

Comunicações por computador

Grupo 5 - PL1

16 de março de 2021

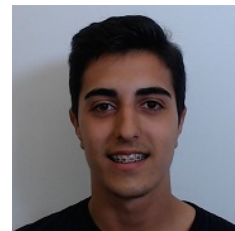
TP1: Protocolos da Camada de Transporte



Luís Martins (A89600)



Renata Teixeira (A89611)



Tiago Alves (A89554)

1 Perguntas

1.1 Pergunta 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 (se aplicável)	Protocolo de Aplicação (se aplicável)	Protocolo de Transporte (se aplicável)	Porta de Atendimento (se aplicável)	Overhead de Transporte em Bytes (se aplicável)
ping	DNS	UDP	53	8
tracert	TRACEROUTE	UDP	33446	8
telnet	TELNET	TCP	23	20
ftp	FTP	TCP	21	20
tftp	TFTP	UDP	69	8
browser / http	HTTP	TCP	80	20
nslookup	DNS	UDP	53	8
ssh	SSH	TCP	22	20

- **ping:** No print que se segue, podemos ver que se usa UDP (User Datagram Protocol), que o protocolo da aplicação é DNS e que a porta de atendimento é a 53. Visto que o protocolo de transporte é UDP, podemos concluir que o overhead tem 8 bytes de tamanho.

1 0.000000000	10.0.2.15	8.8.8.8	DNS	89 Standard query 0xab14 A cisco.di.uminho.pt OPT
2 0.000124758	10.0.2.15	8.8.8.8	DNS	89 Standard query 0x9b0a AAAA cisco.di.uminho.pt OPT
3 0.008172862	8.8.8.8	10.0.2.15	DNS	138 Standard query response 0x9b0a AAAA cisco.di.uminho.pt SOA dns.di.uminho...
4 0.178085217	8.8.8.8	10.0.2.15	DNS	105 Standard query response 0xa514 A cisco.di.uminho.pt A 193.136.19.254 OPT
5 0.178433456	10.0.2.15	193.136.19.254	ICMP	98 Echo (ping) request id=0x0006, seq=1/256, ttl=64 (reply in 6)
6 0.192436288	193.136.19.254	10.0.2.15	ICMP	98 Echo (ping) reply id=0x0006, seq=1/256, ttl=243 (request in 5)
7 0.192776529	10.0.2.15	8.8.8.8	DNS	98 Standard query 0xf672 PTR 254.19.136.193.in-addr.arpa OPT
8 0.272763893	8.8.8.8	10.0.2.15	DNS	139 Standard query response 0xf672 PTR 254.19.136.193.in-addr.arpa PTR cisco...
9 1.181336050	10.0.2.15	193.136.19.254	ICMP	98 Echo (ping) request id=0x0006, seq=2/512, ttl=64 (reply in 10)
10 1.195788937	193.136.19.254	10.0.2.15	ICMP	98 Echo (ping) reply id=0x0006, seq=2/512, ttl=243 (request in 9)
11 2.183850018	10.0.2.15	193.136.19.254	ICMP	98 Echo (ping) request id=0x0006, seq=3/768, ttl=64 (reply in 12)
12 2.198059508	193.136.19.254	10.0.2.15	ICMP	98 Echo (ping) reply id=0x0006, seq=3/768, ttl=243 (request in 11)
13 3.184317587	10.0.2.15	193.136.19.254	ICMP	98 Echo (ping) request id=0x0006, seq=4/1024, ttl=64 (reply in 14)
14 3.198612170	193.136.19.254	10.0.2.15	ICMP	98 Echo (ping) reply id=0x0006, seq=4/1024, ttl=243 (request in 13)
15 4.186260196	10.0.2.15	193.136.19.254	ICMP	98 Echo (ping) request id=0x0006, seq=5/1280, ttl=64 (reply in 16)
16 4.200535238	193.136.19.254	10.0.2.15	ICMP	98 Echo (ping) reply id=0x0006, seq=5/1280, ttl=243 (request in 15)

▶ Frame 7: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface enp0s3, id 0

▶ Ethernet II, Src: PcsCompu_d1:8b:d0 (08:00:27:d1:8b:d0), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)

▶ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 8.8.8.8

▼ User Datagram Protocol, Src Port: 38105, Dst Port: 53

Source Port: 38105

Destination Port: 53

Length: 64

Checksum: 0x1c70 [unverified]

[Checksum Status: Unverified]

[Stream index: 2]

[Timestamps]

▼ Domain Name System (query)

Transaction ID: 0xf672

Flags: 0x0100 Standard query

Questions: 1

Answer RRs: 0

Authority RRs: 0

Additional RRs: 1

Queries

▶ Additional records

[\[Response In: 8\]](#)

- **tracert:** Tal como no comando *ping*, podemos ver, pelo print abaixo, que o comando *tracert* também utiliza como Protocolo de transporte o UDP, tendo, por isso, o overhead tamanho 8. Também podemos ver pela figura que, para o pacote que escolhemos, o destination port é o 33446.

