

Universidade do Minho
Mestrado Integrado em Engenharia Informática



TP1: Protocolos da Camada de Transporte

Comunicações por Computador

Grupo 1 – PL5



Carla Cruz A80564



Ana Ribeiro A82474



Jéssica Lemos A82061

Questões e Resposta

1. Inclua no relatório uma tabela em que identifique, para cada comando executado, qual o protocolo de aplicação, o protocolo de transporte, porta de atendimento e overhead de transporte, como ilustrado no exemplo seguinte:

Comando usado (aplicação)	Protocolo de aplicação (se aplicável)	Protocolo de transporte (se aplicável)	Porta de atendimento (se aplicável)	Overhead de transporte em bytes (se aplicável)
ping	PING	-	-	-
tracert	TRACEROUTE	UDP	33437*	33.3 %
telnet	TELNET	TCP	23	33.3 %
ftp	FTP	TCP	21	43.48 %
Tftp	TFPT	UDP	69	27.78 %
Browser/http	HTTP	TCP	80	11.4 %
nslookup	DNS	UDP	53	3.8 %
ssh	SSHv2	TCP	22	5.7 %

*Este valor é apenas um exemplo de uma porta de atendimento, dado que esta contém um conjunto de portas.

• PING

25 5.013470	10.0.2.15	192.168.100.254	DNS	86 Standard query PTR 240.9.136.193.in-addr.arpa
26 5.016510	192.168.100.254	10.0.2.15	DNS	377 Standard query response PTR marco.uminho.pt
27 6.011392	10.0.2.15	193.136.9.240	ICMP	98 Echo (ping) request id=0x0bec, seq=7/1792, ttl=64
28 6.013751	193.136.9.240	10.0.2.15	ICMP	98 Echo (ping) reply id=0x0bec, seq=7/1792, ttl=61
29 6.014028	10.0.2.15	192.168.100.254	DNS	86 Standard query PTR 240.9.136.193.in-addr.arpa

Figura 1 - Captura de Tráfego Ping

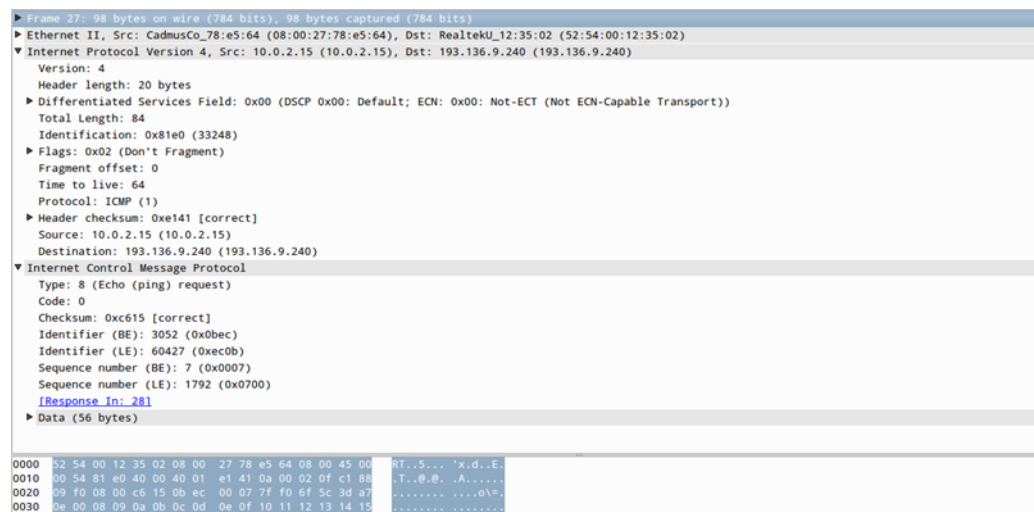


Figura 2 - Trama exemplo

Foi possível verificar que o protocolo de aplicação utilizado é o PING. Como este comando, trabalha diretamente com a camada de rede, o protocolo de transporte não é aplicável, pelo que não existe uma porta de atendimento. Para além disto, também não existe overhead de transporte.

