U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Redes de Computadores

2º Trabalho Laboratorial

Licenciatura em Engenharia Informática e Computação

2023

João Alves **up202108670**@fe.up.pt

Eduardo Sousa **up202103342**@fe.up.pt

Índice

Sumário

Este projeto, desenvolvido no contexto da disciplina de Redes de Computadores, tem como objetivo a implementação de um programa de download utilizando o protocolo FTP, juntamente com a configuração e operação de uma rede de computadores. Ao aplicar os conceitos aprendidos durante as aulas teóricas, conseguimos criar o programa mencionado e realizar a configuração da rede.

Introdução

O propósito deste projeto consistiu no desenvolvimento e teste de um programa de download, utilizando o protocolo FTP, além da configuração de uma rede de computadores conforme as diretrizes do guião. O relatório subsequente está estruturado em seções que abrangem diversos aspetos do projeto, como demonstrado um pouco acima pelo índice.

Aplicação de Download

Arquitetura

Este código foi desenvolvido para implementar um download de um ficheiro presente no servidor da FEUP, seguindo o protocolo FTP. Para isso, tivemos em consideração a norma RFC 959.

Primeiramente, o URL que é passado à aplicação vai ser parsed com recurso à função **parseArgument**, em *user*, *password*, *host*, *resource*, *file* e *ip*. O *user* e *password* são inputs opcionais e são usados para fazer a autenticação no servidor, o *host* serve para especificar o nome do servidor do qual queremos fazer o download do ficheiro, o *resource* que é o path do ficheiro pretendido, o *file* que é a especificação do nome do ficheiro (incluindo a sua extensão), e o *ip* que é obtido através da conversão do host, pelo mecanismo do DNS. Estes parâmetros são então imprimidos no ecrã para verificação da sua validade pelo utilizador.

Em seguida, é criado o socket de controlo com recurso à função **createAndConnectSocket** para envio inicial de pacotes de estabelecimento de ligação do utilizador com o servidor, sendo usadas as funções **authenticateConnection** e **enterPassiveMode**, que servem, respectivamente, para autenticar a conexão do utilizador com o servidor com recurso ao *user* e *password* previamente parsed, e converter a conexão para o modo passivo para então iniciar o *download* propriamente dito.

Neste momento, se tudo tiver corrido como o previsto nos passos anteriores, é criado então outro socket para receber o ficheiro que irá ser transferido através da função **createAndConnectSocket**, e é então pedido e transferido o ficheiro através das funções **requestFTPResource** e **getFTPResource**.

Quase todas as funções previamente referidas fazem uso da função **readFTPServerResponse** que lê o código enviado pelo servidor como resposta aos pedidos efetuados, verificando depois então se correu tudo como o esperado, e se não correu como desejado, é então impressa uma mensagem de erro no ecrã, de modo, ao utilizador conseguir ver o que correu mal durante a execução do código de *download*.

Se tudo tiver corrido como o esperado, então, no final, a conexão do cliente com o servidor é fechada através da função **closeFTPConnection**, finalizando assim a execução da aplicação de *download*, e obtendo o ficheiro desejado.

Testes

Testámos o código com ficheiros de vários tamanhos e formatos, de modo anónimo e autenticado, sendo que todos os testes de transferência foram concluídos com sucesso. O código também apresenta sempre uma mensagem de erro caso algo falhe na transferência. Apresentamos em anexo imagens de transferências concluídas com sucesso.

Configuração de Redes e Análise

Experiência 1 – Configuração de IP de rede

Os logs da experiência e comandos para configuração encontram-se disponíveis em anexo.

O objetivo principal das experiência realizada foi a configuração de endereços IP em duas máquinas, Tux63 e Tux64, e a análise do funcionamento do protocolo ARP (Address Resolution Protocol) e do protocolo ICMP (Internet Control Message Protocol). Para isso, foi necessário utilizar o comando *ifconfig* para atribuir um endereço IP (172.16.60.1/24, para o Tux63 e 172.16.60.254/24, para o Tux64) a cada uma das máquinas e o comando *ping* para verificar a conexão entre elas. Todos os comandos usados para esta experiência estão em anexo.

O protocolo ARP é responsável pelo mapeamento de um endereço IP de uma máquina para um endereço MAC na rede local. Quando uma máquina tenta enviar um pacote para outra na mesma rede local, e a tabela ARP não tem entradas para o IP da máquina destinatária, a máquina remetente envia um pacote ARP, por broadcast, para todas as máquinas da rede local perguntando qual delas tem um endereço MAC que corresponde ao endereço IP do destinatário. O destinatário responde com outro pacote ARP que indica à máquina remetente qual é o seu endereço MAC, permitindo assim a transferência de pacotes.

O protocolo ICMP, por outro lado, é utilizado para trocar mensagens de controlo, indicando sucesso ou erros durante a comunicação com outro endereço IP. No contexto da experiência, foram observados pacotes ICMP transportando mensagens de resposta e de pedido.

A análise dos pacotes trocados e das tabelas ARP permitiu observar os endereços MAC e IP associados a cada máquina. Além disso, foi possível verificar que, quando as entradas das tabelas ARP são eliminadas, ocorre uma troca de pacotes ARP no início da conexão para repor as associações entre os endereços IP e MAC que foram eliminados.

Por fim, a interface loopback, uma interface virtual da rede que permite ao computador receber respostas de si mesmo, foi mencionada como uma ferramenta importante para testar a correta configuração da placa de rede.

What are the MAC and IP addresses of ARP packets and why?

Inicialmente são enviados 2 endereços de IP em Broadcast, que corresponde ao IP da máquina de destino e ao IP da máquina de origem, tendo como IPs, respetivamente, 172.16.60.254 (Tux64) e 172.16.60.1 (Tux53). Estes IPs são enviados no mesmo pacote do tipo MAC. Depois, como resposta, a máquina Tux64 envia através de um pacote do tipo ARP o seu endereço MAC, que no caso é: 00:0f:fe:8c:af:71.

What packets does the ping command generate?

Enquanto não for obtido o endereço MAC da máquina de destino, o comando ping cria pacotes do tipo ARP. Depois de obter o endereço MAC da máquina de destino, o comando ping irá passar a gerar pacotes do tipo ICMP (Internet Control Message Protocol).

What are the MAC and IP addresses of the ping packets?

Os endereços IP e MAC usados em pacotes são os das máquinas de comunicação, ou seja, das máquinas Tux63 e Tux64.

How to determine if a receiving Ethernet frame is ARP, IP, ICMP?
Para determinar qual o tipo de protocolo que foi recebido basta olhar para a coluna “Protocol” do Wireshark e visualizar, ou ao inspecionar o header de ethernet de um pacote (primeiros 2 bytes), conseguimos determinar qual foi o protocolo usado pelo pacote.

How to determine the length of a receiving frame?
O Wireshark apresenta uma coluna que nos indica o tamanho do frame recebido, em bytes. Normalmente, essa informação também pode ser obtida através da análise do cabeçalho dos frames.