20	0.108486119	10.0.2.15	193.136.19.254	UDP	74	55805 → 33446	Len=32
21	0.108534777	10.0.2.15	193.136.19.254	UDP	74	34406 → 33447	Len=32
22	0.108585046	10.0.2.15	193.136.19.254	UDP	74	60836 → 33448	Len=32
23	0.108626549	10.0.2.15	193.136.19.254	UDP	74	57145 → 33449	Len=32
32	10.116845485	10.0.2.15	193.136.19.254	UDP	74	37255 → 33450	Len=32
33	10.116876303	10.0.2.15	193.136.19.254	UDP	74	56184 → 33451	Len=32
34	10.116937037	10.0.2.15	193.136.19.254	UDP	74	48096 → 33452	Len=32

▶	Frame 20: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface enp0s3, id 0
▶	Ethernet II, Src: PcsCompu_d1:8b:d0 (08:00:27:d1:8b:d0), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
▶	Internet Protocol Version 4, Src: 10.0.2.15, Dst: 193.136.19.254
▼	User Datagram Protocol, Src Port: 55805, Dst Port: 33446
▶	Source Port: 55805
▶	Destination Port: 33446
▶	Length: 40
▶	Checksum: 0xe1ce [unverified]
▶	[Checksum Status: Unverified]
▶	[Stream index: 14]
▶	[Timestamps]
▶	Data (32 bytes)

- **telnet:** Como podemos ver pelo print, o comando *telnet* tem como protocolo de transporte o TCP (Transmission Control Protocol), neste caso, a porta de atendimento é a 23 e que o *overhead*(header length) é 20 bytes.

21	0.136224617	10.0.2.15	193.136.9.183	TCP	54	49580 → 23 [ACK]	Seq=136 Ack=58 Win=64183 Len=0
22	0.136322198	10.0.2.15	193.136.9.183	TELNET	57	Telnet Data ...	
23	0.136431175	193.136.9.183	10.0.2.15	TCP	60	23 → 49580 [ACK]	Seq=58 Ack=139 Win=65535 Len=0
24	0.149391952	193.136.9.183	10.0.2.15	TELNET	91	Telnet Data ...	
25	0.189854564	10.0.2.15	193.136.9.183	TCP	54	49580 → 23 [ACK]	Seq=139 Ack=95 Win=64146 Len=0
26	2.042896159	10.0.2.15	193.136.9.183	TELNET	55	Telnet Data ...	
27	2.043044104	193.136.9.183	10.0.2.15	TCP	60	23 → 49580 [ACK]	Seq=95 Ack=140 Win=65535 Len=0

▶	Frame 24: 91 bytes on wire (728 bits), 91 bytes captured (728 bits) on interface enp0s3, id 0
▶	Ethernet II, Src: RealtekU_12:35:02 (52:54:00:12:35:02), Dst: PcsCompu_d1:8b:d0 (08:00:27:d1:8b:d0)
▶	Internet Protocol Version 4, Src: 193.136.9.183, Dst: 10.0.2.15
▼	Transmission Control Protocol, Src Port: 23, Dst Port: 49580, Seq: 58, Ack: 139, Len: 37
▶	Source Port: 23
▶	Destination Port: 49580
▶	[Stream index: 0]
▶	[TCP Segment Len: 37]
▶	Sequence number: 58 (relative sequence number)
▶	Sequence number (raw): 638720059
▶	[Next sequence number: 95 (relative sequence number)]
▶	Acknowledgment number: 139 (relative ack number)
▶	Acknowledgment number (raw): 899015754
▶	0x01 ... Header Length: 20 bytes (5)
▼	Flags: 0x018 (PSH, ACK)
▶	000. = Reserved: Not set
▶	...0 = Nonce: Not set
▶0... = Congestion Window Reduced (CWR): Not set
▶0. = ECN-Echo: Not set
▶0. = Urgent: Not set
▶1. = Acknowledgment: Set
▶1. = Push: Set
▶0. = Reset: Not set
▶0. = Syn: Not set
▶0. = Fin: Not set
▶	[TCP Flags:AP...]
▶	Window size value: 65535
▶	[Calculated window size: 65535]
▶	[Window size scaling factor: -2 (no window scaling used)]
▶	Checksum: 0xbad7 [unverified]
▶	[Checksum Status: Unverified]
▶	Urgent pointer: 0
▶	[SEQ/ACK analysis]
▶	[Timestamps]
▶	TCP payload (37 bytes)
▶	Telnet

- **ftp:** Como podemos ver pelo print, o comando *ftp* tem como protocolo de transporte

o TCP (Transmission Control Protocol), neste caso a porta de atendimento é a 21 e que o *overhead*(header length) é 20 bytes.

