



**Universidade do Minho**  
Escola de Engenharia

# Comunicações por Computador

Relatório do Trabalho Prático n.º 1  
PL5 Grupo 2

LEI - 3º Ano - 1º Semestre

Realizador por:

A98695 Lucas Oliveira

A89292 Mike Pinto

A96208 Rafael Gomes

Braga,  
28 de março de 2024

# Conteúdo

<b>1</b>	<b>Parte I - Questões e Respostas</b>	<b>3</b>
1.1	Questão 1 . . . . .	3
1.2	Questão 2 . . . . .	5
1.3	Questão 3 . . . . .	6
1.4	Questão 4 . . . . .	8
<b>2</b>	<b>Parte II - Questões e Respostas</b>	<b>9</b>
2.1	Questão 1 . . . . .	9
2.1.1	HTTP/Brower . . . . .	10
2.1.2	SSH . . . . .	11
2.1.3	FTP . . . . .	12
2.1.4	TFTP . . . . .	13
2.1.5	Telnet . . . . .	14
2.1.6	nslookup . . . . .	15
2.1.7	Ping . . . . .	15
2.1.8	Traceroute . . . . .	16
<b>3</b>	<b>Conclusão</b>	<b>17</b>

# Lista de Figuras

1.1.1	<i>Ping</i> realizado a partir do <i>PC1</i> para o <i>Servidor1</i> . . . . .	3
1.1.2	<i>Ping</i> realizado a partir do <i>Portátil1</i> para o <i>Servidor1</i> . . . . .	4
1.1.3	Captura de pacotes, através do <i>WireShark</i> na ligação <i>SFTP</i> do <i>PC1</i> para o <i>Servidor1</i> . . . . .	4
1.2.1	Captura de pacotes, através do <i>WireShark</i> na ligação <i>FTP</i> do <i>Portátil1</i> para o <i>Servidor1</i> . . . . .	5
1.2.2	Diagrama temporal da aplicação <i>FTP</i> , com a transferência do <i>file1</i> . .	6
1.3.1	Captura de pacotes, através do <i>WireShark</i> na ligação <i>TFTP</i> do <i>Portátil1</i> para o <i>Servidor1</i> . . . . .	6
1.3.2	Diagrama temporal da aplicação <i>TFTP</i> , com a transferência do <i>file1</i> .	7
2.1.1	HTTP . . . . .	10
2.1.2	FTP . . . . .	11
2.1.3	FTP . . . . .	12
2.1.4	TFTP . . . . .	13
2.1.5	Telnet . . . . .	14
2.1.6	Nslookup . . . . .	15
2.1.7	Ping . . . . .	15
2.1.8	Traceroute . . . . .	16

# Capítulo 1

## Parte I - Questões e Respostas

### 1.1 Questão 1

De que forma as perdas e duplicações de pacotes afetaram 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.

```
--- 10.4.4.1 ping statistics ---
20 packets transmitted, 19 received, 5% packet loss, time 19043ms
rtt min/avg/max/mdev = 5.320/5.901/10.545/1.127 ms
<C1.conf# ping -c 20 10.4.4.1 | tee file-ping-output
PING 10.4.4.1 (10.4.4.1) 56(84) bytes of data.
64 bytes from 10.4.4.1: icmp_seq=1 ttl=61 time=5.65 ms
64 bytes from 10.4.4.1: icmp_seq=2 ttl=61 time=5.15 ms
64 bytes from 10.4.4.1: icmp_seq=3 ttl=61 time=5.34 ms
64 bytes from 10.4.4.1: icmp_seq=4 ttl=61 time=5.33 ms
64 bytes from 10.4.4.1: icmp_seq=5 ttl=61 time=6.03 ms
64 bytes from 10.4.4.1: icmp_seq=7 ttl=61 time=5.31 ms
64 bytes from 10.4.4.1: icmp_seq=7 ttl=61 time=5.80 ms (DUP!)
64 bytes from 10.4.4.1: icmp_seq=8 ttl=61 time=5.34 ms
64 bytes from 10.4.4.1: icmp_seq=9 ttl=61 time=5.35 ms
64 bytes from 10.4.4.1: icmp_seq=10 ttl=61 time=6.03 ms
64 bytes from 10.4.4.1: icmp_seq=11 ttl=61 time=5.34 ms
64 bytes from 10.4.4.1: icmp_seq=12 ttl=61 time=6.33 ms
64 bytes from 10.4.4.1: icmp_seq=14 ttl=61 time=5.36 ms
64 bytes from 10.4.4.1: icmp_seq=15 ttl=61 time=6.05 ms
64 bytes from 10.4.4.1: icmp_seq=16 ttl=61 time=5.35 ms
64 bytes from 10.4.4.1: icmp_seq=18 ttl=61 time=6.02 ms
64 bytes from 10.4.4.1: icmp_seq=19 ttl=61 time=5.95 ms
64 bytes from 10.4.4.1: icmp_seq=20 ttl=61 time=5.36 ms
--- 10.4.4.1 ping statistics ---
20 packets transmitted, 17 received, +1 duplicates, 15% packet loss, time 19094ms
rtt min/avg/max/mdev = 5.311/5.675/6.386/0.360 ms
```

