

Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de Comunicações por Computador

Ano Letivo de 2021/2022

Relatório TP1 Protocolos da Camada de Transporte

Grupo 1 - PL3
Duarte Serrão a83630
Mário Coelho a42865
Rita Gomes a87960

Índice

1	Introdução	1
2	Parte I: Instalação, configuração e utilização de serviços de transferência de ficheiros	2
3	Parte II: Simulações na redes	4
3.1	PING - Packet Internet Grouper	4
3.2	SFTP - SSH File Transfer Protocol	4
3.3	FTP - File Transfer Protocol	5
3.4	TFTP - Trivial File Transfer Protocol	8
3.5	HTTP - Hypertext Transfer Protocol	9
3.6	QUESTÕES - PARTE I	10
3.6.1	Questão 1	10
3.6.2	Questão 2	11
3.6.3	Questão 3	12
3.6.4	Questão 4	13
4	Parte III: Uso da camada de transporte por parte das aplicações	19
4.1	ping www.google.pt	19
4.2	traceroute cisco.di.uminho.pt	20
4.3	telnet cc2022.ddns.net	20
4.4	ftp cc2022.ddns.net	22
4.5	tftp cc2022.ddns.net	22
4.6	http://marco.uminho.pt/disciplinas/CC-LEI/	23
4.7	nslookup www.uminho.pt	24
4.8	ssh cc2022.ddns.net	25
5	Conclusões	26
	Lista de Siglas e Acrónimos	27

1 Introdução

O presente relatório apresenta uma breve descrição das tarefas realizadas no âmbito do trabalho prático TP1, referente ao estudo dos protocolos da camada de transporte, da unidade curricular de Comunicações por Computador.

Conforme sugestão do próprio enunciado, nas secções que se seguem apresentam-se, por ordem, cada uma das perguntas, seguida da respectiva resolução. Por vezes são utilizados printscreen como complemento às respostas.

De modo a conseguir dar resposta às questões colocadas no presente trabalho prático, o grupo teve de pesquisar alguns conceitos. De seguida apresenta-se um breve resumo desses conceitos para permitir a posterior referencia no decorrer da apresentação dos resultados deste trabalho.

O primeiro grande conceito procurado foi o da arquitectura de camadas da pilha de rede definida pelos modelos OSI e TCP/IP. Assim, foi encontrada a Figura 1.1¹ que resume a comparação entre estes dois modelos, com a vantagem de acrescentar ao meio listas com os protocolos de rede mais comuns em cada uma das camadas.

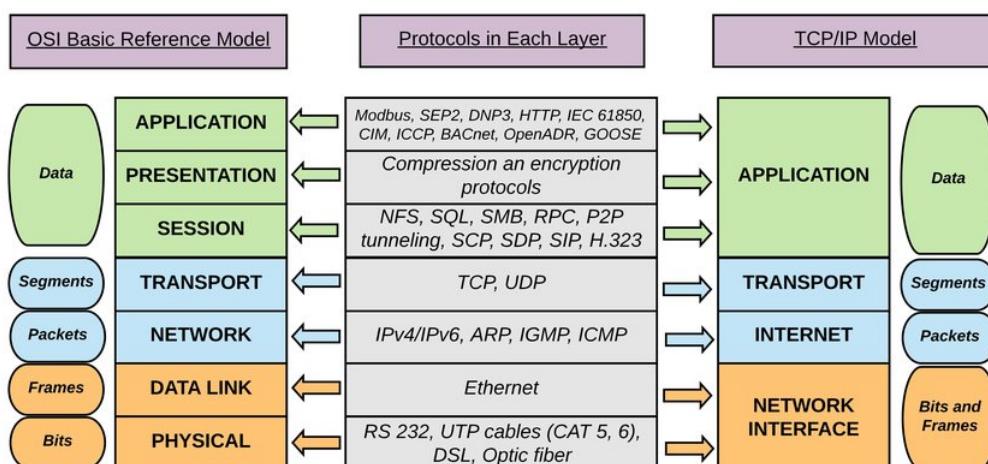


Figura 1.1: Comparação entre os modelos OSI e TCP/IP.

¹Sundararajan, A.; Chavan, A.; Saleem, D.; Sarwat, A.I. A Survey of Protocol-Level Challenges and Solutions for Distributed Energy Resource Cyber-Physical Security. *Energies* 2018, 11, 2360. <https://doi.org/10.3390/en11092360>

2 Parte I: Instalação, configuração e utilização de serviços de transferência de ficheiros

Esta parte do trabalho era apresentada na forma de um guião que, na sua grande maioria, era apenas para replicar e pouco exigia dos alunos. A etapa 1 foi executada sem problemas até porque a maioria das ferramentas requeridas já vinha instalada por defeito na máquina virtual fornecida para a realização do trabalho. A etapa 2 também era bastante direta, apresentando-se na Figura 2.1 o resultado esperado, nomeadamente, uma pasta com os ficheiros a transferir: um ficheiro de texto (file1) e um ficheiro binário (file2).

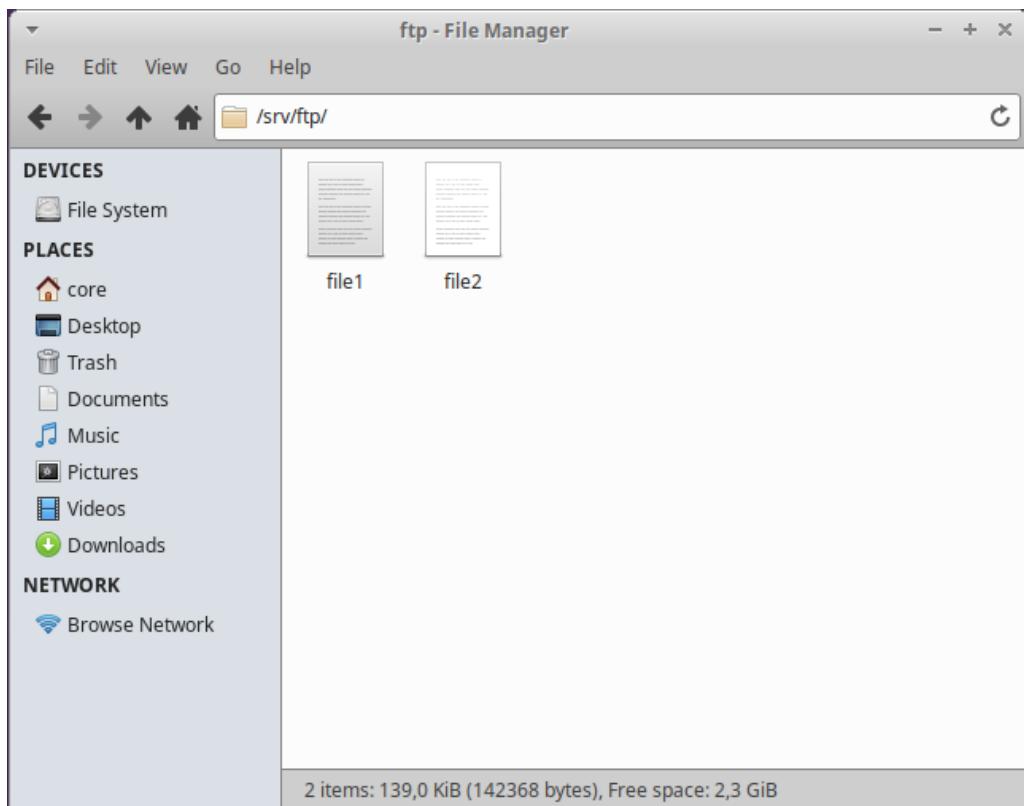


Figura 2.1: Pasta no servidor1 com os ficheiros a transferir.

A etapa 3 consistia em abrir a topologia de rede fornecida e preparar desde já terminais para as máquinas que irão intervir no que se seguia do trabalho (Servidor1, Portatil1, Grilo), conforme ilustrado na Figura 2.2.

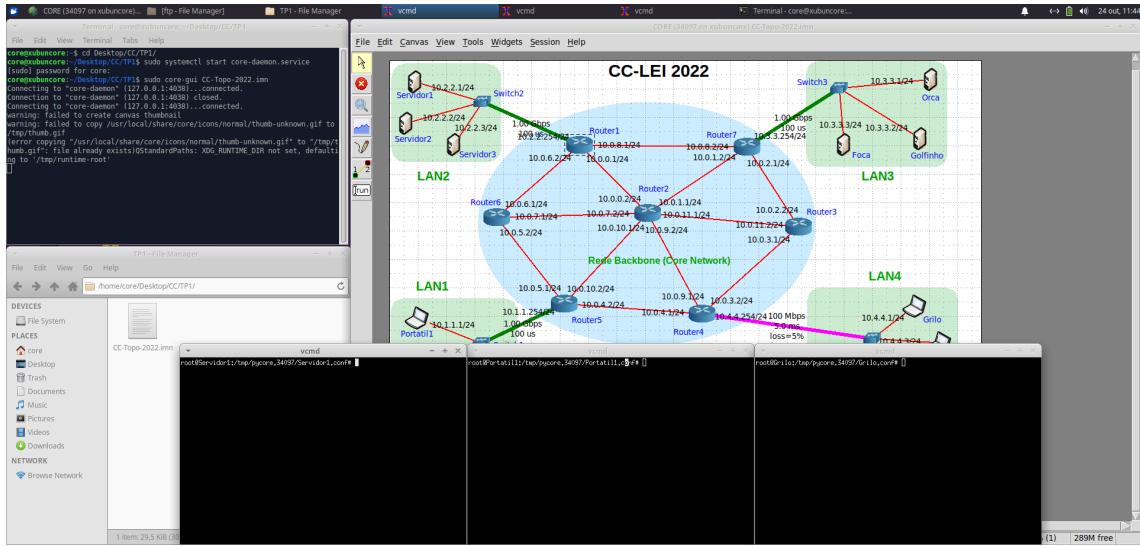


Figura 2.2: Topologia de rede fornecida e terminais necessários para o trabalho.

Adicionalmente, na etapa 3 era também solicitado o lançamento do Wireshark na interface eth2 do Router1 (conforme Figura 2.3). A escolha da interface foi feita de modo a ter um ponto comum no acesso ao servidor1 por parte do Portatil1 e do Grilo. Analisando a topologia de rede, os caminhos entre estas duas máquinas e o servidor1 apenas são o mesmo a partir da ligação entre o router1 e o switch2, pelo que é a interface do router1 que dá para esse caminho que estará monitorizada pelo Wireshark.

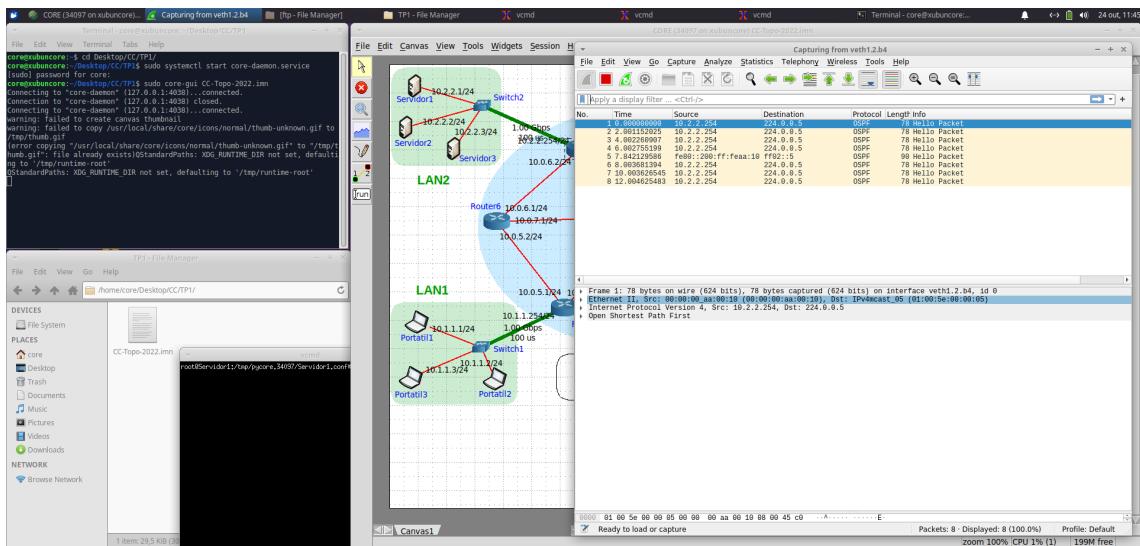


Figura 2.3: Wireshark aberto e a capturar pacotes na interface eth2 do router1.

