porta ether2 do *router* à ether9 da *switch*. No tux43, configuramos os endereços IP das portas ether1 e ether2, com 172.16.1.48/24 e 172.16.41.254/24, respetivamente. Após configurar os IPs, tratamos de configurar as rotas que nos foram pedidas. Para tornar o tux44 no *default router* do tux43, executamos 'route add default gw 172.16.40.254'; definimos o RC como *default router* do tux42 e tux44 executando 'route add default gw 172.16.41.254' nas duas máquinas; finalmente, adicionamos rotas para a rede 172.16.40.0/24 no tux42 e RC com o comando 'route add -net 172.16.40.0/24 gw 172.16.41.253' (no terminal do tux42) e '/ip route add dst-address=172.16.40.0/24 gateway=172.16.41.253' (na interface do *router* MicroTik). Além disto, também definimos uma *default route* para o RC, neste caso, através do endereço 172.16.1.254 (para conseguir ter acesso à *internet*), executando '/ip route add dst-address=0.0.0.0/0 gateway=172.16.1.254' na interface do *router*. No tux43 fizemos *ping* para os outros quatro endereços na rede (172.16.41.1 (imagem 2.4.1), 172.16.40.254 (imagem 2.4.2), 172.16.41.253 (imagem 2.4.3), e 172.16.41.254 (imagem 2.4.4) (todas as imagens podem ser encontradas em Anexos/Imagens)).

No tux42, executamos 'sysctl net.ipv4.conf.eth0.accept_redirects=0' e 'sysctl net.ipv4.conf.all.accept_redirects=0', e removemos a rota para a rede 172.16.40.0/24 com o comando 'route del -net 172.16.40.0/24 gw 172.16.41.253'; enviamos um 'ping' para o tux43, capturando os pacotes com o Wireshark. Esta captura pode ser vista na imagem 2.4.5. Uma vez que removemos a rota direta para a rede do tux43, as mensagens emitidas pelo tux42 são enviadas através do RC (que foi definido anteriormente como o seu *default router*). No final deste passo, revertemos as alterações com 'sysctl net.ipv4.conf.eth0.accept_redirects=1' e 'sysctl net.ipv4.conf.all.accept_redirects=1'.

Desativamos o NAT no RC com o comando '/ip firewall nat disable 0' na interface do *router*, e testamos no tux43 fazer *ping* para o endereço 172.16.1.254, mas sem sucesso. Revertemos o processo com o comando '/ip firewall nat enable 0', e testamos o mesmo *ping* outra vez, desta vez com sucesso.

Perguntas:

1. Como configurar uma rota estática num router comercial?
O procedimento envolve resetar as configurações do dispositivo, adicioná-lo à rede interna, por meio da bridge correspondente, e atribuir tanto um IP interno quanto um IP externo.

2.Quais são os caminhos seguidos pelos pacotes nas experiências realizadas e porquê?
Os pacotes de dados foram redirecionados através do router implementado usando ICMP redirect para alcançar o endereço IP de destino, sem a conexão direta do tux42 ao tux44. Na experiência seguinte, não foi necessário nenhum redirecionamento, porque a rota mais curta na rede estava disponível, permitindo a comunicação direta entre os dispositivos.

3. Como configurar o NAT num router comercial?
É possível configurar o NAT através do comando `/ip firewall nat enable 0` no terminal do router.

4.O que faz o NAT?
O NAT, que significa "Network Address Translation" (Tradução de Endereço de Rede), é um processo usado para mapear endereços IP dentro de uma rede local para um único endereço IP externo. Quando o destino responde, a resposta é direcionada para esse endereço público e, em seguida, traduzida de volta para o endereço local original. Isso permite a economia de endereços públicos, já que vários dispositivos internos podem compartilhar um único endereço público.

Experiência 5 - DNS

Nesta experiência, foi-nos pedido para configurarmos o DNS (Domain Name System) nos três computadores, com o servidor DNS da FEUP (193.136.28.10), para poder aceder a *hosts* com base no seu nome (ex: google.com). Para tal, executamos o comando ‘nano /etc/resolv.conf’ no terminal em cada computador, e no ficheiro resolv.conf adicionamos a linha “nameserver 193.136.28.10”.
Para testar este passo, fizemos ‘ping google.com’ nos três computadores, com sucesso.

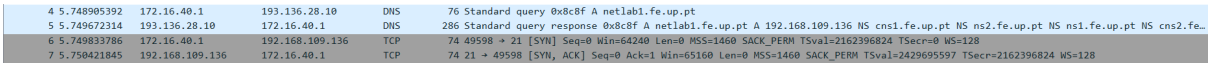
Perguntas:

1. Como configurar o serviço DNS num *host*?
O DNS é configurado adicionando a linha "nameserver <endereço IP do servidor>" no ficheiro `/etc/resolv.conf`.

2. Que pacotes são trocados por DNS e que informação é transportada?
O cliente envia pacotes do tipo *DNS Query Packet*, com informação como o tipo de endereço que procura, o nome do *host*, etc.; o servidor responde com *DNS Response Packets*, que contêm, principalmente, o endereço IP pedido pelo cliente.

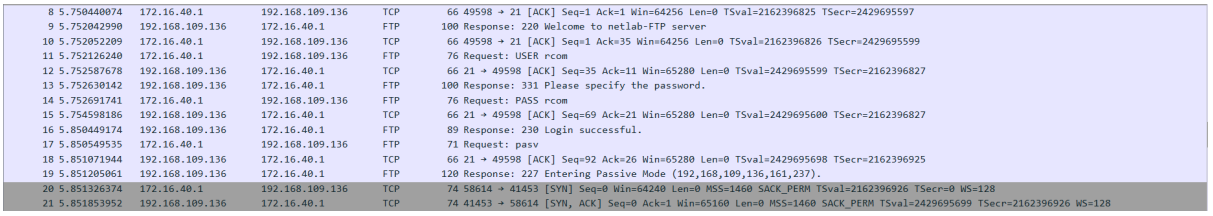
Experiência 6 - Conexões TCP

Aqui foi-nos pedido que testássemos a nossa aplicação de *download* na máquina tux43, com captura de pacotes usando o Wireshark.
Para tal, usamos o comando ‘./download ftp://rcom:rcom@netlab1.fe.up.pt/pipe.txt’.



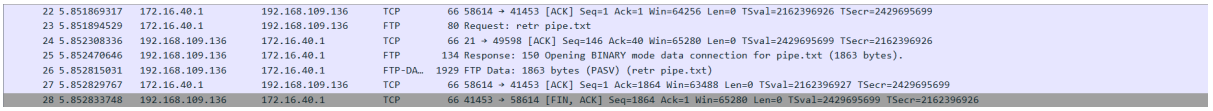
4	5.748905392	172.16.40.1	193.136.28.10	DNS	76 Standard query 0x8c8f A netlab1.fe.up.pt
5	5.749672314	193.136.28.10	172.16.40.1	DNS	286 Standard query response 0x8c8f A netlab1.fe.up.pt A 192.168.109.136 NS cns1.fe.up.pt NS ns2.fe.up.pt NS ns1.fe.up.pt NS cns2.fe…
6	5.749833786	172.16.40.1	192.168.109.136	TCP	74 49598 → 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2162396824 TSecr=0 WS=128
7	5.750421845	192.168.109.136	172.16.40.1	TCP	74 21 → 49598 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=2429695597 TSecr=2162396824 WS=128

tux43 faz um pedido DNS para encontrar o endereço IP do *host* netlab1.fe.up.pt (linhas azuis), e estabelece a ligação ao servidor através do protocolo TCP (linhas cinza).