• TRACEROUTE

11	0.006391	10.0.2.15	193.136.19.254	UDP	74	Source port: 59794	Destination port: 33436
12	0.006488	10.0.2.2	10.0.2.15	ICMP	70	Time-to-live exceeded (Time to live exceeded 1	
13	0.006537	10.0.2.15	193.136.19.254	UDP	74	Source port: 42359	Destination port: 33437
14	0.006644	10.0.2.15	193.136.19.254	UDP	74	Source port: 34743	Destination port: 33438
15	0.006751	10.0.2.15	193.136.19.254	UDP	74	Source port: 55741	Destination port: 33439

Figura 3 - Captura de Tráfego

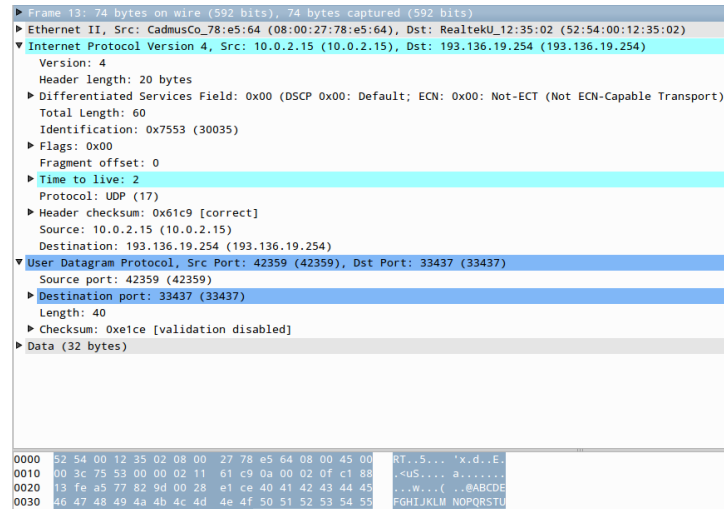


Figura 4 - Trama exemplo

O protocolo de aplicação utilizado foi o TRACEROUTE. No campo *Protocol* da Figura 4, podemos verificar que o protocolo de transporte utilizado foi o UDP. A porta de atendimento para a trama exemplo é a 33437, como é observado no campo DST Port. Contudo, existe um conjunto de portas de atendimento, como pode ser observado na Figura 3, onde as outras tramas recorrem a portas diferentes. Uma vez que o *Header length* é 20 bytes e o *Total length* é 60, concluímos que o overhead de transporte é $20/60 = 0.333$.

• BROWSER/HTTP

29	42.932146	10.0.2.15	193.136.9.240	HTTP	189	GET /disciplinas/CC-MIEI/ HTTP/1.1
41	42.995125	193.136.9.240	10.0.2.15	HTTP	1277	HTTP/1.1 200 OK (text/html)

Figura 6 - Captura de Tráfego HTTP

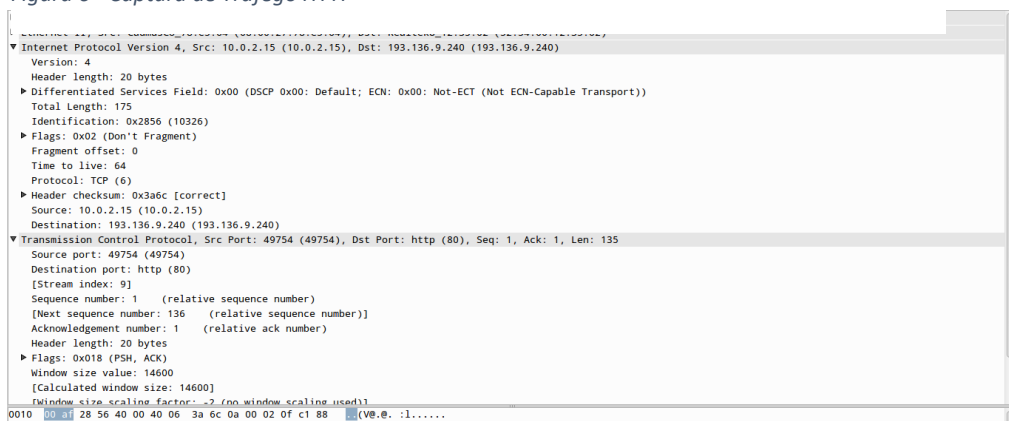


Figura 5 - Trama exemplo

Como podemos verificar pela Figura 5, o protocolo de aplicação é HTTP. O protocolo de transporte usado é o TCP, como verificamos no campo *Protocol*. A porta de atendimento é a 80, visualizada na *Dst Port*. Dado que a *Header length* é 20 bytes e o *Total Length* é 175, concluímos que o overhead é $20/175 = 0.114$.

