Trabalho Prático 2

Redes de Computadores

Turma 1

João Pedro Pinheiro de Lacerda Campos *up201704982@fe.up.pt*

Nuno Miguel Teixeira Cardoso *up201706162@fe.up.pt*

19.12.2019

Índice

Introdução

Este trabalho, desenvolvido na unidade curricular de Redes de Computadores, tem duas finalidades: a configuração de uma rede e o desenvolvimento de uma aplicação de download.

Para a configuração de uma rede foram fornecidas várias experiências a executar. Com cada uma delas aprendemos mais detalhadamente a forma correta de implementação de um sistema de redes de computadores, com o objetivo final de conseguir descarregar um ficheiro de um servidor através de uma app que operasse sobre a rede em questão.

Quanto à aplicação de download, foi desenvolvida utilizando o protocolo *FTP* (*File Transfer Protocol*) através de ligações *TCP* (*Transmission Control Protocol*), a partir de sockets.

Aplicação de download

A aplicação de download foi desenvolvida em C de forma a aceitar como argumento um URL que autenticasse o utilizador e identificasse o ficheiro a receber e o servidor a que aceder. Para esta parte do trabalho fizemos uso dos documentos RFC959, que aborda o protocolo FTP, e RFC1738, que descreve o uso de URL.

Arquitetura

Primeiro é feita a leitura e interpretação do URL. Este tem o formato:

ftp://<user>:<password>@<host>/<path-to-file>

A parte **<user>:<password>@** pode ser omitida para entrar no servidor em modo anónimo.

O URL é passado para a função *get_args()*, juntamente com apontadores para *char* que guardarão as informações obtidas do URL. Primeiro verifica-se o inicio do URL com **ftp://**. A seguir é verificado se o utilizador e password estão explícitos, se não estiverem, o utilizador é definido como **"anonymous"** com password **""**(string vazia). Estas informações são guardadas nas variáveis *user* e *pass*. Depois é retirado o host do ficheiro, que é a string até à primeira barra (**'/'**) encontrada, e guardado em *host*. Finalmente é retirado o caminho até ao ficheiro, sendo que a string desde a última barra até ao final, é o nome do ficheiro. Colocados em *path* e *file*, respetivamente.

Feita a leitura do URL temos quase toda a informação necessária para o funcionamento do programa. Para a conexão ao servidor não podemos usar diretamente a string vinda de *host*. Usamos a função *getip()*, que recebe como argumento *host* e retorna através do char* *ip* o endereço IP do servidor. Para a conexão é utilizada a porta 21.

Podemos finalmente começar a comunicação com o servidor. Para isso usamos a função *start_connection()* que abre um socket, preparado para comunicação nos dois sentidos com através de IP, e liga este socket ao servidor.

Depois de estar ligado ao servidor, o programa segue uma ordem predefinida de comandos por forma a obter o ficheiro do servidor. Sempre que envia um comando, recebe a resposta vinda do servidor. Dependendo do número no início da resposta esta pode ser de confirmação (começando pelos números 1, 2 ou 3) ou de erro (começando por 4 ou 5). No caso de ser uma mensagem de erro, o programa para.

Estes comandos são os seguintes, distribuídos pelas diferentes funções no código:

USER <user> - inicia o processo de autenticação no servidor, enviando *user*;

PASS <pass> - envia a password para autenticação do utilizador;

CWD <path> - muda o caminho atual no servidor, para o diretório com o ficheiro;

PASV - entra em modo passivo, para permitir o download do ficheiro;

Na função *passive_mode()*, em contraste com as outras funções, a informação recebida pelo servidor é utilizada. Na resposta do servidor são dados 6 números que são utilizados para formar um novo endereço IP e porta. A seguir, o programa liga-se a este novo endereço usando um novo socket.

RETR <file> - faz o pedido do ficheiro ao servidor

Após o pedido do ficheiro, a função *receive_file()* trata de receber o ficheiro através do novo socket ligado no modo passivo. Em anexo são apresentados os protótipos das funções usadas para obtenção dos argumentos e para comunicação com o servidor.

Resultado de download

Faremos a análise ao modo anônimo do programa. Para isso corremos o comando

time ./download ftp://ftp.up.pt/pub/debian-multimedia/convert-stats

Com este comando é feito o download do ficheiro convert-stats. Na Figura 1 é possível observar que é escrita no ecrã toda a informação retirada do URL. Todas as respostas vindas do servidor também são escritas no ecrã para facilitar a resolução de problemas, pois cada resposta é acompanhada por um número que pode indicar que está tudo bem ou que aconteceu um erro.

Depois de receber a resposta de entrada no servidor, é feita a autenticação do utilizador, seguida da mudança de diretório. Podemos ver a resposta ao comando **PASV**, com os seis números entre parênteses. A seguir, é feito o pedido e receção do ficheiro e termina o programa.

O ficheiro é recebido com a mensagem **150 Opening BINARY mode data connection for ENDINGS (3954 bytes)**. Assim, vemos que o ficheiro tem um tamanho de 3954 bytes. Na mensagem seguinte vemos que o programa demorou 0.292 segundos a correr, o que indica uma velocidade de transferência de mais de 13 kB/s.

Configuração da rede

Experiência 1 - Configuração de um IP de rede

- **Breve descrição:**

Na primeira experiência é nos pedido para configurar, utilizando *ifconfig* e *route*, os computadores **tux1** e **tux4**. Ligamos as cartas de rede dos dois computadores ao switch. Estes computadores ficam ligados pelo switch na rede 172.16.y0.0/24 (sendo que y depende da bancada utilizada). Para os **tux1** utilizamos a carta *eth0*, à qual atribuímos o endereço de ip 172.16.y0.1. Para o **tux4** usamos a carta *eth0,* à qual atribuímos o endereço de ip 172.16.y0.254. Ao fazer *ping* podemos confirmar que os computadores estão ligados e conseguem comunicar entre si.

- **Conceitos em estudo:**

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

ARP é um acrônimo para *Address Resolution Protocol*, que corresponde a um protocolo de comunicação usado para mapear dinamicamente endereços de rede a um endereço físico (MAC). É usado para descobrir o endereço da camada de ligação associado ao endereço IPv4.

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

Ao testar a conectividade na experiência 1, usamos o comando *ping* do **tux1** para o **tux4** e vice versa. A conclusão alcançada é igual independentemente do sentido. Para o caso em que executamos *ping* do **tux1** para o **tux4,** o **tux1** envia um pacote ARP com

destino *broadcast* a perguntar qual o endereço MAC do tux para o qual está a tentar mandar algo. Esse pedido é constituído pelos endereços IP e MAC do **tux1**, que é o *sender* (172.16.40.1 e 00:0f:fe:8c:af:af, respetivamente) e pelo endereço IP do tux com o qual está a tentar comunicar, ou seja o **tux4** (172.16.40.254), que é o *target*. Uma vez que se desconhece o endereço MAC do tux *target*, este vai no formato 00:00:00:00:00:00. Conclusões retiradas dos *logs* presentes na figura 2 em anexo.

Seguidamente, o **tux4** responde ao pedido enviando um pacote ARP com o seu IP (172.16.40.254), o seu endereço MAC (00:21:5a:5a:7b:ea) e os endereços IP e MAC do **tux1** *target* (ver figura 3).

3. Que pacotes é que o comando *ping* gera?

Inicialmente gera os pacotes ARP anteriormente mencionados. Depois gera dois pacotes ICMP de *request* e *reply* (*Internet Control Message Protocol*). Ver figura 4.

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

Os endereços MAC e IP dos pacotes *ping* correspondem aos endereços dos tux de origem e destino. Para o caso de um *ping* do **tux1** para o **tux4**:

Pacote de pedido (figura 5):

MAC de origem: 00:c0:df:25:40:81 (**tux1**)
MAC de destino: 00:21:5a:c3:78:70 (**tux4**)
IP de origem: 172.16.40.1 (t**ux1**)
IP de destino 172.16.40.4 (**tux4**)

Pacote de resposta (figura 6):

MAC de origem: 00:21:5a:c3:78:70 (**tux4**)
MAC de destino: 00:c0:df:25:40:81 (**tux1**)
IP de origem 172.16.40.4 (**tux4**)
IP de destino: 172.16.40.1 (**tux1**)

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

Examinando o cabeçalho Ethernet de um pacote é possível determinar o tipo da trama recetora. Um valor 0x0800 corresponde a uma trama de tipo IP. Analisando, de seguida, o cabeçalho do IP, caso o valor deste seja 1, então o protocolo usado é o ICMP. Posteriormente, caso o valor do tipo do cabeçalho Ethernet seja 0x0806, então o tipo da trama é ARP. Consultar figuras 7 e 8.

6. Como determinar o comprimento de uma trama recebida?

Através do wireshark, na aba Frame, é possível descobrir o tamanho da trama recebida. Ver figura 9.

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

A interface *loopback* é uma interface de rede que permite que um computador possa receber respostas de si mesmo. É regularmente usada para verificar se a carta de rede do computador está corretamente configurada. Ver figura 10.

Experiência 2 - Implementação de duas VLAN's

- **Breve descrição:**

Nesta experiência, pegando na configuração dos **tux1** e **tux4**, configuramos também o **tux2**, numa rede diferente 172.16.y1.0/24. Para o **tux2** usamos a carta de rede *eth0*, à qual atribuímos o endereço de ip 172.16.y1.1.

Abrindo o terminal do switch, criamos duas *vlan's vlany0* e *vlany1*. Utilizando o número da porta de cada tux, adicionamos a porta dos **tux1** e **tux4** à *vlany0* e à *vlany1* ligamos a porta do **tux2**.