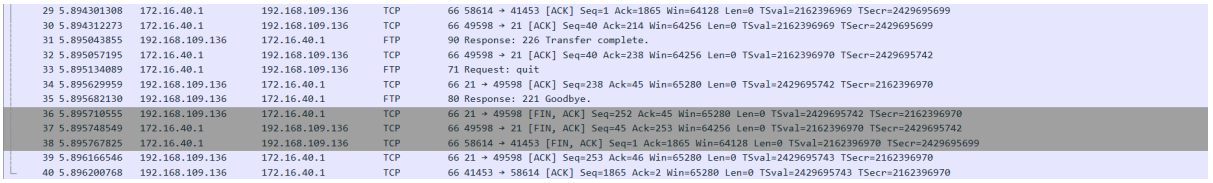
8	5.750440074	172.16.40.1	192.168.109.136	TCP	66 49598 → 21 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2162396825 TSecr=2429695597
9	5.752042990	192.168.109.136	172.16.40.1	FTP	100 Response: 220 Welcome to netlab-FTP server
10	5.752052209	172.16.40.1	192.168.109.136	TCP	66 49598 → 21 [ACK] Seq=1 Ack=35 Win=64256 Len=0 TSval=2162396826 TSecr=2429695599
11	5.752126240	172.16.40.1	192.168.109.136	FTP	76 Request: USER rcom
12	5.752587678	192.168.109.136	172.16.40.1	TCP	66 21 → 49598 [ACK] Seq=35 Ack=11 Win=65280 Len=0 TSval=2429695599 TSecr=2162396827
13	5.752630142	192.168.109.136	172.16.40.1	FTP	100 Response: 331 Please specify the password.
14	5.752691741	172.16.40.1	192.168.109.136	FTP	76 Request: PASS rcom
15	5.754598186	192.168.109.136	172.16.40.1	TCP	66 21 → 49598 [ACK] Seq=69 Ack=21 Win=65280 Len=0 TSval=2429695600 TSecr=2162396827
16	5.850449174	192.168.109.136	172.16.40.1	FTP	89 Response: 230 Login successful.
17	5.850549535	172.16.40.1	192.168.109.136	FTP	71 Request: pasv
18	5.851071944	192.168.109.136	172.16.40.1	TCP	66 21 → 49598 [ACK] Seq=92 Ack=26 Win=65280 Len=0 TSval=2429695698 TSecr=2162396925
19	5.851205061	192.168.109.136	172.16.40.1	FTP	120 Response: 227 Entering Passive Mode (192,168,109,136,161,237).
20	5.851326374	172.16.40.1	192.168.109.136	TCP	74 58614 → 41453 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2162396926 TSecr=0 WS=128
21	5.851853952	192.168.109.136	172.16.40.1	TCP	74 41453 → 58614 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=2429695699 TSecr=2162396926 WS=128

O terminal A do tux43 comunica com o servidor, enviando as informações de início de sessão, e fazendo o pedido 'pasv' (linhas rosa); o terminal B do tux43 estabelece a ligação com a porta obtida após o pedido 'pasv', através do protocolo TCP (linhas cinza).



22	5.851869317	172.16.40.1	192.168.109.136	TCP	66 58614 → 41453 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2162396926 TSecr=2429695699
23	5.851894529	172.16.40.1	192.168.109.136	FTP	80 Request: retr pipe.txt
24	5.852308336	192.168.109.136	172.16.40.1	TCP	66 21 → 49598 [ACK] Seq=146 Ack=40 Win=65280 Len=0 TSval=2429695699 TSecr=2162396926
25	5.852470646	192.168.109.136	172.16.40.1	FTP	134 Response: 150 Opening BINARY mode data connection for pipe.txt (1863 bytes).
26	5.852815031	192.168.109.136	172.16.40.1	FTP-DA…	1929 FTP Data: 1863 bytes (PASV) (retr pipe.txt)
27	5.852829767	172.16.40.1	192.168.109.136	TCP	66 58614 → 41453 [ACK] Seq=1 Ack=1864 Win=63488 Len=0 TSval=2162396927 TSecr=2429695699
28	5.852833748	192.168.109.136	172.16.40.1	TCP	66 41453 → 58614 [FIN, ACK] Seq=1864 Ack=1 Win=65280 Len=0 TSval=2429695699 TSecr=2162396926

O terminal A faz o pedido 'retr pipe.txt' ao servidor, que descarrega o ficheiro para a porta do terminal B (linha 26) (linhas rosa); a porta de *download* do servidor envia um pacote TCP FIN+ACK para o terminal B (linha cinza).



29	5.894301308	172.16.40.1	192.168.109.136	TCP	66 58614 → 41453 [ACK] Seq=1 Ack=1865 Win=64128 Len=0 TSval=2162396969 TSecr=2429695699
30	5.894312273	172.16.40.1	192.168.109.136	TCP	66 49598 → 21 [ACK] Seq=40 Ack=214 Win=64256 Len=0 TSval=2162396969 TSecr=2429695699
31	5.895043855	192.168.109.136	172.16.40.1	FTP	90 Response: 226 Transfer complete.
32	5.895057195	172.16.40.1	192.168.109.136	TCP	66 49598 → 21 [ACK] Seq=40 Ack=238 Win=64256 Len=0 TSval=2162396970 TSecr=2429695742
33	5.895134089	172.16.40.1	192.168.109.136	FTP	71 Request: quit
34	5.895629959	192.168.109.136	172.16.40.1	TCP	66 21 → 49598 [ACK] Seq=238 Ack=45 Win=65280 Len=0 TSval=2429695742 TSecr=2162396970
35	5.895682130	192.168.109.136	172.16.40.1	FTP	80 Response: 221 Goodbye.
36	5.895710555	192.168.109.136	172.16.40.1	TCP	66 21 → 49598 [FIN, ACK] Seq=252 Ack=45 Win=65280 Len=0 TSval=2429695742 TSecr=2162396970
37	5.895748549	172.16.40.1	192.168.109.136	TCP	66 49598 → 21 [FIN, ACK] Seq=45 Ack=253 Win=64256 Len=0 TSval=2162396970 TSecr=2429695742
38	5.895767825	172.16.40.1	192.168.109.136	TCP	66 58614 → 41453 [FIN, ACK] Seq=1 Ack=1865 Win=64128 Len=0 TSval=2162396970 TSecr=2429695699
39	5.896166546	192.168.109.136	172.16.40.1	TCP	66 21 → 49598 [ACK] Seq=253 Ack=46 Win=65280 Len=0 TSval=2429695743 TSecr=2162396970
40	5.896200768	192.168.109.136	172.16.40.1	TCP	66 41453 → 58614 [ACK] Seq=1865 Ack=2 Win=65280 Len=0 TSval=2429695743 TSecr=2162396970

O terminal A continua a comunicação com o servidor, enviando um pedido 'quit' (linhas rosa); o servidor e os terminais A e B trocam pacotes TCP FIN+ACK, fechando a conexão (linhas cinza e duas últimas linhas rosa).

Depois, foi-nos pedido que repetíssemos a transferência no tux43, mas iniciássemos uma transferência no tux42 durante a primeira.

Perguntas:

1. Quantas conexões TCP são abertas pela aplicação?
Duas, uma para ligar tux43 à porta 21 do servidor, e outra para ligar tux43 à porta de download (obtida pela resposta ao pedido 'pasv') do servidor.

2. Em que conexão é transportada a informação de controlo do FTP?
Na conexão entre tux43 e a porta 21 do servidor, através da qual são enviados os comandos relativos ao FTP.

3. Quais são as fases de uma conexão TCP?
Estabelecimento da conexão (SYN, SYN+ACK, ACK), transferência de dados, fecho da conexão (FIN, ACK(+FIN), ACK).

4. Como funciona o mecanismo ARQ TCP? Quais são os campos TCP relevantes? Que informação relevante pode ser observada nos *logs*?
Este mecanismo serve para reenviar pacotes TCP perdidos/corrompidos, caso a comunicação cliente-servidor ocorra numa rede com muito tráfego, na qual a perda de informação é mais provável. Para detetar um pacote perdido/corrompido, o protocolo TCP baseia-se em *timeouts* e nos valores associados às mensagens ACK.

5. Como funcionam os mecanismos de gestão de congestionamento do protocolo TCP? Quais são os campos relevantes? Como é que a taxa de transferência da conexão evolui ao longo do tempo? É de acordo com o mecanismo de controlo de congestão do TCP?
O emissor começa a enviar pacotes com uma taxa baixa ("slow start"), que vai aumentando exponencialmente até atingir a *congestion window*, um parâmetro do protocolo. A partir daqui, o emissor vai aumentar a taxa linearmente, para evitar congestionar a rede. Caso ocorra um *timeout*, o emissor vai voltar à fase inicial, e recomeçar o aumento da taxa de transmissão.