• NSLOOKUP

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.2.15	192.168.100.254	DNS	73	Standard query A www.uminho.pt
2	0.001734	192.168.100.254	10.0.2.15	DNS	544	Standard query response A 193.137.9.114

Figura 7 - Captura de tráfego

▶ Frame 2: 544 bytes on wire (4352 bits), 544 bytes captured (4352 bits)	
▶ Ethernet II, Src: RealtekU_12:35:02 (52:54:00:12:35:02), Dst: CadmusCo_78:e5:64 (08:00:27:78:e5:64)	
▼ Internet Protocol Version 4, Src: 192.168.100.254 (192.168.100.254), Dst: 10.0.2.15 (10.0.2.15)	
Version: 4	
Header length: 20 bytes	
▶ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))	
Total Length: 530	
Identification: 0x033b (827)	
Flags: 0x00	
Fragment offset: 0	
Time to live: 64	
Protocol: UDP (17)	
▶ Header checksum: 0x43eb [correct]	
Source: 192.168.100.254 (192.168.100.254)	
Destination: 10.0.2.15 (10.0.2.15)	
▼ User Datagram Protocol, Src Port: domain (53), Dst Port: 37257 (37257)	
Source port: domain (53)	
Destination port: 37257 (37257)	
Length: 510	
▶ Checksum: 0x48b6 [validation disabled]	
▶ Domain Name System (response)	
0000 08 00 27 78 e5 64 52 54 00 12 35 02 08 00 45 00 ..'x.dRT..S... 0010 02 12 03 3b 00 00 40 11 43 eb c0 a8 64 fe 0a 00C...d... 0020 02 0f 00 35 91 89 01 fe 48 b6 e8 e0 81 80 00 01 ..5....H..... 0030 00 01 00 0a 00 0c 03 77 77 77 06 75 6d 69 6e 68w ww.uminh	

Figura 8 - Trama exemplo

Como podemos visualizar na figura 7, o protocolo de aplicação utilizado é o DNS. Na figura 8, verificamos no campo *Protocol* que o protocolo de transporte é o UDP. Constatamos que a porta de atendimento é a 53. Como o *Header length* é 20 e o *Total length* é 530, o overhead é $20/530 = 0.038$.

• FTP

Filter: ftp			Expression... Clear Apply			
No.	Time	Source	Destination	Protocol	Length	Info
6	0.022459	193.136.9.183	10.0.2.15	FTP	74	Response: 220 (vsFTPd 2.3.5)
10	9.186479	10.0.2.15	193.136.9.183	FTP	70	Request: USER anonymous
12	9.189208	193.136.9.183	10.0.2.15	FTP	88	Response: 331 Please specify the password.
14	12.141589	10.0.2.15	193.136.9.183	FTP	67	Request: PASS gr2018
16	12.154775	193.136.9.183	10.0.2.15	FTP	77	Response: 230 Login successful.
18	12.154994	10.0.2.15	193.136.9.183	FTP	60	Request: SYST
20	12.156199	193.136.9.183	10.0.2.15	FTP	73	Response: 215 UNIX Type: L8

Figura 9 - Captura de tráfego

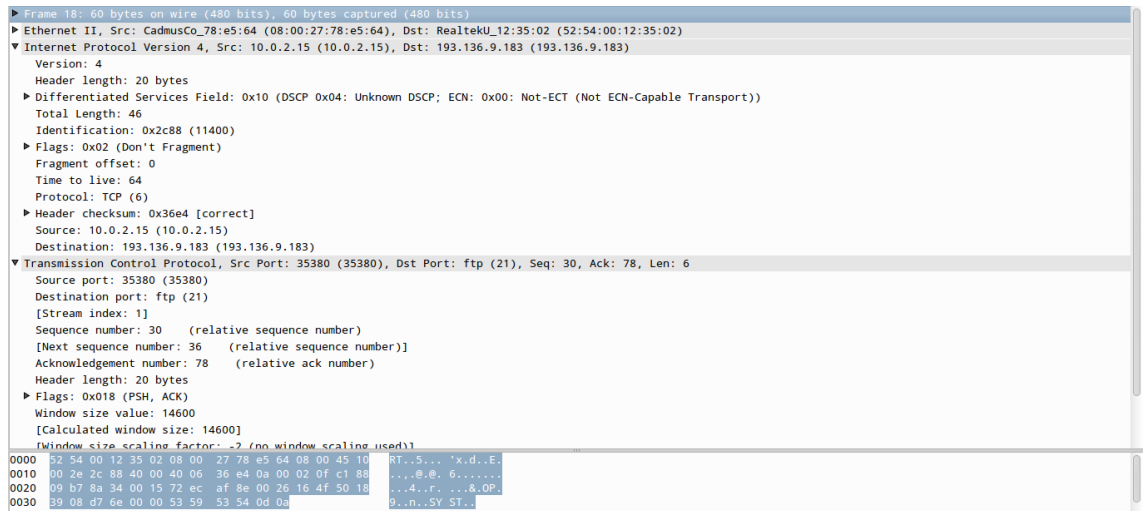


Figura 10 - Trama exemplo

Na Figura 9 verificamos que o protocolo de aplicação utilizado é o FTP. Posteriormente, na Figura 8 verificamos que o protocolo de transporte é o TCP. No *Dst Port* a porta de atendimento é a 21. O *Header length* é 20 e o *Total length* é 46, o overhead é $20/46 = 0.4348$.