78	31.731940395	193.136.9.183	10.0.2.15	TCP	60	21 → 44448 [ACK] Seq=55 Ack=23 Win=65535 Len=0
79	31.822553598	193.136.9.183	10.0.2.15	FTP	77	Response: 230 Login successful.
80	31.822570315	10.0.2.15	193.136.9.183	TCP	54	44448 → 21 [ACK] Seq=23 Ack=78 Win=64163 Len=0
81	31.822805732	10.0.2.15	193.136.9.183	FTP	60	Request: SYST
82	31.822958727	193.136.9.183	10.0.2.15	TCP	60	21 → 44448 [ACK] Seq=78 Ack=29 Win=65535 Len=0
83	31.836977030	193.136.9.183	10.0.2.15	FTP	73	Response: 215 UNIX Type: L8
84	31.837012579	10.0.2.15	193.136.9.183	TCP	54	44448 → 21 [ACK] Seq=29 Ack=97 Win=64144 Len=0
85	31.837226892	10.0.2.15	193.136.9.183	FTP	60	Request: FEAT
86	31.837377737	193.136.9.183	10.0.2.15	TCP	60	21 → 44448 [ACK] Seq=97 Ack=35 Win=65535 Len=0


```

Frame 83: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface enp0s3, id 0
Ethernet II, Src: RealtekU_12:35:02 (52:54:00:12:35:02), Dst: PcsCompu_d1:8b:d0 (08:00:27:d1:8b:d0)
Internet Protocol Version 4, Src: 193.136.9.183, Dst: 10.0.2.15
Transmission Control Protocol, Src Port: 21, Dst Port: 44448, Seq: 78, Ack: 29, Len: 19
  Source Port: 21
  Destination Port: 44448
  [Stream index: 6]
  [TCP Segment Len: 19]
  Sequence number: 78 (relative sequence number)
  Sequence number (raw): 523712079
  [Next sequence number: 97 (relative sequence number)]
  Acknowledgment number: 29 (relative ack number)
  Acknowledgment number (raw): 3208135642
  0101 .... = Header Length: 20 bytes (5)
  Flags: 0x018 (PSH, ACK)
    000. .... = Reserved: Not set
    ...0 .... = Nonce: Not set
    ....0... = Congestion Window Reduced (CWR): Not set
    ....0... = ECN-Echo: Not set
    ....0... = Urgent: Not set
    ....1... = Acknowledgment: Set
    ....1... = Push: Set
    ....0... = Reset: Not set
    ....0... = Syn: Not set
    ....0... = Fin: Not set
    [TCP Flags: .....AP...]
  Window size value: 65535
  [Calculated window size: 65535]
  [Window size scaling factor: -2 (no window scaling used)]
  Checksum: 0xbacc [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
  [SEQ/ACK analysis]
  [Timestamps]
  TCP payload (19 bytes)
  File Transfer Protocol (FTP)
    215 UNIX Type: L8\r\n
      Response code: NAME system type (215)
      Response arg: UNIX Type: L8
      [Current working directory: ]

```

- **tftp**: Como podemos verificar na figura seguinte, o comando *tftp* tem como protocolo de transporte o UDP, por isso tem overhead de 8 bytes e também podemos ver que a porta de atendimento é a 69.

Como estavamos a ter problemas com o uso deste protocolo na máquina virtual fornecida, decidimos, então, testar o *tftp* na topologia core.

1427	1838.156179.	10.4.4.1	10.1.1.1	TFTP	56	Read Request, File: file1, Transfer type: octet
1428	1838.156861.	10.1.1.1	10.4.4.1	TFTP	276	Data Packet, Block: 1 (last)
1429	1838.157076.	10.4.4.1	10.1.1.1	TFTP	46	Acknowledgement, Block: 1


```

Frame 1427: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface veth1.2.dd, id 0
Ethernet II, Src: 00:00:00:aa:00:10 (00:00:00:aa:00:10), Dst: 00:00:00:aa:00:14 (00:00:00:aa:00:14)
Internet Protocol Version 4, Src: 10.4.4.1, Dst: 10.1.1.1
User Datagram Protocol, Src Port: 46800, Dst Port: 69
  Source Port: 46800
  Destination Port: 69
  Length: 22
  Checksum: 0xd40c [unverified]
  [Checksum Status: Unverified]
  [Stream index: 2]
  [Timestamps]
  Trivial File Transfer Protocol
    Opcode: Read Request (1)
    Source File: file1
    Type: octet