6. A taxa de transferência de uma conexão TCP é perturbada pela aparição de uma segunda conexão TCP? Como?
Sim, uma vez que a *bandwidth* terá de ser dividida por mais uma conexão, limitando assim a taxa de transferência da primeira.

Conclusões

A realização deste trabalho permitiu-nos compreender melhor o funcionamento de uma rede de computadores, bem como os protocolos envolvidos na transmissão de dados dentro da mesma. A componente laboratorial foi essencial para a compreensão dos mecanismos essenciais numa rede, como *routers* e *bridges*, e o papel que cada um destes desempenha para garantir o bom funcionamento da mesma.

Referências

Guião:

https://moodle2324.up.pt/pluginfile.php/72936/mod_resource/content/4/Lab%202%20-%20Gui%C3%A3o%20v5.1.pdf

Anexos/Comandos de configuração

tux42.S0 ligado a Switch.ether2
tux42.E0 ligado a T3
tux43.E0 ligado a Switch.ether3
tux44.E0 ligado a Switch.ether4
router.ether2 ligado a Switch.ether9

Experiência 1:

tux43: ifconfig eth0 down
tux43: ifconfig eth0 up
tux43: ifconfig eth0 172.16.40.1/24
tux44: ifconfig eth0 down
tux44: ifconfig eth0 up
tux44: ifconfig eth0 172.16.40.254/24
tux43: route -n
tux43: arp -a
tux44: route -n
tux44: arp -a
tux43: ping 172.16.40.254

Experiência 2:

tux42: ifconfig eth0 down
tux42: ifconfig eth0 up
tux42: ifconfig eth0 172.16.41.1/24
tux42 [Switch controller]: /interface bridge add name=bridge15
tux42 [Switch controller]: /interface bridge add name=bridge25
tux42 [Switch controller]: /interface bridge port remove [find interface=ether2]
tux42 [Switch controller]: /interface bridge port remove [find interface=ether3]
tux42 [Switch controller]: /interface bridge port remove [find interface=ether4]
tux42 [Switch controller]: /interface bridge port add bridge=bridge15 interface=ether3
tux42 [Switch controller]: /interface bridge port add bridge=bridge15 interface=ether4
tux42 [Switch controller]: /interface bridge port add bridge=bridge25 interface=ether2
tux43: ping 172.16.40.254 (step 5)
tux43: ping 172.16.41.1 (step 5)
tux43: ping -b 172.16.40.255 (step 8)
tux42: ping -b 172.16.41.255 (step 10)

Experiência 3:

tux44: ifconfig eth1 down
tux44: ifconfig eth1 up

tux44: ifconfig eth1 172.16.41.253/24
tux42 [Switch controller]: /interface bridge port remove [find interface=ether1]
tux42 [Switch controller]: /interface bridge port add bridge=bridge25 interface=ether1
tux44: sysctl net.ipv4.ip_forward=1
tux44: sysctl net.ipv4.icmp_echo_ignore_broadcasts=0
tux43: route add -net 172.16.41.0/24 gw 172.16.40.254
tux42: route add -net 172.16.40.0/24 gw 172.16.41.253
tux43: ping 172.16.40.254
tux43: ping 172.16.41.253
tux43: ping 172.16.41.1
tux42: arp -d 172.16.41.253
tux43: arp -d 172.16.40.254
tux44: arp -d 172.16.40.1
tux44: arp -d 172.16.41.1
tux43: ping 172.16.41.1

Experiência 4:

tux42 [Router controller]: /ip address add address=172.16.41.254/24 interface=ether2
tux42 [Router controller]: /ip address add address=172.16.1.48/24 interface=ether1
tux42 [Switch controller]: /interface bridge port remove [find interface=ether9]
tux42 [Switch controller]: /interface bridge port add bridge=bridge25 interface=ether9
tux43: route add default gw 172.16.40.254
tux42: route add default gw 172.16.41.254
tux44: route add default gw 172.16.41.254
tux42: route add -net 172.16.40.0/24 gw 172.16.41.253
tux42 [Router controller]: /ip route add dst-address=172.16.40.0/24 gateway=172.16.41.253
tux42 [Router controller]: /ip route add dst-address=0.0.0.0/0 gateway=172.16.1.254
tux43: ping 172.16.41.1
tux43: ping 172.16.40.254
tux43: ping 172.16.41.253
tux43: ping 172.16.41.254
tux42: sysctl net.ipv4.conf.eth0.accept_redirects=0
tux42: sysctl net.ipv4.conf.all.accept_redirects=0
tux42: route del -net 172.16.40.0/24 gw 172.16.41.253
tux42: ping 172.16.40.1
tux42: traceroute tux43
tux42: route add -net 172.16.40.0/24 gw 172.16.41.253
tux42: traceroute tux43
tux42: sysctl net.ipv4.conf.eth0.accept_redirects=1
tux42: sysctl net.ipv4.conf.all.accept_redirects=1
tux43: ping 172.16.1.254
tux42 [Router controller]: /ip firewall nat disable 0
tux43: ping 172.16.1.254
tux42 [Router controller]: /ip firewall nat enable 0

Experiência 5:

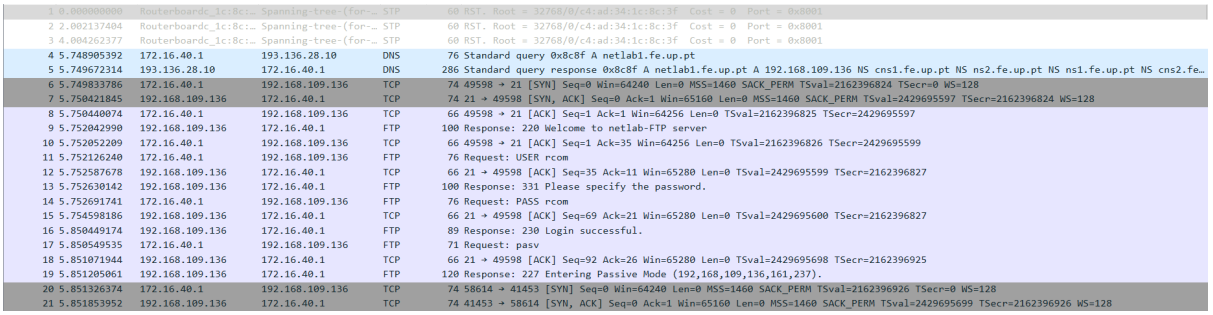
tux42: nano /etc/resolv.conf (adicionar linha: nameserver 193.136.28.10)
tux43: nano /etc/resolv.conf (adicionar linha: nameserver 193.136.28.10)
tux44: nano /etc/resolv.conf (adicionar linha: nameserver 193.136.28.10)
tux42: ping google.com (success)
tux43: ping google.com (success)
tux44: ping google.com (success)

Experiência 6:

tux43: make
tux43: ./download ftp://rcom:rcom@netlab1.fe.up.pt/pipe.txt
tux43: ./download ftp://rcom:rcom@netlab1.fe.up.pt/files/crab.mp4