3 Parte II: Simulações na redes

3.1 PING - Packet Internet Grouper

```
ping -c 20 10.2.2.1 | tee file-ping-output
```

Conforme o próprio manual do comando indica “ping - send ICMP ECHO_REQUEST to network hosts”. Assim, analisando a Figura 3.1, podemos desde logo perceber que estaremos a trabalhar ao nível da camada de rede. O comando solicitado para executar tem ainda a flag “-c 20” que indica que apenas 20 pacotes com pedidos de resposta serão enviados. Adicionalmente, o comando contém uma segunda parte (após o pipe) que, recorrendo ao comando tee, copia o resultado do ping para um ficheiro com o nome file-ping-output. Note-se que, uma vez que o ping será corrido em cada um dos hosts usados no trabalho (Portatil1 e Grilo), este ficheiro ficará guardado em cada uma destas máquinas. Na Figura 3.1 mostra-se o resultado de correr o comando referido em cada uma das máquinas de teste.

The screenshot shows two terminal windows side-by-side. Both are titled 'vcmd' and show the command 'ping -c 20 10.2.2.1 | tee file-ping-output' being run. The left window is for 'Portatil1.conf#', and the right window is for 'Grilo.conf#'. Both outputs show the same sequence of ICMP echo requests and responses, with times ranging from 0.279 ms to 0.685 ms. The right window also shows the final ping statistics at the bottom.

```
<34097/Portatil1.conf# ping -c 20 10.2.2.1 | tee file-ping-output
PING 10.2.2.1 (10.2.2.1) 56(84) bytes of data.
64 bytes from 10.2.2.1: icmp_seq=1 ttl=61 time=0.219 ms
64 bytes from 10.2.2.1: icmp_seq=2 ttl=61 time=0.420 ms
64 bytes from 10.2.2.1: icmp_seq=3 ttl=61 time=0.279 ms
64 bytes from 10.2.2.1: icmp_seq=4 ttl=61 time=0.344 ms
64 bytes from 10.2.2.1: icmp_seq=5 ttl=61 time=0.573 ms
64 bytes from 10.2.2.1: icmp_seq=6 ttl=61 time=0.685 ms
64 bytes from 10.2.2.1: icmp_seq=7 ttl=61 time=0.474 ms
64 bytes from 10.2.2.1: icmp_seq=8 ttl=61 time=0.408 ms
64 bytes from 10.2.2.1: icmp_seq=9 ttl=61 time=0.312 ms
64 bytes from 10.2.2.1: icmp_seq=10 ttl=61 time=0.549 ms
64 bytes from 10.2.2.1: icmp_seq=11 ttl=61 time=0.573 ms
64 bytes from 10.2.2.1: icmp_seq=12 ttl=61 time=0.530 ms
64 bytes from 10.2.2.1: icmp_seq=13 ttl=61 time=0.588 ms
64 bytes from 10.2.2.1: icmp_seq=14 ttl=61 time=0.725 ms
64 bytes from 10.2.2.1: icmp_seq=15 ttl=61 time=0.529 ms
64 bytes from 10.2.2.1: icmp_seq=16 ttl=61 time=0.587 ms
64 bytes from 10.2.2.1: icmp_seq=17 ttl=61 time=0.536 ms
64 bytes from 10.2.2.1: icmp_seq=18 ttl=61 time=0.604 ms
64 bytes from 10.2.2.1: icmp_seq=19 ttl=61 time=0.302 ms
64 bytes from 10.2.2.1: icmp_seq=20 ttl=61 time=0.315 ms
--- 10.2.2.1 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19419ms
rtt min/avg/max/mdev = 0.279/0.570/2.186/0.394 ms
root@Portatil1:/tmp/pycore,34097/Portatil1.conf# |
```

```
<7/Grilo.conf# ping -c 20 10.2.2.1 | tee file-ping-output
PING 10.2.2.1 (10.2.2.1) 56(84) bytes of data.
64 bytes from 10.2.2.1: icmp_seq=1 ttl=61 time=10.4 ms
64 bytes from 10.2.2.1: icmp_seq=2 ttl=61 time=5.69 ms
64 bytes from 10.2.2.1: icmp_seq=3 ttl=61 time=5.26 ms
64 bytes from 10.2.2.1: icmp_seq=4 ttl=61 time=5.97 ms
64 bytes from 10.2.2.1: icmp_seq=5 ttl=61 time=5.47 ms
64 bytes from 10.2.2.1: icmp_seq=6 ttl=61 time=5.22 ms
64 bytes from 10.2.2.1: icmp_seq=7 ttl=61 time=5.22 ms
64 bytes from 10.2.2.1: icmp_seq=8 ttl=61 time=5.34 ms
64 bytes from 10.2.2.1: icmp_seq=9 ttl=61 time=5.35 ms (DUP!)
64 bytes from 10.2.2.1: icmp_seq=10 ttl=61 time=5.58 ms
64 bytes from 10.2.2.1: icmp_seq=11 ttl=61 time=5.28 ms
64 bytes from 10.2.2.1: icmp_seq=12 ttl=61 time=5.42 ms
64 bytes from 10.2.2.1: icmp_seq=13 ttl=61 time=5.68 ms
64 bytes from 10.2.2.1: icmp_seq=14 ttl=61 time=5.21 ms
64 bytes from 10.2.2.1: icmp_seq=15 ttl=61 time=5.23 ms
64 bytes from 10.2.2.1: icmp_seq=16 ttl=61 time=5.21 ms
64 bytes from 10.2.2.1: icmp_seq=17 ttl=61 time=5.23 ms
64 bytes from 10.2.2.1: icmp_seq=18 ttl=61 time=5.84 ms
64 bytes from 10.2.2.1: icmp_seq=19 ttl=61 time=5.30 ms
--- 10.2.2.1 ping statistics ---
20 packets transmitted, 18 received, +1 duplicates, 10% packet loss, time 19052ms
rtt min/avg/max/mdev = 5.213/5.677/10.378/1.130 ms
root@Grilo:/tmp/pycore,34097/Grilo.conf# |
```

Figura 3.1: Terminais nos hosts Portatil1 e Grilo após correr o comando ping.

3.2 SFTP - SSH File Transfer Protocol

```
ps -ef | grep ssh
netstat -n -a
```

Antes de analisar o protocolo SSH foi solicitada a utilização de dois comandos por forma a verificar que o servidor ssh se encontrava em execução. O primeiro comando permitia listar todos (-ef) os processos em execução (ps), fazendo um filtro (grep) para apenas mostrar os processos ssh. Conforme Figura 3.2 é possível verificar no processo 32 que o servidor ssh se encontra à escuta. O segundo comando permite obter algumas estatísticas da rede (netstat) usando endereços numéricas (-n) para todas (-a) as ligações em curso. Conforme Figura 3.2 é possível verificar que o servidor ssh se encontra à escuta na porta 22. De facto, é esta a porta por defeito para este tipo de protocolo.

```

root@Servidor1:/tmp/pycore.34097/Servidor1.conf# ps -ef | grep ssh
root      32  1  0 11:42 ?    00:00:00 sshd: /usr/sbin/sshd -f /etc
/ssh/sshd_config [listener] 0 of 10-100 startups
root      41  33  0 12:13 pts/2  00:00:00 grep --color=auto ssh
root@Servidor1:/tmp/pycore.34097/Servidor1.conf# netstat -n -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address          Foreign Address      State
tcp        0      0 0.0.0.0:22              0.0.0.0:*            LISTEN
tcp6       0      0 ::1:22                ::*                  LISTEN
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type      State      I-Node Path
root@Servidor1:/tmp/pycore.34097/Servidor1.conf# 

```

Figura 3.2: Preparação do ambiente de trabalho para utilização do protocolo SFTP.

Depois de verificadas as condições para avançar com o exercício, pretendia-se agora transferir do servidor1, para o Portatil1 e o Grilo, o ficheiro de texto file1. Na Figura 3.3 apresentam-se os resultados obtidos no terminal. Desde logo é possível verificar que a transferência foi bem-sucedida em ambos os casos (100% transferido). No entanto, a velocidade média de cada transferência foi diferente. Tal prende-se com a capacidade das ligações entre os hosts e o servidor1, que difere dentro da LAN a que cada um pertence.

3.3 FTP - File Transfer Protocol

chmod a-w /srv/ftp

Antes de iniciar a experiência convém ter a certeza que, ao dar acesso à pasta do servidor,

Figura 3.3: Terminais nos hosts Portatil1 e Grilo após utilização do protocolo SFTP.

não será possível escrever lá nada, apenas ler. De facto, após a etapa 2 a directória /srv/ftp do servidor ficou com privilégios de leitura e escrita (ver Figura 3.4).

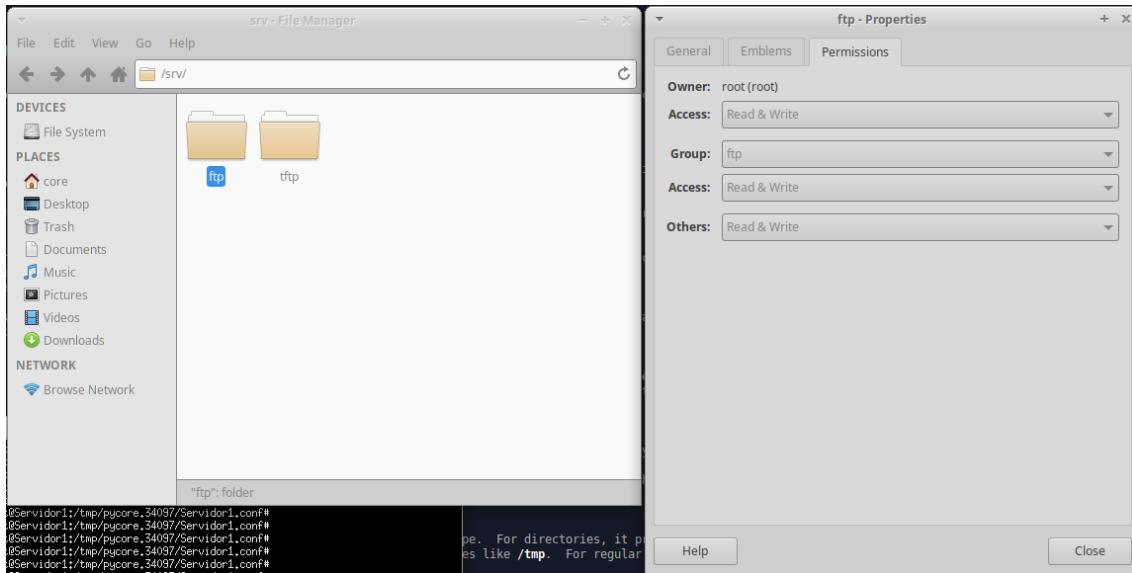


Figura 3.4: Permissões no servidor após etapa 2.