Como **tux1** e **tux4** estão na mesma rede, continua a ser possível usar *ping* de um para o outro. Se tentarmos usar *ping* tanto do **tux1**, como do **tux4**, para o **tux2**, reparamos que não é possível, pois estão em redes diferentes.

- **Conceitos em estudo:**

1. **Como configurar a vlany0?**

Devem ser seguidos os passos da figura em anexo, relativa a configurações de VLAN. Na régua 1, a porta T4 tem de estar ligada à porta *Switch Console* da régua 2, enquanto que a porta T3 da régua 1 tem de estar ligada à porta S0 do tux onde se pretende configurar o *Switch*. No GTKTerm, escrevem-se os seguintes comandos para criação da vlan y0:

configure terminal
vlan y0
end

De seguida, adicionam-se as portas dos **tux1** e **tux4** para permitir o seu acesso:

configure terminal
interface fastethernet 0/[nº da porta]
switchport mode access
switchport access vlan y0
end

2. **Quantos domínios de transmissão existem? Como se pode concluir isso a partir dos registos?**

Existem dois domínios de transmissão, pois quando o **tux1** executa um *ping broadcast* recebe resposta do **tux4** apenas e quando o **tux2** executa um *ping broadcast* não recebe resposta de ninguém. Existem portanto dois domínios: aquele que contém o **tux2** e o que contém os **tux1** e **tux4**. Tal facto resulta da divisão em duas sub-redes, com duas VLANs distintas. Ver figura 11.

Experiência 3 - Configuração do Tux4 como um router

- **Breve descrição:**

Aqui, continuando a partir da experiência anterior, ligamos a segunda carta de rede do **tux4** (carta eth1) a uma nova porta do switch e atribuímos-lhe o endereço de ip 172.16.y1.253. Fazemos as preparações de *ifconfig* e *route* e adicionamos esta porta à *vlany1*. Agora ativamos o *IP forwarding* e desativamos o *ICMP echo-ignore-broadcast*, tornando o **tux4** um router que pode passar informação entre as redes *y0* e *y1*.

Configuramos o **tux1** e **tux2** para utilizarem o **tux4** como *gateway* para se conseguirem aceder mutuamente. Agora é possível usar *ping* para comunicar entre **tux1** e **tux2**.

- **Conceitos em estudo:**

1. **Que rotas há nos tux? Quais os seus significados?**

Tux1: rota para a vlan 0 (172.16.y0.0) pela *gateway* 0.0.0.0 (ele próprio) e rota para a vlan 1 (172.16.y1.0) pela *gateway* 172.16.y0.254.

Tux4: rota para a vlan 0 (172.16.y0.0) pela *gateway* 172.16.y0.254 e rota para a vlan 1 (172.16.y1.0) pela *gateway* 172.16.y1.253.

Tux2: rota para a vlan 1 (172.16.y1.0) pela *gateway* 0.0.0.0 (ele próprio) e rota para a vlan 0 (172.16.y0.0) pela *gateway* 172.16.y1.253.

Numa rota, o ip de destino representa o ip do tux que se pretende alcançar. *Gateway* representa uma porta/via por onde a informação deverá prosseguir para alcançar o destino pretendido.

2. **Que informação contém uma entrada da tabela de *forwarding*?**

Destination: destino da rota.
Gateway: porta/via por onde a informação deverá prosseguir para alcançar o destino pretendido.
Netmask: usada para determinar o ID da rede, tendo o endereço IP do destino.
Flags: informações sobre a rota.
Metric: custo de cada rota.
Use: contador de pesquisas pela rota; dependendo do uso de -F ou -C este será o número de falhas de cache ou o número de sucessos, respetivamente.
Interface: placa de rede responsável pela *gateway* (eth0/eth1).

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

A explicação para esta pergunta já se encontra na resposta à pergunta 2 da experiência 1. A única coisa a mudar aqui é para o caso, por exemplo, em que o **tux1** executa um *ping* para o **tux2** e vice versa. Nestes casos, para registos observados no **tux1**, as mensagens ARP irão conter os endereços correspondentes aos **tux1** e **tux4**, apesar de o destino do *ping* ser o **tux2**, pois o **tux4** serve de *gateway* à comunicação entre os **tux1** e **tux2**. Ver figura 12.

4. **Quais pacotes ICMP são observados e porquê?**

São observados pacotes ICMP de *request* e *reply* que dependem diretamente dos tux usados na comunicação, através do comando *ping*. Ao contrário dos pacotes ARP, os pacotes ICMP podem conter o endereço de IP de qualquer um dos tux, uma vez que neste momento a rede já permite a comunicação entre qualquer um deles. Caso os tux não conseguissem contactar seriam enviados os pacotes ICMP de *Host Unreachable*. Ver figura 12 com exemplos de *ping* a partir o **tux1** para o **tux4** (172.16.40.254 e 172.16.41.253) e **tux2** (172.16.41.1), respetivamente.

5. **Quais são os endereços de IP e MAC associados com os pacotes de ICMP e porquê?**

Os endereços de IP e MAC associados com os pacotes de ICMP são os endereços dos tux de origem e destino envolvidos na comunicação. Por exemplo, para o comando

ping 172.16.51.1

a partir do **tux1**, os endereços de destino serão os do **tux2** e os de origem serão os do **tux1**.

Experiência 4 - Configuração de um router CISCO e implementação de NAT

- **Breve descrição:**

Seguindo a partir da experiência 3, é agora necessário configurar um router comercial **Rc**, na rede 172.16.y1.0/24. À carta de rede do **Rc** que comunica com a interface de rede 0 atribuímos o endereço de ip 172.16.y1.254 e àquela que comunica com a interface de rede 1 atribuímos o endereço de ip 172.16.1.49. Além disso, é necessário adicionar algumas rotas nos tux. Ao **tux1** é necessário adicionar uma rota *default* para o endereço ip da carta *eth0* do **tux4**, enquanto que aos **tux2** e **tux4** é preciso adicionar rotas *default* para o **Rc**.

Numa fase inicial, sem implementação de NAT, o **tux1** terá de ser capaz de comunicar com todas as interfaces de rede do **tux2**, **tux4** e router **Rc.** O **tux2** tem de ter ativa a aceitação de *ICMP redirects*.

Após a implementação de NAT, o **tux1** consegue comunicar com o router do *lab* e todos os computadores tux conseguem conectar-se à Internet, exceto o **tux4**. O motivo pelo qual o **tux4** é o único que não tem acesso à Internet reside nos comandos executados aquando da configuração do router *CISCO*. Nomeadamente, os comandos:

access-list 1 permit 172.16.y0.0 0.0.0.7
access-list 1 permit 172.16.y1.0 0.0.0.7

limitam a que apenas os endereços de ip com valores compreendidos entre 172.16.y0.0 e 172.16.y0.7 e compreendidos entre 172.16.y1.0 e 172.16.y1.7 consigam aceder à Internet. Como os endereços de ip das cartas correspondentes ao **tux4** são, respetivamente, 172.16.y0.254 para a eth0 e 172.16.y1.243 para a eth1, nenhum dos valores se encontra no intervalo referido e, por isso, o **tux4** está impossibilitado de se conectar à Internet, servindo apenas de router entre as duas subredes.

- **Conceitos em estudo:**

As respostas a cada uma das seguintes perguntas assentam nos conceitos observados na figura em anexo relativa a configurações de um router CISCO.

1. **Como se configura uma rota estática num router comercial?**

Para configurar o router comercial, a porta T4 da régua 1 tem de estar ligada à porta do Router Console da régua 2 e a porta T3 da régua 1 tem de estar ligada à porta S0 do tux onde se pretende configurar o router, através do GTKTerm. A configuração de uma rota estática resulta dos comandos:

configure terminal
ip route [ip de destino] [máscara] [ip de gateway]

exit

2. **Quais são as rotas seguidas pelos pacotes durante a experiência e porquê?**

Caso a rota exista, os pacotes seguem-na. No entanto, se a rota não existir, os pacotes são enviados ao router **Rc** que os redireciona para o seu destino, informando-os da existência do **tux4**. Isto é visível na experiência 4 quando a rota 172.16.y0.0/24 via **tux4** é removida. Consultar figura 13.

3. **Como se configura NAT num router comercial?**

A configuração do NAT encontra-se presente em anexo. Numa primeira fase foi necessário configurar a interface interna, entrando na consola de configuração da interface fastethernet 0/0 do router, a partir do comando **interface gigabitethernet 0/0**. De seguida, especificou-se o IP da interface, com o comando **ip address 172.16.y1.254**(ip) **255.255.255.0**(*mask*).

Numa fase seguinte, foi necessário configurar a interface externa, atribuindo um IP à interface 1 (interface que se encontra ligada ao router da sala), com os comandos **interface gigabitethernet 0/1** e **ip address 172.16.1.y9 255.255.255.0**.

Em ambas as configurações, o comando **no shutdown** resulta em que as configurações não sejam perdidas após desligar o router.

Para estipular a gama de endereços foram introduzidos os seguintes comandos: **ip nat pool ovrld 172.16.1.y9 172.16.1.y9 prefix 24** e **ip nat inside source list 1 pool ovrld overload**. Para criar a lista de acessos e permissões de pacotes, para cada uma das sub-redes foram usados os comandos: **access-list 1 permit 172.16.y0.0 0.0.0.7**(ip máximo) e **access-list 1 permit 172.16.y1.0 0.0.0.7**.

Finalmente, foram definidas as rotas internas e externas com os comandos: **ip route 0.0.0.0 0.0.0.0 172.16.1.254** e **ip route 172.16.y0.0 255.255.255 172.16.y1.253**. Este último, por exemplo, cria uma rota em que, quando o IP de destino for 172.16.y0.0, os pacotes são redirecionados para o IP 172.16.y1.253.