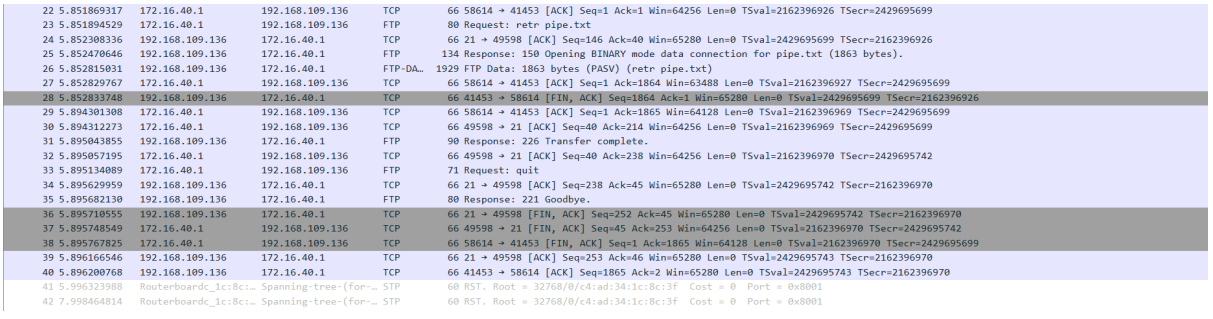
tux43: ./download ftp://rcom:rcom@netlab1.fe.up.pt/files/crab.mp4
tux42: ./download ftp://rcom:rcom@netlab1.fe.up.pt/files/crab.mp4

Anexos/Imagens



1	0.000000000	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
2	2.002137404	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
3	4.004262377	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
4	5.748905392	172.16.40.1	193.136.28.10	DNS	76 Standard query 0x8c8f A netlab1.fe.up.pt
5	5.749672314	193.136.28.10	172.16.40.1	DNS	286 Standard query response 0x8c8f A netlab1.fe.up.pt A 192.168.109.136 NS cns1.fe.up.pt NS ns2.fe.up.pt NS ns1.fe.up.pt NS cns2.fe…
6	5.749833786	172.16.40.1	192.168.109.136	TCP	74 49598 → 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2162396824 TSecr=0 WS=128
7	5.750421845	192.168.109.136	172.16.40.1	TCP	74 21 → 49598 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=2429695597 TSecr=2162396824 WS=128
8	5.750440074	172.16.40.1	192.168.109.136	TCP	66 49598 → 21 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2162396825 TSecr=2429695597
9	5.752042990	192.168.109.136	172.16.40.1	FTP	100 Response: 220 Welcome to netlab-FTP server
10	5.752052209	172.16.40.1	192.168.109.136	TCP	66 49598 → 21 [ACK] Seq=1 Ack=35 Win=64256 Len=0 TSval=2162396826 TSecr=2429695599
11	5.752126240	172.16.40.1	192.168.109.136	FTP	76 Request: USER rcom
12	5.752587678	192.168.109.136	172.16.40.1	TCP	66 21 → 49598 [ACK] Seq=35 Ack=11 Win=65280 Len=0 TSval=2429695599 TSecr=2162396827
13	5.752630142	192.168.109.136	172.16.40.1	FTP	100 Response: 331 Please specify the password.
14	5.752691741	172.16.40.1	192.168.109.136	FTP	76 Request: PASS rcom
15	5.754598186	192.168.109.136	172.16.40.1	TCP	66 21 → 49598 [ACK] Seq=69 Ack=21 Win=65280 Len=0 TSval=2429695600 TSecr=2162396827
16	5.850449174	192.168.109.136	172.16.40.1	FTP	89 Response: 230 Login successful.
17	5.850549535	172.16.40.1	192.168.109.136	FTP	71 Request: pasv
18	5.851071944	192.168.109.136	172.16.40.1	TCP	66 21 → 49598 [ACK] Seq=92 Ack=26 Win=65280 Len=0 TSval=2429695698 TSecr=2162396925
19	5.851205061	192.168.109.136	172.16.40.1	FTP	120 Response: 227 Entering Passive Mode (192,168,109,136,161,237).
20	5.851326374	172.16.40.1	192.168.109.136	TCP	74 58614 → 41453 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2162396926 TSecr=0 WS=128
21	5.851853952	192.168.109.136	172.16.40.1	TCP	74 41453 → 58614 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=2429695699 TSecr=2162396926 WS=128

Imagem 1.1



22	5.851869317	172.16.40.1	192.168.109.136	TCP	66 58614 → 41453 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2162396926 TSecr=2429695699
23	5.851894529	172.16.40.1	192.168.109.136	FTP	80 Request: retr pipe.txt
24	5.852308336	192.168.109.136	172.16.40.1	TCP	66 21 → 49598 [ACK] Seq=146 Ack=40 Win=65280 Len=0 TSval=2429695699 TSecr=2162396926
25	5.852470646	192.168.109.136	172.16.40.1	FTP	134 Response: 150 Opening BINARY mode data connection for pipe.txt (1863 bytes).
26	5.852815031	192.168.109.136	172.16.40.1	FTP-DA…	1929 FTP Data: 1863 bytes (PASV) (retr pipe.txt)
27	5.852829767	172.16.40.1	192.168.109.136	TCP	66 58614 → 41453 [ACK] Seq=1 Ack=1864 Win=63488 Len=0 TSval=2162396927 TSecr=2429695699
28	5.852833748	192.168.109.136	172.16.40.1	TCP	66 41453 → 58614 [FIN, ACK] Seq=1864 Ack=1 Win=65280 Len=0 TSval=2429695699 TSecr=2162396926
29	5.894301308	172.16.40.1	192.168.109.136	TCP	66 58614 → 41453 [ACK] Seq=1 Ack=1865 Win=64128 Len=0 TSval=2162396969 TSecr=2429695699
30	5.894312273	172.16.40.1	192.168.109.136	TCP	66 49598 → 21 [ACK] Seq=40 Ack=214 Win=64256 Len=0 TSval=2162396969 TSecr=2429695699
31	5.895043855	192.168.109.136	172.16.40.1	FTP	90 Response: 226 Transfer complete.
32	5.895057195	172.16.40.1	192.168.109.136	TCP	66 49598 → 21 [ACK] Seq=40 Ack=238 Win=64256 Len=0 TSval=2162396970 TSecr=2429695742
33	5.895134089	172.16.40.1	192.168.109.136	FTP	71 Request: quit
34	5.895629959	192.168.109.136	172.16.40.1	TCP	66 21 → 49598 [ACK] Seq=238 Ack=45 Win=65280 Len=0 TSval=2429695742 TSecr=2162396970
35	5.895682130	192.168.109.136	172.16.40.1	FTP	80 Response: 221 Goodbye.
36	5.895710555	192.168.109.136	172.16.40.1	TCP	66 21 → 49598 [FIN, ACK] Seq=252 Ack=45 Win=65280 Len=0 TSval=2429695742 TSecr=2162396970
37	5.895748549	172.16.40.1	192.168.109.136	TCP	66 49598 → 21 [FIN, ACK] Seq=45 Ack=253 Win=64256 Len=0 TSval=2162396970 TSecr=2429695742
38	5.895767825	172.16.40.1	192.168.109.136	TCP	66 58614 → 41453 [FIN, ACK] Seq=1 Ack=1865 Win=64128 Len=0 TSval=2162396970 TSecr=2429695699
39	5.896166546	192.168.109.136	172.16.40.1	TCP	66 21 → 49598 [ACK] Seq=253 Ack=46 Win=65280 Len=0 TSval=2429695743 TSecr=2162396970
40	5.896200768	192.168.109.136	172.16.40.1	TCP	66 41453 → 58614 [ACK] Seq=1865 Ack=2 Win=65280 Len=0 TSval=2429695743 TSecr=2162396970
41	5.996323988	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
42	7.998464814	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001