Aplicando o comando anterior, que altera o modo de acesso (chmod) àquela directória, para todos os utilizadores (a), retirando-lhes as permissões de escrita (-w), conforme se verifica na Figura 3.5.

```
vsftpd /etc/vsftpd.conf -osecure_chroot_dir=/srv/ftp \\  
-oanonymous_enable=YES}
```

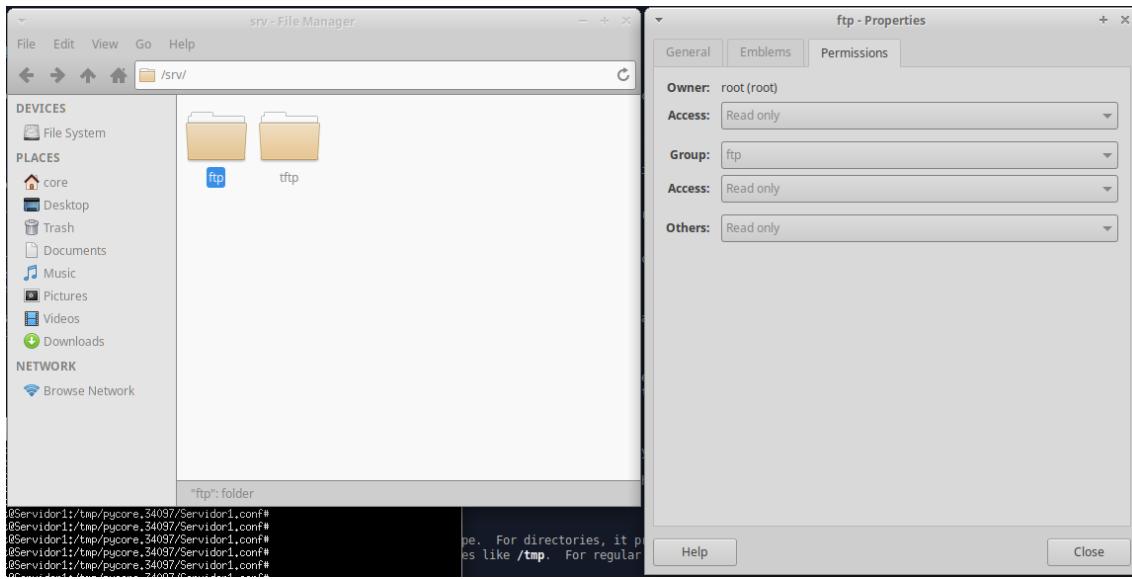


Figura 3.5: Permissões no servidor após alteração.

O comando acima permite inicializar o servidor de FTP sugerido neste trabalho, o vsftpd (Very Secure FTP Daemon). O primeiro parâmetro deste comando é o ficheiro de configuração do servidor. Adicionalmente são passados mais dois argumentos com opções. Por último, aplica-se a sequência de comandos sugerida para dar inicio à transferência do ficheiro de texto file1 via FTP. A Figura 3.6 ilustra o resultado obtido. É possível ver na bash da esquerda que o servidor FTP se encontra em funcionamento no servidor. Nas outras duas bash é possível ver o processo da transferência. Neste, a única grande diferença entre as duas transferências do mesmo ficheiro é que, tal como identificado anteriormente, a transferência para o portátil1 é mais rápida do que a transferência para o grilo. Tal deve-se ao tipo de infraestrutura de rede nomeadamente em termos da capacidade da ligação de cada uma das LAN com a rede de backbone.

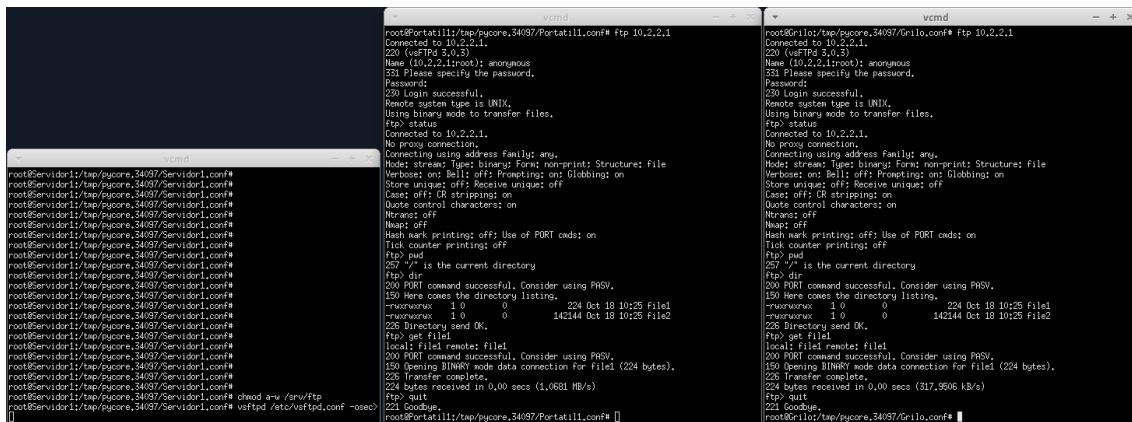


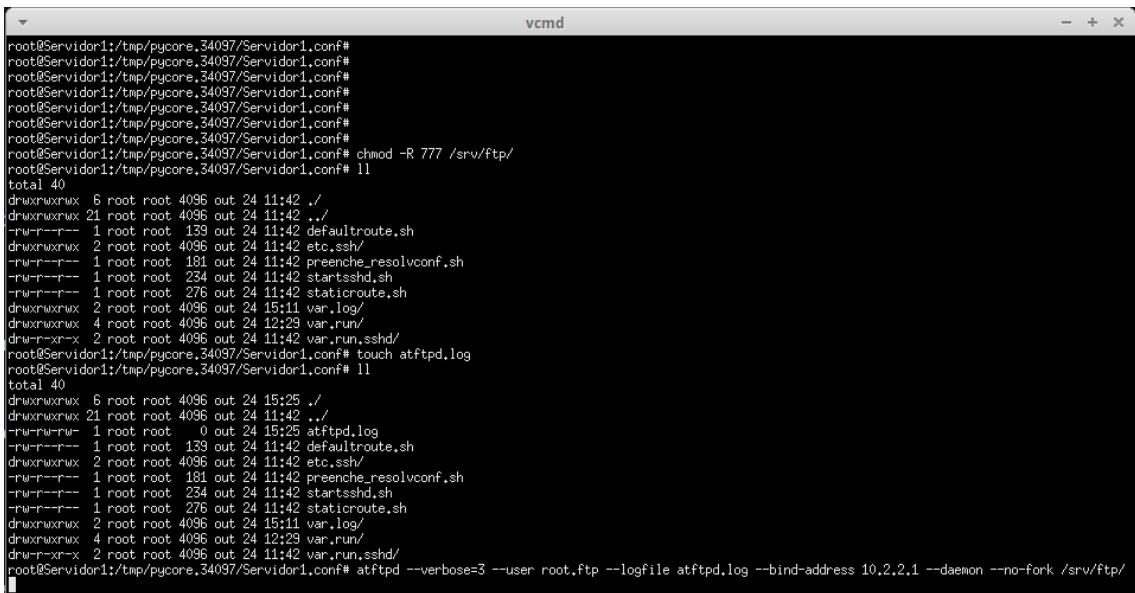
Figura 3.6: Terminais nos hosts Portatil1 e Grilo, e no servidor1, após utilização do protocolo FTP.

3.4 TFTP - Trivial File Transfer Protocol

Antes sequer de inicializar o teste com este protocolo, é possível desde logo verificar que, enquanto o protocolo anterior permitia FTP com elevada segurança, neste caso o processo deverá ser muito menos seguro (daí o termo trivial). Para poder realizar a tarefa pedida, foram realizados os seguintes comandos de preparação:

```
chmod -R 777 /srv/ftp
touch atftpd.log
atftpd -verbose=3 --user root.ftp --logfile atftpd.log --bind-address 10.2.2.1 --daemon
--no-fork /srv/ftp/
```

O primeiro serve para fornecer recursivamente (-R) permissões totais (777 – rwx) para todos os utilizadores (por omissão de flag especificando os utilizadores) na directória /srv/ftp. O segundo serve para criar o ficheiro atftpd.log onde ficarão guardados os logs da actividade do servidor. O terceiro, permite inicializar o servidor TFTP. Na Figura 3.7 é possível constatar que o ficheiro criado não existia antes, e que o servidor ficou em modo escuta para o que se segue.



The screenshot shows a terminal window titled "vcmd". The command history and output are as follows:

```
root@Servidor1:/tmp/pycore_34097/Servidor1.conf# 
root@Servidor1:/tmp/pycore_34097/Servidor1.conf# 
root@Servidor1:/tmp/pycore_34097/Servidor1.conf# 
root@Servidor1:/tmp/pycore_34097/Servidor1.conf# 
root@Servidor1:/tmp/pycore_34097/Servidor1.conf# 
root@Servidor1:/tmp/pycore_34097/Servidor1.conf# 
root@Servidor1:/tmp/pycore_34097/Servidor1.conf# chmod -R 777 /srv/ftp/
root@Servidor1:/tmp/pycore_34097/Servidor1.conf# ll
total 40
drwxrwxrwx 6 root root 4096 out 24 11:42 .
drwxrwxrwx 21 root root 4096 out 24 11:42 ..
-rw-r--r-- 1 root root 139 out 24 11:42 defaultroute.sh
drwxrwxrwx 2 root root 4096 out 24 11:42 etc.ssh/
-rw-r--r-- 1 root root 181 out 24 11:42 preenche_resolvconf.sh
-rw-r--r-- 1 root root 234 out 24 11:42 startssh.sh
-rw-r--r-- 1 root root 276 out 24 11:42 staticroute.sh
drwxrwxrwx 2 root root 4096 out 24 15:11 var.log/
drwxrwxrwx 4 root root 4096 out 24 12:29 var.run/
drw-r-xr-x 2 root root 4096 out 24 11:42 var.run.sshd/
root@Servidor1:/tmp/pycore_34097/Servidor1.conf# touch atftpd.log
root@Servidor1:/tmp/pycore_34097/Servidor1.conf# ll
total 40
drwxrwxrwx 6 root root 4096 out 24 15:25 .
drwxrwxrwx 21 root root 4096 out 24 11:42 ..
-rw-r--r-- 1 root root 0 out 24 15:25 atftpd.log
-rw-r--r-- 1 root root 139 out 24 11:42 defaultroute.sh
drwxrwxrwx 2 root root 4096 out 24 11:42 etc.ssh/
-rw-r--r-- 1 root root 181 out 24 11:42 preenche_resolvconf.sh
-rw-r--r-- 1 root root 234 out 24 11:42 startssh.sh
-rw-r--r-- 1 root root 276 out 24 11:42 staticroute.sh
drwxrwxrwx 2 root root 4096 out 24 15:11 var.log/
drwxrwxrwx 4 root root 4096 out 24 12:29 var.run/
drw-r-xr-x 2 root root 4096 out 24 11:42 var.run.sshd/
root@Servidor1:/tmp/pycore_34097/Servidor1.conf# atftpd --verbose=3 --user root.ftp --logfile atftpd.log --bind-address 10.2.2.1 --daemon --no-fork /srv/ftp/
```

Figura 3.7: Preparação do ambiente de trabalho para utilização do protocolo TFTP.

Por último, aplicam-se os comandos necessários para realizar a transferência do ficheiro de texto file1 do servidor para os mesmos dois hosts que se usaram nas experiências anteriores. De forma semelhante, a única diferença entre as duas transferências esteve associada à velocidade de transferência, conforme ilustrado na Figura 3.8.

Figura 3.8: Terminais nos hosts Portatil1 e Grilo após utilização do protocolo TFTP.

3.5 HTTP - Hypertext Transfer Protocol

Neste caso a preparação consiste em tão simplesmente indicar ao servidor HTTP a direcção onde estará a trabalhar (que é a que contém os ficheiros que queremos transferir). De seguida são aplicados os comandos de transferência de ficheiros. Neste caso foram transferidos os dois ficheiros nos dois hosts. Conforme se pode ver na figura abaixo, e em linha com o comportamento verificado com os protocolos anteriores, apenas as velocidades de transferência foram diferentes.

Figura 3.9: Terminais nos hosts Portatil1 e Grilo, e servidor1, após utilização do protocolo HTTP.

Nesta altura foi desligado o modo de captura do wireshark e procedeu-se à analise dos pacotes transferidos ao longo das experiências realizadas.

3.6 QUESTÕES - PARTE I

3.6.1 Questão 1

De que forma as perdas e duplicações de pacotes afectaram o desempenho das aplicações? Que camada lidou com as perdas e duplicações: transporte ou aplicação? Responda com base nas experiências feitas e nos resultados observados.

O simples facto de haver perdas ou duplicações pode não configurar um problema de desempenho por si só. De facto, podem existir contextos em que não é relevante garantir que os pacotes sejam todos entregues, mas sim garantir que se envia o máximo de pacotes possíveis no mínimo tempo possível, ou outros critérios que façam sentido nesses contextos.

No presente trabalho, nas várias experiências realizadas, verificou-se a existência de pacotes duplicados e perdidos no caso da aplicação do ping (ver Figura 3.1). No entanto, sendo este um protocolo de nível de rede, a camada que lida com esses problemas é a camada de rede.