```

- **browser/http**: Através da seguinte figura, podemos observar que o comando *http* tem como protocolo de transporte o TCP. Neste caso, a porta de atendimento é a 80 e o overhead de transporte é de 20 bytes.

```

+ 199 16.383507473 10.0.2.15 193.136.9.240 HTTP 416 GET /disciplinas/CC-MIEI/ HTTP/1.1
200 16.383686176 193.136.9.240 10.0.2.15 TCP 60 80 → 35852 [ACK] Seq=1 Ack=363 Win=65535 Len=0
201 16.398510892 193.136.9.240 10.0.2.15 TCP 2974 80 → 35852 [ACK] Seq=1 Ack=363 Win=65535 Len=2920 [TC
202 16.398528712 10.0.2.15 193.136.9.240 TCP 54 35852 → 80 [ACK] Seq=363 Ack=2921 Win=62780 Len=0
203 16.398899947 193.136.9.240 10.0.2.15 TCP 1502 80 → 35852 [PSH, ACK] Seq=2921 Ack=363 Win=65535 Len=
204 16.398910161 10.0.2.15 193.136.9.240 TCP 54 35852 → 80 [ACK] Seq=363 Ack=4369 Win=62780 Len=0

+ Frame 199: 416 bytes on wire (3328 bits), 416 bytes captured (3328 bits) on interface enp0s3, id 0
+ Ethernet II, Src: PcsCompu_d1:8b:d0 (08:00:27:d1:8b:d0), 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
+ Transmission Control Protocol, Src Port: 35852, Dst Port: 80, Seq: 1, Ack: 1, Len: 362
  Source Port: 35852
  Destination Port: 80
  [Stream index: 15]
  [TCP Segment Len: 362]
  Sequence number: 1 (relative sequence number)
  Sequence number (raw): 1851164500
  [Next sequence number: 363 (relative sequence number)]
  Acknowledgment number: 1 (relative ack number)
  Acknowledgment number (raw): 315648002
  0101 .... = Header Length: 20 bytes (5)
  Flags: 0x018 (PSH, ACK)
    000. .... = Reserved: Not set
    ...0 .... = Nonce: Not set
    ....0... = Congestion Window Reduced (CWR): Not set
    ....0.. = ECN-Echo: Not set
    ....0. .... = Urgent: Not set
    ....0. .... = Acknowledgment: Set
    ....1... = Push: Set
    ....1... = Reset: Not set
    ....10... = Syn: Not set
    ....10... = Fin: Not set
    [TCP Flags: .....AP...]
  Window size value: 64240
  [Calculated window size: 64240]
  [Window size scaling factor: -2 (no window scaling used)]
  Checksum: 0xd90b [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
  [SEQ/ACK analysis]
  [Timestamps]
  TCP payload (362 bytes)
+ Hypertext Transfer Protocol
+ GET /disciplinas/CC-MIEI/ HTTP/1.1\r\n
  Host: marco.uminho.pt\r\n
  User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:86.0) Gecko/20100101 Firefox/86.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
  Accept-Language: en-US,en;q=0.5\r\n
  Accept-Encoding: gzip, deflate\r\n
  DNT: 1\r\n
  Connection: keep-alive\r\n
  Upgrade-Insecure-Requests: 1\r\n
  \r\n
  [Full request URI: http://marco.uminho.pt/disciplinas/CC-MIEI/]
  [HTTP request 1/2]
  [Response in frame: 209]
  [Next request in frame: 211]

```

- **nslookup**: A partir da análise da figura seguinte, podemos verificar que o comando *nslookup* tem como protocolo de aplicação o DNS, como protocolo de transporte o UDP, ou seja 8 bytes de overhead de transporte. E a porta de atendimento é a 53.

1	0.808000000	10.0.2.15	8.8.8.8	DNS	80 Standard query 0x6f33 A cc2021.ddns.net OPT
2	0.808000000	10.0.2.15	8.8.8.8	DNS	1028 Standard query response 0x6f33 A cc2021.ddns.net A 193.136.9.183 OPT
3	0.804199583	10.0.2.15	8.8.8.8	DNS	80 Standard query 0x34c2 AAAA cc2021.ddns.net OPT
4	0.807809439	8.8.8.8	10.0.2.15	DNS	146 Standard query response 0x34c2 AAAA cc2021.ddns.net SOA nf1.no-ip.com OPT
5	5.109984453	PcsCompu_d1:8b:d0	RealtekU_12:35:02	ARP	42 Who has 10.0.2.2? Tell 10.0.2.15
6	5.110263423	RealtekU_12:35:02	PcsCompu_d1:8b:d0	ARP	60 10.0.2.2 is at 52:54:00:12:35:02

```

+ Frame 2: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface enp0s3, id 0
+ Ethernet II, Src: RealtekU_12:35:02 (52:54:00:12:35:02), Dst: PcsCompu_d1:8b:d0 (08:00:27:d1:8b:d0)
+ Internet Protocol Version 4, Src: 8.8.8.8, Dst: 10.0.2.15
+ User Datagram Protocol, Src Port: 53, Dst Port: 43674
  Source Port: 53
  Destination Port: 43674
  Length: 68
  Checksum: 0xaf86 [unverified]
  [Checksum Status: Unverified]
  [Source and Destination]
  [Timestamps]
+ Domain Name System (response)
  Transaction ID: 0x6f33
  Flags: 0x8180 Standard query response, No error
  Questions: 1
  Answer RRs: 1
  Authority RRs: 0
  Additional RRs: 1
  [Request In: 1]
  [Time: 0.041528208 seconds]

```

- **ssh**: De acordo com a seguinte figura, observamos que o comando *ssh* tem como protocolo de transporte o TCP, tem 20 bytes de overhead de transporte e a porta de

atendimento é a 22.

15	0.115596750	10.0.2.15	193.136.9.183	SSHv2	134 Client: Elliptic Curve Diffie-Hellman Key Exchange In
16	0.115738991	193.136.9.183	10.0.2.15	TCP	60 22 → 55864 [ACK] Seq=1026 Ack=1634 Win=65535 Len=0
17	0.117787512	193.136.9.183	10.0.2.15	SSHv2	366 server: Elliptic Curve Diffie-Hellman Key Exchange Re
18	0.137803154	10.0.2.15	193.136.9.183	TCP	54 55864 → 22 [ACK] Seq=1634 Ack=1338 Win=63960 Len=0
19	0.138580101	10.0.2.15	193.136.9.183	SSHv2	70 Client: New Keys
20	0.138677227	193.136.9.183	10.0.2.15	TCP	60 22 → 55864 [ACK] Seq=1338 Ack=1650 Win=65535 Len=0
21	0.138937069	10.0.2.15	193.136.9.183	SSHv2	94 Client: Encrypted packet (len=40)