Imagem 1.2

5	6.647782864	HewlettPacka_61:2f:…	Broadcast	ARP	42 Who has 172.16.40.254? Tell 172.16.40.1
6	6.647905853	HewlettPacka_5a:7b:…	HewlettPacka_61:2f:…	ARP	60 172.16.40.254 is at 00:21:5a:5a:7b:ea
7	6.647915072	172.16.40.1	172.16.40.254	ICMP	98 Echo (ping) request id=0x148e, seq=1/256, ttl=64 (reply in 8)
8	6.648002513	172.16.40.254	172.16.40.1	ICMP	98 Echo (ping) reply id=0x148e, seq=1/256, ttl=64 (request in 7)
9	6.712185143	fe80::221:5aff:fe5a…	ff02::2	ICMPv6	70 Router Solicitation from 00:21:5a:5a:7b:ea
10	7.651404736	172.16.40.1	172.16.40.254	ICMP	98 Echo (ping) request id=0x148e, seq=2/512, ttl=64 (reply in 11)
11	7.651507193	172.16.40.254	172.16.40.1	ICMP	98 Echo (ping) reply id=0x148e, seq=2/512, ttl=64 (request in 10)
12	8.008821778	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:29 Cost = 0 Port = 0x8017
13	8.675409781	172.16.40.1	172.16.40.254	ICMP	98 Echo (ping) request id=0x148e, seq=3/768, ttl=64 (reply in 14)
14	8.675513146	172.16.40.254	172.16.40.1	ICMP	98 Echo (ping) reply id=0x148e, seq=3/768, ttl=64 (request in 13)
15	9.699401488	172.16.40.1	172.16.40.254	ICMP	98 Echo (ping) request id=0x148e, seq=4/1024, ttl=64 (reply in 16)
16	9.699504503	172.16.40.254	172.16.40.1	ICMP	98 Echo (ping) reply id=0x148e, seq=4/1024, ttl=64 (request in 15)
17	10.011028469	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:29 Cost = 0 Port = 0x8017
18	10.723400598	172.16.40.1	172.16.40.254	ICMP	98 Echo (ping) request id=0x148e, seq=5/1280, ttl=64 (reply in 19)
19	10.723529524	172.16.40.254	172.16.40.1	ICMP	98 Echo (ping) reply id=0x148e, seq=5/1280, ttl=64 (request in 18)
20	11.832251997	HewlettPacka_5a:7b:…	HewlettPacka_61:2f:…	ARP	60 Who has 172.16.40.1? Tell 172.16.40.254
21	11.832272111	HewlettPacka_61:2f:…	HewlettPacka_5a:7b:…	ARP	42 172.16.40.1 is at 00:21:5a:61:2f:d4

Imagem 2.1.1

4	5.006733113	0.0.0.0	255.255.255.255	MNDP	159 5678 → 5678 Len=117
5	5.006758046	Routerboardc_1c:8c:…	CDP/VTP/DTP/PAgP/UD…	CDP	93 Device ID: MikroTik Port ID: bridge40
6	5.006805398	Routerboardc_1c:8c:…	LLDP_Multicast	LLDP	110 MA/c4:ad:34:1c:8c:3f IN/bridge40 120 SysN=MikroTik SysD=MikroTik Ro
7	5.996936553	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
8	7.999252214	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
9	10.001561869	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
10	12.003868032	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
11	14.006184604	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
12	14.818568629	172.16.40.1	172.16.40.254	ICMP	98 Echo (ping) request id=0x15cf, seq=1/256, ttl=64 (reply in 13)
13	14.818727656	172.16.40.254	172.16.40.1	ICMP	98 Echo (ping) reply id=0x15cf, seq=1/256, ttl=64 (request in 12)
14	15.841742794	172.16.40.1	172.16.40.254	ICMP	98 Echo (ping) request id=0x15cf, seq=2/512, ttl=64 (reply in 15)
15	15.841901193	172.16.40.254	172.16.40.1	ICMP	98 Echo (ping) reply id=0x15cf, seq=2/512, ttl=64 (request in 14)
16	16.008494541	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
17	16.865740732	172.16.40.1	172.16.40.254	ICMP	98 Echo (ping) request id=0x15cf, seq=3/768, ttl=64 (reply in 18)
18	16.865874478	172.16.40.254	172.16.40.1	ICMP	98 Echo (ping) reply id=0x15cf, seq=3/768, ttl=64 (request in 17)
19	17.889743350	172.16.40.1	172.16.40.254	ICMP	98 Echo (ping) request id=0x15cf, seq=4/1024, ttl=64 (reply in 20)
20	17.889875140	172.16.40.254	172.16.40.1	ICMP	98 Echo (ping) reply id=0x15cf, seq=4/1024, ttl=64 (request in 19)
21	18.010797425	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
22	18.913742476	172.16.40.1	172.16.40.254	ICMP	98 Echo (ping) request id=0x15cf, seq=5/1280, ttl=64 (reply in 23)
23	18.913872031	172.16.40.254	172.16.40.1	ICMP	98 Echo (ping) reply id=0x15cf, seq=5/1280, ttl=64 (request in 22)
24	19.937741323	172.16.40.1	172.16.40.254	ICMP	98 Echo (ping) request id=0x15cf, seq=6/1536, ttl=64 (reply in 25)
25	19.937873461	172.16.40.254	172.16.40.1	ICMP	98 Echo (ping) reply id=0x15cf, seq=6/1536, ttl=64 (request in 24)
26	20.013094793	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
27	20.029277185	HewlettPacka_5a:7b:…	HewlettPacka_61:2f:…	ARP	60 Who has 172.16.40.1? Tell 172.16.40.254
28	20.029284379	HewlettPacka_61:2f:…	HewlettPacka_5a:7b:…	ARP	42 172.16.40.1 is at 00:21:5a:61:2f:d4
29	20.065698881	HewlettPacka_61:2f:…	HewlettPacka_5a:7b:…	ARP	42 Who has 172.16.40.254? Tell 172.16.40.1
30	20.065805807	HewlettPacka_5a:7b:…	HewlettPacka_61:2f:…	ARP	60 172.16.40.254 is at 00:21:5a:5a:7b:ea

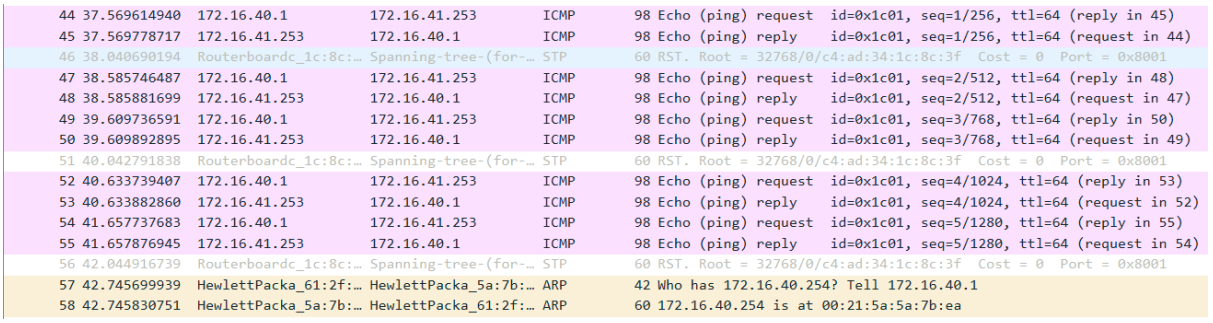
Imagem 2.2.1

9	14.298210561	172.16.40.1	172.16.41.1	ICMP	98 Echo (ping) request id=0x1bf3, seq=1/256, ttl=64 (reply in 10)
10	14.298567029	172.16.41.1	172.16.40.1	ICMP	98 Echo (ping) reply id=0x1bf3, seq=1/256, ttl=63 (request in 9)
11	15.321735456	172.16.40.1	172.16.41.1	ICMP	98 Echo (ping) request id=0x1bf3, seq=2/512, ttl=64 (reply in 12)
12	15.322018661	172.16.41.1	172.16.40.1	ICMP	98 Echo (ping) reply id=0x1bf3, seq=2/512, ttl=63 (request in 11)
13	16.017166546	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
14	16.345732126	172.16.40.1	172.16.41.1	ICMP	98 Echo (ping) request id=0x1bf3, seq=3/768, ttl=64 (reply in 15)
15	16.346006670	172.16.41.1	172.16.40.1	ICMP	98 Echo (ping) reply id=0x1bf3, seq=3/768, ttl=63 (request in 14)
16	17.369732915	172.16.40.1	172.16.41.1	ICMP	98 Echo (ping) request id=0x1bf3, seq=4/1024, ttl=64 (reply in 17)
17	17.370007390	172.16.41.1	172.16.40.1	ICMP	98 Echo (ping) reply id=0x1bf3, seq=4/1024, ttl=63 (request in 16)
18	18.018866884	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
19	18.393732658	172.16.40.1	172.16.41.1	ICMP	98 Echo (ping) request id=0x1bf3, seq=5/1280, ttl=64 (reply in 20)
20	18.394025361	172.16.41.1	172.16.40.1	ICMP	98 Echo (ping) reply id=0x1bf3, seq=5/1280, ttl=63 (request in 19)
21	19.449701051	HewlettPacka_61:2f:…	HewlettPacka_5a:7b:…	ARP	42 Who has 172.16.40.254? Tell 172.16.40.1
22	19.449820619	HewlettPacka_5a:7b:…	HewlettPacka_61:2f:…	ARP	60 172.16.40.254 is at 00:21:5a:5a:7b:ea