4. **O que faz o NAT?**

NAT significa *Network Address Translation* e permite que os endereços de IP sejam conservados, de forma a que redes IP privadas que usam endereços não registados se possam ligar a uma qualquer rede pública (ou Internet). Ou seja, por outras palavras, NAT permite que computadores de uma rede interna possam comunicar com o exterior, sendo exigido um único endereço IP para representar um conjunto de computadores fora da sua rede interna. NAT opera sobre um *router* e oferece algumas funções de segurança.

Experiência 5 - Configuração de DNS

- **Breve descrição:**

Nesta experiência, configuramos o *DNS* (*Domain Name System*) em cada um dos computadores **tux1**, **tux2** e **tux4**. A partir desta configuração é agora possível comunicar através de *hostnames*, que poderão corresponder a domínios de *sites* comuns. O comando ***ping hostname* (exemplo: ping www.facebook.com)** permite testar a comunicação.

- **Conceitos em estudo:**

1. **Como se configura o serviço DNS num *host*?**

Para configurar DNS executa-se o seguinte comando em cada um dos tux:

vi /etc/resolv.conf search netlab.fe.up.pt nameserver 172.16.1.1

O comando especificado altera o ficheiro resolv.conf, localizado em etc no tux (*host*), adicionando-lhe a informação expressa no comando (**search netlab.fe.up.pt** - servidor DNS e **nameserver 172.16.1.1** - endereço de IP).

2. **Quais pacotes são trocados pelo DNS e que informação é transportada?**

Consultando os registos da figura 14, é possível verificar o resultado de dois *pings* a www.google.com. Para cada um ocorre o seguinte:

Linha 78, pacote enviado pelo *host* **tux1** para o servidor que contém o *hostname* desejado (www.google.com), requerindo o seu endereço de IP.

Linha 80, o servidor responde enviando um pacote com o endereço de IP do *hostname*.

Experiência 6 - Conexões TCP

- **Breve descrição:**

Tendo neste momento a rede estabelecida, vemos-nos prontos a executar a aplicação de *download* criada. Correndo a aplicação no **tux1** e especificando o *path* do ficheiro a ser descarregado, a aplicação terá de ser capaz de o transferir para o diretório do **tux1** em que a aplicação está a operar. Isto será válido também para o **tux2**. Além disso, é possível executar duas descargas ao mesmo tempo: uma a partir do **tux1** e outra a partir do **tux2**.

- **Conceitos em estudo:**

1. **Quantas conexões TCP são abertas pela aplicação ftp?**

Foram abertas duas conexões: uma para enviar comandos FTP por parte do cliente e receber respostas do servidor e outra para receber dados enviados pelo servidor e enviar respostas por parte do cliente.

2. **Em qual conexão é transportado o controlo de informação TCP?**

Na primeira conexão referida anteriormente, ou seja, a conexão responsável pela troca de comandos FTP entre servidor e cliente.

3. **Quais são as fases de uma conexão TCP?**

Uma conexão TCP é constituída por três fases: estabelecimento da conexão (figura 15), troca de dados (figura 16) e encerramento da conexão (figura 17).

4. **Como funciona o mecanismo ARQ TCP? Quais são os campos de TCP relevantes? Que informação relevante pode ser observada nos registos?**

O TCP (*Transmission Control Protocol*) usa o mecanismo ARQ (*Automatic Repeat Request*) para garantir um boa transmissão dos dados, com controlo de erros. Deste modo, o recetor responde com mensagens de reconhecimento constituídas por vários componentes, dos quais se destacam:

acknowledgement numbers - indicam que a trama em questão foi recebida corretamente;
window size - indica a gama dos pacotes que o emissor está possibilitado a enviar;
sequence number - indica o número do pacote que o recetor pretende receber. Consultar figura 18.

5. **Como é que funciona o mecanismo de controlo de congestão TCP? Quais são os campos relevantes? Como é que o fluxo de dados da conexão evolui ao longo do tempo? Está de acordo com o mecanismo de controlo de congestão TCP?**

O mecanismo de controlo de congestão TCP consiste no cálculo estimado do número de octetos que a rede é capaz de encaminhar, diminuindo forçosamente a taxa de transmissão da rede de forma a que o núcleo da rede não seja sobrecarregado.

O fluxo de dados da conexão está de acordo com o mecanismo referido, uma vez que uma maior congestão da rede leva necessariamente a uma taxa de transmissão menor.

Na figura 19 é possível visualizar um gráfico I/O para execuções simultâneas da aplicação de download nos **tux1** e **tux2** (registo obtido no **tux1**), mostrando a variação da taxa de transmissão de pacotes de dados ao longo do tempo.

6. **O fluxo de dados da conexão TCP é afetado pelo aparecimento de uma segunda conexão TCP? Como?**

Com o aparecimento de uma segunda conexão TCP, a taxa de transferência é distribuída igualitariamente pelas duas ligações e, assim, pode ocorrer uma diminuição da taxa de transmissão por estarem a ocorrer duas transmissões de dados em simultâneo.

Conclusão

O trabalho desenvolvido teve como dois principais objetivos a criação de uma rede, segundo o protocolo ARP, e a implementação de uma aplicação *download*.

As seis experiências desenvolvidas auxiliaram na consolidação dos conhecimentos teóricos lecionados nas aulas, bem como nos elucidaram para a vertente prático-laboratorial, tanto em conhecimentos relativos a *software* (preparação e desenvolvimento da app download, configurações de router, switch e tux's) , como relativos a *hardware* (cabos de rede, switch e router).

Também aprendemos o funcionamento do protocolo TCP, que apesar de ser um protocolo de nível baixo, consegue evitar o congestionamento da rede, aumentando a eficácia da conexão.

Em suma, consideramos que todos os objetivos foram cumpridos com sucesso e que o trabalho laboratorial desenvolvido promoveu uma aprendizagem mais aprofundada do tema em questão.

Referências

https://www.ietf.org/rfc/rfc0959
https://www.ietf.org/rfc/rfc1738
https://blog.pantuza.com/artigos/o-protocolo-arp-address-resolution-protocol
https://www.cisco.com/c/en/us/support/docs/ip/network-address-translation-nat/13772-12.html

Anexos

Imagens

```
***************************************************
* User:                                anonymous *
* Pass:                                        *
* Host:                               ftp.up.pt *
* Path:                               pub/CPAN *
* File:                                ENDINGS *
* IP  :                           193.137.29.15 *
***************************************************


Connected 193.137.29.15:21
220-Welcome to the University of Porto's mirror archive (mirrors.up.pt)
220----------------------------------------------------------------
220-
220-All connections and transfers are logged. The max number of connections is 200.
220-
220-For more information please visit our website: http://mirrors.up.pt/
220-Questions and comments can be sent to mirrors@uporto.pt
220-
220-
220
USER anonymous
331 Please specify the password.
PASS
230 Login successful.
CWD pub/CPAN
250-The Comprehensive Perl Archive Network (http://www.cpan.org/)
250-master site has been from the very beginning (1995) hosted at FUNET,
250-the Finnish University NETwork.
250-
250-
250 Directory successfully changed.
PASV
227 Entering Passive Mode (193,137,29,15,207,252).
Connected 193.137.29.15:53244
RETR ENDINGS
150 Opening BINARY mode data connection for ENDINGS (3954 bytes).
./download ftp://ftp.up.pt/pub/CPAN/ENDINGS  0,02s user 0,00s system 5% cpu 0,292 total
```