Figura 1.1.1: *Ping* realizado a partir do *PC1* para o *Servidor1*.

```

vcmd
64 bytes from 10.4.4.1: icmp_seq=2 ttl=61 time=0.941 ms
64 bytes from 10.4.4.1: icmp_seq=3 ttl=61 time=7.27 ms
64 bytes from 10.4.4.1: icmp_seq=4 ttl=61 time=15.6 ms
64 bytes from 10.4.4.1: icmp_seq=5 ttl=61 time=10.5 ms
64 bytes from 10.4.4.1: icmp_seq=6 ttl=61 time=3.06 ms
64 bytes from 10.4.4.1: icmp_seq=7 ttl=61 time=1.77 ms
64 bytes from 10.4.4.1: icmp_seq=8 ttl=61 time=1.40 ms
64 bytes from 10.4.4.1: icmp_seq=9 ttl=61 time=0.963 ms
64 bytes from 10.4.4.1: icmp_seq=10 ttl=61 time=5.19 ms
64 bytes from 10.4.4.1: icmp_seq=11 ttl=61 time=4.83 ms
64 bytes from 10.4.4.1: icmp_seq=12 ttl=61 time=1.40 ms
64 bytes from 10.4.4.1: icmp_seq=13 ttl=61 time=2.93 ms
64 bytes from 10.4.4.1: icmp_seq=14 ttl=61 time=2.99 ms
64 bytes from 10.4.4.1: icmp_seq=15 ttl=61 time=6.75 ms
64 bytes from 10.4.4.1: icmp_seq=16 ttl=61 time=2.51 ms
64 bytes from 10.4.4.1: icmp_seq=17 ttl=61 time=3.19 ms
64 bytes from 10.4.4.1: icmp_seq=18 ttl=61 time=3.49 ms
64 bytes from 10.4.4.1: icmp_seq=19 ttl=61 time=1.62 ms
64 bytes from 10.4.4.1: icmp_seq=20 ttl=61 time=0.677 ms

--- 10.4.4.1 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19095ms
rtt min/avg/max/mdev = 0.677/4.082/15.613/3.586 ms
root@Portatil1:/tmp/pycore.42087/Portatil1.conf#

```

Figura 1.1.2: Ping realizado a partir do *Portatil1* para o *Servidor1*

Visando testar a conectividade entre os *hosts PC1* e *Portatil1* foi executado o comando *Ping* com destino ao *Servidor1* como podemos observar nas figuras 1.1.1 e 1.1.2. Para ambos os casos foram enviados 20 pacotes, tendo todos os pacotes do *Portatil1* sido entregues. Contudo, apenas 17 dos 20 pacotes enviados pelo *PC1* foram entregues ao *Servidor1* havendo assim 15% de *packet loss* e um pacote duplicado.

No.	Time	Source	Destination	Protocol	Length	Info
1051	1895.1940995	10.2.2.1	10.4.4.1	TCP	66	46336 -> 22 [ACK] Seq=1692 Ack=2182 Win=64128 Len=0 TSval=2648776166 TSecr=4840688814
1052	1896.9693487	10.4.4.254	10.4.4.1	OSPF	78	Hello Packet
1053	1897.1898469	10.2.2.1	10.4.4.1	SSHv2	82	Client: New Keys
1054	1897.1034576	10.4.4.1	10.2.2.1	TCP	66	22 -> 46336 [ACK] Seq=2182 Ack=1818 Win=64128 Len=0 TSval=4840689927 TSecr=2648772872
1055	1897.1898069	10.2.2.1	10.4.4.1	TCP	118	46336 -> 22 [PSH, ACK] Seq=1588 Ack=2182 Win=64128 Len=44 TSval=2648772876 TSecr=4840689927 [TCP segment of a reassembled...
1056	1897.1875438	10.4.4.1	10.2.2.1	TCP	66	22 -> 46336 [ACK] Seq=2182 Ack=1662 Win=64128 Len=0 TSval=4840689933 TSecr=2648772876
1057	1897.1879948	10.4.4.1	10.2.2.1	SSHv2	118	Server: Encrypted packet (len=44)
1058	1897.1140534	10.2.2.1	10.4.4.1	TCP	126	46336 -> 22 [PSH, ACK] Seq=1662 Ack=2226 Win=64128 Len=60 TSval=2648772885 TSecr=4840689934 [TCP segment of a reassembled...
1059	1897.1140534	10.4.4.1	10.2.2.1	TCP	66	22 -> 46336 [ACK] Seq=2226 Ack=1722 Win=64128 Len=0 TSval=4840689940 TSecr=2648772885
1060	1897.1140534	10.4.4.1	10.2.2.1	TCP	66	22 -> 46336 [ACK] Seq=2226 Ack=1722 Win=64128 Len=0 TSval=4840689940 TSecr=2648772885
1061	1897.1140534	10.4.4.1	10.2.2.1	TCP	118	Server: Encrypted packet (len=52)
1062	1897.1140534	10.4.4.1	10.2.2.1	SSHv2	66	46336 -> 22 [ACK] Seq=1722 Ack=2278 Win=64128 Len=0 TSval=2648772899 TSecr=4840689947
1063	1897.1279812	10.2.2.1	10.4.4.1	TCP	118	46336 -> 22 [PSH, ACK] Seq=1722 Ack=2278 Win=64128 Len=84 TSval=2648772876 TSecr=4840689947 [TCP segment of a reassembled...
1064	1898.0873649	10.2.2.1	10.4.4.1	TCP	66	22 -> 46336 [ACK] Seq=2278 Ack=1896 Win=64128 Len=0 TSval=4840691733 TSecr=2648772876
1065	1898.9888555	10.4.4.1	10.2.2.1	TCP	66	22 -> 46336 [ACK] Seq=2278 Ack=1896 Win=64128 Len=0 TSval=4840691733 TSecr=2648772876
1066	1898.9338431	10.4.4.1	10.2.2.1	SSHv2	84	Server: Encrypted packet (len=28)
1067	1898.9391816	10.2.2.1	10.4.4.1	TCP	66	46336 -> 22 [ACK] Seq=1896 Ack=2306 Win=64128 Len=0 TSval=2648773911 TSecr=4840691759

Figura 1.1.3: Captura de pacotes, através do *WireShark* na ligação *SFTP* do *PC1* para o *Servidor1*

A camada responsável pela perda e/ou duplicação de pacotes é a camada de transporte mediante o uso de protocolos fiáveis como o *TCP*, orientado à ligação, uma vez que garante que todos os pacotes são entregues ao destino e na ordem correta, pois para todos os pacotes recebidos é enviado um pacote *ACK* (*Acknowledgment*). Contudo, redes de pior qualidade, como a rede do *PC1*, são mais suscetíveis a uma perda e/ou duplicação de pacotes, o que obriga ao envio de mensagens de erro, como podemos observar nas tramas n.º 1059 e 1061 da figura 1.1.3, o que poderá levar a uma sobrecarga na rede e eventualmente atrasos na aplicação.