Experiência 2 – Implementar *bridges* num switch

Os logs da experiência e comandos para configuração encontram-se disponíveis em anexo.

Na experiência 2, o foco foi a criação e configuração de VLANs, utilizando switches para segmentar o tráfego de rede e organizar os dispositivos em domínios de broadcast distintos. Foram configuradas duas bridges com os Tuxs63 e Tux64 em uma e o Tux62 em outra, utilizando duas bridges no switch. Os comandos usados para a configuração das bridges estão em anexo.

Para a configuração das VLANs, foi necessário configurar o endereço IP 172.16.61.1/24 para o Tux62 e criar as bridges 60 e 61 no switch. As interfaces conectadas a cada um dos Tuxs foram adicionadas às bridges correspondentes. Isso resultou em dois domínios de broadcast separados, onde o Tux63 podia alcançar o Tux64, mas não o Tux62, que estava em uma LAN isolada.

No final da experiência, existem dois domínios de broadcast porque existem duas bridges implementadas. Pela análise dos logs concluímos que o Tux63 conseguiu resposta do ping ao Tux64 e não conseguiu resposta do ping ao Tux62, o que faz sentido, pois o Tux64 encontra-se na mesma bridge do Tux63 e o Tux62 encontra-se numa bridge diferente.

Experiência 3 – Configurar um router em Linux

Os logs da experiência e comandos para configuração encontram-se disponíveis em anexo.

Partindo da configuração obtida na experiência 2, para configurar o Tux64 como router, ligamos o eth1 do Tux64 ao switch e configuramos o IP 172.16.61.253/24, removemos a interface que estava conectada ao Tux64 e adicioná-la à *bridge61*. Ativamos o *ip_forward* e desativamos o *icmp_echo_ignore_broadcasts*, e adicionamos as rotas nos Tux62 e Tux63 para as LANs 172.16.60.0/24 e 172.16.61.0/24 respetivamente utilizando os IPs do Tux64 como gateway (172.16.61.253 e 172.16.60.254 respetivamente).

Estas *routes* permitem utilizar o Tux64 como router entre as LANs 172.16.60.0/24 e 172.16.61.0/24, conectando assim os Tux63 e Tux62. Quando fazemos *ping* de um para o outro, os pacotes chegam ao destino com sucesso.

What routes are there in the tuxes? What is their meaning?
Nos Tux63 e Tux62 existem duas rotas que usam o Tux64 como *gateway*, que permitem a comunicação entre eles dado que o Tux64 é o único ponto comum entre as duas LANs.

What information does an entry of the forwarding table contain?
Cada entrada na tabela contém o endereço de destino, assim como o *gateway* necessário para lá chegar.

What ARP messages, and associated MAC addresses, are observed and why?
Ao fazer *ping* do Tux62 ao Tux63 por exemplo, os pacotes ARP apenas têm os endereços MAC dos Tux62 e Tux64. Como o Tux62 não conhece o endereço do Tux63, o Tux62 só pode utilizar o endereço da *gateway*, que é o Tux64, que está definida para ser utilizada para a ligação à LAN onde está o Tux63.

What ICMP packets are observed and why?
Ao fazer *ping* do Tux62 ao Tux63, os pacotes ICMP têm o endereço de origem, que neste caso é o IP do Tux62, e o endereço de destino, que é o IP do Tux63.

What are the IP and MAC addresses associated with ICMP packets and why?
Ao fazer *ping* do Tux62 ao Tux63, como explicado na pergunta anterior, os pacotes ICMP têm como endereço de origem o IP do Tux62 e como endereço de destino o IP do Tux63. Como o Tux64 é que faz a ligação entre as duas bridges, os pacotes também têm o endereço MAC do Tux64.

Experiência 4 – Configurar router comercial e implementar o NAT

Os logs da experiência encontram-se disponíveis em anexo.

Partindo da configuração obtida na experiência 3, ligamos uma entrada do router à régua e outra entrada ao *switch*. Adicionamos a interface da bridge 61 e trocamos o cabo do Tux62 ao *switch* para o router, e configuramos os IPs de cada interface.

No final, adicionamos rotas em cada um dos Tuxs para serem usadas por defeito. Nos Tux64 e Tux62, estas rotas usam o IP do router como *gateway*, e no Tux63, a *gateway* é o IP do Tux64 (172.16.60.254). Verificamos que o Tux63 pode fazer *ping* a qualquer um dos outros elementos.

Após, no Tux63, desativarmos o accept_redirects e remover a rota para 172.16.60.0/24 através do Tux64, verificamos que os pacotes, quando o Tux62 faz *ping* ao Tux63, são enviados através do router e depois pelo Tux64. Ao reativarmos o accept_redirects ou readicionando a route através do Tux64, verificamos que os pacotes passam a ser enviados através do Tux64 diretamente (sem passar pelo router).

Fazendo *ping* ao endereço 172.16.1.254 a partir do Tux63, verificamos que a conexão está estabelecida. No entanto, desativando a NAT no router e repetindo o *ping*, a conexão já não é estabelecida. Isto acontece porque a NAT traduz endereços públicos para endereços da rede local (e vice-versa). Quando um pacote é enviado para uma rede externa à rede local, o endereço de origem que possui é o endereço público, que é portanto o endereço de destino da resposta. Quando a NAT está desativada, o pacote chega ao router e como o endereço que tem é o público ao invés do local, o router não o reencaminha para o Tux63.

How to configure a static route in a commercial router?
Para configurar uma static route num router comercial, faz-se reset às configurações que ele tem, adiciona-se o router à rede local, e atribui-se-lhe um IP interno e um externo.

How to configure NAT in a commercial router?
Para configurar a NAT utiliza-se o comando */ip firewall nat enable 0*, executando-o no terminal do router.

Experiência 5 – DNS

Os logs da experiência encontram-se disponíveis em anexo.

O objetivo principal das experiências realizadas foi configurar e entender o funcionamento do DNS (Domain Name System) dentro de uma rede. Para isso, foi necessário alterar o conteúdo do ficheiro `/etc/resolv.conf` em cada máquina da rede, inserindo a linha `nameserver 172.16.1.1`, que representa o endereço IP do servidor DNS. Também pode ser utilizado o comando `search`, que representa o nome do servidor DNS.

A troca de pacotes DNS na rede ocorre antes de qualquer outro protocolo, pois é ele que realiza a tradução do nome de domínio para o endereço IP que será utilizado pelos demais protocolos. Nesse processo, o host envia para o servidor um pacote com o hostname, esperando que seja retornado o seu endereço IP. O servidor, por sua vez, responde com um pacote contendo o endereço IP correspondente ao hostname.

Os resultados dessas experiências foram verificados através do comando `ping` para o google.com e da análise dos logs de captura de pacotes. Através dessas práticas, foi possível observar como a configuração do DNS altera a maneira como as máquinas se conectam à Internet.

Experiência 6 – Conexão TCP

Os logs da experiência estão disponíveis em anexo.