Imagem 2.4.1

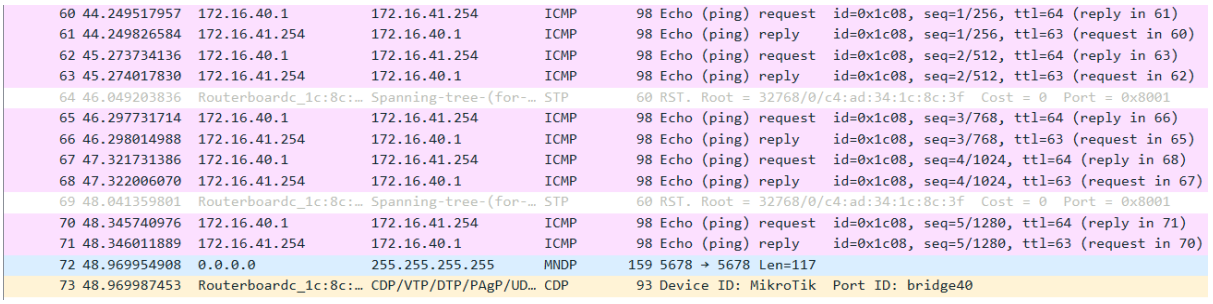
27	26.953876498	172.16.40.1	172.16.40.254	ICMP	98 Echo (ping) request id=0x1bfa, seq=1/256, ttl=64 (reply in 28)
28	26.954042439	172.16.40.254	172.16.40.1	ICMP	98 Echo (ping) reply id=0x1bfa, seq=1/256, ttl=64 (request in 27)
29	27.961737916	172.16.40.1	172.16.40.254	ICMP	98 Echo (ping) request id=0x1bfa, seq=2/512, ttl=64 (reply in 30)
30	27.961875642	172.16.40.254	172.16.40.1	ICMP	98 Echo (ping) reply id=0x1bfa, seq=2/512, ttl=64 (request in 29)
31	28.029037703	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
32	28.985733747	172.16.40.1	172.16.40.254	ICMP	98 Echo (ping) request id=0x1bfa, seq=3/768, ttl=64 (reply in 33)
33	28.985875105	172.16.40.254	172.16.40.1	ICMP	98 Echo (ping) reply id=0x1bfa, seq=3/768, ttl=64 (request in 32)
34	30.009737121	172.16.40.1	172.16.40.254	ICMP	98 Echo (ping) request id=0x1bfa, seq=4/1024, ttl=64 (reply in 35)
35	30.009898313	172.16.40.254	172.16.40.1	ICMP	98 Echo (ping) reply id=0x1bfa, seq=4/1024, ttl=64 (request in 34)
36	30.032097493	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
37	31.033731067	172.16.40.1	172.16.40.254	ICMP	98 Echo (ping) request id=0x1bfa, seq=5/1280, ttl=64 (reply in 38)
38	31.033864253	172.16.40.254	172.16.40.1	ICMP	98 Echo (ping) reply id=0x1bfa, seq=5/1280, ttl=64 (request in 37)
39	31.989995285	HewlettPacka_5a:7b:…	HewlettPacka_61:2f:…	ARP	60 Who has 172.16.40.1? Tell 172.16.40.254
40	31.990017564	HewlettPacka_61:2f:…	HewlettPacka_5a:7b:…	ARP	42 172.16.40.1 is at 00:21:5a:61:2f:d4

Imagem 2.4.2



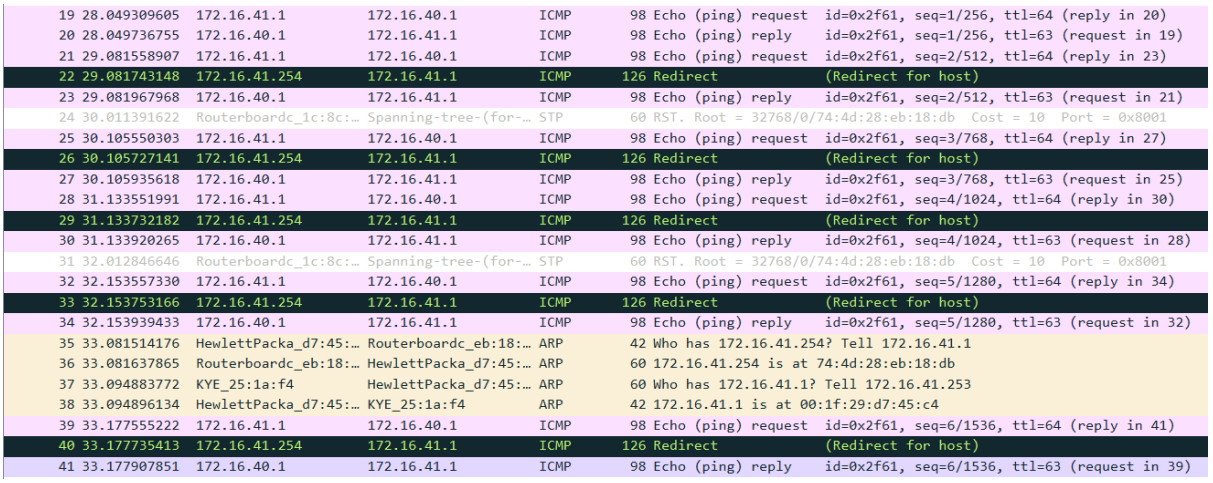
44	37.569614940	172.16.40.1	172.16.41.253	ICMP	98 Echo (ping) request id=0x1c01, seq=1/256, ttl=64 (reply in 45)
45	37.569778717	172.16.41.253	172.16.40.1	ICMP	98 Echo (ping) reply id=0x1c01, seq=1/256, ttl=64 (request in 44)
46	38.040690194	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
47	38.585746487	172.16.40.1	172.16.41.253	ICMP	98 Echo (ping) request id=0x1c01, seq=2/512, ttl=64 (reply in 48)
48	38.585881699	172.16.41.253	172.16.40.1	ICMP	98 Echo (ping) reply id=0x1c01, seq=2/512, ttl=64 (request in 47)
49	39.609736591	172.16.40.1	172.16.41.253	ICMP	98 Echo (ping) request id=0x1c01, seq=3/768, ttl=64 (reply in 50)
50	39.609892895	172.16.41.253	172.16.40.1	ICMP	98 Echo (ping) reply id=0x1c01, seq=3/768, ttl=64 (request in 49)
51	40.042791838	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
52	40.633739407	172.16.40.1	172.16.41.253	ICMP	98 Echo (ping) request id=0x1c01, seq=4/1024, ttl=64 (reply in 53)
53	40.633882860	172.16.41.253	172.16.40.1	ICMP	98 Echo (ping) reply id=0x1c01, seq=4/1024, ttl=64 (request in 52)
54	41.657737683	172.16.40.1	172.16.41.253	ICMP	98 Echo (ping) request id=0x1c01, seq=5/1280, ttl=64 (reply in 55)
55	41.657876945	172.16.41.253	172.16.40.1	ICMP	98 Echo (ping) reply id=0x1c01, seq=5/1280, ttl=64 (request in 54)
56	42.044916739	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
57	42.745699939	HewlettPacka_61:2f:…	HewlettPacka_5a:7b:…	ARP	42 Who has 172.16.40.254? Tell 172.16.40.1
58	42.745830751	HewlettPacka_5a:7b:…	HewlettPacka_61:2f:…	ARP	60 172.16.40.254 is at 00:21:5a:5a:7b:ea