Figura 1

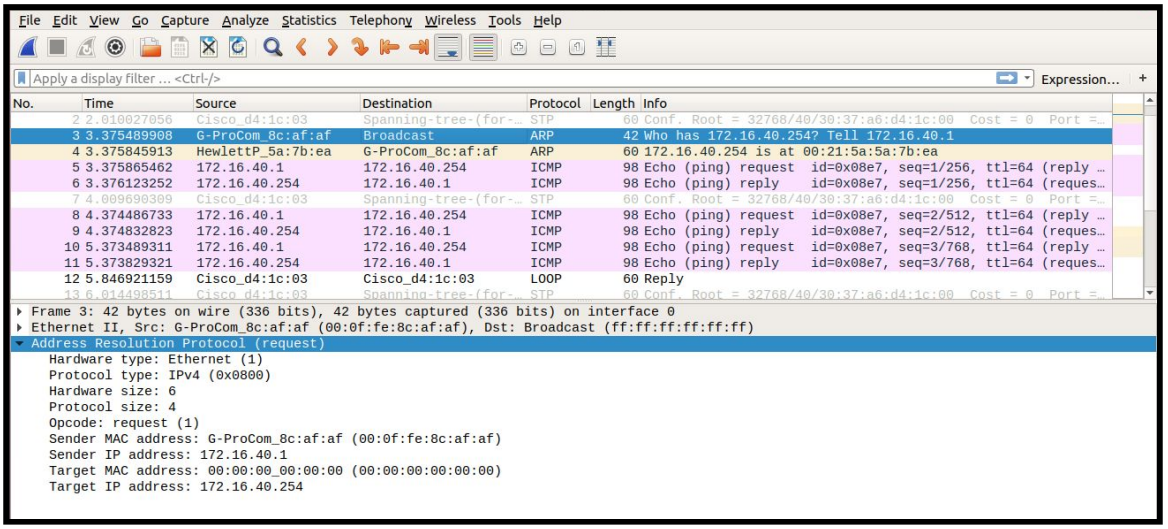


Figura 2

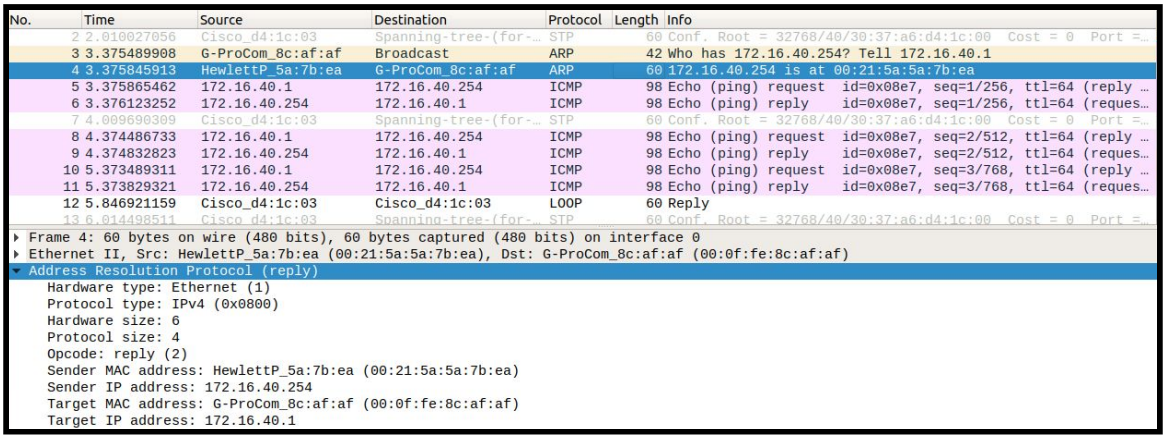


Figura 3

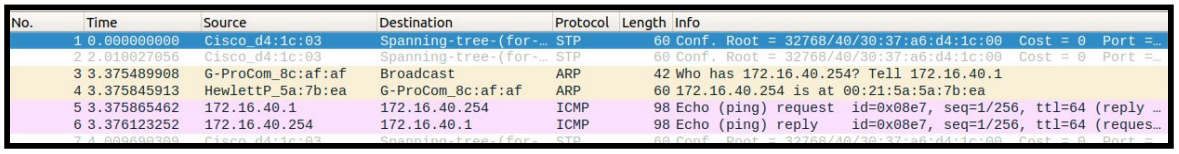


Figura 4

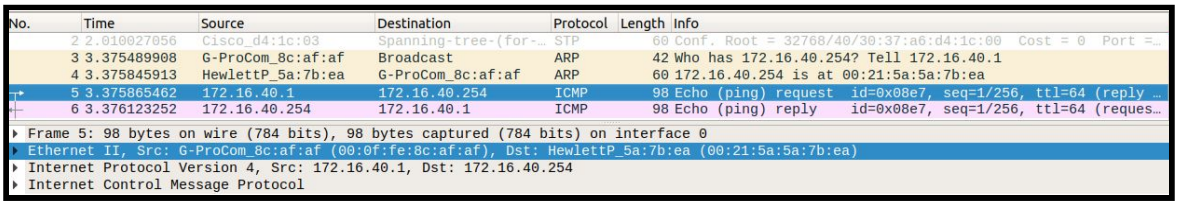


Figura 5

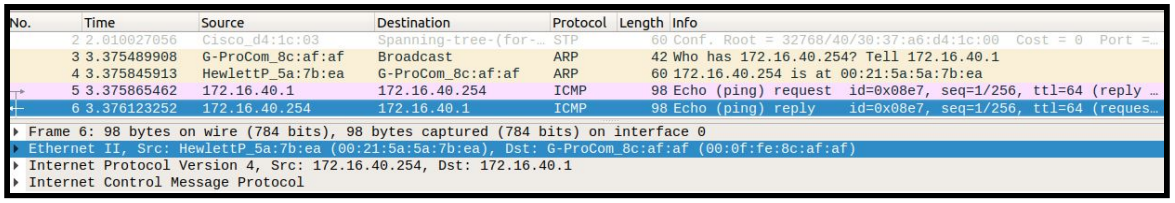


Figura 6

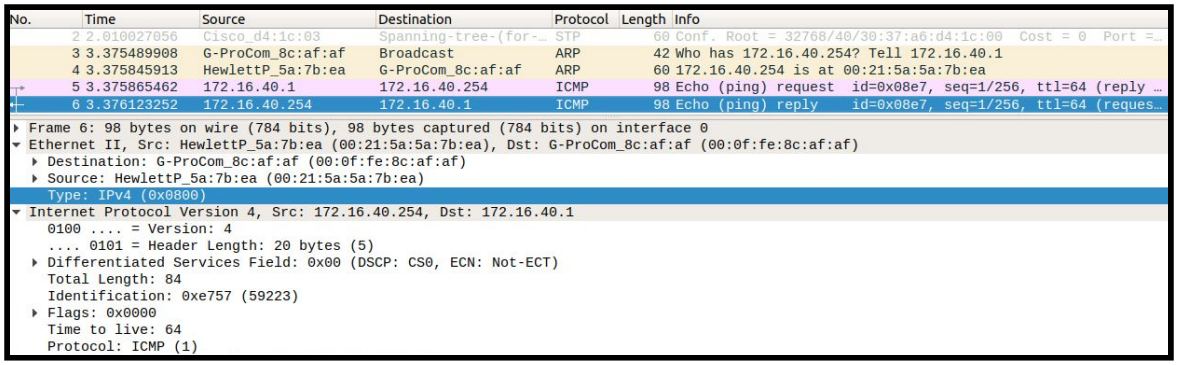


Figura 7

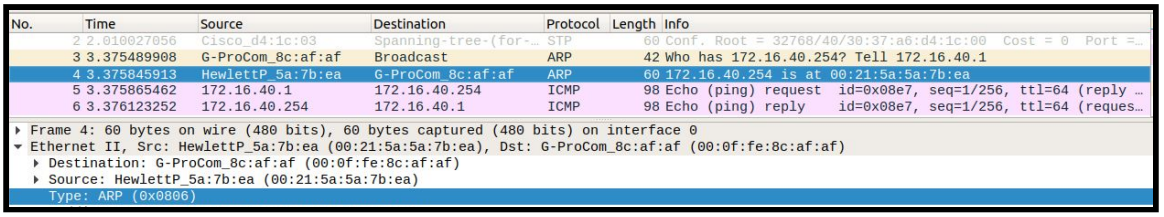


Figura 8

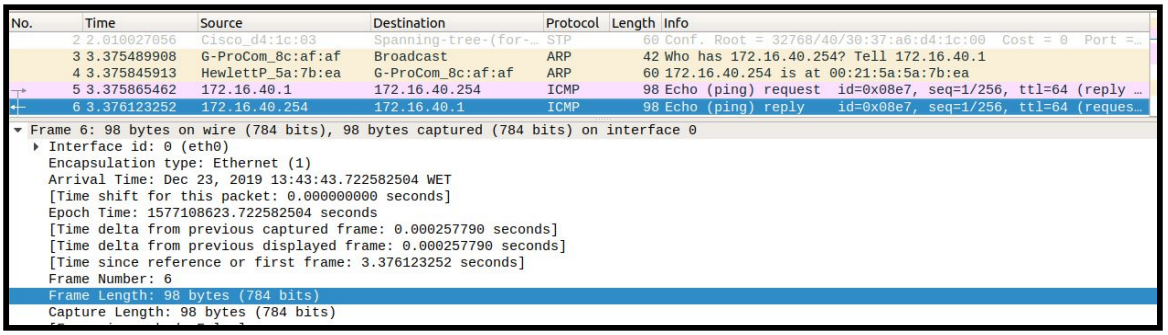


Figura 9

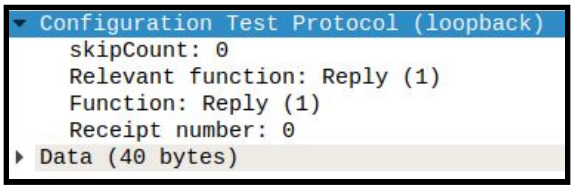


Figura 10

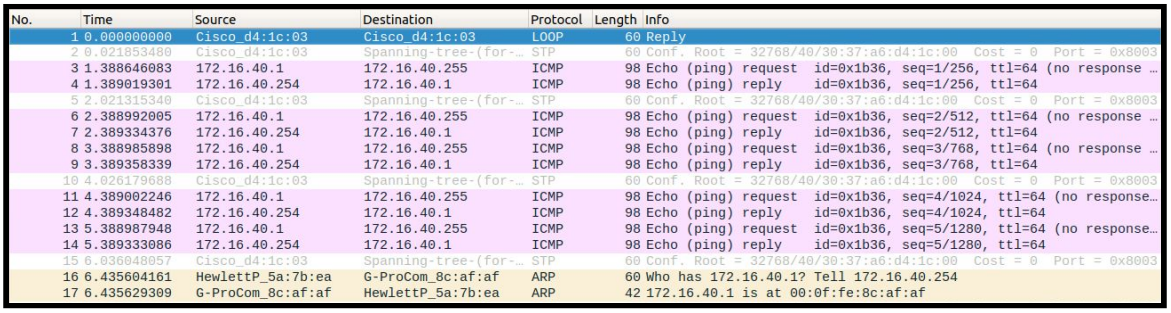


Figura 11

No.	Time	Source	Destination	Protocol	Length	Info
2	1.794158044	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x1d3d, seq=1/256, ttl=64 (reply …
3	1.794312986	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1d3d, seq=1/256, ttl=64 (reques…
4	1.998728952	Cisco_d4:1c:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port =…
5	2.794462327	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x1d3d, seq=2/512, ttl=64 (reply …
6	2.794802964	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1d3d, seq=2/512, ttl=64 (reques…
7	4.003450187	Cisco_d4:1c:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port =…
8	6.007913156	Cisco_d4:1c:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port =…
9	6.798418439	G-ProCom_8c:af:af	HewlettP_5a:7b:ea	ARP	42	Who has 172.16.40.254? Tell 172.16.40.1
10	6.798763226	HewlettP_5a:7b:ea	G-ProCom_8c:af:af	ARP	60	172.16.40.254 is at 00:21:5a:5a:7b:ea
11	6.835523258	HewlettP_5a:7b:ea	G-ProCom_8c:af:af	ARP	60	Who has 172.16.40.1? Tell 172.16.40.254
12	6.835539724	G-ProCom_8c:af:af	HewlettP_5a:7b:ea	ARP	42	172.16.40.1 is at 00:0f:fe:8c:af:af
13	8.018157691	Cisco_d4:1c:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port =…
14	8.885401545	Cisco_d4:1c:03	Cisco_d4:1c:03	LOOP	60	Reply
15	9.458534287	172.16.40.1	172.16.41.253	ICMP	98	Echo (ping) request id=0x1d41, seq=1/256, ttl=64 (reply …
16	9.458730172	172.16.41.253	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1d41, seq=1/256, ttl=64 (reques…
17	10.017947616	Cisco_d4:1c:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port =…
18	10.458477879	172.16.40.1	172.16.41.253	ICMP	98	Echo (ping) request id=0x1d41, seq=2/512, ttl=64 (reply …
19	10.458753297	172.16.41.253	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1d41, seq=2/512, ttl=64 (reques…
20	12.023042470	Cisco_d4:1c:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port =…
21	14.032494288	Cisco_d4:1c:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port =…
22	16.032329471	Cisco_d4:1c:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port =…
23	17.417971618	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1d4e, seq=1/256, ttl=64 (reply …
24	17.418576796	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1d4e, seq=1/256, ttl=63 (reques…
25	18.037659558	Cisco_d4:1c:03	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port =…
26	18.418457330	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1d4e, seq=2/512, ttl=64 (reply …
27	18.418904007	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1d4e, seq=2/512, ttl=63 (reques…
28	18.884895817	Cisco_d4:1c:03	Cisco_d4:1c:03	LOOP	60	Reply
29	19.418458701	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1d4e, seq=3/768, ttl=64 (reply …
30	19.418680796	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1d4e, seq=3/768, ttl=63 (reques…

Figura 12

No.	Time	Source	Destination	Protocol	Length	Info
18	11.156928476	Cisco_e3:df:10	HewlettP_d7:45:c4	ARP	60	172.16.41.254 is at 68:ef:bd:e3:df:10
19	12.038437430	Cisco_d4:1c:05	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = …
20	12.254293790	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) request id=0x0c5d, seq=1/256, ttl=64 (reply in
21	12.254652076	172.16.41.254	172.16.41.1	ICMP	70	Redirect (Redirect for host)
22	12.255093822	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) reply id=0x0c5d, seq=1/256, ttl=63 (request
23	13.268672923	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) request id=0x0c5d, seq=2/512, ttl=64 (reply in