Na experiência final, para além de utilizarmos a rede que foi configurada ao longo dos passos anteriores, também fizemos uso do código *download* desenvolvido para podermos visualizar o envio e receção de *frames* e os mecanismos de controlo do fluxo e congestionamento de *data* que são comuns para ligações do tipo TCP.

Primeiramente, começamos por compilar o código do download e fazer a transferência de um ficheiro disponibilizado no servidor ftp da feup, observando que tipo de pacotes eram transferidos. Verificamos que inicialmente foram transferidos pacotes do tipo DNS, de modo, a ser possível traduzir o nome do servidor para um endereço de IP, e que de seguida houve troca de pacotes FTP para estabelecer um handshake entre o servidor e o cliente. Após isso, verificamos a troca de pacotes do tipo FTP que transportavam *data* que seriam os pacotes com as informações do ficheiro a fazer download, e no final vimos pacotes do tipo FTP com intuito de encerrar a ligação entre o server e o cliente.

Foram abertas duas conexões TCP (Transfer Control Protocol) durante a ligação FTP, sendo uma destinada para o envio de pacotes de controlo para o servidor e a outra para tratar da receção dos pacotes de *data* do ficheiro que queremos fazer *download.* Todas as conexões TCP usam ARQ (Automatic Repeat reQuest) para controlar erros, empregando mensagens ACK para confirmar a recepção correta de pacotes e timeouts para verificar o tempo de receção. Se um timeout é excedido, assume-se que o pacote foi perdido e é retransmitido pela rede. Este mecanismo assegura a transferência de dados, incorporando controle de fluxo e controle de congestionamento. A deteção de perda de um pacote ocorre por três ACKs consecutivos ou por *timeout*, levando a uma significativa redução no número de pacotes enviados na transmissão a seguir.

O controlo de congestionamento, executado pelos emissores, implica o uso do método Selective Repeat, onde os pacotes são enviados sem esperar pelos respetivos ACK. Para determinar a capacidade da rede durante a transferência, o emissor envia vários pacotes de uma só vez, utilizando uma de duas abordagens. Pode usar a abordagem de começo lento, que envia um número de pacotes exponencial em relação à transferência anterior, ou a abordagem de aumento aditivo, que é semelhante à anterior, mas com aumentos de apenas mais um pacote em relação à transferência anterior.

O controlo de fluxo possibilita que o destinatário gerencie a quantidade de pacotes recebidos. Os buffers nos lados do destinatário e do emissor têm capacidades limitadas, evitando a perda de pacotes na rede. Em cada pacote enviado, é incluído um tamanho máximo para que o emissor saiba quantos bytes pode enviar sem causar problemas. Durante a experiência, observámos que o tamanho máximo de bytes a serem enviados diminuía em momentos de tráfego intenso na rede.

A perda de um pacote é detetada por timeout ou três ACKs consecutivos, resultando numa redução drástica no número de pacotes enviados na próxima transmissão, aproximando-se pela metade do valor que causou a perda.

No final da experiência, verificamos que a taxa de transferência de uma ligação de dados TCP é afetada pela presença de uma segunda ligação TCP. Isto ocorre porque o Tux63 e o Tux62 partilham o mesmo percurso de rede, competindo por largura de banda e recursos disponíveis. Isso resultou numa diminuição da taxa de transferência para cada ligação. Apesar dos mecanismos de controlo de congestionamento tentarem gerir a situação ajustando as taxas de transmissão, a interferência entre as ligações TCP dos Tuxs impacta negativamente a eficiência da transferência de dados, ou seja, a velocidade de *download* nos Tuxs irá ser bastante inferior em relação ao caso em que não existiria uma segunda ligação TCP a usar o mesmo percurso de rede.

Conclusão

O segundo projeto da cadeira de RCOM teve como propósito configurar uma rede e implementar um cliente de download. Durante o processo, exploramos e assimilamos novos conceitos relacionados com funcionalidades comuns no nosso dia a dia, assim como o protocolo em estudo. Em resumo, o projeto foi concluído, atingindo praticamente todos os objetivos propostos. A sua elaboração contribuiu de forma positiva para aprofundar o conhecimento, tanto teórico quanto prático, no âmbito do tema abordado.

Referências

- RFC 959 disponibilizado no moodle
- Beej's Guide to Network Programming - Using Internet Sockets

Anexos

Experiência 1 – Configuração de IP de rede

What are the commands required to configure this experience?

> /system config reset

$ ifconfig eth0 up

$ ifconfig eth0 172.16.60.1/24

$ ifconfig eth0 172.16.60.254/24

$ ping 172.16.60.1 -c 5

$ ping 172.16.60.254 -c 5

$ arp -a

$ arp -d 172.16.60.254/24

$ arp -a

$ ping 172.16.60.254 -c 5

22	20.743875	172.16.60.1	172.16.60.255	ICMP	98 Echo (ping) request id=0x1724, seq=5/1280, ttl=64 (no response found!)
23	20.743905	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1724, seq=5/1280, ttl=64
24	21.743880	172.16.60.1	172.16.60.255	ICMP	98 Echo (ping) request id=0x1724, seq=6/1536, ttl=64 (no response found!)
25	21.743908	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1724, seq=6/1536, ttl=64
26	21.755691	HewlettPacka_c5:61:…	GProComputer_8c:af:…	ARP	42 Who has 172.16.60.1? Tell 172.16.60.254
27	21.755981	GProComputer_8c:af:…	HewlettPacka_c5:61:…	ARP	60 172.16.60.1 is at 00:0f:fe:8c:af:71
28	22.064101	Cisco_3a:f1:06	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8006
29	22.623057	Cisco_3a:f1:06	Cisco_3a:f1:06	LOOP	60 Reply
30	22.743878	172.16.60.1	172.16.60.255	ICMP	98 Echo (ping) request id=0x1724, seq=7/1792, ttl=64 (no response found!)
31	22.743905	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1724, seq=7/1792, ttl=64
32	23.743863	172.16.60.1	172.16.60.255	ICMP	98 Echo (ping) request id=0x1724, seq=8/2048, ttl=64 (no response found!)
33	23.743892	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1724, seq=8/2048, ttl=64
34	24.071463	Cisco_3a:f1:06	Spanning-tree-(for-…	STP	60 Conf. Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8006
35	24.743874	172.16.60.1	172.16.60.255	ICMP	98 Echo (ping) request id=0x1724, seq=9/2304, ttl=64 (no response found!)
36	24.743900	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1724, seq=9/2304, ttl=64
37	24.907358	Cisco_3a:f1:06	CDP/VTP/DTP/PAgP/UD…	CDP	435 Device ID: tux-sw6 Port ID: FastEthernet0/4
38	25.743863	172.16.60.1	172.16.60.255	ICMP	98 Echo (ping) request id=0x1724, seq=10/2560, ttl=64 (no response found!)

Experiência 2 – Implementar *bridges* num switch

How to configure bridgeY0?