Imagem 2.4.3



60	44.249517957	172.16.40.1	172.16.41.254	ICMP	98 Echo (ping) request id=0x1c08, seq=1/256, ttl=64 (reply in 61)
61	44.249826584	172.16.41.254	172.16.40.1	ICMP	98 Echo (ping) reply id=0x1c08, seq=1/256, ttl=63 (request in 60)
62	45.273734136	172.16.40.1	172.16.41.254	ICMP	98 Echo (ping) request id=0x1c08, seq=2/512, ttl=64 (reply in 63)
63	45.274017830	172.16.41.254	172.16.40.1	ICMP	98 Echo (ping) reply id=0x1c08, seq=2/512, ttl=63 (request in 62)
64	46.049203836	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
65	46.297731714	172.16.40.1	172.16.41.254	ICMP	98 Echo (ping) request id=0x1c08, seq=3/768, ttl=64 (reply in 66)
66	46.298014988	172.16.41.254	172.16.40.1	ICMP	98 Echo (ping) reply id=0x1c08, seq=3/768, ttl=63 (request in 65)
67	47.321731386	172.16.40.1	172.16.41.254	ICMP	98 Echo (ping) request id=0x1c08, seq=4/1024, ttl=64 (reply in 68)
68	47.322006070	172.16.41.254	172.16.40.1	ICMP	98 Echo (ping) reply id=0x1c08, seq=4/1024, ttl=63 (request in 67)
69	48.041359801	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8c:3f Cost = 0 Port = 0x8001
70	48.345740976	172.16.40.1	172.16.41.254	ICMP	98 Echo (ping) request id=0x1c08, seq=5/1280, ttl=64 (reply in 71)
71	48.346011889	172.16.41.254	172.16.40.1	ICMP	98 Echo (ping) reply id=0x1c08, seq=5/1280, ttl=63 (request in 70)
72	48.969954908	0.0.0.0	255.255.255.255	MNDP	159 5678 → 5678 Len=117
73	48.969987453	Routerboardc_1c:8c:…	CDP/VTP/DTP/PAgP/UD…	CDP	93 Device ID: MikroTik Port ID: bridge40

Imagem 2.4.4



19	28.049309605	172.16.41.1	172.16.40.1	ICMP	98 Echo (ping) request id=0x2f61, seq=1/256, ttl=64 (reply in 20)
20	28.049736755	172.16.40.1	172.16.41.1	ICMP	98 Echo (ping) reply id=0x2f61, seq=1/256, ttl=63 (request in 19)
21	29.081558907	172.16.41.1	172.16.40.1	ICMP	98 Echo (ping) request id=0x2f61, seq=2/512, ttl=64 (reply in 23)
22	29.081743148	172.16.41.254	172.16.41.1	ICMP	126 Redirect (Redirect for host)
23	29.081967968	172.16.40.1	172.16.41.1	ICMP	98 Echo (ping) reply id=0x2f61, seq=2/512, ttl=63 (request in 21)
24	30.011391622	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/74:4d:28:eb:18:db Cost = 10 Port = 0x8001
25	30.105550303	172.16.41.1	172.16.40.1	ICMP	98 Echo (ping) request id=0x2f61, seq=3/768, ttl=64 (reply in 27)
26	30.105727141	172.16.41.254	172.16.41.1	ICMP	126 Redirect (Redirect for host)
27	30.105935618	172.16.40.1	172.16.41.1	ICMP	98 Echo (ping) reply id=0x2f61, seq=3/768, ttl=63 (request in 25)
28	31.133551991	172.16.41.1	172.16.40.1	ICMP	98 Echo (ping) request id=0x2f61, seq=4/1024, ttl=64 (reply in 30)
29	31.133732182	172.16.41.254	172.16.41.1	ICMP	126 Redirect (Redirect for host)
30	31.133920265	172.16.40.1	172.16.41.1	ICMP	98 Echo (ping) reply id=0x2f61, seq=4/1024, ttl=63 (request in 28)
31	32.012846646	Routerboardc_1c:8c:…	Spanning-tree-(for-…	STP	60 RST. Root = 32768/0/74:4d:28:eb:18:db Cost = 10 Port = 0x8001
32	32.153557330	172.16.41.1	172.16.40.1	ICMP	98 Echo (ping) request id=0x2f61, seq=5/1280, ttl=64 (reply in 34)
33	32.153753166	172.16.41.254	172.16.41.1	ICMP	126 Redirect (Redirect for host)
34	32.153939433	172.16.40.1	172.16.41.1	ICMP	98 Echo (ping) reply id=0x2f61, seq=5/1280, ttl=63 (request in 32)
35	33.081514176	HewlettPacka_d7:45:…	Routerboardc_eb:18:…	ARP	42 Who has 172.16.41.254? Tell 172.16.41.1
36	33.081637865	Routerboardc_eb:18:…	HewlettPacka_d7:45:…	ARP	60 172.16.41.254 is at 74:4d:28:eb:18:db
37	33.094883772	KYE_25:1a:f4	HewlettPacka_d7:45:…	ARP	60 Who has 172.16.41.1? Tell 172.16.41.253
38	33.094896134	HewlettPacka_d7:45:…	KYE_25:1a:f4	ARP	42 172.16.41.1 is at 00:1f:29:d7:45:c4
39	33.177555222	172.16.41.1	172.16.40.1	ICMP	98 Echo (ping) request id=0x2f61, seq=6/1536, ttl=64 (reply in 41)
40	33.177735413	172.16.41.254	172.16.41.1	ICMP	126 Redirect (Redirect for host)
41	33.177907851	172.16.40.1	172.16.41.1	ICMP	98 Echo (ping) reply id=0x2f61, seq=6/1536, ttl=63 (request in 39)

Imagem 2.4.5

Anexos/Código

Disposição do código

dir
 |-bin
 |-include
   |-clientTCP.h
   |-getIP.h
   |-main.h
  |-src
   |-clientTCP.c
   |-getIP.c
  |-main.c