24	13.269022618	172.16.41.254	172.16.41.1	ICMP	70	Redirect (Redirect for host)
25	13.269452352	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) reply id=0x0c5d, seq=2/512, ttl=63 (request
26	14.043314656	Cisco_d4:1c:05	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = …
27	14.292669829	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) request id=0x0c5d, seq=3/768, ttl=64 (reply in
28	14.292991169	172.16.41.254	172.16.41.1	ICMP	70	Redirect (Redirect for host)
29	14.293397785	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) reply id=0x0c5d, seq=3/768, ttl=63 (request
30	14.971449722	Cisco_d4:1c:05	CDP/VTP/DTP/PAgP/UD…	CDP	432	Device ID: tux-sw4 Port ID: FastEthernet0/3

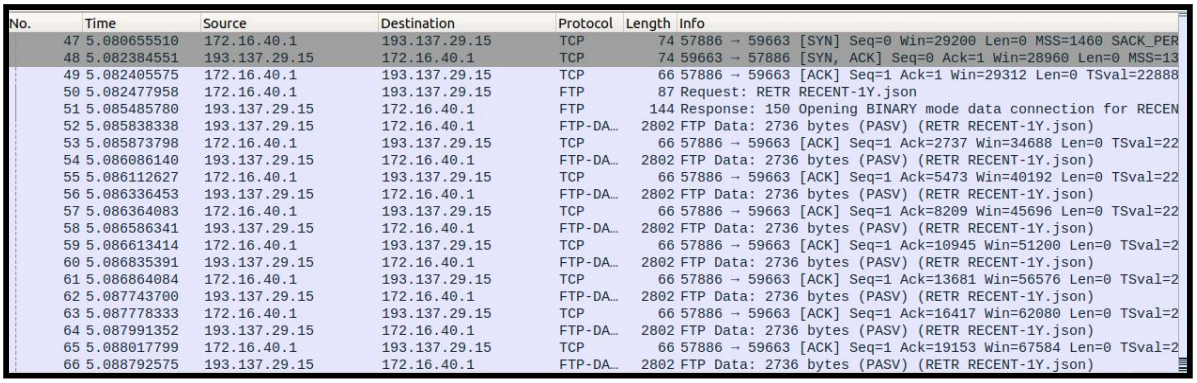
Figura 13

	Time	Source	Destination	Protocol	Length	Info
78	8.559336973	172.16.40.1	172.16.1.1	DNS	74	Standard query 0xd7b1 A www.google.com
79	8.559372026	172.16.40.1	172.16.1.1	DNS	74	Standard query 0x76ce AAAA www.google.com
80	8.560929580	172.16.1.1	172.16.40.1	DNS	338	Standard query response 0xd7b1 A www.google.com A 172.217.17…
81	8.560960132	172.16.1.1	172.16.40.1	DNS	350	Standard query response 0x76ce AAAA www.google.com AAAA 2a00:…
82	8.562070744	172.16.40.1	172.217.17.4	ICMP	98	Echo (ping) request id=0x0fc1, seq=1/256, ttl=64 (reply in 8…
83	8.577489176	172.217.17.4	172.16.40.1	ICMP	98	Echo (ping) reply id=0x0fc1, seq=1/256, ttl=49 (request in…
84	8.577678074	172.16.40.1	172.16.1.1	DNS	85	Standard query 0x6fe7 PTR 4.17.217.172.in-addr.arpa
85	8.579029281	172.16.1.1	172.16.40.1	DNS	381	Standard query response 0x6fe7 PTR 4.17.217.172.in-addr.arpa …
86	9.563197559	172.16.40.1	172.217.17.4	ICMP	98	Echo (ping) request id=0x0fc1, seq=2/512, ttl=64 (reply in 8…
87	9.578291726	172.217.17.4	172.16.40.1	ICMP	98	Echo (ping) reply id=0x0fc1, seq=2/512, ttl=49 (request in…
88	10.000059941	172.16.40.1	172.16.1.1	DNS	87	Standard query 0x0028 PTR 170.213.58.216.in-addr.arpa
89	10.001762722	172.16.1.1	172.16.40.1	DNS	415	Standard query response 0x0028 PTR 170.213.58.216.in-addr.arp…

Figura 14

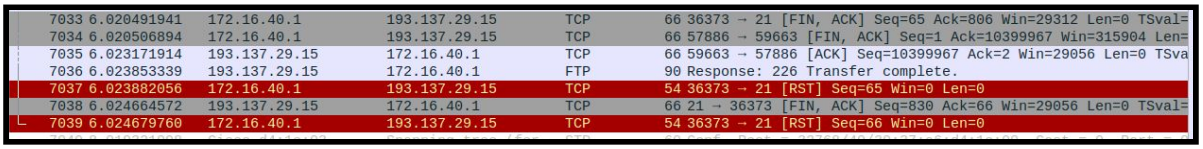
No.	Time	Source	Destination	Protocol	Length	Info
7	4.982746130	172.16.40.1	193.137.29.15	TCP	74	36373 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1
8	4.986037616	193.137.29.15	172.16.40.1	TCP	74	21 → 36373 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1380
9	4.986066459	172.16.40.1	193.137.29.15	TCP	66	36373 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=22888832
10	4.993318603	193.137.29.15	172.16.40.1	FTP	139	Response: 220-Welcome to the University of Porto's mirror a
11	4.993330914	193.137.29.15	172.16.40.1	FTP	135	Response: 220----------
12	4.993334513	193.137.29.15	172.16.40.1	FTP	72	Response: 220-
13	4.993337310	193.137.29.15	172.16.40.1	FTP	151	Response: 220-All connections and transfers are logged. The
14	4.993340132	193.137.29.15	172.16.40.1	FTP	72	Response: 220-
15	4.993362935	172.16.40.1	193.137.29.15	TCP	66	36373 → 21 [ACK] Seq=1 Ack=74 Win=29312 Len=0 TSval=228888
16	4.993374058	172.16.40.1	193.137.29.15	TCP	66	36373 → 21 [ACK] Seq=1 Ack=143 Win=29312 Len=0 TSval=228888
17	4.993379206	172.16.40.1	193.137.29.15	TCP	66	36373 → 21 [ACK] Seq=1 Ack=149 Win=29312 Len=0 TSval=228888
18	4.993383723	172.16.40.1	193.137.29.15	TCP	66	36373 → 21 [ACK] Seq=1 Ack=234 Win=29312 Len=0 TSval=228888
19	4.993387969	172.16.40.1	193.137.29.15	TCP	66	36373 → 21 [ACK] Seq=1 Ack=240 Win=29312 Len=0 TSval=228888
20	4.993589734	193.137.29.15	172.16.40.1	FTP	140	Response: 220-For more information please visit our website
21	4.993599569	193.137.29.15	172.16.40.1	FTP	127	Response: 220-Questions and comments can be sent to mirrors
22	4.993602517	193.137.29.15	172.16.40.1	FTP	72	Response: 220-
23	4.993605043	193.137.29.15	172.16.40.1	FTP	72	Response: 220-
24	4.993607559	193.137.29.15	172.16.40.1	FTP	72	Response: 220
25	4.993624983	172.16.40.1	193.137.29.15	TCP	66	36373 → 21 [ACK] Seq=1 Ack=314 Win=29312 Len=0 TSval=228888
26	4.993633901	172.16.40.1	193.137.29.15	TCP	66	36373 → 21 [ACK] Seq=1 Ack=375 Win=29312 Len=0 TSval=228888