Recorrendo ao protocolo *UDP*, apesar deste ser mais “rápido” e de fácil utilização, poderá ser a própria aplicação responsável pela garantia de entrega dos pacotes no caso destes não serem entregues, visto o protocolo de transporte *UDP* não ser orientado à ligação, não possuindo capacidade de verificação de entrega de pacotes, o poderá levar a um eventual atraso na aplicação.

## 1.2 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.

145	146.425465201	10.1.1.1	10.4.4.1	FTP	78 Request: RETR file1
146	146.426426888	10.4.4.1	10.1.1.1	TCP	74 20 → 44495 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 T...
147	146.426596503	10.1.1.1	10.4.4.1	TCP	74 44495 → 20 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SA...
148	146.426720587	10.4.4.1	10.1.1.1	TCP	66 20 → 44495 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3414420547...
149	146.426842341	10.4.4.1	10.1.1.1	FTP	130 Response: 150 Opening BINARY mode data connection for file1 (...)
150	146.426845201	10.4.4.1	10.1.1.1	FTP-DA...	290 FTP Data: 224 bytes (PORT) (RETR file1)
151	146.426846067	10.4.4.1	10.1.1.1	TCP	66 20 → 44495 [FIN, ACK] Seq=225 Ack=1 Win=64256 Len=0 TSval=341...
152	146.427063395	10.1.1.1	10.4.4.1	TCP	66 44495 → 20 [ACK] Seq=1 Ack=225 Win=65024 Len=0 TSval=36164904...
153	146.427418753	10.1.1.1	10.4.4.1	TCP	66 44495 → 20 [FIN, ACK] Seq=1 Ack=226 Win=65024 Len=0 TSval=361...
154	146.427569990	10.4.4.1	10.1.1.1	TCP	66 20 → 44495 [ACK] Seq=226 Ack=2 Win=64256 Len=0 TSval=34144205...
155	146.427848443	10.4.4.1	10.1.1.1	FTP	90 Response: 226 Transfer complete.

Figura 1.2.1: Captura de pacotes, através do *WireShark* na ligação *FTP* do *Portátil1* para o *Servidor1*

Através da imagem 1.2.1 verificamos que o servidor realiza um pedido de conexão com o cliente ao enviar um pacote *TCP* com a *flag* [SYN]. O cliente após receber o pacote de pedido de conexão responde com um pacote com as *flags* [SYN,ACK] estabelecendo-se assim a conexão. Foi efetuada a transferência de dados, como podemos observar pela trama n.º 150. Por fim o servidor inicia o fim da conexão enviando um pacote *TCP* com a *flag* [FIN,ACK], seguidamente o cliente ao receber este responde com um pacote com as *flags* [FIN, ACK] como consta na trama n.º 153.

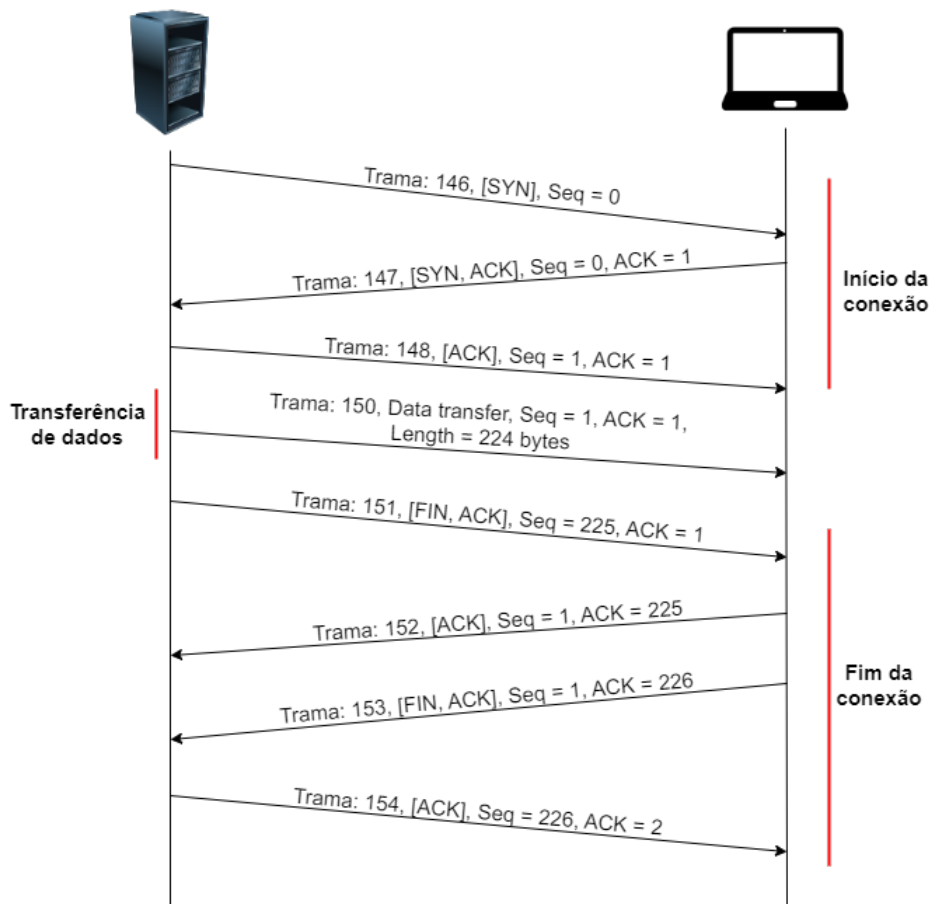


Figura 1.2.2: Diagrama temporal da aplicação *FTP*, com a transferência do *file1*