Nas demais experiências, não ocorreram situações de perda ou duplicação de pacotes. Ainda assim, através de uma pesquisa bibliográfica foi possível identificar que os protocolos de aplicação que usam como protocolo de transporte o TCP, terão disponíveis métodos para controlo de erros e reenvio de pacotes perdidos. Contrariamente, os que utilizam o protocolo de transporte UDP não possuem tais mecanismos, sendo simplesmente descartados os pacotes com erros (i.e. são detectados os erros mas não é feito nada para os corrigir).

Uso de ACKS no protocolo TCP

Para os protocolos de aplicação que estão sobre o **protocolo de transmissão TCP**, é possível verificar que este utiliza um tipo de pacotes especial chamado "ACK"(do Inglês, *acknowledgement*). Este pacote serve para dizer à fonte que o destinatário recebeu a mensagem.

Com este sistema, a quantidade de perdas é muito menor, pois funciona à base de timeouts. Por este mesmo motivo, acontecem às vezes duplicações quando o tempo de retorno do ACK é mais demorado que o timeout predefinido.

Por consequência, verifica-se que quem lida com perdas e duplicações é o protocolo de transmissão e não o de aplicação.

Perdas no protocolo UDP

Como o protocolo de transmissão UDP não funciona à base de ACKS, ou algo parecido, quando este encontra um erro no pacote, simplesmente descarta-o, o que resulta na perda total desse pacote.

Como é o UDP que faz o controlo de erros e descarta os pacotes que os têm, verifica-se que quem lida com as perdas é mais uma vez o protocolo de transmissão e não o de aplicação.

Desempenho na Aplicação

Caso na aplicação do receptor se notar que existiu perda de pacotes, esta irá ter de passar pelo processo todo outra vez e pedir de novo o ficheiro pretendido. Este processo claramente irá atrasar a funcionalidade da aplicação em questão.

Caso tenham havido duplicações, o desempenho será inferior (embora menos que no caso de perdas), pois a aplicação terá os pacotes identificados pelo número de sequência, descartando os que tiver duplicados.

Adicionalmente, no caso do protocolo SFTP existem ainda pacotes adicionais relacionados com a encriptação SSH fornecida pela camada de aplicação. Tal é visível na Figura 3.14. O facto de existirem estas verificações adicionais vai naturalmente introduzir mais uma camada de complexidade à transferência de pacotes, logo de perda de desempenho.

3.6.2 Questão 2

Obtenha a partir do wireshark, ou desenhe manualmente, um diagrama temporal para a transferência de file1 por FTP. Foque-se apenas na transferência de dados [ftp-data] e não na conexão de controlo, pois o FTP usa mais que uma conexão em simultâneo. Identifique, se aplicável, as fases de início de conexão, transferência de dados e fim de conexão. Identifique também os tipos de segmentos trocados e os números de sequência usados quer nos dados como nas confirmações.

Na Figura 3.10 apresenta-se o diagrama temporal para a transferência de file1 por FTP. Nos parágrafos que se seguem são tecidos alguns comentários detalhando os aspectos mais importantes ilustrados no diagrama.

3247 12383.329484.. 10.2.2.1	10.1.1.1	TCP	74.20 - 45591 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=449036710 TSerr=0 WS=128
3248 12383.329996.. 10.1.1.1	10.2.2.1	TCP	74.45591 - 20 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=3909819391 TSerr=449036710 WS=128
3249 12383.339165.. 10.2.2.1	10.1.1.1	TCP	66.20 - 45591 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=449036711 TSerr=3909819391
3250 12383.339257.. 10.2.2.1	10.1.1.1	FTP	150< response: 150 Opening BINARY mode data connection for file1 (224 bytes).
3251 12383.339414.. 10.2.2.1	10.1.1.1	FTP-DATA	280< FTP DATA: 280 File file1 ready for transfer.
3252 12383.339414.. 10.2.2.1	10.1.1.1	TCP	66.20 - 45591 [FIN, ACK] Seq=225 Ack=3 Win=64256 Len=0 TSval=449036711 TSerr=3909819391
3253 12383.339491.. 10.1.1.1	10.2.2.1	TCP	66.49310 - 21 [ACK] Seq=113 Ack=391 Win=64256 Len=0 TSval=3909819391 TSerr=449036711
3254 12383.339749.. 10.1.1.1	10.2.2.1	TCP	66.45591 - 20 [ACK] Seq=1 Ack=225 Win=65024 Len=0 TSval=3909819392 TSerr=449036711
3255 12383.339932.. 10.1.1.1	10.2.2.1	TCP	66.45591 - 20 [FIN, ACK] Seq=1 Ack=226 Win=65024 Len=0 TSval=3909819392 TSerr=449036711
3256 12383.339880.. 10.2.2.1	10.1.1.1	TCP	66.20 - 45591 [ACK] Seq=226 Ack=2 Win=64256 Len=0 TSval=449036712 TSerr=3909819392

Figura 3.10: Captura de pacotes obtida no Wireshark para a conexão FTP entre o servidor (10.2.2.1) e o portatil1 (10.1.1.1).

1. Conexão

- (i) Através do TCP, o portatil1 (10.1.1.1) tenta estabelecer uma ligação com o servidor1 (10.2.2.1), enviando um pacote SYN;
- (ii) Através do TCP, o servidor1 (10.2.2.1) tenta estabelecer uma ligação com o portatil1 (10.1.1.1), enviando um pacote SYN e ao mesmo tempo envia um pacote ACK a avisar que recebeu o pacote anterior;
- (iii) Através do TCP, o portatil1 (10.1.1.1) envia um ACK ao servidor1, estabelecendo assim uma ligação entre os dois.

2. Transferência do Ficheiro

- (i) Através do FTP, o servidor1 (10.2.2.1) envia o file1 num só pacote;
- (ii) Através do TCP, o portatil1 (10.1.1.1) envia um ACK a confirmar a recepção.

3. Fim de Conexão

- (i) Através do TCP, o servidor1 (10.2.2.1) tenta acabar a ligação com o portatil1 (10.1.1.1), enviando um pacote FIN e um ACK;
- (ii) Através do TCP, o portatil1 (10.1.1.1) enviará de volta um ACK;
- (iii) Através do TCP, o portatil1 (10.1.1.1) tenta acabar de volta a ligação com o servidor1 (10.2.2.1), enviando um pacote SYN e ao mesmo tempo envia um pacote ACK;
- (iv) Através do TCP, o servidor1 (10.2.2.1) envia um ACK ao portátil1, acabando assim a ligação entre os dois.

3.6.3 Questão 3

Obtenha a partir do wireshark, ou desenhe manualmente, um diagrama temporal para a transferência de file1 por TFTP. Identifique, se aplicável, as fases de início de conexão, transferência de dados e fim de conexão. Identifique também os tipos de segmentos trocados e os números de sequência usados quer nos dados como nas confirmações.

Na Figura 3.11 apresenta-se o diagrama temporal para a transferência de file1 por TFTP. Conforme ilustrado no diagrama, a conexão começa com o envio de um Read Request do portatil1 (10.1.1.1) para o servidor1 (10.2.2.1), o qual contém o nome do ficheiro a transferir (file1).

Posteriormente, o servidor1 responde com um pacote de dados (Data Packet), isto é, ocorre a fase de transferência de dados. Após receber os dados, inicia-se a fase de fim de conexão e o servidor envia um pacote ACK a informar o sucesso da recepção desses dados.

ip.addr == 10.1.1.1 and ip.addr == 10.2.2.1					
No.	Time	Source	Destination	Protocol	Length Info
4292	13:07:07,146777	10.1.1.1	10.2.2.1	TFTP	56 Read Request, File: file1, Transfer type: octet
4293	13:07:07,147877	10.1.1.1	10.2.2.1	TFTP	278 Data Packet, Block: 1 (last)
4294	13:07:07,148522	10.1.1.1	10.2.2.1	TFTP	46 Acknowledgement, Block: 1

Figura 3.11: Captura de pacotes obtida no Wireshark para a conexão TFTP entre o servidor (10.2.2.1) e o portatil1 (10.1.1.1).

3.6.4 Questão 4

Compare sucintamente as quatro aplicações de transferência de ficheiros que usou nos seguintes pontos (i) uso da camada de transporte; (ii) eficiência; (iii) complexidade; (iv) segurança.

De modo a complementar as respostas que se seguem, apresenta-se nas seguintes figuras prints representativos da utilização dos protocolos HTTP na transferência dos ficheiros file1 (Figura 3.12) e file2 (Figura 3.13), bem como do SSH (Figura 3.14). Note-se que em todos estes casos a comunicação ocorre entre o servidor1 (10.2.2.1) e o portatil1 (10.1.1.1).

ip.addr == 10.1.1.1 and ip.addr == 10.2.2.1					
No.	Time	Source	Destination	Protocol	Length Info
4598	14:45:16.2416	10.1.1.1	10.2.2.1	TCP	74 38262 - 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 Tsva=3911893223 Tscr=0 WS=128
4599	14:45:16.2517	10.1.1.1	10.2.2.1	TCP	74 88 - 38262 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 Tsva=451110544 Tscr=3911893223 WS=128
4600	14:45:16.3212	10.1.1.1	10.2.2.1	TCP	66 38262 - 38262 [ACK] Seq=1 Ack=1 Win=64256 Len=0 Tsva=3911893224 Tscr=451110544
4681	14:45:16.3517	10.1.1.1	10.2.2.1	HTTP	200 GET /file1.htm
4682	14:45:16.3608	10.2.2.1	10.1.1.1	TCP	66 88 - 38262 [ACK] Seq=1 Ack=141 Win=65924 Len=0 Tsva=451110545 Tscr=3911893224
4683	14:45:16.3886	10.2.2.1	10.1.1.1	HTTP	598 HTTP/1.1 200 Ok (text/plain)
4684	14:45:16.8878	10.1.1.1	10.2.2.1	TCP	66 88 - 38262 [FIN, ACK] Seq=141 Ack=444 Win=64128 Len=0 Tsva=3911893228 Tscr=451110545
4685	14:45:16.8833	10.2.2.1	10.1.1.1	TCP	66 88 - 38262 [ACK] Seq=444 Ack=142 Win=65924 Len=0 Tsva=451110549 Tscr=3911893228

Figura 3.12: Captura de pacotes obtida no Wireshark para a conexão HTTP entre o servidor (10.2.2.1) e o portatil1 (10.1.1.1) na transferência do file1.

Uso da Camada de Transporte

- **FTP:** Utiliza o protocolo TCP
- **SFTP:** Utiliza o protocolo TCP
- **TFTP:** Utiliza o protocolo UDP
- **HTTP:** Utiliza o protocolo TCP

Eficiência

Um protocolo é eficiente pelo seu método de controlo de erros, pela quantidade de dados que precisa de mandar por pacote e o quanto rápido é o seu intervalo de transmissão entre os pacotes.

- **FTP**
 - Utiliza ACKs, logo consegue perceber se um pacote foi ou não entregue, retransmitindo-o imediatamente. Logo, é bastante eficiente em caso de falha;
 - Caso todos os pacotes sejam bem entregues, será um pouco mais lento do que protocolos que utilizem UDP (Assumindo que estes foram também entregues com sucesso).
- **SFTP**

- É o que tem o overhead maior devido à introdução de encriptação fornecida pelo protocolo de aplicação SSH, logo, terá de enviar mais dados que os outros;
- Como também utiliza TCP, terá uma eficiência menor com o FTP, pois também estará a utilizar o sistema dos ACKs para gestão de erros.

- **TFTP**

- É o que tem um overhead de transporte mais pequeno, logo, terá de mandar menos dados em relação aos outros três;
- Em caso de sucesso, é o mais rápido deles todos, já que não espera que o destinatário lhe confirme que recebeu os pacotes através de ACKs, tornando-o eficiente;
- Em caso de insucesso torna-se extremamente ineficiente, pois terá de retransmitir os pacotes.

- **HTTP**

- Como também utiliza TCP, terá uma eficiencia parecida com o FTP, pois também estará a utilizar o sistema dos ACKs para gestão de erros.

Complexidade

Um protocolo é complexo pelo conteúdo no seu overhead e as suas funcionalidades (controlo de erros, pedidos de transferência de dados em paralelo, entre outros).

- **FTP**

- Tem um overhead de 20 bytes;
- É capaz de suportar diferentes pedidos de transferência de dados em paralelo;
- Tem uma boa capacidade de controlo de erros, devido a usar TCP;
- Pode-se considerar um protocolo complexo.