Figura 15



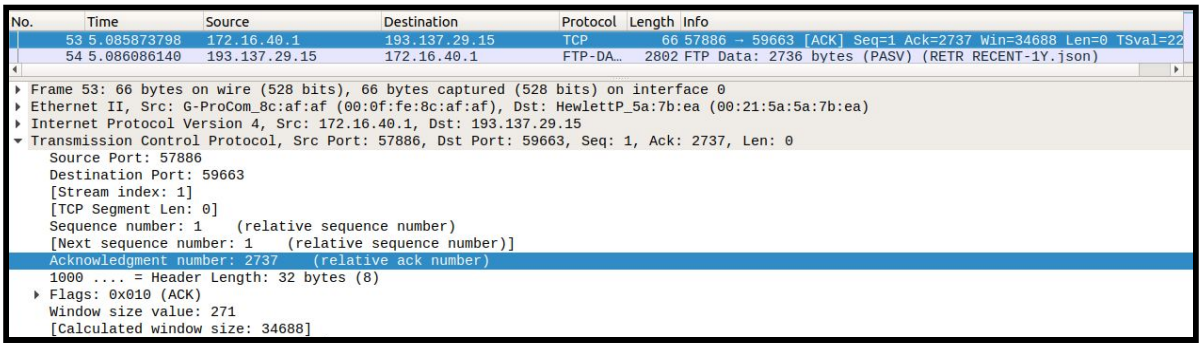
No.	Time	Source	Destination	Protocol	Length	Info
47	5.080655510	172.16.40.1	193.137.29.15	TCP	74	57886 → 59663 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PER
48	5.082384551	193.137.29.15	172.16.40.1	TCP	74	59663 → 57886 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=13
49	5.082405575	172.16.40.1	193.137.29.15	TCP	66	57886 → 59663 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=22888
50	5.082477958	172.16.40.1	193.137.29.15	FTP	87	Request: RETR RECENT-1Y.json
51	5.085485780	193.137.29.15	172.16.40.1	FTP	144	Response: 150 Opening BINARY mode data connection for RECEN
52	5.085838338	193.137.29.15	172.16.40.1	FTP-DA…	2802	FTP Data: 2736 bytes (PASV) (RETR RECENT-1Y.json)
53	5.085873798	172.16.40.1	193.137.29.15	TCP	66	57886 → 59663 [ACK] Seq=1 Ack=2737 Win=34688 Len=0 TSval=22
54	5.086086140	193.137.29.15	172.16.40.1	FTP-DA…	2802	FTP Data: 2736 bytes (PASV) (RETR RECENT-1Y.json)
55	5.086112627	172.16.40.1	193.137.29.15	TCP	66	57886 → 59663 [ACK] Seq=1 Ack=5473 Win=40192 Len=0 TSval=22
56	5.086336453	193.137.29.15	172.16.40.1	FTP-DA…	2802	FTP Data: 2736 bytes (PASV) (RETR RECENT-1Y.json)
57	5.086364083	172.16.40.1	193.137.29.15	TCP	66	57886 → 59663 [ACK] Seq=1 Ack=8209 Win=45696 Len=0 TSval=22
58	5.086586341	193.137.29.15	172.16.40.1	FTP-DA…	2802	FTP Data: 2736 bytes (PASV) (RETR RECENT-1Y.json)
59	5.086613414	172.16.40.1	193.137.29.15	TCP	66	57886 → 59663 [ACK] Seq=1 Ack=10945 Win=51200 Len=0 TSval=2
60	5.086835391	193.137.29.15	172.16.40.1	FTP-DA…	2802	FTP Data: 2736 bytes (PASV) (RETR RECENT-1Y.json)
61	5.086864084	172.16.40.1	193.137.29.15	TCP	66	57886 → 59663 [ACK] Seq=1 Ack=13681 Win=56576 Len=0 TSval=2
62	5.087743700	193.137.29.15	172.16.40.1	FTP-DA…	2802	FTP Data: 2736 bytes (PASV) (RETR RECENT-1Y.json)
63	5.087778333	172.16.40.1	193.137.29.15	TCP	66	57886 → 59663 [ACK] Seq=1 Ack=16417 Win=62080 Len=0 TSval=2
64	5.087991352	193.137.29.15	172.16.40.1	FTP-DA…	2802	FTP Data: 2736 bytes (PASV) (RETR RECENT-1Y.json)
65	5.088017799	172.16.40.1	193.137.29.15	TCP	66	57886 → 59663 [ACK] Seq=1 Ack=19153 Win=67584 Len=0 TSval=2
66	5.088792575	193.137.29.15	172.16.40.1	FTP-DA…	2802	FTP Data: 2736 bytes (PASV) (RETR RECENT-1Y.json)

Figura 16



7033	6.020491941	172.16.40.1	193.137.29.15	TCP	66	36373 → 21 [FIN, ACK] Seq=65 Ack=806 Win=29312 Len=0 TSval=
7034	6.020506894	172.16.40.1	193.137.29.15	TCP	66	57886 → 59663 [FIN, ACK] Seq=1 Ack=10399967 Win=315904 Len=
7035	6.023171914	193.137.29.15	172.16.40.1	TCP	66	59663 → 57886 [ACK] Seq=10399967 Ack=2 Win=29056 Len=0 TSva
7036	6.023853339	193.137.29.15	172.16.40.1	FTP	90	Response: 226 Transfer complete.
7037	6.023882056	172.16.40.1	193.137.29.15	TCP	54	36373 → 21 [RST] Seq=65 Win=0 Len=0
7038	6.024664572	193.137.29.15	172.16.40.1	TCP	66	21 → 36373 [FIN, ACK] Seq=830 Ack=66 Win=29056 Len=0 TSval=
7039	6.024679760	172.16.40.1	193.137.29.15	TCP	54	36373 → 21 [RST] Seq=66 Win=0 Len=0

Figura 17



No.	Time	Source	Destination	Protocol	Length	Info
53	5.085873798	172.16.40.1	193.137.29.15	TCP	66	57886 → 59663 [ACK] Seq=1 Ack=2737 Win=34688 Len=0 TSval=22
54	5.086086140	193.137.29.15	172.16.40.1	FTP-DA…	2802	FTP Data: 2736 bytes (PASV) (RETR RECENT-1Y.json)

```
▸ Frame 53: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▸ Ethernet II, Src: G-ProCom_8c:af:af (00:0f:fe:8c:af:af), Dst: HewlettP_5a:7b:ea (00:21:5a:5a:7b:ea)
▸ Internet Protocol Version 4, Src: 172.16.40.1, Dst: 193.137.29.15
▾ Transmission Control Protocol, Src Port: 57886, Dst Port: 59663, Seq: 1, Ack: 2737, Len: 0
    Source Port: 57886
    Destination Port: 59663
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 1    (relative sequence number)]
    Acknowledgment number: 2737    (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
  ▸ Flags: 0x010 (ACK)
    Window size value: 271
    [Calculated window size: 34688]
```