main.c

```
#include "main.h"

int checkResponse(const int socket, char* buf) {
    char byte;
    int i = 0;
    int code;
    char message[300];
    while(1) {
        i = 0;
        memset(buf, 0, 300); // clear buf

        do {
            read(socket, &byte, 1);
            if(byte != '\n') {
                buf[i++] = byte;
            }
        } while (byte != '\n'); // read the whole server response into
buf

        printf("%s\n", buf);
        if((sscanf(buf, "%d-%[^\n]", &code, message) != 2)) {
            break;
        }
    }

    return code; // return the response code
}

int connectToServer(const int socket, const char* user, const char*
password) {
    char user_input[5+strlen(user)+1];
    sprintf(user_input, "USER %s\r\n", user); // user_input = "USER
<user>"

    char password_input[5+strlen(password)+1];
    sprintf(password_input, "PASS %s\r\n", password); // password_input
= "PASS <password>"

    char response[300];

    printf("%s", user_input);
```

```
    write(socket, user_input, strlen(user_input)); // send "USER
<user>" message to server

    /* after sending USER message, read server's response;
    if server responds with a code other than 331, end execution */
    if (checkResponse(socket, response) != 331) {
        printf("Unable to authenticate user '%s', server responded with
code %s.\n", user, response);
        exit(-1);
    }
    printf("%s", password_input);
    write(socket, password_input, strlen(password_input)); // send
"PASS <password>" message to server

    /* after sending PASS message, read (and return) server's response
*/
    return checkResponse(socket, response);
}

int enterPassiveMode(const int socket, char* ip_address, int* port) {
    char response[300];
    int ip_byte1, ip_byte2, ip_byte3, ip_byte4, port_byte1, port_byte2;

    write(socket, "pasv\r\n", 6); // send "pasv" message to server
    printf("pasv\n");

    /* after sending "pasv" message, read server's response;
    if server responds with a code other than 227, end execution */
    if (checkResponse(socket, response) != 227){
        return -1;
    }

    // obtain values for the IP and Port bytes, and use them to set the
values of the ip_address and port arguments
    sscanf(response, "%*[^(](%d,%d,%d,%d,%d,%d)%*[^\n$)]", &ip_byte1,
&ip_byte2, &ip_byte3, &ip_byte4, &port_byte1, &port_byte2);
    *port = (port_byte1 * 256) + port_byte2;
    sprintf(ip_address, "%d.%d.%d.%d", ip_byte1, ip_byte2, ip_byte3,
ip_byte4);
    return 227;
}

int requestFile(const int socket, char* path_to_file) {
```

```
    char file_input[5+strlen(path_to_file)+1];
    char response[300];
    sprintf(file_input, "retr %s\r\n", path_to_file); // file_input =
"retr <path>"
    printf("%s", file_input);
    write(socket, file_input, strlen(file_input)); // write "retr
<path>" message to server

    /* after sending "retr" message, read and return server's response
*/
    return checkResponse(socket, response);
}

int main(int argc, char* argv[]) {
    if (argc > 2) { // wrong usage of command
        printf("ERROR: usage is %s
ftp://[<user>:<password>@]<host>/<url-path>", argv[0]);
        exit(-1);
    }

    char user[300]; // username for authentication
    char password[300]; // password for authentication
    char host[300]; // connection host
    char path[300]; // path to file
    char file_to_download[300]; // name of the file to download
    char ip[300]; // IP address of the server

    regex_t regex;
    regcomp(&regex, "/", 0);

    if(regexec(&regex, argv[1], 0, NULL, 0)) {
        return -1;
    }

    regcomp(&regex, "@", 0);

    if (regexec(&regex, argv[1], 0, NULL, 0) != 0) { // there is no '@'
in the provided argument; it is of type ftp://<host>/<url-path>
        sscanf(argv[1], "ftp://%300[^/]", host);
        strcpy(user, "anonymous");
        strcpy(password, "anonymous");

    }
```

```
    else { // there is a '@' in the provided argument; it is of type
ftp://<user>:<password>@<host>/<url-path>
        sscanf(argv[1], "ftp://%300[^:]:%300[^@]@%300[^/]", user,
password, host);
    }

    sscanf(argv[1], "ftp://%*[^/]/%300s", path); // obtain path to file
    strcpy(file_to_download, strrchr(argv[1], '/') + 1); // obtain file
name

    (void) getIP(host, ip); // set ip's value to the IP address of the
host

    printf("Hostname: %s\nPath: %s\nFile to Download: %s\nUser:
%s\nPassword: %s\nIP: %s\n", host, path, file_to_download, user,
password, ip);

    char response_buf[300];

    int socket_A = clientTCP(ip, 21); // set socket_A to be the file
descriptor of port 21 on the IP address of the host
    int socket_A_val = socket_A;

    if(socket_A < 0 || checkResponse(socket_A, response_buf) != 220) {
// if server does not respond with code 220, something went wrong
        printf("Error connecting to port 21.\n");
        exit(-1);
    }

    /* try to connect to the socket using the credentials obtained
above;
    if the server responds with a code other than 230, end execution */
    if(connectToServer(socket_A, user, password) != 230) {
        printf("Error authenticating.\n");
        exit(-1);
    }

    uint16_t download_port;
    char download_ip[300];


    /* enter Passive Mode on socket_A, and store the IP address and
Port for the connection to socket_B
```

```
    in download_ip and download_port, respectively;
    if the server responds with a code other than 227, end execution */
    if (enterPassiveMode(socket_A, download_ip, &download_port) != 227)
{
        printf("Couldn't enter passive mode.\n");
        exit(-1);
    }

    printf("downloading from port %d on address %s\n", download_port,
download_ip);
    socket_A = socket_A_val;

    int socket_B = clientTCP(download_ip, download_port); // set
socket_B as the file descriptor of port <download_port> on the IP
address <download_ip>
    if (socket_B < 0) {
        printf("Error connecting to port %d\n", download_port);
        exit(-1);
    }

    /* send "retr" request to socket_A, requesting the file present in
path;
    if the server responds with a code other than 150, end execution */
    int requestFileResponse = requestFile(socket_A, path);
    if (requestFileResponse!=150 && requestFileResponse!=125) {
        printf("Couldn't find requested file in the port.\n");
        exit(-1);
    }

    FILE* fd = fopen(file_to_download, "w"); // open the file
<file_to_download> with "w" so we can write to it
    if (fd == NULL) {
        printf("Couldn't open the requested file.\n");
        exit(-1);
    }

    char data_buffer[300];
    int bytes;

    do {
        bytes = read(socket_B, data_buffer, 300); // read <=300 bytes
each time from socket_B
```

```
        if (fwrite(data_buffer, bytes, 1, fd) < 0) { // write those
bytes to the opened file
            return -1;
        }
        printf("wrote %d bytes to file %s\n", bytes, file_to_download);
    } while (bytes);

    fclose(fd); // after we finish writing to the file, close it

    /* after the file is fully downloaded, we must check the message
from the server;
    if the server responds with a code other than 226, end execution */
    if (checkResponse(socket_A, response_buf) != 226) {
        printf("Error transfering file.\n");
        exit(-1);
    }

    write(socket_A, "quit\r\n", 7); // send "quit" message to socket_A
in order to close the connection
    printf("quit\n");

    /* after sending "quit" message, check for the server's response;
    if the server responds with a code other than 221, end execution */
    if(checkResponse(socket_A, response_buf) != 221) {
        return -1;
    }

    /* close the sockets */
    if(close(socket_A)!=0 || close(socket_B)!=0) {
        printf("Error: Couldn't close sockets.\n");
        exit(-1);
    }

    printf("SUCCESS\n");
    return 0;
}
```

main.h

```
#include "clientTCP.h"
#include "getIP.h"
#include <regex.h>
```

```
int checkResponse(const int socket, char* buf);

int connectToServer(const int socket, const char* user, const char*
password);

int enterPassiveMode(const int socket, char* ip_address, int* port);

int main(int argc, char* argv[]);
```

getIP.c

```
#include "getIP.h"

int getIP(const char* hostname, char* ip) {
    struct hostent *h;

/**
 * The struct hostent (host entry) with its terms documented

    struct hostent {
        char *h_name;    // Official name of the host.
        char **h_aliases;    // A NULL-terminated array of alternate
names for the host.
        int h_addrtype;    // The type of address being returned;
usually AF_INET.
        int h_length;    // The length of the address in bytes.
        char **h_addr_list;    // A zero-terminated array of network
addresses for the host.
        // Host addresses are in Network Byte Order.
    };

    #define h_addr h_addr_list[0]   The first address in h_addr_list.
*/
    if ((h = gethostbyname(hostname)) == NULL) {
        herror("gethostbyname()");
        exit(-1);
    }

    // printf("Host name  : %s\n", h->h_name);
    // printf("IP Address : %s\n", inet_ntoa(*((struct in_addr *)
h->h_addr)));

    strcpy(ip, inet_ntoa(*((struct in_addr *) h->h_addr)));
    return (int) inet_ntoa(*((struct in_addr *) h->h_addr));
```

```
}
```

getIP.h

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

int getIP(const char* hostname, char* ip);
```

clientTCP.c

```
#include "clientTCP.h"

int clientTCP(const char* ip, uint16_t 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 a TCP socket*/
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket()");
        exit(-1);
    }
    /*connect to the server*/
    if (connect(sockfd,
                (struct sockaddr *) &server_addr,
                sizeof(server_addr)) < 0) {
        perror("connect()");
        exit(-1);
    }
```

```
    return sockfd;
}
```

clientTCP.h

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>

#include <string.h>

#define SERVER_PORT 6000
#define SERVER_ADDR "192.168.28.96"

int clientTCP(const char* ip, uint16_t port);
```

Makefile

```
# Makefile to build the project
# NOTE: This file must not be changed.

# Parameters
CC = gcc
CFLAGS = -Wall

SRC = src/
INCLUDE = include/
BIN = bin/

.PHONY: application
application: $(BIN)/download

$(BIN)/download: main.c $(SRC)/*.c
    $(CC) $(CFLAGS) -o download $^ -I$(INCLUDE)

.PHONY: clean
clean:
    rm -rf ./download
```