Para configurar uma bridge, primeiro temos de a criar, de modo a podermos então depois configurá-la de modo a criar uma sub-rede que irá conter o Tux63 e o Tux64. Após a sua criação eliminamos as configurações default atribuídas pelo switch aquando da criação da bridge, de modo a podermos configurá-la como é suposto. Os comandos para fazer a configuração são os seguintes (M e N representam o número da porta do switch onde os Tuxs estão ligados):

> /interface bridge add name=bridge60

> /interface bridge port remove [find interface=ether1]

> /interface bridge port remove [find interface=ether2]

> /interface bridge port remove [find interface=ether3]

> /interface bridge port add bridge=bridge60 interface=ether1

> /interface bridge port add bridge=bridge60 interface=ether2

> /interface bridge port add bridge=bridge61 interface=ether3

13	16.038632	Cisco_3a:f1:03	Spanning-tree-(for-…	STP	60 Conf. TC + Root = 32768/60/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8003
14	16.318592	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x1608, seq=2/512, ttl=64 (reply in 15)
15	16.318932	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1608, seq=2/512, ttl=64 (request in 14)
16	17.318593	172.16.60.1	172.16.60.254	ICMP	98 Echo (ping) request id=0x1608, seq=3/768, ttl=64 (reply in 17)
17	17.318958	172.16.60.254	172.16.60.1	ICMP	98 Echo (ping) reply id=0x1608, seq=3/768, ttl=64 (request in 16)

```
 8 10.059104   172.16.60.1        172.16.61.1        ICMP   98 Echo (ping) request  id=0x1691, seq=1/256, ttl=64 (no response found!)
 9 11.058200   172.16.60.1        172.16.61.1        ICMP   98 Echo (ping) request  id=0x1691, seq=2/512, ttl=64 (no response found!)
10 12.028995   Cisco_3a:f1:03     Spanning-tree-(for-… STP  60 Conf. Root = 32768/60/fc:fb:fb:3a:f1:00  Cost = 0  Port = 0x8003
11 12.058194   172.16.60.1        172.16.61.1        ICMP   98 Echo (ping) request  id=0x1691, seq=3/768, ttl=64 (no response found!)
12 13.058203   172.16.60.1        172.16.61.1        ICMP   98 Echo (ping) request  id=0x1691, seq=4/1024, ttl=64 (no response found!)
13 14.033798   Cisco_3a:f1:03     Spanning-tree-(for-… STP  60 Conf. Root = 32768/60/fc:fb:fb:3a:f1:00  Cost = 0  Port = 0x8003
14 14.058263   172.16.60.1        172.16.61.1        ICMP   98 Echo (ping) request  id=0x1691, seq=5/1280, ttl=64 (no response found!)
15 15.058205   172.16.60.1        172.16.61.1        ICMP   98 Echo (ping) request  id=0x1691, seq=6/1536, ttl=64 (no response found!)
```

Experiência 3 – Configurar um router em Linux

What are the commands required to configure this experience?

$ ifconfig eth1 up
$ ifconfig eth1 172.16.61.253/24
> /interface bridge port remove [find interface=ether4]
> /interface bridge port add bridge=bridge61 interface=ether4
$ sysctlnet.ipv4.ip_forward=1
$ sysctl net.ipv4.icmp_echo_ignore_broadcasts=0
$ route add -net 172.16.60.0/24 gw 172.16.61.253
$ route add -net 172.16.61.0/24 gw 172.16.60.254
$ arp -d 172.16.60.1
$ arp -d 172.16.61.1
$ arp -d 172.16.61.253
$ arp -d 172.16.60.254

```
11 15.606178   GProComputer_8c:af:… Broadcast          ARP    60 Who has 172.16.60.254? Tell 172.16.60.1
12 15.606214   HewlettPacka_c5:61:… GProComputer_8c:af:… ARP  42 172.16.60.254 is at 00:21:5a:c5:61:bb
13 15.606553   172.16.60.1        172.16.61.1        ICMP   98 Echo (ping) request  id=0x5338, seq=1/256, ttl=64 (reply in 14)
14 15.606871   172.16.61.1        172.16.60.1        ICMP   98 Echo (ping) reply    id=0x5338, seq=1/256, ttl=63 (request in 13)
15 16.607332   172.16.60.1        172.16.61.1        ICMP   98 Echo (ping) request  id=0x5338, seq=2/512, ttl=64 (reply in 16)
16 16.607544   172.16.61.1        172.16.60.1        ICMP   98 Echo (ping) reply    id=0x5338, seq=2/512, ttl=63 (request in 15)
17 17.229918   Cisco_3a:f1:06     Spanning-tree-(for-… STP  60 Conf. Root = 32768/60/fc:fb:fb:3a:f1:00  Cost = 0  Port = 0x8006
18 17.606331   172.16.60.1        172.16.61.1        ICMP   98 Echo (ping) request  id=0x5338, seq=3/768, ttl=64 (reply in 19)
19 17.606525   172.16.61.1        172.16.60.1        ICMP   98 Echo (ping) reply    id=0x5338, seq=3/768, ttl=63 (request in 18)
20 18.606003   172.16.60.1        172.16.61.1        ICMP   98 Echo (ping) request  id=0x5338, seq=4/1024, ttl=64 (reply in 21)
21 18.606194   172.16.61.1        172.16.60.1        ICMP   98 Echo (ping) reply    id=0x5338, seq=4/1024, ttl=63 (request in 20)
```

Experiência 4 – Configurar router comercial e implementar o NAT

> /interface bridge port remove [find interface=ether5]
> /interface bridge port add bridge=bridge61 interface=ether5
> /system reset-configuration
> /ip address add address=172.16.1.59/24 interface=ether1
> /ip address add address=172.16.61.254/24 interface=ether2
$ route add default gw 172.16.61.254 (Tux62)
$ route add default gw 172.16.60.254 (Tux63)
$ route add default gw 172.16.61.254 (Tux64)
> /ip route add dst-address=172.16.60.0/24 gateway=172.16.61.253
> /ip route add dst-address=0.0.0.0/0 gateway=172.16.1.254
$ echo 0 > /proc/sys/net/ipv4/conf/eth0/accept_redirects (Tux62)
$ echo 0 > /proc/sys/net/ipv4/conf/all/accept_redirects (Tux62)
$ route del -net 172.16.60.0 gw 172.16.61.253 netmask 255.255.255.0 (Tux62)
> /ip firewall nat disable 0
> /ip firewall nat enable 0

51 42.649832	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) request id=0x1048, seq=6/1536, ttl=63 (reply in 52)
52 42.649853	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1048, seq=6/1536, ttl=64 (request in 51)
53 42.656503	HewlettPacka_5a:74:…	Netronix_71:73:da	ARP	42	Who has 172.16.61.253? Tell 172.16.61.1
54 42.656593	Netronix_71:73:da	HewlettPacka_5a:74:…	ARP	60	172.16.61.253 is at 00:08:54:71:73:da
55 43.649859	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) request id=0x1048, seq=7/1792, ttl=63 (reply in 56)
56 43.649887	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1048, seq=7/1792, ttl=64 (request in 55)