### 1.3 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.

100	134.11747...	10.1.1.1	10.4.4.1	TFTP	56 Read Request, File: file1, Transfer type: octet
101	134.12221...	10.4.4.1	10.1.1.1	TFTP	270 Data Packet, Block: 1 (last)
102	134.12285...	10.1.1.1	10.4.4.1	TFTP	46 Acknowledgement, Block: 1

Figura 1.3.1: Captura de pacotes, através do *WireShark* na ligação *TFTP* do *Portátil1* para o *Servidor1*

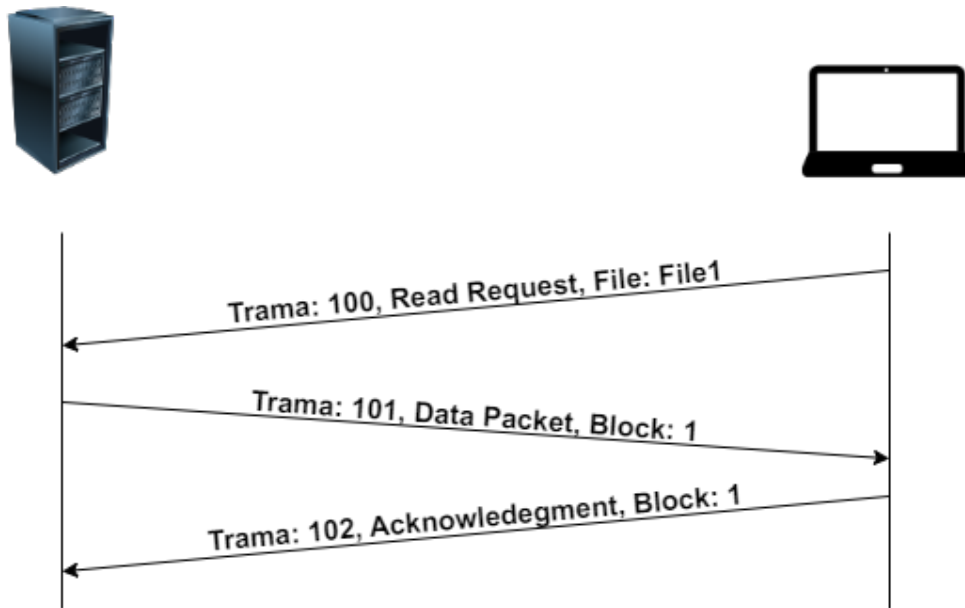


Figura 1.3.2: Diagrama temporal da aplicação *TFTP*, com a transferência do *file1*

Através da imagem 1.3.2, podemos observar que o cliente envia um *Read Request* para o servidor, que em resposta envia um *Data Packet*(pacote de dados). Por fim, o utilizador envia um pacote de *ACK*, de modo a confirmar o sucesso da operação.



## 1.4 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;

Tabela 1.1: Comparação do uso da camada de transporte, eficiência, complexidade e segurança de diferentes aplicações de transferência de ficheiros

Aplicações	SFTP	FTP	TFTP	HTTP
<b>Camada de Transporte</b>	Protocolo TCP	Protocolo TCP	Protocolo UDP	Protocolo TCP
<b>Eficiência</b>	Possui uma boa eficiência	Eficiência decente	É altamente eficiente, mas não se responsabiliza pela entrega de dados	Elevada eficiência
<b>Complexidade</b>	Elevada complexidade	Elevada complexidade	Simples e direto com baixa complexidade	Baixa complexidade
<b>Segurança</b>	Muito seguro, pois recorre ao uso de autenticação e codificação dos dados	Pouco seguro, apesar de possuir autenticação	Inseguro, pois não possui autenticação e codificação de dados	Pouco seguro

## Capítulo 2

# Parte II - Questões e Respostas

### 2.1 Questão 1

Com base no trabalho realizado, tanto na parte I como na parte II, identifique para cada aplicação executada, qual o protocolo de aplicação, o protocolo de transporte, porta de atendimento e overhead de transporte.

Tabela 2.1: Identificação de protocolos de aplicação, transporte, porta de atendimento e overhead de transporte por aplicação

Comando Usado	Protocolo de Aplicação	Protocolo de Transporte	Porta de Atendimento	Overhead de Transporte em Bytes
wget, lynx ou browser	HTTP	TCP	80	$(20/406) \times 100 \approx 4,93\%$
ssh,sftp	SSH	TCP	22	$(20/67) \times 100 \approx 29,85\%$
FTP	FTP	TCP	21 (controle conexão) 20 (transferencia de dados)	$(20/60) \times 100 \approx 33,33\%$
tftp	TFTP	UDP	69	$(8/72) \times 100 \approx 11,11\%$
telnet	TELNET	TCP	23	$(20/67) \times 100 \approx 29,85\%$
nslookup ou dig	DNS	UDP	53	$(8/70) \times 100 \approx 11,43\%$
ping	-	-	-	0
tracert	-	UDP	33434 (em particular)	$(8/60) \times 100 \approx 13,33\%$