- TELNET

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.2.15	192.168.100.254	DNS	75	Standard query AAAA gr2018.ddns.net
2	0.001606	192.168.100.254	10.0.2.15	DNS	135	Standard query response
3	0.001714	10.0.2.15	192.168.100.254	DNS	91	Standard query AAAA gr2018.ddns.net.sa.di.uminho.pt
4	0.003199	192.168.100.254	10.0.2.15	DNS	136	Standard query response, No such name
5	0.004212	10.0.2.15	192.168.100.254	DNS	75	Standard query A gr2018.ddns.net
6	2.055043	192.168.100.254	10.0.2.15	DNS	348	Standard query response A 193.136.9.183
7	2.055731	10.0.2.15	193.136.9.183	TCP	74	44785 > telnet [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=239449 TSecr=0 WS=16
8	3.055542	10.0.2.15	193.136.9.183	TCP	74	44785 > telnet [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=239699 TSecr=0 WS=16
9	5.010684	CadmusCo_78:e5:64	RealtekU_12:35:02	ARP	42	Who has 10.0.2.2? Tell 10.0.2.15
10	5.011180	RealtekU_12:35:02	CadmusCo_78:e5:64	ARP	60	10.0.2.2 is at 52:54:00:12:35:02
11	5.058746	10.0.2.15	193.136.9.183	TCP	74	44785 > telnet [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=240200 TSecr=0 WS=16
12	9.067867	10.0.2.15	193.136.9.183	TCP	74	44785 > telnet [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=241202 TSecr=0 WS=16
13	17.090673	10.0.2.15	193.136.9.183	TCP	74	44785 > telnet [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=243208 TSecr=0 WS=16
14	33.123839	10.0.2.15	193.136.9.183	TCP	74	44785 > telnet [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=247216 TSecr=0 WS=16

Figura 11 - Captura de tráfego

13 17.090673 10.0.2.15 193.136.9.183 TCP 74 44785 > telnet [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=243208 TSecr=0 WS=16			
▶ Frame 13: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)			
▼ Ethernet II, Src: CadmusCo_78:e5:64 (08:00:27:78:e5:64), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)			
▶ Destination: RealtekU_12:35:02 (52:54:00:12:35:02)			
▶ Source: CadmusCo_78:e5:64 (08:00:27:78:e5:64)			
Type: IP (0x0800)			
▼ Internet Protocol Version 4, Src: 10.0.2.15 (10.0.2.15), Dst: 193.136.9.183 (193.136.9.183)			
Version: 4			
Header length: 20 bytes			
▶ Differentiated Services Field: 0x10 (DSCP 0x04: Unknown DSCP; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))			
Total Length: 60			
Identification: 0x8697 (34455)			
▶ Flags: 0x02 (Don't Fragment)			
Fragment offset: 0			
Time to live: 64			
Protocol: TCP (6)			
▶ Header checksum: 0xdcc6 [correct]			
Source: 10.0.2.15 (10.0.2.15)			
Destination: 193.136.9.183 (193.136.9.183)			
▶ Transmission Control Protocol, Src Port: 44785 (44785), Dst Port: telnet (23), Seq: 0, Len: 0			
0000	52 54 00 12 35 02 08 00	27 78 e5 64 08 00 45 10	81..S... "x.d...E..
0010	00 3c 86 97 40 00 40 06	dc c6 0a 00 02 0f c1 88	..<..E..B..
0020	09 b7 ae f1 00 17 d1 5e	21 ca 00 00 00 00 a0 02^
0030	39 08 d7 7c 00 00 02 04	05 b4 04 02 08 0a 00 03	9..

Figura 12 - Trama exemplo

O protocolo de aplicação utilizado é o TELNET. Na Figura 12 o campo *Protocol* indica-nos que o protocolo de transporte é o TCP. Com o *Dst Port* verificamos que a porta de atendimento é a 23. O overhead é de 0.333 pois o *Header length* é 20 e o *Total length* é 60.