▶	Frame 17: 366 bytes on wire (2928 bits), 366 bytes captured (2928 bits) on interface enp0s3, id 0
▶	Ethernet II, Src: RealtekU_12:35:02 (52:54:00:12:35:02), Dst: PcsCompu_d1:8b:d0 (08:00:27:d1:8b:d0)
▶	Internet Protocol Version 4, Src: 193.136.9.183, Dst: 10.0.2.15
▼	Transmission Control Protocol, Src Port: 22, Dst Port: 55864, Seq: 1026, Ack: 1634, Len: 312
	Source Port: 22
	Destination Port: 55864
	[Stream index: 0]
	[TCP Segment Len: 312]
	Sequence number: 1026 (relative sequence number)
	Sequence number (raw): 679873027
	[Next sequence number: 1338 (relative sequence number)]
	Acknowledgment number: 1634 (relative ack number)
	Acknowledgment number (raw): 636712589
	0101 ... = Header Length: 20 bytes (5)
▼	Flags: 0x018 (PSH, ACK)
	000. = Reserved: Not set
	...0. = Nonce: Not set
0. = Congestion Window Reduced (CWR): Not set
0. = ECN-Echo: Not set
0. = Urgent: Not set
1. = Acknowledgment: Set
1. = Push: Set
0. = Reset: Not set
0. = Syn: Not set
0. = Fin: Not set
	[TCP Flags:AP...]
	Window size value: 65535
	[Calculated window size: 65535]
	[Window size scaling factor: -2 (no window scaling used)]
	Checksum: 0xa168 [unverified]
	[Checksum Status: Unverified]
	Urgent pointer: 0
▶	[SEQ/ACK analysis]
▶	[Timestamps]
	TCP payload (312 bytes)
▼	SSH Protocol
	SSH Version 2 (encryption:aes128-ctr mac:umac-64@openssh.com compression:none)
	SSH Version 2 (encryption:aes128-ctr mac:umac-64@openssh.com compression:none)
	[Direction: server-to-client]

Posto isto, consideramos ainda importante mencionar algumas diferenças entre o TCP e o UDP. Como já referimos anteriormente, o tamanho do cabeçalho do protocolo TCP é de 20 bytes enquanto o do UDP é de apenas 8 bytes. Para além disso, ao contrário do protocolo UDP, o protocolo TCP é orientado à conexão.

O protocolo TCP é capaz de fazer a deteção e correção de erros, fazendo com que os pacotes com erros sejam retransmitidos. Por outro lado, o protocolo UDP é capaz de detetar erros mas não os tenta corrigir, simplesmente descarta os pacotes que tenham erros.

No que toca à velocidade de transferência, o UDP é mais eficiente que o TCP, uma vez que não efetua correção de erros.