## 2.1.1 HTTP/Brower

No.	Time	Source	Destination	Protocol	Length	Info
7	0.436567944	23.239.87.12	10.0.2.15	TCP	60	[TCP ACKed unseen segment] 80 → 59830
8	0.432853116	10.0.2.15	44.249.241.152	TCP	54	46410 → 443 [ACK] Seq=1 Ack=1 Win=62780
9	0.432578975	44.249.241.152	10.0.2.15	TCP	60	[TCP ACKed unseen segment] 443 → 46410
10	0.689060282	10.0.2.15	193.136.9.240	HTTP	420	GET /disciplinas/CC-LEI/ HTTP/1.1
11	0.685979768	193.136.9.240	10.0.2.15	TCP	60	80 → 43666 [ACK] Seq=1 Ack=367 Win=6553
12	0.722890906	193.136.9.240	10.0.2.15	TCP	1466	80 → 43666 [PSH, ACK] Seq=1 Ack=367 Win=6
13	0.722890906	10.0.2.15	193.136.9.240	TCP	54	43666 → 80 [ACK] Seq=367 Ack=1413 Win=6
14	0.722890906	193.136.9.240	10.0.2.15	TCP	1514	80 → 43666 [ACK] Seq=1413 Ack=367 Win=6
15	0.726891755	10.0.2.15	193.136.9.240	TCP	54	43666 → 80 [ACK] Seq=367 Ack=2873 Win=6
16	0.733726938	193.136.9.240	10.0.2.15	TCP	5894	80 → 43666 [ACK] Seq=2873 Ack=367 Win=6

▶ Frame 10: 420 bytes on wire (3360 bits), 420 bytes captured (3360 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: 193.136.9.240  
 ▶ 0100 .... = Version: 4  
 .... 0101 = Header Length: 20 bytes (5)  
 ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)  
 Total Length: 406  
 Identification: 0x729d (29341)  
 Flags: 0x4000, Don't fragment  
 Fragment offset: 0  
 Time to live: 64  
 Protocol: TCP (6)  
 Header checksum: 0xef3d [validation disabled]  
 [Header checksum status: Unverified]  
 Source: 10.0.2.15  
 Destination: 193.136.9.240  
 ▶ Transmission Control Protocol, Src Port: 43666, Dst Port: 80, Seq: 1, Ack: 1, Len: 366  
 Source Port: 43666  
 Destination Port: 80  
 [Stream index: 0]  
 [TCP Segment Len: 366]  
 Sequence number: 1 (relative sequence number)  
 Sequence number (raw): 1748362975  
 [Next sequence number: 367 (relative sequence number)]  
 Acknowledgment number: 1 (relative ack number)  
 Acknowledgment number (raw): 9280002  
 0101 .... = Header Length: 20 bytes (5)  
 ▶ Flags: 0x018 (PSH, ACK)  
 Window size value: 64240  
 [Calculated window size: 64240]  
 [Window size scaling factor: -2 (no window scaling used)]  
 Checksum: 0xd90f [unverified]  
 [Checksum Status: Unverified]  
 Urgent pointer: 0  
 ▶ [SEQ/ACK analysis]  
 ▶ [Timestamps]  
 TCP payload (366 bytes)  
 ▶ Hypertext Transfer Protocol

Figura 2.1.1: HTTP

Após a análise da captura de tráfego da figura 2.1.1 é possível verificar que o Protocolo de aplicação utilizado é o HTTP, o protocolo de transporte o TCP, a porta de atendimento é a 80. O tamanho total da trama é de 201 bytes com 20 bytes de header de transporte, tendo assim a trama um overhead de transporte de  $(20/406) \times 100 \approx 4,92\%$

## 2.1.2 SSH

No.	Time	Source	Destination	Protocol	Length	Info
4	5.116038351	10.0.2.15	193.136.9.201	TCP	74	37218 → 22 [SYN] Seq=0 Win=6...
5	5.128793222	193.136.9.201	10.0.2.15	TCP	60	22 → 37218 [SYN, ACK] Seq=0 ...
6	5.128975493	10.0.2.15	193.136.9.201	TCP	54	37218 → 22 [ACK] Seq=1 Ack=1...
7	5.129764255	10.0.2.15	193.136.9.201	SSHv2	95	Client: Protocol (SSH-2.0-Op...
8	5.130355280	193.136.9.201	10.0.2.15	TCP	60	22 → 37218 [ACK] Seq=1 Ack=4...
9	5.245113023	193.136.9.201	10.0.2.15	SSHv2	95	Server: Protocol (SSH-2.0-Op...
10	5.245163889	10.0.2.15	193.136.9.201	TCP	54	37218 → 22 [ACK] Seq=42 Ack=...
11	5.246806796	193.136.9.201	10.0.2.15	SSHv2	1134	Server: Key Exchange Init
12	5.246831016	10.0.2.15	193.136.9.201	TCP	54	37218 → 22 [ACK] Seq=42 Ack=...
13	5.248657725	10.0.2.15	193.136.9.201	SSHv2	1566	Client: Key Exchange Init