21 26.061363	Cisco_3a:f1:06	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/1/fc:fb:fb:3a:f1:00 Cost = 0 Port = 0x8006
22 26.432283	172.16.60.1	172.16.60.254	ICMP	98	Echo (ping) request id=0x1431, seq=3/768, ttl=64 (reply in 23)
23 26.432313	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1431, seq=3/768, ttl=64 (request in 22)
24 27.431281	172.16.60.1	172.16.60.254	ICMP	98	Echo (ping) request id=0x1431, seq=4/1024, ttl=64 (reply in 25)
25 27.431309	172.16.60.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1431, seq=4/1024, ttl=64 (request in 24)
26 27.953579	Cisco_3a:f1:06	Cisco_3a:f1:06	LOOP	60	Reply

46 12.859760	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) request id=0x1c54, seq=10/2560, ttl=64 (reply in 48)
47 12.860565	172.16.61.254	172.16.61.1	ICMP	70	Redirect (Redirect for host)
48 12.860853	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) reply id=0x1c54, seq=10/2560, ttl=63 (request in 46)
49 13.860923	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) request id=0x1c54, seq=11/2816, ttl=64 (reply in 51)
50 13.861723	172.16.61.254	172.16.61.1	ICMP	70	Redirect (Redirect for host)
51 13.862026	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) reply id=0x1c54, seq=11/2816, ttl=63 (request in 49)
52 14.034144	Cisco_7b:ce:82	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/61/00:1e:14:7b:ce:80 Cost = 0 Port = 0x8002

5 6.141252	172.16.60.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x1408, seq=1/256, ttl=64 (no response found!)
6 6.142184	172.16.61.254	172.16.60.1	ICMP	70	Destination unreachable (Host unreachable)
7 7.142045	172.16.60.1	172.16.1.254	ICMP	98	Echo (ping) request id=0x1408, seq=2/512, ttl=64 (no response found!)
8 7.142940	172.16.61.254	172.16.60.1	ICMP	70	Destination unreachable (Host unreachable)

Experiência 5 – DNS

6 7.398025	172.16.60.1	172.16.1.1	DNS	70	Standard query 0x7a83 A google.com
7 7.399533	172.16.1.1	172.16.60.1	DNS	222	Standard query response 0x7a83 A google.com A 216.58.201.142 NS ns3.google.com NS ns2.google.com NS n…
8 7.399943	172.16.60.1	216.58.201.142	ICMP	98	Echo (ping) request id=0x67fc, seq=1/256, ttl=64 (reply in 9)
9 7.413373	216.58.201.142	172.16.60.1	ICMP	98	Echo (ping) reply id=0x67fc, seq=1/256, ttl=51 (request in 8)
10 7.414437	172.16.60.1	172.16.1.1	DNS	87	Standard query 0xe25e PTR 142.201.58.216.in-addr.arpa
11 7.415575	172.16.1.1	172.16.60.1	DNS	303	Standard query response 0xe25e PTR 142.201.58.216.in-addr.arpa PTR mad06s25-in-f142.1e100.net PTR mad…

Experiência 6 – Conexão TCP

Início de conexão

44 1.441916	2a00:1450:4003:808:…	2001:8a0:fdea:1c00:…	QUIC	86	Protected Payload (KP0)
45 1.638209	192.168.1.68	239.255.255.250	UDP	1090	1069 → 8082 Len=1048
46 1.776738	192.168.1.72	192.168.1.254	DNS	69	Standard query 0xbd69 A ftp.up.pt
47 1.781396	192.168.1.254	192.168.1.72	DNS	85	Standard query response 0xbd69 A ftp.up.pt A 193.137.29.15
48 1.803953	192.168.1.72	193.137.29.15	TCP	62	52303 → 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM
49 1.816816	193.137.29.15	192.168.1.72	TCP	62	21 → 52303 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1380 SACK_PERM
50 1.866885	192.168.1.72	193.137.29.15	TCP	54	52303 → 21 [ACK] Seq=1 Ack=1 Win=64240 Len=0
51 1.874030	162.159.134.234	192.168.1.72	TLSv1.2	100	Application Data
52 1.883672	193.137.29.15	192.168.1.72	FTP	127	Response: 220-Welcome to the University of Porto's mirror archive (mirrors.up.pt)
53 1.884158	193.137.29.15	192.168.1.72	FTP	129	Response: 220----------
54 1.884158	193.137.29.15	192.168.1.72	FTP	139	Response: 220-All connections and transfers are logged. The max number of connections is 200.
55 1.884394	193.137.29.15	192.168.1.72	FTP	72	[TCP Previous segment not captured] Response: 220-
56 1.885684	193.137.29.15	192.168.1.72	TCP	195	[TCP Out-Of-Order] 21 → 52303 [PSH, ACK] Seq=234 Ack=1 Win=64240 Len=141
57 1.929973	192.168.1.72	193.137.29.15	TCP	54	52303 → 21 [ACK] Seq=1 Ack=149 Win=64092 Len=0
58 1.930108	192.168.1.72	193.137.29.15	TCP	66	52303 → 21 [ACK] Seq=1 Ack=234 Win=64007 Len=0 SLE=375 SRE=393
59 1.930145	192.168.1.72	193.137.29.15	TCP	54	52303 → 21 [ACK] Seq=1 Ack=393 Win=63848 Len=0

Fim de conexão

73 2.239797	192.168.1.72	193.137.29.15	FTP	82	Request: retr pub/kodi/timestamp.txt
74 2.253181	193.137.29.15	192.168.1.72	FTP	134	Response: 150 Opening BINARY mode data connection for pub/kodi/timestamp.txt (11 bytes).
75 2.255441	193.137.29.15	192.168.1.72	FTP-DA…	65	FTP Data: 11 bytes (PASV) (pass password)
76 2.256050	193.137.29.15	192.168.1.72	TCP	60	55922 → 52304 [FIN, ACK] Seq=12 Ack=1 Win=64240 Len=0
77 2.302447	192.168.1.72	193.137.29.15	TCP	54	52304 → 55922 [ACK] Seq=1 Ack=13 Win=64229 Len=0
78 2.315233	193.137.29.15	192.168.1.72	FTP	78	Response: 226 Transfer complete.
79 2.333751	192.168.1.72	193.137.29.15	TCP	54	52303 → 21 [ACK] Seq=63 Ack=582 Win=63659 Len=0
80 2.364493	192.168.1.72	193.137.29.15	FTP	59	Request: quit
81 2.377190	193.137.29.15	192.168.1.72	FTP	68	Response: 221 Goodbye.
82 2.377423	193.137.29.15	192.168.1.72	TCP	60	21 → 52303 [FIN, ACK] Seq=620 Ack=68 Win=64173 Len=0
83 2.393846	168.95.245.3	192.168.1.72	ICMP	60	Echo (ping) reply id=0x0001, seq=20274/12879, ttl=53 (request in 71)
84 2.410587	192.168.1.72	193.137.29.15	TCP	54	52303 → 21 [ACK] Seq=68 Ack=621 Win=63621 Len=0
85 2.410805	192.168.1.72	193.137.29.15	TCP	54	52303 → 21 [FIN, ACK] Seq=68 Ack=621 Win=63621 Len=0
86 2.410852	192.168.1.72	193.137.29.15	TCP	54	52304 → 55922 [FIN, ACK] Seq=1 Ack=13 Win=64229 Len=0
87 2.424915	193.137.29.15	192.168.1.72	TCP	60	21 → 52303 [ACK] Seq=621 Ack=69 Win=64173 Len=0