- SSH

Filter: ssh			Expression... Clear Apply			
No.	Time	Source	Destination	Protocol	Length	Info
10	2.161478	193.136.9.183	10.0.2.15	SSHv2	95	Server: Protocol: SSH-2.0-OpenSSH_5.9p1 Debian-Subuntu1.4\r
12	2.162254	10.0.2.15	193.136.9.183	SSHv2	96	Client: Protocol: SSH-2.0-OpenSSH_5.9p1 Debian-Subuntu1.10\r
14	2.166162	193.136.9.183	10.0.2.15	SSHv2	1038	Server: Key Exchange Init
15	2.168261	10.0.2.15	193.136.9.183	SSHv2	1326	Client: Key Exchange Init
17	2.174363	10.0.2.15	193.136.9.183	SSHv2	134	Client: Diffie-Hellman Key Exchange Init
19	2.187444	193.136.9.183	10.0.2.15	SSHv2	366	Server: New Keys
21	17.086424	10.0.2.15	193.136.9.183	SSHv2	70	Client: New Keys

Figura 13 - Captura de tráfego

19 2.187444 193.136.9.183 10.0.2.15 SSHv2 366 Server: New Keys			
▶ Frame 19: 366 bytes on wire (2928 bits), 366 bytes captured (2928 bits)			
▶ Ethernet II, Src: RealtekU_12:35:02 (52:54:00:12:35:02), Dst: CadmusCo_78:e5:64 (08:00:27:78:e5:64)			
▼ Internet Protocol Version 4, Src: 193.136.9.183 (193.136.9.183), Dst: 10.0.2.15 (10.0.2.15)			
Version: 4			
Header length: 20 bytes			
▶ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))			
Total Length: 352			
Identification: 0x032b (811)			
▶ Flags: 0x00			
Fragment offset: 0			
Time to live: 64			
Protocol: TCP (6)			
▶ Header checksum: 0x9f1f [correct]			
Source: 193.136.9.183 (193.136.9.183)			
Destination: 10.0.2.15 (10.0.2.15)			
▼ Transmission Control Protocol, Src Port: ssh (22), Dst Port: 59898 (59898), Seq: 1026, Ack: 1395, Len: 312			
Source port: ssh (22)			
Destination port: 59898 (59898)			
[Stream index: 3]			
Sequence number: 1026 (relative sequence number)			
[Next sequence number: 1338 (relative sequence number)]			
Acknowledgement number: 1395 (relative ack number)			
Header length: 20 bytes			
▶ Flags: 0x018 (PSH, ACK)			
Window size value: 65535			
[Calculated window size: 65535]			

Figura 14 - Trama exemplo

Constatado pela captura de tráfego realizada, SSHv2 é o protocolo de aplicação utilizado. Foi verificado que o protocolo de transporte é o TCP e que a porta de atendimento é a 22. Dado que o *Header length* é 20 e o *Total length* é 352, o overhead é $20/352 = 0.057$.

- TFTP

7 0.058501	10.0.2.15	193.136.9.183	TFTP	86 Read Request, File: file1, Transfer type: octet, tsize\000=0\000, blksize\000=512\000, timeout\000=6
8 6.066196	10.0.2.15	193.136.9.183	TFTP	86 Read Request, File: file1, Transfer type: octet, tsize\000=0\000, blksize\000=512\000, timeout\000=6
11 12.073931	10.0.2.15	193.136.9.183	TFTP	86 Read Request, File: file1, Transfer type: octet, tsize\000=0\000, blksize\000=512\000, timeout\000=6
12 18.086232	10.0.2.15	193.136.9.183	TFTP	86 Read Request, File: file1, Transfer type: octet, tsize\000=0\000, blksize\000=512\000, timeout\000=6
13 24.093027	10.0.2.15	193.136.9.183	TFTP	86 Read Request, File: file1, Transfer type: octet, tsize\000=0\000, blksize\000=512\000, timeout\000=6
14 30.099618	10.0.2.15	193.136.9.183	TFTP	86 Read Request, File: file1, Transfer type: octet, tsize\000=0\000, blksize\000=512\000, timeout\000=6

Figura 15 - Captura de tráfego TFTP

▶ Frame 13: 86 bytes on wire (688 bits), 86 bytes captured (688 bits)
▶ Ethernet II, Src: CadmusCo_78:e5:64 (08:00:27:78:e5:64), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
▼ Internet Protocol Version 4, Src: 10.0.2.15 (10.0.2.15), Dst: 193.136.9.183 (193.136.9.183)
Version: 4
Header length: 20 bytes
▶ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
Total Length: 72
Identification: 0x9cd1 (40145)
Flags: 0x02 (Don't Fragment)
Fragment offset: 0
Time to live: 64
Protocol: UDP (17)
▶ Header checksum: 0xc685 [correct]
Source: 10.0.2.15 (10.0.2.15)
Destination: 193.136.9.183 (193.136.9.183)
▼ User Datagram Protocol, Src Port: 35409 (35409), Dst Port: tftp (69)
Source port: 35409 (35409)
Destination port: tftp (69)
Length: 52
▶ Checksum: 0xd793 [validation disabled]
▶ Trivial File Transfer Protocol