▶ Frame 9: 95 bytes on wire (760 bits), 95 bytes captured (760 bits) on interface enp0s3, id 0  
 ▶ Ethernet II, Src: RealtekU 12:35:02 (52:54:00:12:35:02), Dst: PcsCompu\_06:03:48 (08:00:27:06:03:48)  
 ▶ Internet Protocol Version 4, Src: 193.136.9.201, Dst: 10.0.2.15  
 0100 .... = Version: 4  
 .... 0101 = Header Length: 20 bytes (5)  
 ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)  
 Total Length: 81  
 Identification: 0x018d (397)  
 ▶ Flags: 0x0000  
 Fragment offset: 0  
 Time to live: 64  
 Protocol: TCP (6)  
 Header checksum: 0xa1ba [validation disabled]  
 [Header checksum status: Unverified]  
 Source: 193.136.9.201  
 Destination: 10.0.2.15  
 ▶ Transmission Control Protocol, Src Port: 22, Dst Port: 37218, Seq: 1, Ack: 42, Len: 41  
 Source Port: 22  
 Destination Port: 37218  
 [Stream index: 0]  
 [TCP Segment Len: 41]  
 Sequence number: 1 (relative sequence number)  
 Sequence number (raw): 2816002  
 [Next sequence number: 42 (relative sequence number)]  
 Acknowledgment number: 42 (relative ack number)  
 Acknowledgment number (raw): 2707937426  
 0101 .... = Header Length: 20 bytes (5)  
 ▶ Flags: 0x018 (PSH, ACK)  
 Window size value: 65535  
 [Calculated window size: 65535]  
 [Window size scaling factor: -2 (no window scaling used)]  
 Checksum: 0xa97c [unverified]  
 [Checksum Status: Unverified]  
 Urgent pointer: 0  
 ▶ [SEQ/ACK analysis]  
 ▶ [Timestamps]  
 TCP payload (41 bytes)  
 ▶ SSH Protocol

Figura 2.1.2: FTP

Pela figura 2.1.2, verificamos que o protocolo de aplicação utilizado é o SSH, o protocolo de transporte o TCP e a porta de atendimento a 22. O tamanho total da trama é de 67 bytes com 20 bytes de header de transporte, tendo assim a trama um overhead de transporte de  $(20/67) \times 100 \approx 29,85\%$

### 2.1.3 FTP

7	5.0901863...	10.0.2.15	193.137.214.36	TCP	74	46520 → 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 ...
8	5.0965534...	193.137.214.36	10.0.2.15	TCP	60	21 → 46520 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=...
9	5.0965951...	10.0.2.15	193.137.214.36	TCP	54	46520 → 21 [ACK] Seq=1 Ack=1 Win=64240 Len=0
10	5.1092277...	193.137.214.36	10.0.2.15	FTP	74	Response: 220 (vsFTPD 3.0.3)
11	5.1092524...	10.0.2.15	193.137.214.36	TCP	54	46520 → 21 [ACK] Seq=1 Ack=21 Win=64220 Len=0
12	8.1528786...	10.0.2.15	193.137.214.36	FTP	64	Request: USER ftp
13	8.1533054...	193.137.214.36	10.0.2.15	TCP	60	21 → 46520 [ACK] Seq=21 Ack=11 Win=65535 Len=0

```

Frame 10: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface enp0s3, id 0
Ethernet II, Src: RealtekU 12:35:02 (52:54:00:12:35:02), Dst: PcsCompu_06:03:48 (08:00:27:06:03:48)
Internet Protocol Version 4, Src: 193.137.214.36, Dst: 10.0.2.15
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 60
  Identification: 0x0f25 (3877)
  Flags: 0x0000
  Fragment offset: 0
  Time to live: 64
  Protocol: TCP (6)
  Header checksum: 0xc7da [validation disabled]
  [Header checksum status: Unverified]
  Source: 193.137.214.36
  Destination: 10.0.2.15
Transmission Control Protocol, Src Port: 21, Dst Port: 46520, Seq: 1, Ack: 1, Len: 20
  Source Port: 21
  Destination Port: 46520
  [Stream index: 0]
  [TCP Segment Len: 20]
  Sequence number: 1 (relative sequence number)
  Sequence number (raw): 29184002
  [Next sequence number: 21 (relative sequence number)]
  Acknowledgment number: 1 (relative ack number)
  Acknowledgment number (raw): 3415171310
  0101 .... = Header Length: 20 bytes (5)
  Flags: 0x018 (PSH, ACK)
  Window size value: 65535
  [Calculated window size: 65535]
  [Window size scaling factor: -2 (no window scaling used)]

```

Figura 2.1.3: FTP

Como podemos observar pela figura 2.1.3 o protocolo de aplicação utilizado é o FTP, o de transporte o TCP e a porta de atendimento a porta 21(utilizada para controlo de conexão) e a porta 20(utilizada para transferência de dados). O tamanho do pacote é de 60 bytes com 20 bytes de header, tendo a trama um overhead de transporte de  $(20/60) \times 100 \approx 33,33\%$ .

## 2.1.4 TFTP

24	7.248558244	10.0.2.15	193.136.9.201	TFTP	86	Read Request, File: file1, To
25	8.009630146	10.0.2.15	88.157.128.22	NTP	90	NTP Version 4, client
26	8.029795339	88.157.128.22	10.0.2.15	NTP	90	NTP Version 4, server
27	9.017603691	10.0.2.15	91.209.16.78	NTP	90	NTP Version 4, client
28	9.019931478	10.0.2.15	185.125.190.57	NTP	90	NTP Version 4, client
29	9.040684928	91.209.16.78	10.0.2.15	NTP	90	NTP Version 4, server
30	9.008601816	185.125.190.57	10.0.2.15	MTD	90	MTD Version 4, server

▶ Frame 24: 86 bytes on wire (688 bits), 86 bytes captured (688 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: 193.136.9.201  
 0100 .... = Version: 4  
 .... 0101 = Header Length: 20 bytes (5)  
 ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)  
 Total Length: 72  
 Identification: 0x2307 (8967)  
 ▶ Flags: 0x4000, Don't fragment  
 Fragment offset: 0  
 Time to live: 64  
 Protocol: UDP (17)  
 Header checksum: 0x403e [validation disabled]  
 [Header checksum status: Unverified]  
 Source: 10.0.2.15  
 Destination: 193.136.9.201  
 ▶ User Datagram Protocol, Src Port: 46067, Dst Port: 69  
 Source Port: 46067  
 Destination Port: 69  
 Length: 52  
 Checksum: 0xd7a5 [unverified]  
 [Checksum Status: Unverified]  
 [Stream index: 2]  
 [Timestamps]  
 ▶ Trivial File Transfer Protocol  
 Opcode: Read Request (1)  
 Source File: file1  
 Type: octet  
 ▶ Option: tsize = 0  
 ▶ Option: blksize = 512  
 ▶ Option: timeout = 6

Figura 2.1.4: TFTP

Pela figura 2.1.4 verificamos que o protocolo de aplicação utilizado é o TFTP, o protocolo de transporte o UDP e a porta de atendimento a 69. O tamanho do pacote é de 72 bytes e um header UDP possui o tamanho de 8 bytes, tendo a trama um overhead de transporte de  $(8/72) \times 100 \approx 11,11\%$ .