download.c

```c
#include "../include/download.h"


int parseArgument(const char *input, char *user, char *pass, char *host,
char *resource, char *filename, char *ip) {

    const char *start = "ftp://";

    int index = 0;

    int i = 6;


    // Extract host from the FTP URL

    while (input[i] != '/') {

        host[index++] = input[i++];

    }

    host[index] = '\0';

    i++;

    index = 0;


    // Extract resource from the FTP URL

    while (input[i] != '\0') {

        resource[index++] = input[i++];

    }

    resource[index] = '\0';


    int auth = 0;

    // Check if authentication information is present in the host

    for (int i = 0; i < strlen(host); i++) {

        if (host[i] == '@') {
```

```
            auth = 1;
        }
    }

    // Parse authentication information if present, otherwise use
anonymous credentials
    if (auth) {
        index = 0;
        while (host[0] != ':') {
            user[index++] = host[0];
            for (int j = 0; j < strlen(host); j++) {
                host[j] = host[j + 1];
            }
        }
        for (int j = 0; j < strlen(host); j++) {
            host[j] = host[j + 1];
        }

        index = 0;
        while (host[0] != '@') {
            pass[index++] = host[0];
            for (int j = 0; j < strlen(host); j++) {
                host[j] = host[j + 1];
            }
        }
        for (int j = 0; j < strlen(host); j++) {
            host[j] = host[j + 1];
```

```
        }
    } else {
        strcpy(user, "anonymous");
        strcpy(pass, "anonymous");
    }

    int counter = 0;
    // Count the number of '/' characters in the resource path
    for (int i = 0; i < strlen(resource); i++) {
        if (resource[i] == '/') counter++;
    }
    int counter2 = 0;
    index = 0;

    // Extract filename from the resource path
    for (int i = 0; i < strlen(resource); i++) {
        if (counter2 == counter) {
            filename[index++] = resource[i];
        }
        if (resource[i] == '/') counter2++;
    }
    filename[index] = '\0';

    struct hostent *h;
    // Check if the host is valid and retrieve its IP address
    if (strlen(host) == 0) return -1;
    if ((h = gethostbyname(host)) == NULL) {
```

```
        printf(" > Invalid hostname '%s'\n", host);
        exit(-1);
    }
    strcpy(ip, inet_ntoa(*((struct in_addr *) h->h_addr)));

    // Return 0 on success, -1 on failure
    return (strlen(host) && strlen(user) && strlen(pass) &&
strlen(resource) && strlen(filename)) ? 0 : 1;
}


// 100% copied from given code, cannot be wrong
int createAndConnectSocket(char *ip, int port) {
    int sockfd;
    struct sockaddr_in server_addr;

    // Initialize the server address structure
    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*/

    // Create a TCP socket
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket()");
        exit(-1);
```

```
    }


    // Connect to the specified IP address and port
    if (connect(sockfd, (struct sockaddr *) &server_addr,
sizeof(server_addr)) < 0) {
        perror("connect()");
        exit(-1);
    }


    printf(" > Create socket worked fine\n");


    return sockfd;
}


int authenticateConnection(const int socket, const char *username, const
char *password){
    char uCmd[MAX_SIZE];
    char pCmd[MAX_SIZE];
    char response[MAX_SIZE];


    // Construct user and pass commands
    snprintf(uCmd, sizeof(uCmd), "user %s\n", username);
    snprintf(pCmd, sizeof(pCmd), "pass %s\n", password);


    // Send the user command to the server
    write(socket, uCmd, strlen(uCmd));


    // Check the server response for user command
```

```
    if (readFTPServerResponse(socket, response) != FTP_USER_NAME_OKAY) {
        printf(" > Error finding user");
        exit(EXIT_FAILURE);
    }

    // Send the password command to the server
    write(socket, pCmd, strlen(pCmd));

    // Return the server response for password command
    return readFTPServerResponse(socket, response);
}

int readFTPServerResponse(const int socket, char *buffer) {
    char byte;
    int index = 0, responseCode, state = 0;

    // Initialize the buffer with zeros
    memset(buffer, 0, MAX_SIZE);

    // Continue reading until the end of the response
    while (state != 3) {
        // Read one byte from the socket
        read(socket, &byte, 1);

        // Interpret the byte based on the current state
        if (state == 0) {
            if (byte == ' ') state = 1;
```

```
            else if (byte == '-') state = 2;
            else if (byte == '\n') state = 3;
            else buffer[index++] = byte;
        } else if (state == 1) {
            if (byte == '\n') state = 3;
            else buffer[index++] = byte;
        } else if (state == 2) {
            if (byte == '\n') {
                // Reset buffer on multi-line response
                memset(buffer, 0, MAX_SIZE);
                state = 0;
                index = 0;
            } else {
                buffer[index++] = byte;
            }
        }
    }

    // Parse the response code from the buffer
    sscanf(buffer, "%d", &responseCode);

    // Return the FTP server response code
    return responseCode;
}

int requestFTPResource(const int socket, const char *resource) {
    char answer[MAX_SIZE];
```

```
    // Construct the FTP retr command with the resource
    char fileCommand[MAX_SIZE];
    snprintf(fileCommand, sizeof(fileCommand), "retr %s\n", resource);

    // Send the command to the server
    write(socket, fileCommand, strlen(fileCommand));

    // Read and return the server response
    return readFTPServerResponse(socket, answer);
}



int enterPassiveMode(const int socket, char *ip, int *port) {
    char answer[MAX_SIZE];
    int ip1, ip2, ip3, ip4, port1, port2;

    // Send the "pasv" command to the server
    write(socket, "pasv\n", 5);

    // Read the server response and check if it indicates entering passive
mode
    if (readFTPServerResponse(socket, answer) !=
FTP_ENTERING_PASSIVE_MODE)
        return -1;

    // Find the opening parenthesis in the response
    char *start = strchr(answer, '(');
```

```
    if (start == NULL)
        return -1;

    // Extract IP address and port from the response
    if (sscanf(start + 1, "%d,%d,%d,%d,%d,%d", &ip1, &ip2, &ip3, &ip4,
&port1, &port2) != 6)
        return -1;

    // Format the IP address
    sprintf(ip, "%d.%d.%d.%d", ip1, ip2, ip3, ip4);

    // Calculate the port number
    *port = port1 * 256 + port2;

    return FTP_ENTERING_PASSIVE_MODE;
}

int getFTPResource(const int controlSocket, const int dataSocket, char
*filename) {
    FILE *fd = fopen(filename, "wb+");

    if (fd == NULL) {
        printf(" > Error opening file\n");
        exit(-1);
    }

    char buffer[MAX_SIZE];
    int bytes = 1;
```

```
    // Read data from the data socket and write it to the local file
    while (bytes > 0){
        bytes = read(dataSocket, buffer, MAX_SIZE);
        if (bytes > 0 && fwrite(buffer, bytes, 1, fd) < 0) {
            fclose(fd);
            return -1;
        }
    }

    fclose(fd);
    return readFTPServerResponse(controlSocket, buffer);
}

int closeFTPConnection(const int controlSocket, const int dataSocket){
    char answer[MAX_SIZE];

    // Send the quit command to the server
    write(controlSocket, "quit\n", 5);

    // Read the server response and check if it indicates successful
service closing
    if (readFTPServerResponse(controlSocket, answer) !=
FTP_SERVICE_CLOSING) return -1;

    // Close both control and data sockets
    return close(controlSocket) || close(dataSocket);
}
```

```c
void printURLInfo(char *user, char *password, char *host, char *resource,
char *file, char *ip) {
    printf(" > Host: %s\n > Resource: %s\n > File: %s\n > User: %s\n >
Password: %s\n > IP Address: %s\n\n", host, resource, file, user,
password, ip);
}

void printSocketError(const char *destination, const char *ip, int port) {
     printf(" > Socket to '%s:%d' for %s failed\n", ip, port,
destination);
     exit(EXIT_FAILURE);
}

void printError(const char *message) {
     printf(" > Error: %s\n", message);
     exit(EXIT_FAILURE);
}

int main(int argc, char *argv[]) {
    int port;
    char answer[MAX_SIZE];
    char user[MAX_SIZE];
    char password[MAX_SIZE];
    char host[MAX_SIZE];
    char resource[MAX_SIZE];
    char file[MAX_SIZE];
    char ip[MAX_SIZE];
```

```
    // Initialize arrays
    memset(answer, 0, MAX_SIZE);
    memset(user, 0, MAX_SIZE);
    memset(password, 0, MAX_SIZE);
    memset(host, 0, MAX_SIZE);
    memset(resource, 0, MAX_SIZE);
    memset(file, 0, MAX_SIZE);
    memset(ip, 0, MAX_SIZE);

    // Check the number of command-line arguments
    if (argc != 2)
        printError(" > Usage: ./download
ftp://[<user>:<password>@]<host>/<url-path>");

    // Parse the FTP URL
    if (parseArgument(argv[1], user, password, host, resource, file, ip)
!= 0)
        printError(" > Parse error. Usage: ./download
ftp://[<user>:<password>@]<host>/<url-path>");

    // Print parsed FTP URL information
    printURLInfo(user, password, host, resource, file, ip);

    // Create and connect the control socket
    int controlSocket = createAndConnectSocket(ip, SERVER_PORT);

    // Check the control socket and server response
```

```
    if (controlSocket < 0 || readFTPServerResponse(controlSocket, answer)
!= FTP_SERVICE_READY)
        printSocketError("control connection", ip, SERVER_PORT);

    // Authenticate the FTP connection
    if (authenticateConnection(controlSocket, user, password) !=
FTP_USER_LOGGED_IN)
        printError(" > Authentication failed");

    // Enter passive mode and get the data port
    if (enterPassiveMode(controlSocket, ip, &port) !=
FTP_ENTERING_PASSIVE_MODE)
        printError(" > Passive mode failed");

    // Create and connect the data socket
    int dataSocket = createAndConnectSocket(ip, port);

    // Check the data socket
    if (dataSocket < 0)
        printSocketError("data connection", ip, port);

    // Request the FTP resource
    if (requestFTPResource(controlSocket, resource) !=
FTP_FILE_STATUS_OKAY)
        printError(" > Unknown resource");

    // Get the FTP resource and check the data connection status
    if (getFTPResource(controlSocket, dataSocket, file) !=
FTP_CLOSING_DATA_CONNECTION)
```

```
        printError(" > Error transferring file");


    // Close both control and data sockets
    if (closeFTPConnection(controlSocket, dataSocket) != 0)
        printError(" > Sockets close error");


    // Print success message
    printf(" > File downloaded!\n");
    return 0;
}
```