- **SFTP**

- Tem um overhead de 40 bytes (20 do FTP + 20 do SSH);
- É capaz de suportar diferentes pedidos de transferência de dados em paralelo;
- Tem uma boa capacidade de controlo de erros devido a usar TCP;
- Tem a funcionalidade de encriptação de dados fornecida pelo SSH;
- Pode-se considerar um protocolo bastante complexo.

- **TFTP**

- Tem um overhead de 8 bytes;
- É uma alternativa simplificada do FTP, o que leva a ter menos funcionalidades;
- Não tem uma boa capacidade de controlo de erros, devido a usar UDP;
- Pode-se considerar um protocolo não muito complexo.

- **HTTP**

- Tem um overhead de 20 bytes;
- Oferece um suporte para hypermedia;
- Tem escalabilidade e desacoplamento de sistemas;
- Tem uma boa capacidade de controlo de erros, devido a usar TCP;

- Pode-se considerar um protocolo bastante complexo.

Segurança

Um protocolo tem mais segurança caso ofereça as funcionalidades de encriptação de mensagens, autenticação, entre outros.

- **FTP**

- Não oferece encriptação dos dados, o que o torna menos seguro;
- É possível fazer intercepção dos dados e ver claramente que informações estão contidas neles, nomeadamente nomes de utilizadores, emails, palavras passes, etc;
- Tem autenticação;
- Não é um protocolo muito seguro.

- **SFTP**

- Faz uso do SSH, o que o torna bastante seguro;
- SSH faz uso de uma camada de transporte, uma de autenticação e uma de conexão;
- Fornece encriptação, autenticação do servidor e protecção na íntegra dos dados.

- **TFTP**

- Como é uma versão ainda mais simples que o FTP, este não só não tem encriptação (com as consequências previamente ditas), como também não tem autenticação;
- Conclusão, claramente não é um protocolo seguro.

- **HTTP**

- Não oferece encriptação dos dados, o que o torna menos seguro;
- É possível fazer intercepção dos dados e ver claramente que informações estão contidas neles, nomeadamente nomes de utilizadores, emails, palavras passes, etc;
- Conclusão, não é um protocolo seguro.

Figura 3.13: Captura de pacotes obtida no Wireshark para a conexão HTTP entre o servidor (10.2.2.1) e o portátil1 (10.1.1.1) na transferência do file2.

No.	Time	Source	Destination	Protocol	Length Info
1688	2629.8329392 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	74 42420 - 22 [SYN] Seq=0 Win=64300 Len=0 TStamp=19710608000000000 TSval=3997700033 TSecr=3997700033 WS=128
1689	2629.8323261 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	74 42420 - 22 [SYN] Seq=0 Win=64300 Len=0 MSS=1469 SACK_PERM=1 TSval=3997700033 TSecr=3997700033 WS=128
1690	2629.8330786 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	66 42440 - 22 [ACK] Seq=1 Ack=1 Win=64300 Len=0 TSval=3997700033 TSecr=446917355
1691	2629.8330786 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	107 Client: Protocol [SSH-2.0-OpenSSH_8.2pi Ubuntu-4ubuntu10.3]
1692	2629.8321988 . 10.2.2.1	10.1.1.1	10.1.1.1	TCP	66 22 - 42440 [ACK] Seq=1 Ack=42 Win=65152 Len=0 TSval=446917354 TSecr=3997700034
1693	2629.8321454 . 10.2.2.1	10.1.1.1	10.1.1.1	SSHv2	107 Server: Protocol [SSH-2.0-OpenSSH_8.2pi Ubuntu-4ubuntu10.3]
1694	2629.8326522 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 22 - 42440 [ACK] Seq=1 Ack=42 Win=64256 Len=0 TSval=3997700040 TSecr=446917360
1695	2629.8328837 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	1514 42440 - 22 [ACK] Seq=42 Ack=42 Win=64256 Len=1448 TSval=3997700040 TSecr=446917360 [TCP segment of a reassembled PDU]
1696	2629.8328835 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	138 Client: Key Exchange Init
1697	2629.8339710 . 10.2.2.1	10.1.1.1	10.1.1.1	TCP	108 Server: Key Exchange Init
1698	2629.8339710 . 10.2.2.1	10.1.1.1	10.1.1.1	SSHv2	138 Client: Key Exchange Init
1699	2629.8339738 . 10.2.2.1	10.1.1.1	10.1.1.1	TCP	66 22 - 42440 [ACK] Seq=1066 Ack=1066 Win=64128 Len=0 TSval=446917360 TSecr=3997700040
1700	2629.8401256 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 22 - 42440 [ACK] Seq=1 Ack=42 Win=65152 Len=0 TSval=446917354 TSecr=3997700034
1701	2629.8417423 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	114 Client: Diffie-Hellman Key Exchange Init
1702	2629.8417423 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 22 - 42440 [ACK] Seq=1 Ack=42 Win=64128 Len=0 TSval=446917360 TSecr=3997700040
1703	2629.8401047 . 10.2.2.1	10.1.1.1	10.1.1.1	SSHv2	1182 Server: Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=228)
1704	2629.8401047 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 22 - 42440 [ACK] Seq=1062 Ack=2182 Win=64128 Len=0 TSval=3997700049 TSecr=446917369
1705	2628.5250113 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	82 Client: New Keys
1706	2628.5250113 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 22 - 42440 [ACK] Seq=2100 Ack=1019 Win=64128 Len=0 TSval=44692564 TSecr=3997707725
1707	2628.52523164 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	119 Client: Encrypted packet (len=44)
1715	2628.5254278 . 10.2.2.1	10.1.1.1	10.1.1.1	TCP	66 22 - 42440 [ACK] Seq=1066 Ack=1066 Win=64128 Len=0 TSval=44692564 TSecr=3997707726
1716	2628.5254542 . 10.2.2.1	10.1.1.1	10.1.1.1	SSHv2	119 Server: Encrypted packet (len=44)
1717	2628.5256091 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 22 - 42440 [ACK] Seq=1066 Ack=1066 Win=64128 Len=0 TSval=44692564 TSecr=3997707726
1718	2628.5256091 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	128 Client: Encrypted packet (len=60)
1719	2628.5262773 . 10.2.2.1	10.1.1.1	10.1.1.1	TCP	66 22 - 42440 [ACK] Seq=2286 Ack=1722 Win=64128 Len=0 TSval=44692564 TSecr=3997707726
1720	2628.5340276 . 10.2.2.1	10.1.1.1	10.1.1.1	SSHv2	118 Server: Encrypted packet (len=52)
1721	2628.5340276 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 22 - 42440 [ACK] Seq=2272 Ack=1722 Win=64128 Len=0 TSval=44692565 TSecr=3997707726
1722	2628.5340276 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	159 Client: Encrypted packet (len=60)
1723	2628.5339595 . 10.2.2.1	10.1.1.1	10.1.1.1	TCP	66 22 - 42440 [ACK] Seq=2279 Ack=1806 Win=64128 Len=0 TSval=446927553 TSecr=3997710232
1727	2631.0413787 . 10.2.2.1	10.1.1.1	10.1.1.1	SSHv2	94 Server: Encrypted packet (len=28)
1728	2631.0423785 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 22 - 42440 [ACK] Seq=1380 Ack=1380 Win=64128 Len=0 TSval=3997710242 TSecr=446927562
1729	2631.0423785 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	119 Client: Encrypted packet (len=44)
1730	2631.0423785 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 22 - 42440 [ACK] Seq=2308 Ack=1912 Win=64128 Len=0 TSval=446927563 TSecr=3997710242
1731	2631.051932 . 10.2.2.1	10.1.1.1	10.1.1.1	SSHv2	534 Server: Encrypted packet (len=48)
1732	2631.051932 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 22 - 42440 [ACK] Seq=1912 Ack=2774 Win=64128 Len=0 TSval=3997710697 TSecr=446927973
1733	2631.051932 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	110 Client: Encrypted packet (len=52)
1734	2631.0519142 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 22 - 42440 [ACK] Seq=1912 Ack=2818 Win=64128 Len=0 TSval=3997710699 TSecr=446928018
1735	2631.0519146 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	182 Client: Encrypted packet (len=756)
1736	2631.0519299 . 10.2.2.1	10.1.1.1	10.1.1.1	TCP	66 22 - 42440 [ACK] Seq=2674 Ack=2674 Win=64128 Len=0 TSval=446928822 TSecr=3997710699
1737	2631.0519299 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	183 Client: Encrypted packet (len=60)
1738	2631.5519127 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 22 - 42440 [ACK] Seq=2674 Ack=2899 Win=64128 Len=0 TSval=3997710701 TSecr=446928820
1739	2631.5510277 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	119 Client: Encrypted packet (len=44)
1740	2631.5510277 . 10.2.2.1	10.1.1.1	10.1.1.1	TCP	66 22 - 42440 [ACK] Seq=2396 Ack=2718 Win=64128 Len=0 TSval=446928822 TSecr=3997710701
1741	2631.5510277 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	66 Server: Encrypted packet (len=52)
1742	2631.5510277 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 22 - 42440 [ACK] Seq=2718 Ack=2718 Win=64128 Len=0 TSval=3997710706 TSecr=446928826
1743	2631.5510911 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	118 Client: Encrypted packet (len=52)
1744	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 22 - 42440 [ACK] Seq=2718 Ack=2818 Win=64128 Len=0 TSval=3997710699 TSecr=446928818
1745	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	182 Client: Encrypted packet (len=756)
1746	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 22 - 42440 [ACK] Seq=2818 Ack=2818 Win=64128 Len=0 TSval=446928822 TSecr=3997710699
1747	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	183 Client: Encrypted packet (len=60)
1748	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 22 - 42440 [ACK] Seq=2818 Ack=2899 Win=64128 Len=0 TSval=3997710701 TSecr=446928820
1749	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	119 Client: Encrypted packet (len=44)
1750	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 22 - 42440 [ACK] Seq=2899 Ack=2718 Win=64128 Len=0 TSval=446928822 TSecr=3997710701
1751	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	1514 Server: Encrypted packet (len=48)
1752	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	1446 Server: Encrypted packet (len=138)
1753	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	66 42240 - 22 [ACK] Seq=2882 Ack=4069 Win=64128 Len=0 TSval=3997711976 TSecr=446989298
1754	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 42240 - 22 [ACK] Seq=2882 Ack=4069 Win=64128 Len=0 TSval=3997711978 TSecr=446989298
1755	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	118 Client: Encrypted packet (len=52)
1756	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 42240 - 22 [ACK] Seq=2882 Ack=4070 Win=64128 Len=0 TSval=3997711978 TSecr=446989298
1757	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	118 Client: Encrypted packet (len=52)
1758	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 42240 - 22 [ACK] Seq=2882 Ack=4070 Win=64128 Len=0 TSval=3997711978 TSecr=446989298
1759	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	118 Client: Encrypted packet (len=52)
1760	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 42240 - 22 [ACK] Seq=2882 Ack=4070 Win=64128 Len=0 TSval=3997711978 TSecr=446989298
1761	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	118 Client: Encrypted packet (len=52)
1762	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 42240 - 22 [ACK] Seq=2882 Ack=4070 Win=64128 Len=0 TSval=3997711978 TSecr=446989298
1763	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	118 Client: Encrypted packet (len=52)
1764	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 42240 - 22 [ACK] Seq=2882 Ack=4070 Win=64128 Len=0 TSval=3997711978 TSecr=446989298
1765	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	118 Client: Encrypted packet (len=52)
1766	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 42240 - 22 [ACK] Seq=2882 Ack=4070 Win=64128 Len=0 TSval=3997711978 TSecr=446989298
1767	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	118 Client: Encrypted packet (len=52)
1768	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 42240 - 22 [ACK] Seq=2882 Ack=4070 Win=64128 Len=0 TSval=3997711978 TSecr=446989298
1769	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	118 Client: Encrypted packet (len=52)
1770	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 42240 - 22 [ACK] Seq=2882 Ack=4070 Win=64128 Len=0 TSval=3997711978 TSecr=446989298
1771	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	118 Client: Encrypted packet (len=52)
1772	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 42240 - 22 [ACK] Seq=2882 Ack=4070 Win=64128 Len=0 TSval=3997711978 TSecr=446989298
1773	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	118 Client: Encrypted packet (len=52)
1774	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 42240 - 22 [ACK] Seq=2882 Ack=4070 Win=64128 Len=0 TSval=3997711978 TSecr=446989298
1775	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	118 Client: Encrypted packet (len=52)
1776	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 42240 - 22 [ACK] Seq=2882 Ack=4070 Win=64128 Len=0 TSval=3997711978 TSecr=446989298
1777	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	118 Client: Encrypted packet (len=52)
1778	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 42240 - 22 [ACK] Seq=2882 Ack=4070 Win=64128 Len=0 TSval=3997711978 TSecr=446989298
1779	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	118 Client: Encrypted packet (len=52)
1780	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 42240 - 22 [ACK] Seq=2882 Ack=4070 Win=64128 Len=0 TSval=3997711978 TSecr=446989298
1781	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	118 Client: Encrypted packet (len=52)
1782	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 42240 - 22 [ACK] Seq=2882 Ack=4070 Win=64128 Len=0 TSval=3997711978 TSecr=446989298
1783	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	118 Client: Encrypted packet (len=52)
1784	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 42240 - 22 [ACK] Seq=2882 Ack=4070 Win=64128 Len=0 TSval=3997711978 TSecr=446989298
1785	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	118 Client: Encrypted packet (len=52)
1786	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 42240 - 22 [ACK] Seq=2882 Ack=4070 Win=64128 Len=0 TSval=3997711978 TSecr=446989298
1787	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	118 Client: Encrypted packet (len=52)
1788	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 42240 - 22 [ACK] Seq=2882 Ack=4070 Win=64128 Len=0 TSval=3997711978 TSecr=446989298
1789	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	SSHv2	118 Client: Encrypted packet (len=52)
1790	2631.5510912 . 10.1.1.1	10.2.2.1	10.2.2.1	TCP	66 42240 - 22 [ACK] Seq=2882 Ack=4070 Win=64128 Len=0 TSval=3997711978 TSecr=4469892