Figura 18

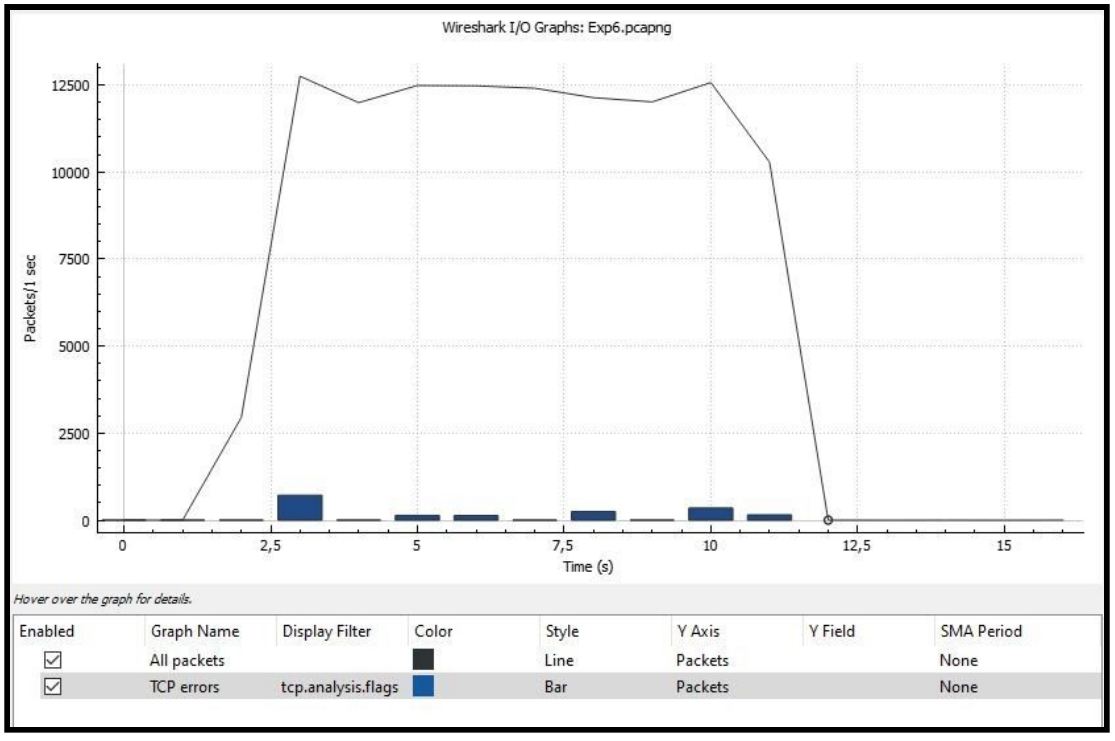


Figura 19

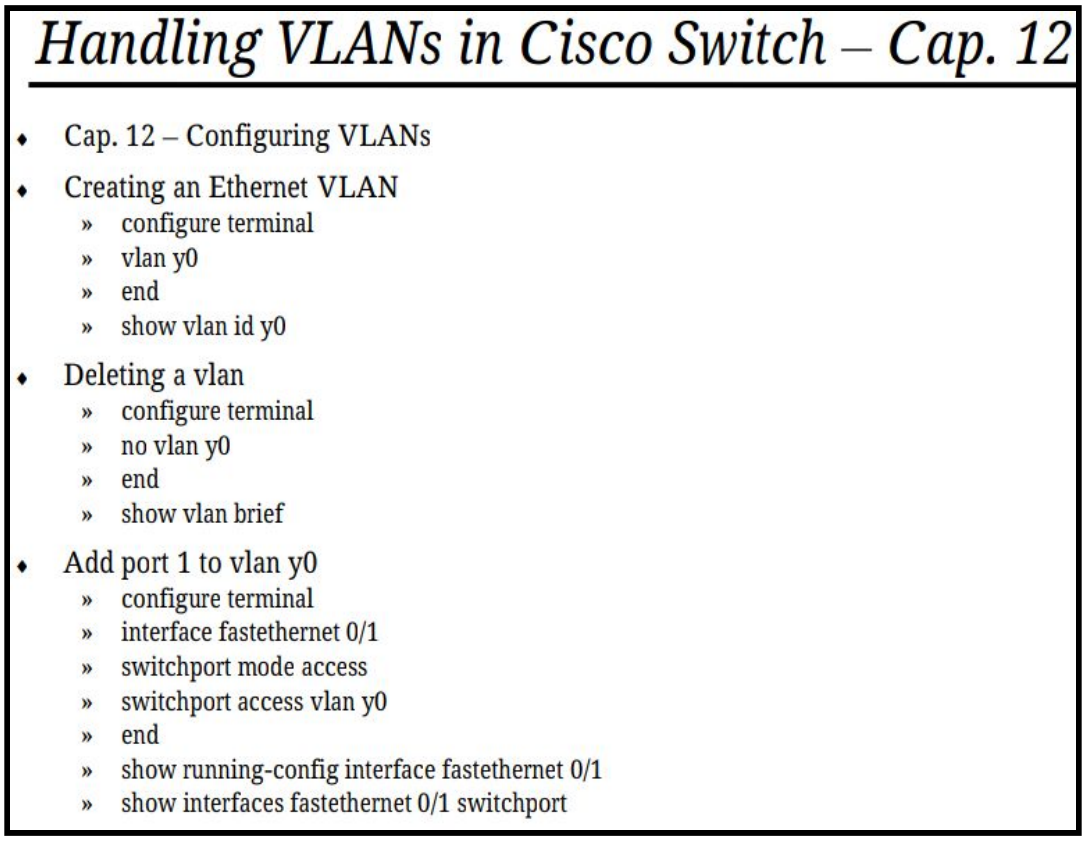


Figura 20

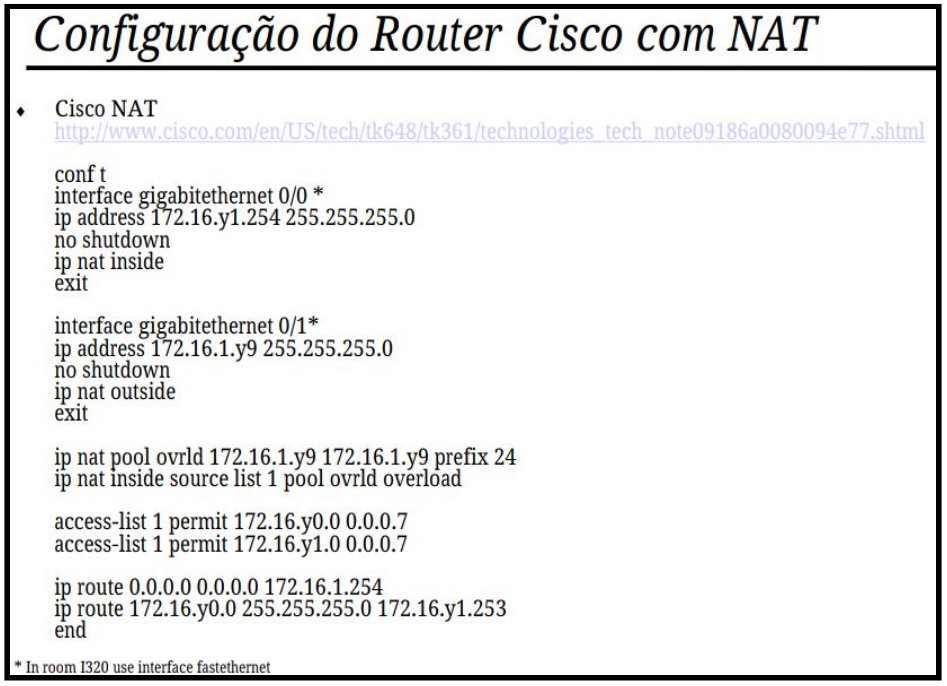


Figura 21

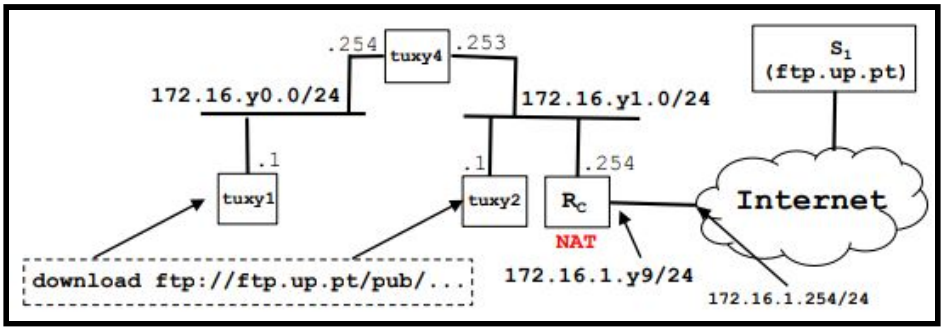


Figura 22

main.c

```
/* RCOM 2019/2020
* Joao Campos and Nuno Cardoso
* Client main file
*/

#include "client.h"

int main(int argc , char *argv[]) {
 int socket_fd;
 host_t *h;
 struct sockaddr_in server_addr;
 char user[MAX_STRING_SIZE]; memset(user, 0, MAX_STRING_SIZE);
 char pass[MAX_STRING_SIZE]; memset(pass, 0, MAX_STRING_SIZE);
 char host[MAX_STRING_SIZE]; memset(host, 0, MAX_STRING_SIZE);
 char path[MAX_STRING_SIZE]; memset(path, 0, MAX_STRING_SIZE);
 char file[MAX_STRING_SIZE]; memset(file, 0, MAX_STRING_SIZE);
 char ip[MAX_STRING_SIZE];
 char buf[MAX_BUF_SIZE];

 if(get_args(argv[1], user, pass, host, path, file)){
  printf("Usage: %s ftp://[<user>:<password>@]<host>/<path>\n", argv[0]);
  return -1;
 }

 if(getip(host, ip)){
  printf("Error getting ip.\n");
  return -1;
 }
```

```
system("clear");
printf(" ************************************************\n");
printf(" * User: %*s *\n", (MAX_STRING_SIZE + 4), user);
printf(" * Pass: %*s *\n", (MAX_STRING_SIZE + 4), pass);
printf(" * Host: %*s *\n", (MAX_STRING_SIZE + 4), host);
printf(" * Path: %*s *\n", (MAX_STRING_SIZE + 4), path);
printf(" * File: %*s *\n", (MAX_STRING_SIZE + 4), file);
printf(" * IP  : %*s *\n", (MAX_STRING_SIZE + 4), ip);
printf(" ************************************************\n\n\n");

if(start_connection(&socket_fd, ip, SERVER_PORT)){
  printf("Error connecting.\n");
}

/* Receive opening message */
if(receive_msg(socket_fd)){
  printf("Error reading opening message.\n");
  return -1;
}

/* Login */
if(login(socket_fd, user, pass)){
  printf("Error in login.\n");
  return -1;
}

/* Change directory */
if(change_directory(socket_fd, path)){
  printf("Error in cwd.\n");
  return -1;
}

/* Passive mode */
int pasv_sock_fd;
if(passive_mode(socket_fd, &pasv_sock_fd)){
  printf("Error in pasv.\n");
  return -1;
}
```

```c
/* Send filename */
if(send_filename(socket_fd, file)){
  printf("Error in retr.\n");
  return -1;
}

/* Receive file */
if(receive_file(pasv_sock_fd, file)){
  printf("Error receiving file.\n");
  return -1;
}

/* End of communication */
close(socket_fd);
close(pasv_sock_fd);

return 0;
}
```

client.h

```c
/* RCOM 2019/2020
* Joao Campos and Nuno Cardoso
* Client header file
*/

#include "socket.h"

#define MAX_STRING_SIZE 40     // Max string size

typedef struct hostent host_t;

/* Get info from url argument */
int get_args(char *arg, char *user, char *pass, char *host, char *path, char* file);

/* Get ip of given host */
int getip(char* host, char* ip);

/* Open socket and connect with IP and port*/
```

```c
int start_connection(int *socket_fd, char* ip, int port);

/* Login into server */
int login(int socket_fd, char* user, char* pass);

/* Change directory in server */
int change_directory(int socket_fd, char* path);

/* Enter passive mode */
int passive_mode(int socket_fd, int* pasv_sock_fd);

/* Send filename */
int send_filename(int socket_fd, char* filename);
```