## 2.1.5 Telnet

1	0.00000000...	10.0.2.15	193.136.9.33	TCP	74	54906 → 23 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 ...
2	0.0064719...	193.136.9.33	10.0.2.15	TCP	60	23 → 54906 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=...
3	0.0065133...	10.0.2.15	193.136.9.33	TCP	54	54906 → 23 [ACK] Seq=1 Ack=1 Win=64240 Len=0
4	0.0069728...	10.0.2.15	193.136.9.33	TELN...	81	Telnet Data ...
5	0.0073448...	193.136.9.33	10.0.2.15	TCP	60	23 → 54906 [ACK] Seq=1 Ack=28 Win=65535 Len=0
6	0.0121966...	193.136.9.33	10.0.2.15	TELN...	66	Telnet Data ...
7	0.0122163...	10.0.2.15	193.136.9.33	TCP	54	54906 → 23 [ACK] Seq=28 Ack=13 Win=64228 Len=0

```

Frame 4: 81 bytes on wire (648 bits), 81 bytes captured (648 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: 193.136.9.33
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x10 (DSCP: Unknown, ECN: Not-ECT)
  Total Length: 67
  Identification: 0x36e1 (14049)
  Flags: 0x4000, Don't fragment
  Fragment offset: 0
  Time to live: 64
  Protocol: TCP (6)
  Header checksum: 0xd0c [validation disabled]
  [Header checksum status: Unverified]
  Source: 10.0.2.15
  Destination: 193.136.9.33
Transmission Control Protocol, Src Port: 54906, Dst Port: 23, Seq: 1, Ack: 1, Len: 27
  Source Port: 54906
  Destination Port: 23
  [Stream index: 0]
  [TCP Segment Len: 27]
  Sequence number: 1 (relative sequence number)
  Sequence number (raw): 3062554437
  [Next sequence number: 28 (relative sequence number)]
  Acknowledgment number: 1 (relative ack number)
  Acknowledgment number (raw): 30592002
  0101 .... = Header Length: 20 bytes (5)
  Flags: 0x018 (PSH, ACK)
  Window size value: 64240
  [Calculated window size: 64240]
  [Window size scaling factor: -2 (no window scaling used)]

```

Figura 2.1.5: Telnet

Após a análise da captura da figura 2.1.5, verificamos que o protocolo de aplicação utilizado é o TELNET, o protocolo de transporte o TCP e a porta de atendimento a 23. O tamanho desta trama é de 67 bytes com um header de 20 bytes, tendo então um overhead de transporte de  $(20/67) \times 100 \approx 29,85\%$

## 2.1.6 nslookup

```
1 0.00000000 10.0.2.15 193.137.16.65 DNS 84 Standard query 0x54c7 AAAA www.uminho.pt OPT
2 0.003680250 193.137.16.65 10.0.2.15 DNS 147 Standard query response 0x54c7 AAAA www.uminho.pt SO_

Terminal - core@xubuncore: ~
File Edit View Terminal Tabs Help

core@xubuncore:~$ nslookup www.uminho.pt
Server:
127.0.0.53
Address:
127.0.0.53#53

Non-authoritative answer:
Name: www.uminho.pt
Address: 193.137.9.114

... 0101 = Header Length: 20 bytes (5)
... Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
... Total Length: 70
... Identification: 0x7c7d (31869)
... Flags: 0x4000, Don't fragment
... Fragment offset: 0
... Time to live: 64
... Protocol: UDP (17)
... Header checksum: 0xe050 [validation disabled]
... [Header checksum status: Unverified]
... Source: 10.0.2.15
... Destination: 193.137.16.65
... User Datagram Protocol, Src Port: 43285, Dst Port: 53
... Source Port: 43285
... Destination Port: 53
... Length: 50
... Checksum: 0xdec [unverified]
... [Checksum status: Unverified]
... [Stream index: 0]
... [Timestamps]
... Domain Name System (query)
```

Figura 2.1.6: Nslookup

Pela figura 2.1.6 temos que o protocolo de aplicação utilizado é o DNS, o de transporte o UDP e a porta de atendimento a 53. Esta trama possui um tamanho de 70 bytes eo protoloco UDP um header de 8 bytes, tendo então um overhead de transporte de  $(8/70) \times 100 \approx 11,42\%$ .