1.2 Pergunta 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

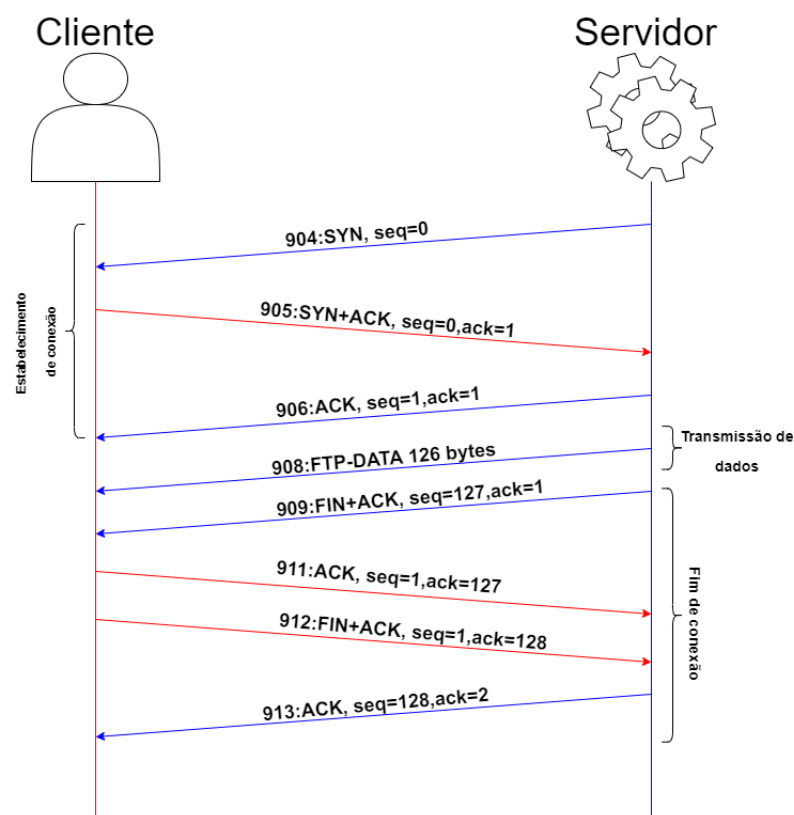


Figura 1: Diagrama Temporal FTP

Através da análise dos vários pacotes de dados, podemos concluir que o diagrama temporal da transferência seria o da seguinte figura. Mediante a sua análise, é possível afirmar que a conexão se inicia com um envio por parte do servidor de um pacote SYN, com número de sequência de valor 0. Seguidamente, o cliente responde com um segmento SYN + ACK, de forma a confirmar o sucesso da receção do segmento TCP anterior. Em resposta ao SYN anteriormente enviado pelo cliente, é encaminhada pelo servidor uma trama ACK, seguida dos dados pretendidos. O servidor envia, posteriormente, um segmento FIN + ACK que comunica a conclusão do envio dos dados. Em seguida, o cliente responde com uma trama ACK e com um pacote FIN + ACK. Por fim, o servidor informa o sucesso no recebimento do segmento TCP FIN, enviando uma trama ACK.

Na figura 2, podemos ver todos os segmentos representados no esquema representado na figura 1. Foram filtrados, no Wireshark, os segmentos correspondentes à conexão de dados.

904	1046.731124...	10.1.1.1	10.4.4.1	TCP	74	20 → 47685	[SYN] Seq=0 Win=64240 Len=0 MSS=1460
905	1046.731267...	10.4.4.1	10.1.1.1	TCP	74	47685 → 20	[SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0
906	1046.731391...	10.1.1.1	10.4.4.1	TCP	66	20 → 47685	[ACK] Seq=1 Ack=1 Win=64256 Len=0 TS=0
907	1046.731456...	10.1.1.1	10.4.4.1	FTP	105	Response: 150	Here comes the directory listing.
908	1046.731564...	10.1.1.1	10.4.4.1	FTP-DA...	192	FTP Data: 126 bytes (PORT)	(LIST)
909	1046.731565...	10.1.1.1	10.4.4.1	TCP	66	20 → 47685	[FIN, ACK] Seq=127 Ack=1 Win=64256 Len=0
910	1046.731589...	10.4.4.1	10.1.1.1	TCP	66	37104 → 21	[ACK] Seq=81 Ack=187 Win=64256 Len=0
911	1046.731887...	10.4.4.1	10.1.1.1	TCP	66	47685 → 20	[ACK] Seq=1 Ack=127 Win=65152 Len=0
912	1046.731887...	10.4.4.1	10.1.1.1	TCP	66	47685 → 20	[FIN, ACK] Seq=1 Ack=128 Win=65152 Len=0
913	1046.732021...	10.1.1.1	10.4.4.1	TCP	66	20 → 47685	[ACK] Seq=128 Ack=2 Win=64256 Len=0
914	1046.732052...	10.1.1.1	10.4.4.1	FTP	90	Response: 226	Directory send OK.

Figura 2: Segmento de Wireshark da transferência FTP

- TFTP

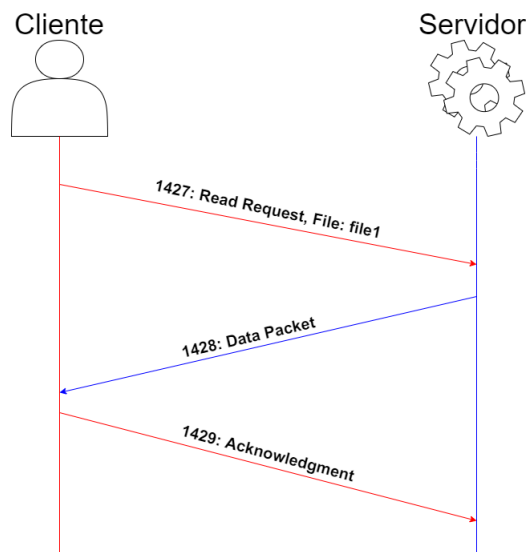


Figura 3: Diagrama Temporal TFTP

De acordo com a figura acima, podemos observar que, primeiramente, o cliente envia um Read Request ao servidor. De seguida, o servidor envia ao cliente um Data Packet com os dados que pretende enviar, no caso dos dados serem maiores do que 512 bytes, são enviados em vários pacotes distintos com esse tamanho máximo. Por último, o cliente, aquando da receção dos dados, envia uma trama ACK ao servidor, confirmando o sucesso da receção dos dados.

Podemos comprovar este comportamento, retirando do Wireshark a seguinte imagem:

1427	1838.156179...	10.4.4.1	10.1.1.1	TFTP	56	Read Request, File: file1, Transfer type: octet
1428	1838.156861...	10.1.1.1	10.4.4.1	TFTP	276	Data Packet, Block: 1 (last)
1429	1838.157076...	10.4.4.1	10.1.1.1	TFTP	46	Acknowledgement, Block: 1

1.3 Pergunta 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;

(i) **Uso da camada de transporte**

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

(ii) **Eficiência na transferência**

- **SFTP**: Idêntico ao FTP, mas os dados são encriptados.
- **FTP**: Visto que usa o protocolo TCP, é garantido, através do uso de *acknowledges*, que o segmento vai ser transmitido, porém há uma perda de eficiência, visto que é necessário esperar pelo *acknowledge* para continuar.
- **TFTP**: Devido ao uso do protocolo UDP por parte desta aplicação, esta torna-se menos viável. Este protocolo perde viabilidade pois não usa *acknowledgements*, não sendo, por essa razão, possível averiguar se o pacote foi entregue com sucesso, sendo, por vezes, necessário a retransmissão dos mesmos. No entanto, em caso de sucesso, este é mais rápido que o FTP.
- **HTTP**: permite que vários HTTP requests sejam enviados numa única ligação TCP sem que seja necessário esperar pelas respostas correspondentes.