Figura 16 - Trama exemplo

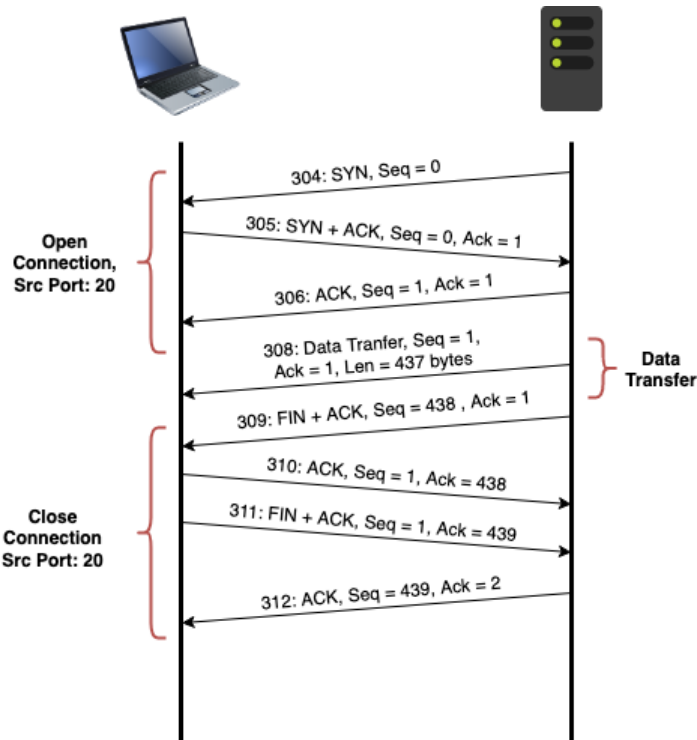
Na Figura 15, podemos constatar que o protocolo de aplicação é o TFTP. Através do campo *Protocol* na Figura 16, verificamos que o protocolo de transporte usado foi o UDP. A porta de atendimento é a 69. O overhead é obtido fazendo $20/72 = 0.2778$.

2. Uma representação num diagrama temporal das transferências da file1 por FTP e TFTP respetivamente. Se for caso disso, identifique as fases de estabelecimento de conexão, transferência de dados e fim de conexão. Identifica também claramente os tipos de segmentos trocados e os números de sequência usados quer nos dados como nas confirmações.

- FTP

303 429.552422	10.4.4.1	10.1.1.1	FTP	78 Request: RETR file1
304 429.552540	10.1.1.1	10.4.4.1	TCP	74 ftp-data > 40045 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=161156 TSecr=0 WS=16
305 429.552695	10.4.4.1	10.1.1.1	TCP	74 40045 > ftp-data [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PERM=1 TSval=161156 TSecr=16
306 429.552828	10.1.1.1	10.4.4.1	TCP	66 ftp-data > 40045 [ACK] Seq=1 Ack=1 Win=14608 Len=0 TSval=161156 TSecr=161156
307 429.552872	10.1.1.1	10.4.4.1	FTP	130 Response: 150 Opening BINARY mode data connection for file1 (437 bytes).
308 429.552880	10.1.1.1	10.4.4.1	FTP-DAT	503 FTP Data: 437 bytes
309 429.552881	10.1.1.1	10.4.4.1	TCP	66 ftp-data > 40045 [FIN, ACK] Seq=438 Ack=1 Win=14608 Len=0 TSval=161156 TSecr=161156
310 429.553114	10.4.4.1	10.1.1.1	TCP	66 40045 > ftp-data [ACK] Seq=1 Ack=438 Win=15552 Len=0 TSval=161156 TSecr=161156
311 429.553240	10.4.4.1	10.1.1.1	TCP	66 40045 > ftp-data [FIN, ACK] Seq=1 Ack=439 Win=15552 Len=0 TSval=161156 TSecr=161156
312 429.553547	10.1.1.1	10.4.4.1	TCP	66 ftp-data > 40045 [ACK] Seq=439 Ack=2 Win=14608 Len=0 TSval=161156 TSecr=161156