ip.addr == 10.4.4.1 and ip.addr == 10.2.2.1						
No.	Time	Source	Destination	Protocol	Length	Info
813	1244.9684616	10.4.4.1	10.2.2.1	ICMP	98	Echo (ping) request id=0x001f, seq=1/256, ttl=61 (request in 814)
814	1244.9686995	10.2.2.1	10.4.4.1	ICMP	98	Echo (ping) reply id=0x001f, seq=1/256, ttl=64 (reply in 813)
815	1244.9687053	10.4.4.1	10.2.2.1	ICMP	98	Echo (ping) request id=0x001f, seq=2/256, ttl=61 (request in 815)
816	1245.9687053	10.2.2.1	10.4.4.1	ICMP	98	Echo (ping) reply id=0x001f, seq=2/256, ttl=64 (reply in 815)
818	1246.9683995	10.4.4.1	10.2.2.1	ICMP	98	Echo (ping) request id=0x001f, seq=3/256, ttl=61 (request in 819)
819	1246.9665317	10.2.2.1	10.4.4.1	ICMP	98	Echo (ping) reply id=0x001f, seq=3/256, ttl=64 (reply in 818)
822	1248.9732227	10.4.4.1	10.2.2.1	ICMP	98	Echo (ping) request id=0x001f, seq=5/288, ttl=61 (request in 823)
823	1248.9732227	10.2.2.1	10.4.4.1	ICMP	98	Echo (ping) reply id=0x001f, seq=5/288, ttl=64 (reply in 822)
824	1249.9784895	10.4.4.1	10.2.2.1	ICMP	98	Echo (ping) request id=0x001f, seq=6/256, ttl=61 (request in 822)
825	1249.9750895	10.2.2.1	10.4.4.1	ICMP	98	Echo (ping) reply id=0x001f, seq=6/256, ttl=64 (request in 824)
827	1250.9763858	10.4.4.1	10.2.2.1	ICMP	98	Echo (ping) request id=0x001f, seq=7/256, ttl=61 (request in 828)
829	1251.9769630	10.4.4.1	10.2.2.1	ICMP	98	Echo (ping) reply id=0x001f, seq=7/256, ttl=64 (reply in 827)
831	1251.9771020	10.2.2.1	10.4.4.1	ICMP	98	Echo (ping) request id=0x001f, seq=8/2048, ttl=61 (request in 830)
832	1252.9774828	10.4.4.1	10.2.2.1	ICMP	98	Echo (ping) reply id=0x001f, seq=8/2048, ttl=64 (request in 829)
833	1252.9776128	10.2.2.1	10.4.4.1	ICMP	98	Echo (ping) request id=0x001f, seq=9/2384, ttl=61 (no response found!)
834	1252.9776128	10.2.2.1	10.4.4.1	ICMP	98	Echo (ping) reply id=0x001f, seq=9/2384, ttl=64 (request in 833)
835	1252.9776136	10.2.2.1	10.4.4.1	ICMP	98	Echo (ping) request id=0x001f, seq=10/2560, ttl=61 (reply in 837)
837	1253.9792190	10.4.4.1	10.2.2.1	ICMP	98	Echo (ping) reply id=0x001f, seq=10/2560, ttl=64 (request in 836)
839	1254.9888409	10.4.4.1	10.2.2.1	ICMP	98	Echo (ping) request id=0x001f, seq=11/2816, ttl=61 (request in 840)
840	1254.9889781	10.2.2.1	10.4.4.1	ICMP	98	Echo (ping) reply id=0x001f, seq=11/2816, ttl=64 (request in 839)
842	1257.9849834	10.4.4.1	10.2.2.1	ICMP	98	Echo (ping) request id=0x001f, seq=13/3232, ttl=61 (request in 843)
843	1258.9856817	10.2.2.1	10.4.4.1	ICMP	98	Echo (ping) reply id=0x001f, seq=13/3232, ttl=64 (request in 842)
844	1258.9866817	10.4.4.1	10.2.2.1	ICMP	98	Echo (ping) request id=0x001f, seq=14/3584, ttl=61 (request in 845)
845	1258.9868120	10.2.2.1	10.4.4.1	ICMP	98	Echo (ping) reply id=0x001f, seq=14/3584, ttl=64 (request in 844)
848	1259.9979654	10.4.4.1	10.2.2.1	ICMP	98	Echo (ping) request id=0x001f, seq=15/3840, ttl=61 (reply in 849)
849	1259.9888941	10.2.2.1	10.4.4.1	ICMP	98	Echo (ping) reply id=0x001f, seq=15/3840, ttl=64 (request in 848)
850	1260.9984447	10.4.4.1	10.2.2.1	ICMP	98	Echo (ping) request id=0x001f, seq=16/4096, ttl=61 (request in 851)
851	1260.9984447	10.2.2.1	10.4.4.1	ICMP	98	Echo (ping) reply id=0x001f, seq=16/4096, ttl=64 (request in 850)
853	1261.9195056	10.4.4.1	10.2.2.1	ICMP	98	Echo (ping) request id=0x001f, seq=17/4352, ttl=61 (request in 854)
854	1261.9195074	10.2.2.1	10.4.4.1	ICMP	98	Echo (ping) reply id=0x001f, seq=17/4352, ttl=64 (request in 853)
855	1262.9124791	10.4.4.1	10.2.2.1	ICMP	98	Echo (ping) request id=0x001f, seq=18/4688, ttl=61 (request in 856)
856	1262.9128117	10.2.2.1	10.4.4.1	ICMP	98	Echo (ping) reply id=0x001f, seq=18/4688, ttl=64 (request in 855)
866	1263.9143655	10.4.4.1	10.2.2.1	ICMP	98	Echo (ping) request id=0x001f, seq=19/4864, ttl=61 (request in 861)
861	1263.9144988	10.2.2.1	10.4.4.1	ICMP	98	Echo (ping) reply id=0x001f, seq=19/4864, ttl=64 (request in 860)
864	1264.9157098	10.4.4.1	10.2.2.1	ICMP	98	Echo (ping) request id=0x001f, seq=20/5120, ttl=61 (Reply in 865)
865	1264.9158513	10.2.2.1	10.4.4.1	ICMP	98	Echo (ping) reply id=0x001f, seq=20/5120, ttl=64 (request in 864)

Figura 3.15: Captura de pacotes obtida no Wireshark para a conexão ping entre o servidor (10.2.2.1) e o grilo (10.4.4.1).

4 Parte III: Uso da camada de transporte por parte das aplicações

Nesta segunda parte do trabalho pretendia-se testar, agora fora do ambiente virtual do CORE, a utilização da camada de transporte, e dos serviços que esta fornece, por parte de algumas aplicações comuns na Internet.

Na Tabela 4.1 apresentam-se os resultados obtidos, enquanto nas secções que se seguem se explica a forma como estes foram obtidos.

Comando Usado	Protocolo da aplicação	Protocolo da transporte	Porta de atendimento	Overhead de Transporte
ping	PING	-	-	-
traceroute	TRACEROUTE	UDP	33434	8
telnet	TELNET	TCP	23	20
ftp	FTP	TCP	21	20
tftp	TFTP	UDP	69	8
http	HTTP	TCP	80	20
nslookup	DNS	UDP	53	8
ssh	SSH	TCP	22	20

Tabela 4.1: Informação sobre comandos usados no CORE

4.1 ping www.google.pt

Na utilização deste serviço estamos ao nível da camada de rede, pelo que não são utilizados protocolos da camada de transporte. Assim sendo, apenas é possível preencher na Tabela 4.1 o protocolo de aplicação. Nas Figuras 4.1 e 4.2 apresentam-se imagens elucidativas da aplicação do ping.

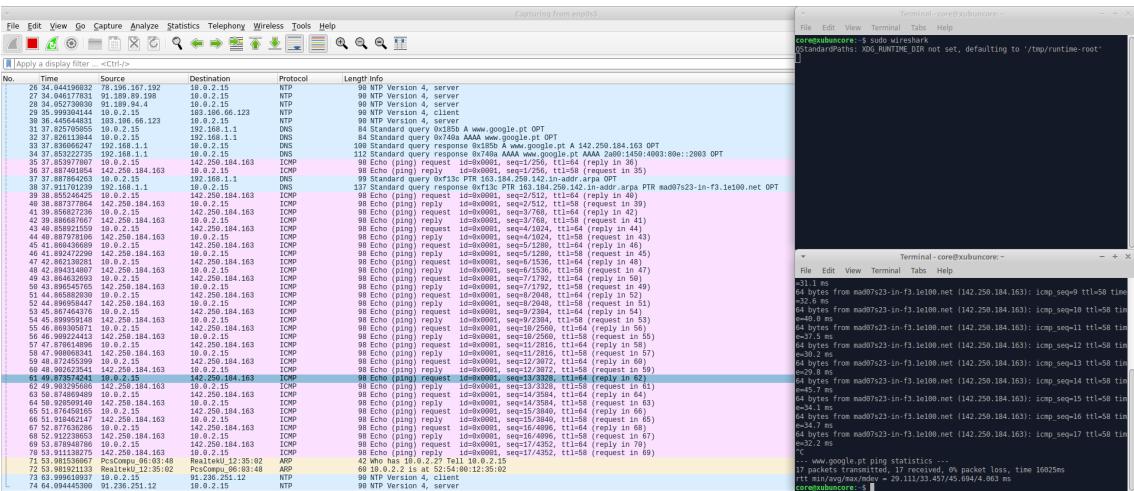


Figura 4.1: Aplicação do ping para testar a conexão ao site www.google.pt.

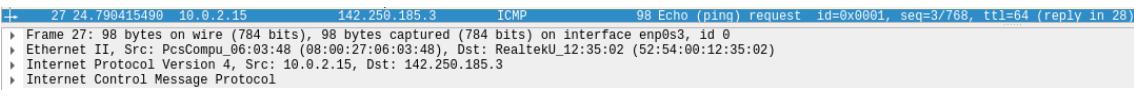


Figura 4.2: Output do wireshark após aplicação do ping para testar a conexão ao site www.google.pt.

4.2 traceroute cisco.di.uminho.pt

No caso do serviço traceroute já é possível verificar a utilização de protocolo de rede, neste caso do protocolo UDP (ver Figura 4.3). Analisando os resultados obtidos no wireshark, é possível completar Tabela 4.1 indicando todos os campos. O valor da porta de atendimento indicado é apenas o do primeiro datagrama UDP entregue. Para os seguintes, a porta de entrega é sucessivamente incrementada de uma unidade. Quanto ao overhead de transporte, é possível verificar na Figura 4.4 que a diferença entre os dados e o tamanho total do datagrama são 8 bytes.