(iii) **Complexidade**

- **SFTP**: Uma vez que o protocolo SFTP é muito fiável e possibilita o acesso, transferência e gestão de dados, e visto que estas funcionalidades apresentam custos elevados de processamento, este protocolo revela-se bastante complexo.
- **FTP**: Dado que o protocolo FTP é capaz de suportar múltiplos pedidos de transferência de dados concorrentemente em que realiza uma nova conexão para cada uma das transferências, têm de existir diferentes velocidades de transferência. A elevada frequência de novas conexões torna este protocolo bastante complexo.
- **TFTP**: Através do nome deste protocolo TFTP (Trivial File Transfer Protocol), podemos perceber que este é uma alternativa simplificada do protocolo anterior FTP (File Transfer Protocol). O protocolo TFTP, para além de ser mais simples, suporta muito menos funcionalidades do que a versão com maior complexidade. Para além disto, juntando também o facto deste protocolo ser baseado em UDP e não em TCP, permite-nos concluir que este protocolo não é muito complexo.

- **HTTP** Este protocolo tem sido usado pela WWW desde 1990. A primeira versão de HTTP, chamada HTTP/0.9, era um protocolo simples para a transferência de dados no formato de texto ASCII pela Internet. Mesmo com mais complexidade introduzida no HTTP/2.0 por encapsular mensagens HTTP em quadros (frames), o HTTP foi projetado para ser simples e legível às pessoas. Sendo, neste momento, um protocolo de complexidade moderada.

(iv) Segurança

- **SFTP:** O SFTP (assemelhando-se ao protocolo FTPS) oferece uma proteção extra aos arquivos e alterações feitas na hospedagem. No entanto, o SFTP utiliza-se da tecnologia SSH (Secure Shell) para autenticar o contacto e estabelecer conexões seguras entre as máquinas. O SSH usa uma arquitetura em camadas, em termos de segurança a camada de transporte fornece encriptação de dados e autenticação do servidor e a camada de autenticação é responsável por manusear a autenticação do utilizador, logo, afirma-se que este protocolo consegue garantir uma elevada segurança.
- **FTP:** Este protocolo utiliza autenticação, não proporcionando encriptação de dados, tornando-o suscetível a ter bastantes falhas na segurança. Desta forma, e devido às transmissões não serem encriptadas, este protocolo é extremamente inseguro.
- **TFTP:** Protocolo que não fornece autenticação; como não protege os dados a serem transferidos, é considerado relativamente inseguro.
- **HTTP:** Protocolo da camada de aplicação que não é encriptado, tendo a informação representada em texto. Por esta razão, embora utilize autenticação, é vulnerável a adulteros dos dados, não garantindo segurança a esse nível.

1.4 Pergunta 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).

Ao contrário da situação decorrida na LAN4 em que não houve qualquer problema a nível de ligações de rede, na LAN3, também alusiva à topologia dada no enunciado, repara-se que se sucedeu perda e duplicação de pacotes IP a níveis de Transporte e Aplicação. Durante as experiências feitas nesta LAN, foram usados dois protocolos de aplicação distintos - TFTP e FTP. Tendo em conta que, numa transferência de dados, o protocolo de transporte usado está relacionado com o protocolo de aplicação, conseguimos, a partir dos resultados obtidos, comparar o UDP e TCP, respetivamente.

Através da realização deste projeto e de acordo com o conteúdo estudado nas aulas teóricas, podemos afirmar que o protocolo TCP é capaz de detetar e corrigir possíveis erros, realizando, nestes casos, uma retransmissão do pacote.

De outra forma, o protocolo UDP, apesar de, igualmente, conseguir detetar erros, não tem capacidade de os corrigir, descartando os pacotes nessas situações. Sendo impossível a deteção das perdas de pacotes.