Figura 17 - Captura de tráfego

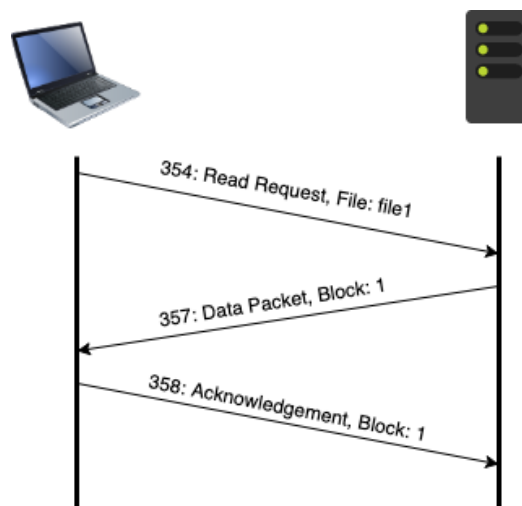


Inicialmente o servidor começa a conexão através da porta 20, enviando um pacote SYN cujo número de sequência é 0. O utilizador responde com um SYN+ACK. De seguida, o servidor envia um ACK de volta ao utilizador. Após enviar os dados pretendidos, o servidor inicia a fase de encerramento de conexão, que contém quatro etapas. É importante realçar que a flag FIN indica que se pretende terminar a conexão, ficando à espera de uma resposta ACK. Tal pode ser verificado através do diagrama temporal apresentado que foi construído com base na captura de tráfego observada na Figura 17.

- TFTP

354	598.175626	10.4.4.1	10.1.1.1	TFTP	56 Read Request, File: file1, Transfer type: octet
357	598.176112	10.1.1.1	10.4.4.1	TFTP	483 Data Packet, Block: 1 (last)
358	598.176327	10.4.4.1	10.1.1.1	TFTP	46 Acknowledgement, Block: 1

Figura 18 - Captura de tráfego

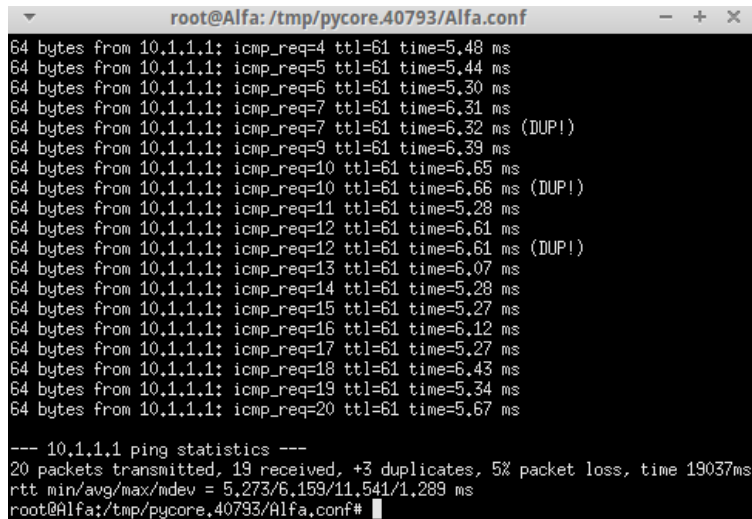


O utilizador envia um Read Request para o servidor que irá conter diversas informações, entre as quais o nome do ficheiro. O servidor irá responder com um pacote de dados. Após a receção dos dados por parte do utilizador este irá enviar um pacote ACK.

3. Com base nas experiências realizadas, distinga e compare sucintamente as quatro aplicações de transferência de ficheiros que usou nos seguintes pontos (i) uso da camada de transporte; (ii) eficiência na transferência; (iii) complexidade; (iv) segurança;

	SFTP	FTP	TFTP	HTTP
Uso da camada de transporte	Protocolo TCP	Protocolo TCP	Protocolo UDP	Protocolo TCP
Eficiência na transferência	A fiabilidade da transferência de dados compromete a sua eficiência.	Fiável o que conduz a um maior overhead.	Não se responsabiliza pela entrega dos dados, pelo que possui um menor overhead.	Um protocolo com uma grande eficiência.
Complexidade	Como disponibiliza inúmeras funcionalidades, é muito complexo.	Complexo, dado que garante segurança na transferência de dados.	O protocolo de transporte utilizado é o UDP e as funcionalidades existentes são mais reduzidas. Sendo assim, é uma versão simplificada do FTP.	Pouco complexo.
Segurança	Seguro, pois recorre à autenticação e à encriptação dos dados.	Utiliza autenticação, contudo é pouco seguro.	Sem mecanismos de autenticação ou encriptação de dados, sendo pouco seguro.	Nesta versão é pouco seguro, uma vez que não utiliza encriptação. Apenas recorre à autenticação.