4.3 telnet cc2022.ddns.net

No caso do serviço telnet o protocolo de rede utilizado foi o TCP (ver Figura 4.5). Analisando os resultados obtidos no wireshark, é possível completar Tabela 4.1 indicando o valor da porta de atendimento, que se mantém igual em todos os sucessivos envios de dados Telnet. Quanto ao overhead de transporte, neste caso é possível verificar na Figura 4.6 que o tamanho do cabeçalho são 20 bytes.

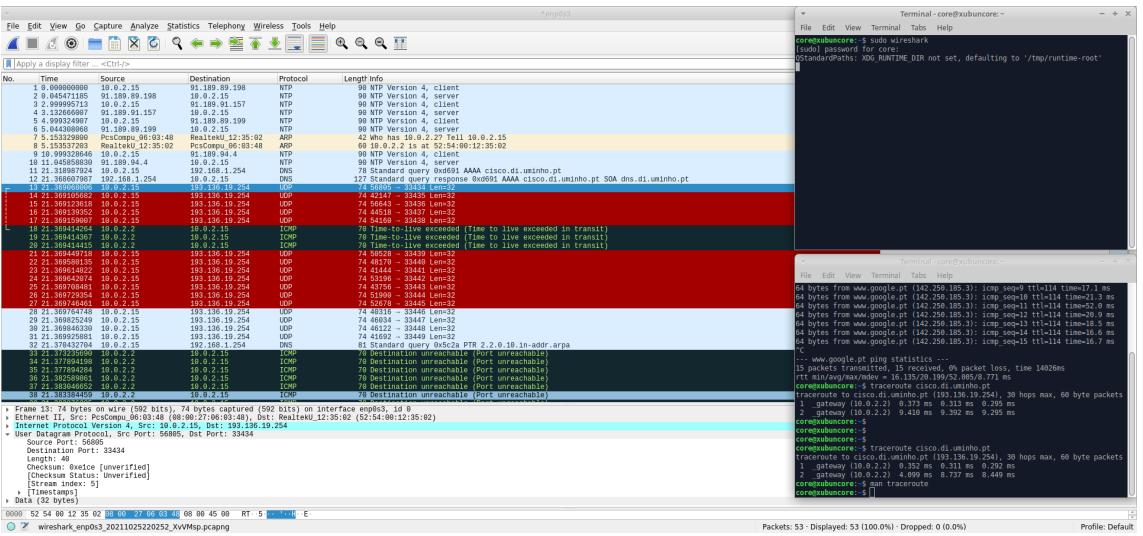


Figura 4.3: Aplicação do traceroute para testar a conexão ao site cisco.di.uminho.pt.

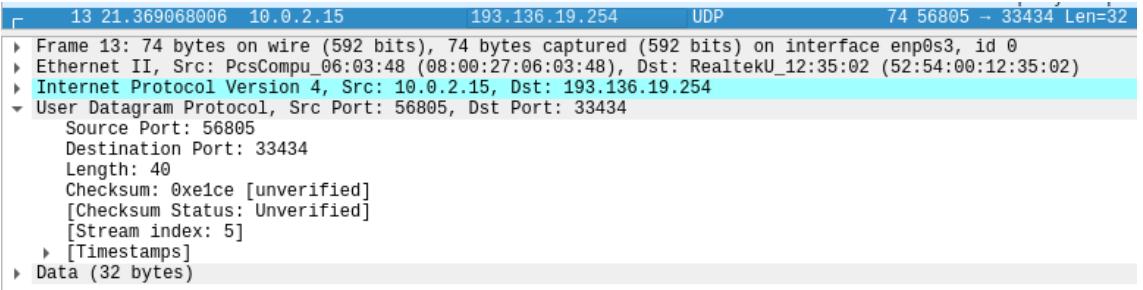


Figura 4.4: Output do wireshark após aplicação do traceroute para testar a conexão ao site cisco.di.uminho.pt.

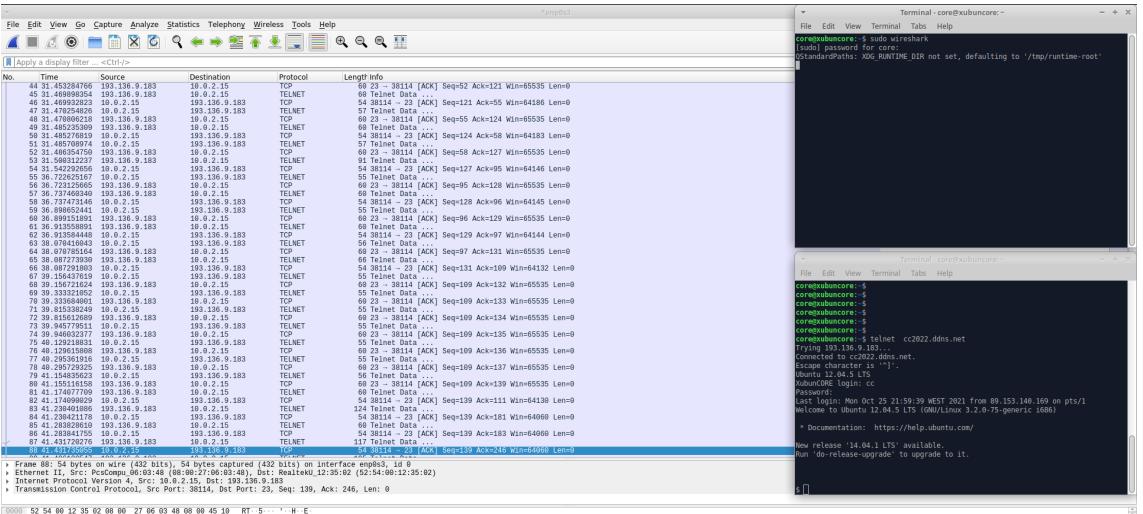


Figura 4.5: Aplicação do telnet para testar a conexão ao site cc2022.ddns.net.

```

| 88 41.431735055 10.0.2.15          193.136.9.183      TCP      54 38114 → 23 [ACK] Seq=139 Ack=246 Win=64060 Len=0
> Frame 88: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface enp0s3, id 0
> Ethernet II, Src: PcsCompu_06:03:48 (08:00:27:06:03:48), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
> Internet Protocol Version 4, Src: 10.0.2.15, Dst: 193.136.9.183
> Transmission Control Protocol, Src Port: 38114, Dst Port: 23, Seq: 139, Ack: 246, Len: 0
    Source Port: 38114
    Destination Port: 23
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence number: 139 (relative sequence number)
    Sequence number (raw): 1214642917
    [Next sequence number: 139 (relative sequence number)]
    Acknowledgment number: 246 (relative ack number)
    Acknowledgment number (raw): 6016247
    0101 ... = Header Length: 20 bytes (5)
    Flags: 0x010 (ACK)
    Window size value: 64060
    [Calculated window size: 64060]
    [Window size scaling factor: -2 (no window scaling used)]
    Checksum: 0xd768 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
    ► [SEQ/ACK analysis]
    ► [Timestamps]


```

Figura 4.6: Output do wireshark após aplicação do telnet para testar a conexão ao site cc2022.ddns.net.

4.4 ftp cc2022.ddns.net

No caso do serviço ftp, tal como já verificado na parte II do presente trabalho, o protocolo de rede utilizado foi o TCP (ver Figura 4.7). Analisando os resultados obtidos no wireshark, é possível completar Tabela 4.1 indicando o valor da porta de atendimento, que se mantém igual em todos os sucessivos envios de dados ftp. Quanto ao overhead de transporte, neste caso é possível verificar na Figura 4.8 que o tamanho do cabeçalho são 20 bytes.

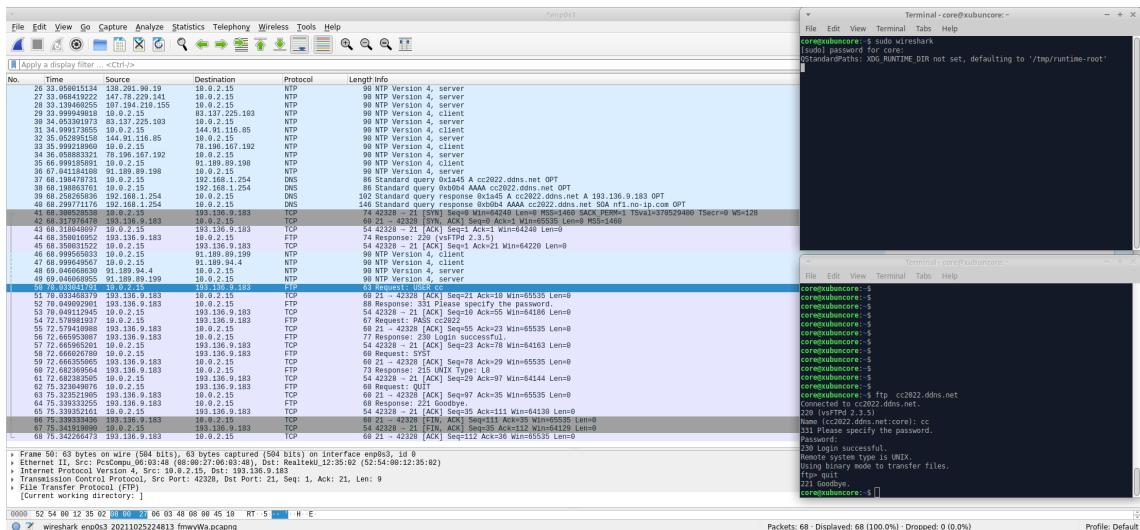


Figura 4.7: Aplicação do ftp para testar a conexão ao site cc2022.ddns.net.

4.5 tftp cc2022.ddns.net

No caso do serviço tftp, tal como já verificado na parte II do presente trabalho, o protocolo de rede utilizado foi o UDP (ver Figura 4.9). Analisando os resultados obtidos

```

| 50 70 033041791 19.0.2.15          193.136.9.183      FTP           63 Request: USER cc
|> Frame 50: 63 bytes on wire (504 bits), 63 bytes captured (504 bits) on interface enp0s3, id 0
|> Ethernet II, Src: PcsCompu_06:03:48 (08:00:27:06:03:48), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
|> Internet Protocol Version 4, Src: 10.0.2.15, Dst: 193.136.9.183
|> Transmission Control Protocol, Src Port: 42328, Dst Port: 21, Seq: 1, Ack: 21, Len: 9
    Source Port: 42328
    Destination Port: 21
    [Stream index: 0]
    [TCP Segment Len: 9]
    Sequence number: 1 (relative sequence number)
    Sequence number (raw): 2933655965
    [Next sequence number: 10 (relative sequence number)]
    Acknowledgment number: 21 (relative ack number)
    Acknowledgment number (raw): 159616022
    O101 .... = Header Length: 20 bytes (5)
|> Flags: 0x018 (PSH, ACK)
    Window size value: 64220
    [Calculated window size: 64220]
    [Window size scaling factor: -2 (no window scaling used)]
    Checksum: 0xd771 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
|> [SEQ/ACK analysis]
|> [Timestamps]
    TCP payload (9 bytes)
|> File Transfer Protocol (FTP)
    [Current working directory: ]

```

Figura 4.8: Output do wireshark após aplicação do ftp para testar a conexão ao site cc2022.ddns.net.

no wireshark, é possível completar Tabela 4.1 indicando o valor da porta de atendimento. Quanto ao overhead de transporte, neste caso é possível verificar na Figura 4.10 que o tamanho do cabeçalho não é dito explicitamente. No entanto, seleccionado o datagrama UDP é possível verificar na parte inferior da figura que surgem selecionados 8 octetos, correspondendo aos 8 bytes do overhead introduzido pelo protocolo UDP.