download.h

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


#define MAX_SIZE        1000
#define SERVER_PORT     21
```

```
// https://datatracker.ietf.org/doc/html/rfc959
// PAG 39 - 42
// like hmtl response codes
// maybe need some more (?)
#define FTP_FILE_STATUS_OKAY          150  // File status okay; about to open data connection
#define FTP_SERVICE_READY             220  // Service ready for new user
#define FTP_SERVICE_CLOSING           221  // Service closing control connection
#define FTP_CLOSING_DATA_CONNECTION   226  // Closing data connection
#define FTP_ENTERING_PASSIVE_MODE     227  // Entering Passive Mode (h1,h2,h3,h4,p1,p2)
#define FTP_USER_LOGGED_IN            230  // User logged in, proceed
#define FTP_USER_NAME_OKAY            331  // User name okay, need password


/***** MAIN FUNCTIONS *****/


/**
 * @brief Reads and parses the FTP server response from the given socket.
 *
 * This function reads the response from the server character by character,
 * interpreting the FTP protocol's response codes. The response code is
 * extracted from the server's message and returned as an integer.
 *
 * @param socket The socket connected to the FTP server.
 * @param buffer A character buffer to store the server's response.
 * @return The FTP server response code.
```

```
 */
int readFTPServerResponse(const int socket, char *buffer);


/**
 * @brief Requests a resource from an FTP server.
 *
 * This function constructs an FTP "retr" command with the specified
resource,
 * sends the command to the server, and reads the server's response.
 *
 * @param socket The control socket connected to the FTP server.
 * @param resource A pointer to a char array containing the resource to
request.
 * @return Returns the FTP server response code.
 *         Exits the program with an error message on failure.
 */
int requestFTPResource(const int socket, const char *resource);


/**
 * @brief Authenticates the connection with an FTP server.
 *
 * This function sends the user and password commands to the server,
 * reads the server's responses, and returns the final authentication
status.
 *
 * @param socket The control socket connected to the FTP server.
 * @param username A pointer to a char array containing the FTP username.
 * @param password A pointer to a char array containing the FTP password.
```

```
 * @return Returns the FTP server response code for successful
authentication.
 *         Exits the program with an error message on failure.
 */
int authenticateConnection(const int socket, const char *username, const
char *password);


/**
 * @brief Creates and connects a socket to the specified IP address and
port.
 *
 * This function creates a TCP socket, sets up the server address
structure, and
 * connects to the specified IP address and port.
 *
 * @param ip A pointer to a char array containing the IP address to
connect to.
 * @param port The port number to connect to.
 * @return Returns the socket file descriptor on success.
 *         Exits the program with an error message on failure.
 */
int createAndConnectSocket(char *ip, int port);


/**
 * @brief Retrieves a file from the FTP server and saves it to a local
file.
 *
 * This function reads data from the data socket and writes it to a local
file specified by the filename.
```