4. As características das ligações de rede têm uma enorme influência nos níveis de Transporte e de Aplicação. Discuta, relacionando a resposta com as experiências realizadas, as influências das situações de perda ou duplicação de pacotes IP no desempenho global de Aplicações fiáveis (se possível, relacionando com alguns dos mecanismos de transporte envolvidos).



```

root@Alfa: /tmp/pycore.40793/Alfa.conf
64 bytes from 10.1.1.1: icmp_req=4 ttl=61 time=5.48 ms
64 bytes from 10.1.1.1: icmp_req=5 ttl=61 time=5.44 ms
64 bytes from 10.1.1.1: icmp_req=6 ttl=61 time=5.30 ms
64 bytes from 10.1.1.1: icmp_req=7 ttl=61 time=6.31 ms
64 bytes from 10.1.1.1: icmp_req=7 ttl=61 time=6.32 ms (DUPL)
64 bytes from 10.1.1.1: icmp_req=9 ttl=61 time=6.39 ms
64 bytes from 10.1.1.1: icmp_req=10 ttl=61 time=6.65 ms
64 bytes from 10.1.1.1: icmp_req=10 ttl=61 time=6.66 ms (DUPL)
64 bytes from 10.1.1.1: icmp_req=11 ttl=61 time=5.28 ms
64 bytes from 10.1.1.1: icmp_req=12 ttl=61 time=6.61 ms
64 bytes from 10.1.1.1: icmp_req=12 ttl=61 time=6.61 ms (DUPL)
64 bytes from 10.1.1.1: icmp_req=13 ttl=61 time=6.07 ms
64 bytes from 10.1.1.1: icmp_req=14 ttl=61 time=5.28 ms
64 bytes from 10.1.1.1: icmp_req=15 ttl=61 time=5.27 ms
64 bytes from 10.1.1.1: icmp_req=16 ttl=61 time=6.12 ms
64 bytes from 10.1.1.1: icmp_req=17 ttl=61 time=5.27 ms
64 bytes from 10.1.1.1: icmp_req=18 ttl=61 time=6.43 ms
64 bytes from 10.1.1.1: icmp_req=19 ttl=61 time=5.34 ms
64 bytes from 10.1.1.1: icmp_req=20 ttl=61 time=5.67 ms

--- 10.1.1.1 ping statistics ---
20 packets transmitted, 19 received, +3 duplicates, 5% packet loss, time 19037ms
rtt min/avg/max/mdev = 5.273/6.159/11.541/1.289 ms
root@Alfa: /tmp/pycore.40793/Alfa.conf#

```

Figura 19 - Ping

Como podemos observar na Figura 19, no ping realizado na LAN3, 5% dos pacotes foram perdidos e 3 foram duplicados. Tal deve-se ao facto de existirem problemas ao nível da ligação de rede. Os protocolos de transporte UDP e TCP tratam esta situação de formas distintas. O primeiro não dispõe de mecanismos de deteção e recuperação de perdas de pacotes, pelo que se forem perdidos, os protocolos que correm em cima do UDP poderão identificar e corrigir estas falhas. Para tal poderá ser atribuído um número de identificação ao pacote de modo a verificar a sua receção. Ao invés, o TCP possui tais mecanismos de deteção e recuperação, assim caso haja perda de pacote esse será retransmitido. É importante realçar que o TCP garante que todos os pacotes são entregues.

Em suma, aquando do uso do protocolo TCP verifica-se o reenvio de pacotes enquanto não chegarem ao seu destino, o que não se verifica no caso do UDP dado que os pacotes podem ser perdidos. Assim, no caso do host alfa apresentado na Figura 19 os pacotes podem ser recuperados através do reenvio no caso do TCP.

Conclusões

Neste trabalho, foi possível estudar o modo como as diversas aplicações recorrem aos serviços da camada inferior. Para tal, observámos os diferentes protocolos quer de aplicação quer de transporte utilizados pelas mesmas, bem como as portas de atendimento usadas e o overhead associado ao transporte.

Numa segunda fase do trabalho realizámos a instalação, configuração e utilização de serviços de transferência de ficheiros. Assim, foi possível transferir um mesmo ficheiro recorrendo a quatro serviços diferentes, nomeadamente SFTP, FTP, TFTP e HTTP. Elaborámos um diagrama temporal para o FTP e TFTP, identificando as fases de estabelecimento de conexão, transferência de dados e fim de conexão. De seguida, analisámos as diferentes características de cada um. Por fim, verificámos o modo como o TCP e o UDP lidam com a perda e duplicação de pacotes.

Em última instância, consideramos este trabalho importante, na medida em que nos permitiu consolidar os conhecimentos a cerca da camada de transporte.