client.c

```c
/* RCOM 2019/2020
* Joao Campos and Nuno Cardoso
* Client file
*/

#include "client.h"

int get_args(char *arg, char *user, char *pass, char *host, char *path, char* file){
 /* ftp://[<user>:<password>@]<host>/<path> */

 char* start = "ftp://";
 char* url = malloc(strlen(arg));

 strcpy(url, arg);
 /* Check start */
 if(strncmp(url, start, strlen(start))){
   printf("Invalid url.");
   return -1;
 }
 url += strlen(start);

 if(strchr(url, ':') != NULL){
   /* Read username */
```

```
  while (*url != ':') {
    strncat(user, url, 1);
    url++;
  }
  url++;

  /* Read password */
  while (*url != '@') {
    strncat(pass, url, 1);
    url++;
  }
  url++;
} else {
  strcpy(user, "anonymous");
  strcpy(pass, "");
}

/* Read host */
while (*url != '/') {
  strncat(host, url, 1);
  url++;
}
url++;

/* Read path */
strncpy(path, url, strrchr(url, '/')-url);

/* Read filename */
strcpy(file, strrchr(url, '/')+1);

return 0;
}

int getip(char* host, char* ip){
 host_t *h;
/* Get host struct */
if((h = gethostbyname(host)) == NULL){
  perror("gethostbyname");
  return -1;
```

```
}
/* Get ip from host */
strcpy(ip, inet_ntoa(*((struct in_addr *)h->h_addr)));

return 0;
}

int start_connection(int *socket_fd, char* ip, int port){
/* Open socket */
if((*socket_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
  printf("Error with socket.\n");
  return -1;
}

/* Create socket address */
struct sockaddr_in server_addr;
bzero((char*)&server_addr, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = inet_addr(ip);
server_addr.sin_port = htons(port);

/* Connect to socket */
if(connect(*socket_fd, (struct sockaddr*) &server_addr, sizeof(server_addr)) < 0){
  printf("Error with connect.\n");
  return -1;
}

printf("Connected %s:%d\r\n", ip, port);

return 0;
}

int login(int socket_fd, char* user, char* pass){
char buf[MAX_BUF_SIZE];
char res[MAX_BUF_SIZE];

/* Send username */
memset(buf, 0, MAX_BUF_SIZE);
sprintf(buf, "USER %s\r\n", user);
```

```
printf("%s", buf);
if(send_msg(socket_fd, buf)){
  printf("Error sending username.\n");
  return -1;
}

/* Receive response */
if(receive_msg(socket_fd)){
  printf("Error reading username response.\n");
  return -1;
}

/* Send password */
memset(buf, 0, MAX_BUF_SIZE);
sprintf(buf, "PASS %s\r\n", pass);
printf("%s", buf);
if(send_msg(socket_fd, buf)){
  printf("Error sending password.\n");
  return -1;
}

/* Receive response */
if(receive_msg(socket_fd)){
  printf("Error reading password response.\n");
  return -1;
}

return 0;
}

int change_directory(int socket_fd, char* path){
char buf[MAX_BUF_SIZE];
char res[MAX_BUF_SIZE];

/* Send cwd */
memset(buf, 0, MAX_BUF_SIZE);
sprintf(buf, "CWD %s\r\n", path);
printf("%s", buf);
if(send_msg(socket_fd, buf)){
```

```
  printf("Error sending cwd.\n");
  return -1;
}

/* Receive response */
if(receive_msg(socket_fd)){
  printf("Error reading cwd response.\n");
  return -1;
}

return 0;
}

int passive_mode(int socket_fd, int* pasv_sock_fd){
 char buf[MAX_BUF_SIZE];
 char res[MAX_BUF_SIZE];

/* Send pasv */
memset(buf, 0, MAX_BUF_SIZE);
sprintf(buf, "PASV\r\n");
printf("%s", buf);
if(send_msg(socket_fd, buf)){
  printf("Error sending pasv.\n");
  return -1;
}

/* Receive response */
if(receive_msg_(socket_fd, res)){
  printf("Error reading pasv response.\n");
  return -1;
}

/* Get passive ip and port */
char pasv[1024];
memset(pasv, 0, MAX_BUF_SIZE);
int ip1,ip2,ip3,ip4,port1,port2,port;
sscanf(res, "227 Entering Passive Mode (%d,%d,%d,%d,%d,%d)",&ip1,&ip2,&ip3,&ip4,&port1,&port2);
sprintf(pasv, "%d.%d.%d.%d",ip1,ip2,ip3,ip4);
port = port1 * 256 + port2;
```

```
/* Connect to passive socket */
if(start_connection(pasv_sock_fd, pasv, port)){
  printf("Error connecting.\n");
  return -1;
}

return 0;
}

int send_filename(int socket_fd, char* filename){
 char buf[MAX_BUF_SIZE];
 char res[MAX_BUF_SIZE];

 /* Send retr */
 memset(buf, 0, MAX_BUF_SIZE);
 sprintf(buf, "RETR %s\r\n", filename);
 printf("%s", buf);
 if(send_msg(socket_fd, buf)){
  printf("Error sending retr.\n");
  return -1;
 }

 /* Receive response */
 if(receive_msg(socket_fd)){
  printf("Error reading retr response.\n");
  return -1;
 }

 return 0;
}
```

socket.h

```
/* RCOM 2019/2020
* Joao Campos and Nuno Cardoso
* Socket header file
*/
```

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <netdb.h>
#include <string.h>
#include <fcntl.h>
#include <libgen.h>

#define SERVER_PORT 21              // port of socket
#define MAX_BUF_SIZE 512            // Max buffer size

/* Receive messages from server */
int receive_msg(int socket_fd);

/* Receive messages from server into buf */
int receive_msg_(int socket_fd, char* buf);

/* Send messages to server */
int send_msg(int socket_fd, char *msg);

/* Receive file from server */
int receive_file(int socket_fd, char* filename);
```

socket.c

```c
/* RCOM 2019/2020
 * Joao Campos and Nuno Cardoso
 * Socket file
 */

#include "socket.h"

int receive_msg(int socket_fd){
 /* Open as FILE* to use fgets() */
```

```c
FILE* sock_file = fdopen(socket_fd, "r");
if(sock_file == NULL){
  perror("receive");
  return -1;
}

char buf[MAX_BUF_SIZE];
int n;

/* Read line-by-line until find space in forth */
do {
  memset(buf, 0, MAX_BUF_SIZE);
  if(fgets(buf, MAX_BUF_SIZE, sock_file) == NULL) break;
  printf("%s", buf);
} while (buf[3] != ' ');

/* In case of error */
if(buf[0] < '1' || buf[0] >= '5'){
  printf("Unexpected error.\n");
  exit(1);
}

return 0;
}

int receive_msg_(int socket_fd, char* buf){
/* Open as FILE* to use fgets() */
FILE* sock_file = fdopen(socket_fd, "r");
if(sock_file == NULL){
  perror("receive");
  return -1;
}

int n;

/* Read line-by-line until find space in forth */
while (buf[3] != ' ') {
  memset(buf, 0, MAX_BUF_SIZE);
  if(fgets(buf, MAX_BUF_SIZE, sock_file) == NULL) break;
```

```
  printf("%s", buf);
}

/* In case of error */
if(buf[0] < '1' || buf[0] >= '5'){
  printf("Unexpected error.\n");
  exit(1);
}


return 0;
}

int send_msg(int socket_fd, char *buf){
/* Write buf to socket */
int n = write(socket_fd, buf, strlen(buf));
if(n < 0){
  printf("Error on send msg.\n");
  return -1;
}

return 0;
}

int receive_file(int socket_fd, char *filename){
/* Create output file */
int file_fd = creat(filename, 0777);
if(file_fd < 0){
  perror("creat()");
  return -1;
}

/* Read from socket and write to file */
char data[MAX_BUF_SIZE];
int read_bytes, write_bytes;
while ((read_bytes = read(socket_fd, data, MAX_BUF_SIZE))) {
  if(read_bytes < 0){
    perror("read()");
    return -1;
```

```
  }
  write_bytes = write(file_fd, data, read_bytes);
  if(write_bytes < 0){
    perror("write()");
    return -1;
  }
 }

 close(file_fd);
 return 0;
}
```

tuxPermission.sh

```
#!/bin/bash
chmod -x tux41.sh
chmod -x tux42.sh
chmod -x tux44.sh
```

tux41.sh

```
/bin/bash
ifconfig eth0 up
ifconfig eth0 172.16.40.1/24
route add -net 172.16.41.0/24 gw 172.16.40.254
route add default gw 172.16.40.254
vi /etc/resolv.conf search netlab.fe.up.pt nameserver 172.16.1.1
```

tux42.sh

```
/bin/bash
ifconfig eth0 up
ifconfig eth0 172.16.41.1/24
route add default gw 172.16.41.254
echo 1 > /proc/sys/net/ipv4/conf/eth0/accept_redirects
echo 1 > /proc/sys/net/ipv4/conf/all/accept_redirects
vi /etc/resolv.conf search netlab.fe.up.pt nameserver 172.16.1.1
```

tux44.sh

```
/bin/bash
ifconfig eth0 up
ifconfig eth0 172.16.40.254/24
ifconfig eth1 up
ifconfig eth1 172.16.41.253/24
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
route add default gw 172.16.41.254
vi /etc/resolv.conf search netlab.fe.up.pt nameserver 172.16.1.1
```