```
 *
 * @param controlSocket The control socket for FTP communication.
 * @param dataSocket The data socket for transferring file data.
 * @param filename The name of the local file to save the retrieved data.
 * @return Returns the FTP server response code after completing the file transfer.
 */
int getFTPResource(const int controlSocket, const int dataSocket, char *filename);


/**
 * @brief Closes the FTP connection by sending a quit command to the server.
 *
 * This function sends a quit command to the FTP server, reads the server response,
 * and closes both the control and data sockets.
 *
 * @param controlSocket The control socket for FTP communication.
 * @param dataSocket The data socket for transferring file data.
 * @return Returns 0 on success, -1 on failure.
 */
int closeFTPConnection(const int controlSocket, const int dataSocket);


/**
 * Parses an FTP URL and extracts relevant information such as
 * user, password, host, resource, filename, and IP address.
 *
```

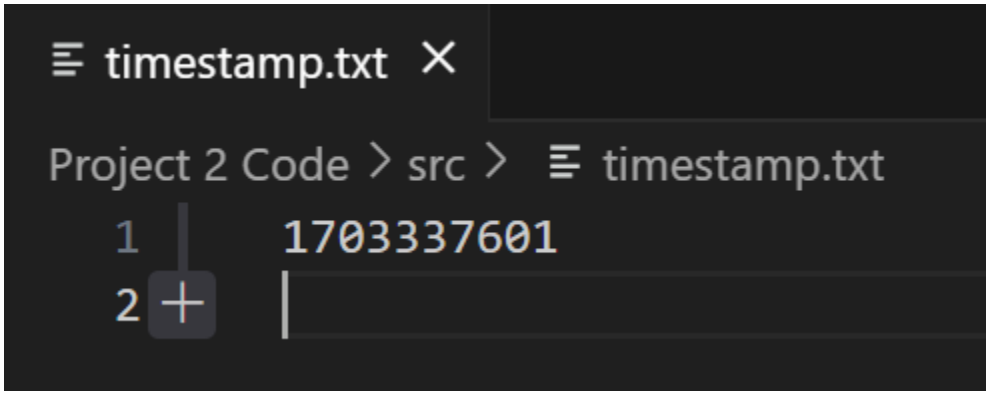
```
 * @param input The FTP URL to be parsed.
 * @param user Output parameter for the username.
 * @param pass Output parameter for the password.
 * @param host Output parameter for the host.
 * @param resource Output parameter for the resource path.
 * @param filename Output parameter for the filename.
 * @param ip Output parameter for the IP address.
 * @return Returns 0 on success, -1 on failure.
 */
int parseArgument(const char *argument, char *user, char *pass, char
*host, char *path, char *filename, char *ip);


/**
 * @brief Enters passive mode for FTP data transfer.
 *
 * This function sends a "pasv" command to the FTP server, reads the
server response,
 * and extracts the IP address and port for passive mode data transfer.
 *
 * @param socket The control socket for FTP communication.
 * @param ip A pointer to a char array to store the extracted IP address.
 * @param port A pointer to an integer to store the extracted port number.
 * @return Returns FTP_ENTERING_PASSIVE_MODE on success, -1 on failure.
 */
int enterPassiveMode(const int socket, char *ip, int *port);


/***** AUXILIARY FUNCTIONS *****/
```

```c
/**
 * @brief Prints URL information, including user, password, host, resource, file, and IP address.
 *
 * This function takes the components of a URL and prints them to the console.
 *
 * @param user The username for authentication.
 * @param password The password for authentication.
 * @param host The host address of the URL.
 * @param resource The resource part of the URL.
 * @param file The filename part of the URL.
 * @param ip The IP address associated with the host.
 */
void printURLInfo(char *user, char *password, char *host, char *resource, char *file, char *ip);


/**
 * @brief Prints an error message related to socket connection failure.
 *
 * This function prints an error message when a socket connection to a destination fails.
 *
 * @param destination The destination information.
 * @param ip The IP address of the destination.
 * @param port The port number of the destination.
 */
void printSocketError(const char *destination, const char *ip, int port);
```

```c
/**
 * @brief Prints a general error message and exits the program.
 *
 * This function prints a general error message to the console and exits the program with a failure status.
 *
 * @param message The error message to be printed.
 */
void printError(const char *message);
```

Ficheiros Transferidos

```
joaoalves@DESKTOP-9P6O7OK:/mnt/c/RCOM/Project 2 Code/src$ ./download ftp://ftp.up.pt/pub/kodi/timestamp.txt
 > Host: ftp.up.pt
 > Resource: pub/kodi/timestamp.txt
 > File: timestamp.txt
 > User: anonymous
 > Password: anonymous
 > IP Address: 193.137.29.15

 > Create socket worked fine
 > Create socket worked fine
 > File downloaded!
```



```
dsl@dsl-VivoBook-ASUSLaptop-X530UFO:~/temporario/src$ ./download ftp://rcom:rcom
@netlab1.fe.up.pt/pipe.txt
 > Host: netlab1.fe.up.pt
 > Resource: pipe.txt
 > File: pipe.txt
 > User: rcom
 > Password: rcom
 > IP Address: 192.168.109.136

 > Create socket worked fine
 > Create socket worked fine
 > File downloaded!
```

```
dsl@dsl-VivoBook-ASUSLaptop-X530UFO:~/temporario/src$ cat pipe.txt
..... ...  .          .  .  .
........ . .....    .  . ........   .     .  .
........ . ... .   . ......  .  ...    .
....   . . ..,*//////*..  .. .
 ...  . %#/,*******,**#&&&%                                  (&&&&&&%&%%&%&&&&%*
 ......,#(*,,((%%%%%&&&@@@&                              /&%(,,,/#&@@@@&@@(,..../(.
.......%#(,,*(#%%%%%&&&@@@&*                        &%(,.,#&@@@@@#..
..     .%#(,,*(#%%%%%&&@@@@&(.                  (%/*../&@@@@@(.
   ....%#(,,*(#%%%&%&&@@@@@/               ##/***(#%#&@@&
...  . #%(*,*(#%%%%%&@@@@@@*          ,##//**(#%&&&&%.
.  .   .%#/,*(#%%&%&&@@@@@@,     *%#/**,*(#%&@@&&%
       %#(//(#%&%%&&@@@@@@@&#((/**,*/(#%&@@@&&%
..  .   .%#(((#%%%&&@@@@@@@%#(((#%%%&@@@@@&%
. .      .%%%%%%%&@@@@@@@@@@@@@@@@@@@@@&&.
..  ..     /&&&&&@@@@@@@@@@@@@@@@@@@&&%
..            .&&&&@@@@@@@@@@@@&&%.
..  . .          ..  .
..  ....
......     .           .  .
......      ..%  ..   .. .. .   ..  %   .     /    . ,*  *   %# * %* ##
.. . . ....#*..%.%.(...  %/ ././#%%. ../(.#/,#  %(,, /%.(  *.(,.* %.%#./
.................. ..    .     .    . .                  .        .... .   ..
.....................  ...   ... .                      .           .......
```