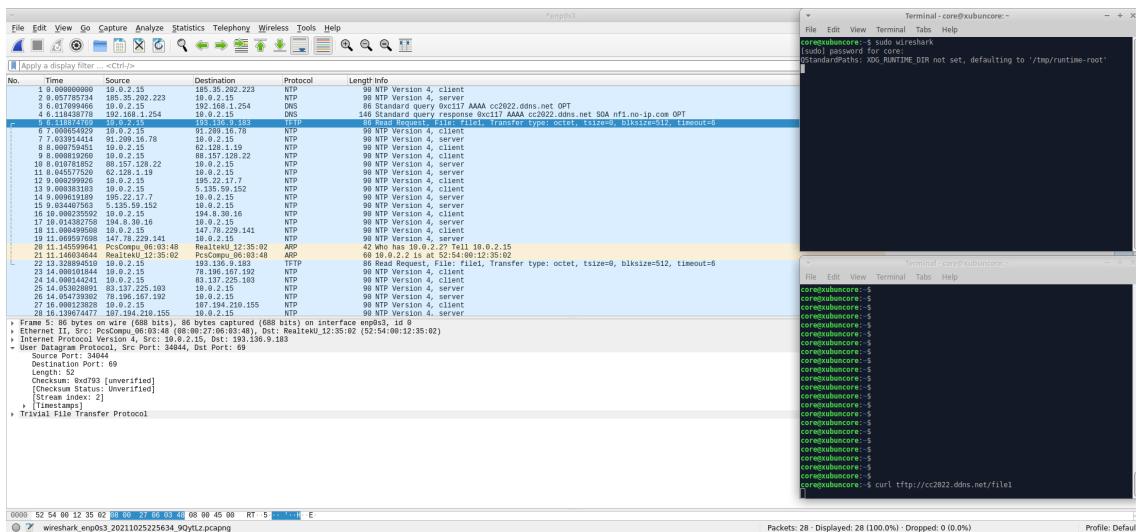


Figura 4.9: Aplicação do tftp para testar a conexão ao site cc2022.ddns.net.

4.6 http://marco.uminho.pt/disciplinas/CC-LEI/

No caso do serviço http, tal como já verificado na parte II do presente trabalho, o protocolo de rede utilizado foi o TCP (ver Figura 4.11). Analisando os resultados obtidos

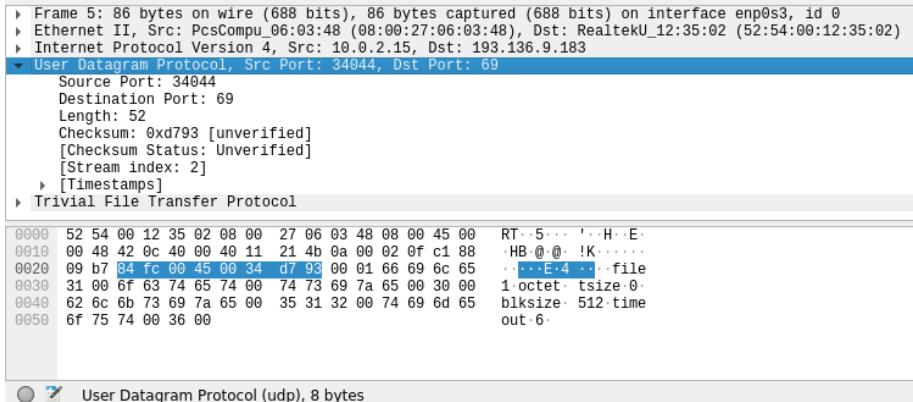


Figura 4.10: Output do wireshark após aplicação do tftp para testar a conexão ao site cc2022.ddns.net.

no wireshark, é possível completar Tabela 4.1 indicando o valor da porta de atendimento. Quanto ao overhead de transporte, neste caso é possível verificar na Figura 4.10 que o tamanho do cabeçalho são os 20 bytes do overhead mínimo introduzido pelo protocolo TCP.

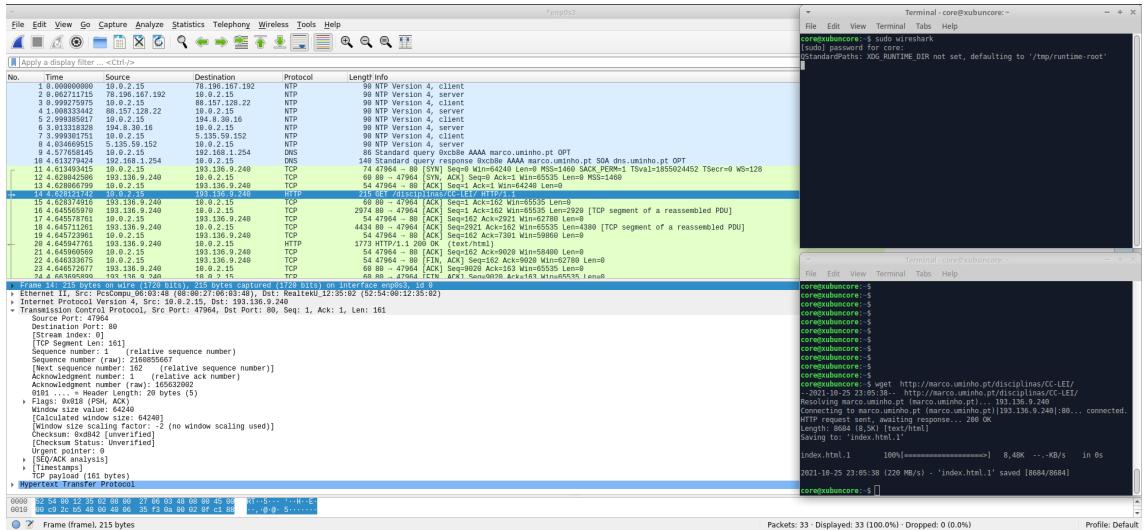


Figura 4.11: Aplicação do http para testar a conexão ao site <http://marco.uminho.pt/disciplinas/CC-LEI/>.

4.7 nslookup www.uminho.pt

No caso do serviço nslookup, o protocolo de rede utilizado foi o UDP (ver Figura 4.12). Analisando os resultados obtidos no wireshark, é possível completar Tabela 4.1 indicando o valor da porta de atendimento. Quanto ao overhead de transporte, neste caso é possível verificar na Figura 4.10 que o tamanho do cabeçalho são os 8 bytes do overhead típicos

do protocolo UDP (obtidos analisando o formato hexadecimal do datagrama).

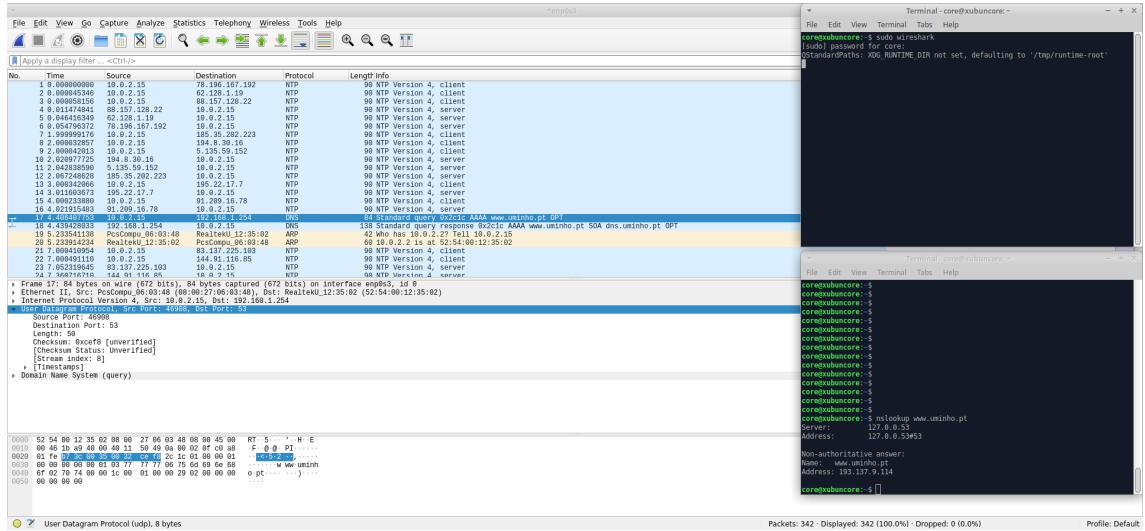


Figura 4.12: Aplicação do nslookup para testar a conexão ao site www.uminho.pt.

4.8 ssh cc2022.ddns.net

No caso do serviço ssh, tal como visto na parte II, o protocolo de rede utilizado foi o TCP (ver Figura 4.13). Analisando os resultados obtidos no wireshark, é possível completar Tabela 4.1 indicando o valor da porta de atendimento. Quanto ao overhead de transporte, neste caso é possível verificar na Figura 4.13 que o tamanho do cabeçalho são os 20 bytes do overhead mínimo do protocolo TCP.

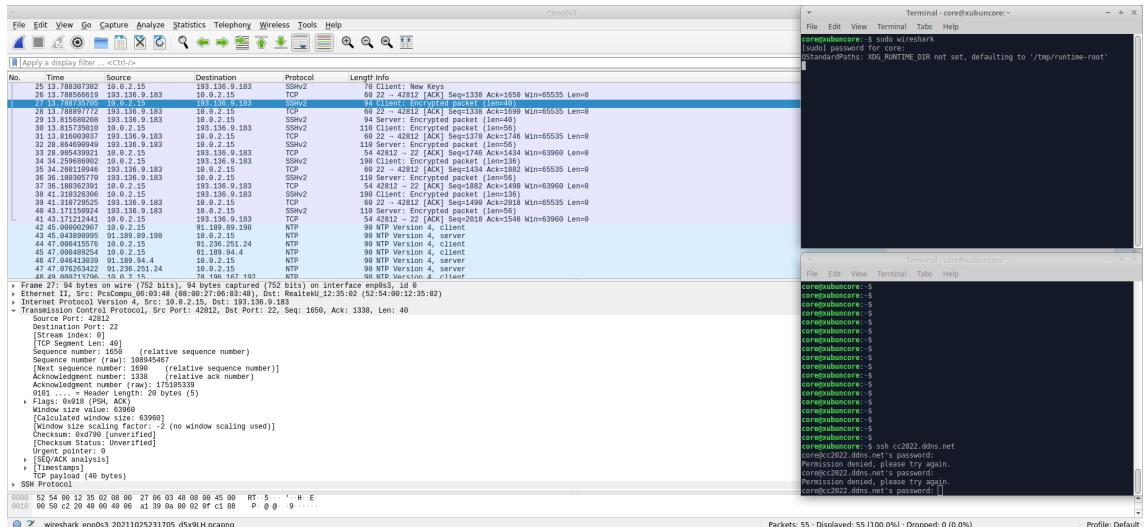


Figura 4.13: Aplicação do ssh para testar a conexão ao site cc2022.ddns.net.

5 Conclusões

Com a realização deste trabalho foi possível aprofundar os conceitos abordados em sala de aula acerca dos diversos Protocolos da Camada de Transporte e ainda alguns Protocolos da Camada de Aplicação.

Com o auxílio das ferramentas Core e Wireshark, começou por se analisar quatro serviços distintos (SFTP, FTP, TFTP, HTTP) relativamente à transferência de ficheiros. Seguidamente, compararam-se estas quatro aplicações de transferência de ficheiros quanto ao uso da camada de transporte, eficiência, complexidade e segurança.

Por fim, foram analisados protocolos da camada de aplicação e transporte, as portas que estes utilizavam e os overheads associados através da análise de tráfego.

A principal conclusão a retirar deste trabalho, e que tem aplicação no futuro profissional dos elementos do grupo, é que se se quiser realizar uma transferência de dados e garantir que todos os dados enviados são recebidos, terá de se adoptar uma aplicação que utilize o protocolo TCP. No entanto, ficou-se também a perceber que este não é o protocolo mais eficiente porque é um pouco mais lento devido ao facto de existir controlo e correcção de erros, o que introduz transferências adicionais de pacotes. Por outro lado, se a perda de pacotes não for um problema maior e se a situação exigir uma conexão rápida, a opção deverá recair em aplicações que utilizem o protocolo UDP, aproveitando, deste modo, a velocidade máxima de transferência possível.

Lista de Siglas e Acrónimos

Protocolos de Camada de Aplicação

FTP File Transfer Protocol

HTTP HyperText Transfer Protocol

STFP Secure File Transfer Protocol

TFTP Trivial File Transfer Protocol

Protocolos de Camada de Transferência

TCP Transmission Control Protocol

UDP User Datagram Protocol

Outros

ACK Acknowledgement

SSH Secure SHell