## 2.1.7 Ping

No.	Time	Source	Destination	Protocol	Length Info
1	0.000000000	10.0.2.15	193.137.16.65	DNS	84 Standard query 0xb346 A www.google.pt OPT
2	0.004478353	193.137.16.65	10.0.2.15	DNS	100 Standard query response 0xb346 A www.google.pt A 142...
3	0.000184929	10.0.2.15	142.250.201.67	ICMP	98 Echo (ping) request id=0x0002, seq=1/256, ttl=64 (r...
4	0.036148963	142.250.201.67	10.0.2.15	ICMP	98 Echo (ping) reply id=0x0002, seq=1/256, ttl=113 (-...
5	0.036798108	10.0.2.15	193.137.16.65	DNS	98 Standard query 0xc9a6 PTR 67.201.250.142.in-addr.ar...
6	0.947792917	193.137.16.65	10.0.2.15	DNS	136 Standard query response 0xc9a6 PTR 67.201.250.142.in...
7	1.006363556	10.0.2.15	142.250.201.67	ICMP	98 Echo (ping) request id=0x0002, seq=2/512, ttl=64 (r...
8	1.029954137	142.250.201.67	10.0.2.15	ICMP	98 Echo (ping) reply id=0x0002, seq=2/512, ttl=113 (-...
9	2.000192522	10.0.2.15	142.250.201.67	ICMP	98 Echo (ping) request id=0x0002, seq=3/768, ttl=64 (r...
10	2.031155532	142.250.201.67	10.0.2.15	ICMP	98 Echo (ping) reply id=0x0002, seq=3/768, ttl=113 (-...
11	3.011685683	10.0.2.15	142.250.201.67	ICMP	98 Echo (ping) request id=0x0002, seq=4/1024, ttl=64 (-...
12	3.03919370	142.250.201.67	10.0.2.15	ICMP	98 Echo (ping) reply id=0x0002, seq=4/1024, ttl=113 (-...
13	4.015519754	10.0.2.15	142.250.201.67	ICMP	98 Echo (ping) request id=0x0002, seq=5/1280, ttl=64 (-...
14	4.040321972	142.250.201.67	10.0.2.15	ICMP	98 Echo (ping) reply id=0x0002, seq=5/1280, ttl=113 (-...

```
... 0101 = Header Length: 20 bytes (5)
... Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
... Total Length: 84
... Identification: 0xdd93 (56723)
... Flags: 0x4000, Don't fragment
... Fragment offset: 0
... Time to live: 64
... Protocol: ICMP (1)
... Header checksum: 0xf8c8 [validation disabled]
... [Header checksum status: Unverified]
... Source: 10.0.2.15
... Destination: 142.250.201.67
... Internet Control Message Protocol
```

Figura 2.1.7: Ping

Pela figura 2.1.7, é possível observar que nenhum protocolo de aplicação ou transporte é utilizado devido ao comando ping trabalhar diretamente sobre a camada de rede, logo não possui nenhuma porta de atendimento e um overhead de transporte de 0 bytes.



## 2.1.8 Traceroute

1	0.0000000...	10.0.2.15	193.137.16.65	DNS	89	Standard query 0x67a8 A cisco.di.uminho.pt OPT
2	0.0002335...	10.0.2.15	193.137.16.65	DNS	89	Standard query 0xfeaf AAAA cisco.di.uminho.pt OPT
3	0.0075612...	193.137.16.65	10.0.2.15	DNS	138	Standard query response 0xfeaf AAAA cisco.di.umi...
4	0.0104916...	193.137.16.65	10.0.2.15	DNS	105	Standard query response 0x67a8 A cisco.di.uminho...
5	0.0109117...	10.0.2.15	193.136.19.254	UDP	74	38605 → 33434 Len=32
6	0.0109496...	10.0.2.15	193.136.19.254	UDP	74	53137 → 33435 Len=32
7	0.0109792...	10.0.2.15	193.136.19.254	UDP	74	53781 → 33436 Len=32
8	0.0110579...	10.0.2.15	193.136.19.254	UDP	74	42170 → 33437 Len=32
9	0.0110997...	10.0.2.15	193.136.19.254	UDP	74	40580 → 33438 Len=32

<ul style="list-style-type: none"> <li>Frame 5: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface enp0s3, id 0</li> <li>Ethernet II, Src: PcsCompu_06:03:48 (08:00:27:06:03:48), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)</li> <li>Internet Protocol Version 4, Src: 10.0.2.15, Dst: 193.136.19.254 <ul style="list-style-type: none"> <li>0100 .... = Version: 4</li> <li>... 0101 = Header Length: 20 bytes (5)</li> <li>Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)</li> <li>Total Length: 60</li> <li>Identification: 0x01b6 (438)</li> <li>Flags: 0x0000</li> <li>Fragment offset: 0</li> <li>Time to live: 1</li> <li>Protocol: UDP (17)</li> <li>Header checksum: 0xd666 [validation disabled]</li> <li>[Header checksum status: Unverified]</li> <li>Source: 10.0.2.15</li> <li>Destination: 193.136.19.254</li> </ul> </li> <li>User Datagram Protocol, Src Port: 38605, Dst Port: 33434 <ul style="list-style-type: none"> <li>Source Port: 38605</li> <li>Destination Port: 33434</li> <li>Length: 40</li> <li>Checksum: 0xe1ce [unverified]</li> <li>[Checksum Status: Unverified]</li> <li>[Stream index: 2]</li> <li>[Timestamps]</li> <li>Data (32 bytes)</li> </ul> </li> </ul>
--

Figura 2.1.8: Traceroute

Como podemos observar pela figura 2.1.8 nenhum protocolo de aplicação é utilizado na execução do comando traceroute. O protocolo de transporte é o UDP e a porta de atendimento a 33434 neste caso particular (podendo outras implementações do traceroute utilizar outras portas). Esta trama possui um overhead de transporte de  $(8/60) \times 100 \approx 13,33\%$

## Capítulo 3

# Conclusão

Ao realizarmos este trabalho prático, conseguimos visualizar o funcionamento dos vários tipos de serviço sobre a transferência de ficheiros numa rede, mais precisamente os seus protocolos de transporte, os desempenhos, o modo como estabelecem conexão com o servidor, a sua complexidade e também a sua segurança.

Com o conhecimento adquirido durante as aulas teóricas, consideramos ter alcançado todos os objetivos propostos neste trabalho prático.