3	0.500794613	10.3.3.3	10.1.1.1	FTP	88 Request: PORT 10,3,3,207,67
4	0.501273351	10.1.1.1	10.3.3.3	FTP	117 Response: 200 PORT command successful. Consider using F
5	0.506624213	10.3.3.3	10.1.1.1	TCP	66 44244 → 21 [ACK] Seq=23 Ack=52 Win=502 Len=0 TSval=3267
6	0.506624674	10.3.3.3	10.1.1.1	FTP	78 Request: RETR file1
7	0.507420247	10.1.1.1	10.3.3.3	TCP	74 20 → 53059 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PE
8	0.512740696	10.3.3.3	10.1.1.1	TCP	74 53059 → 20 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1
9	0.512741216	10.3.3.3	10.1.1.1	TCP	74 [TCP Out-Of-Order] 53059 → 20 [SYN, ACK] Seq=0 Ack=1 W
10	0.512911703	10.1.1.1	10.3.3.3	TCP	66 20 → 53059 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=4291
11	0.512913411	10.1.1.1	10.3.3.3	TCP	66 [TCP Dup ACK 10#1] 20 → 53059 [ACK] Seq=1 Ack=1 Win=642
12	0.513279928	10.1.1.1	10.3.3.3	FTP	130 Response: 150 Opening BINARY mode data connection for f
13	0.513282391	10.1.1.1	10.3.3.3	FTP-DA...	296 FTP Data: 230 bytes (PORT) (RETR file1)
14	0.513283281	10.1.1.1	10.3.3.3	TCP	66 20 → 53059 [FIN, ACK] Seq=231 Ack=1 Win=64256 Len=0 TSV
15	0.519252006	10.3.3.3	10.1.1.1	TCP	66 44244 → 21 [ACK] Seq=35 Ack=116 Win=502 Len=0 TSval=326
16	0.519252646	10.3.3.3	10.1.1.1	TCP	66 53059 → 20 [ACK] Seq=1 Ack=231 Win=65024 Len=0 TSval=32
17	0.519253027	10.3.3.3	10.1.1.1	TCP	66 53059 → 20 [FIN, ACK] Seq=1 Ack=232 Win=65024 Len=0 TSV
18	0.519429264	10.1.1.1	10.3.3.3	TCP	66 20 → 53059 [ACK] Seq=232 Ack=2 Win=64256 Len=0 TSval=42
19	0.520129902	10.1.1.1	10.3.3.3	FTP	90 Response: 226 Transfer complete.
20	0.528961687	10.3.3.3	10.1.1.1	TCP	66 44244 → 21 [ACK] Seq=35 Ack=140 Win=502 Len=0 TSval=326
21	0.528962161	10.3.3.3	10.1.1.1	TCP	66 [TCP Dup ACK 20#1] 44244 → 21 [ACK] Seq=35 Ack=140 Win=

Figura 4: Computador Corvo a fazer download usando o FTP

13	7.793173153	10.3.3.3	10.1.1.1	TFTP	56 Read Request, File: file1, Transfer type: octet
14	7.793173639	10.3.3.3	10.1.1.1	TFTP	56 Read Request, File: file1, Transfer type: octet
15	7.793484232	10.1.1.1	10.3.3.3	TFTP	276 Data Packet, Block: 1 (last)
16	7.793681911	10.1.1.1	10.3.3.3	TFTP	276 Data Packet, Block: 0 (last)
17	7.802070369	10.3.3.3	10.1.1.1	ICMP	304 Destination unreachable (Port unreachable)
21	12.794824296	10.1.1.1	10.3.3.3	TFTP	276 Data Packet, Block: 1 (last)
22	12.800508533	10.3.3.3	10.1.1.1	ICMP	304 Destination unreachable (Port unreachable)

Figura 5: Computador Corvo a fazer download usando o TFTP

Nas figuras anteriores podemos ver o download do ficheiro "file1", usando num dos casos o protocolo FTP e no outro o protocolo TFTP. Visto que na LAN3 existe apenas 5% de package loss e 10% de package duplication, foi necessário fazer vários requests, uma vez que a maior parte deles foram feitos com sucesso. No entanto, após várias tentativas foram encontrados erros em algumas transmissões.

No caso do FTP verificamos que haviam segmentos de Acknowledgment que apareciam repetidos, para além disso haviam segmentos SYN + ACK que chegavam fora de ordem ao destino.

No caso do TFTP, por vezes o ficheiro não chegava ao seu destino final e o host demorava bastante tempo até notar que algo de errado tinha sucedido.

Por outro lado, nos casos em que não aconteciam erros, as transferências em que o protocolo FTP era usado eram mais lentas comparando com o uso do TFTP.

Deste modo, concluímos que, se a velocidade de transmissão for a prioridade, é recomendado o uso do protocolo TFTP; por outro lado, se a prioridade for o ficheiro efetivamente chegar ao seu destino, é aconselhado o uso do protocolo FTP, uma vez que garante a transferência de todos os dados.

2 Conclusão

Neste trabalho foram abordados e explorados diversos Protocolos de Camada de Aplicação e de Transporte. Com recurso à máquina virtual xubuncore e à ferramenta Wireshark, foram realizados vários testes de forma a contemplar e acompanhar o processo de transferência de dados, com o objetivo de se estudar os protocolos, e as vantagens e desvantagens da sua utilização.

Baseada nos resultados, é possível obter a conclusão de que, na eventualidade de uma transferência de dados, ambos os protocolos de transporte têm vantagens a serem tidas em conta, dependendo das prioridades do envio. No caso de se desejar uma transferência de alta fiabilidade, com a garantia de que os dados enviados sejam, de facto, recebidos, devemos optar por uma aplicação que recorra ao protocolo TCP para efetuar o transporte, consentindo com uma menor eficiência.

De outro modo, se uma maior velocidade de transferência for uma prioridade, negligenciando possíveis perdas de pacotes, o protocolo UDP seria o mais indicado.

Em geral, consideramos chegar a conclusões pertinentes e ter consolidado conhecimentos úteis para a Unidade Curricular de Comunicações por Computador, tendo cumprido os objetivos previstos.