

Protocolos da Camada de Transporte

Comunicações por Computador

Etienne Costa(A76089)

Joana Cruz(A76270)

Hugo Moreira(A43148)

Resumo

Neste primeiro trabalho abordamos os diferentes protocolos de aplicação e de transporte. Para cada protocolo de aplicação exploramos qual o respetivo protocolo de transporte utilizado, a porta de atendimento, e comparamos esses diversos protocolos conforme diversos parâmetros pedidos.

Questão 1

Através dos comandos executados ao longo do trabalho, conseguimos retirar as seguintes informações.

Comando utilizado	Protocolo de Aplicação	Protocolo de Transporte	Porta de atendimento	Overhead de transporte
ping	-----	-----	-----	-----
tracert	-----	UDP	33434-33450	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

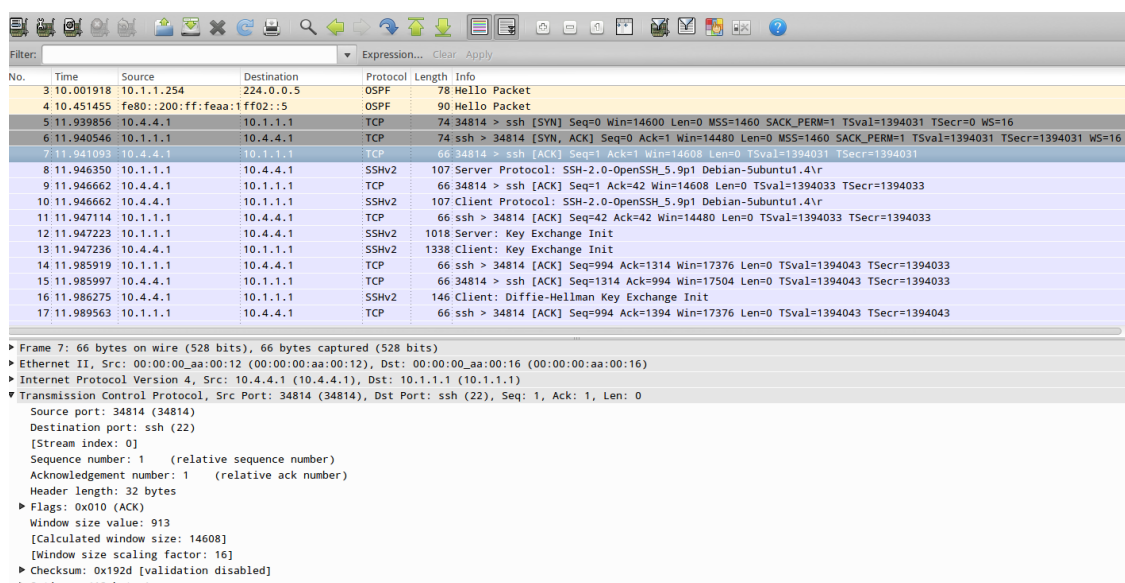


Figura 1 – Captura no wireshark usando o comando SSH

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.1.1.254	224.0.0.5	OSPF	78	Hello Packet
2	0.364213	fe80::200:ff:feaa:1ff02::5	224.0.0.5	OSPF	90	Hello Packet
3	6.990023	10.4.4.1	10.1.1.1	TCP	74	38100 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=1325287 TSecr=0 WS=16
4	6.990208	10.1.1.1	10.4.4.1	TCP	74	http > 38100 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PERM=1 TSval=1325287 TSecr=1325287 WS=16
5	6.991709	10.4.4.1	10.1.1.1	TCP	66	38100 > http [ACK] Seq=1 Ack=1 Win=14608 Len=0 TSval=1325287 TSecr=1325287
6	6.991738	10.4.4.1	10.1.1.1	HTTP	179	GET /file1 HTTP/1.1
7	6.992167	10.1.1.1	10.4.4.1	TCP	66	http > 38100 [ACK] Seq=1 Ack=114 Win=14480 Len=0 TSval=1325288 TSecr=1325287
8	6.992846	10.1.1.1	10.4.4.1	TCP	289	[TCP segment of a reassembled PDU]
9	6.992847	10.1.1.1	10.4.4.1	HTTP	259	HTTP/1.1 200 OK (text/plain)
10	6.993454	10.4.4.1	10.1.1.1	TCP	66	38100 > http [ACK] Seq=114 Ack=224 Win=15680 Len=0 TSval=1325288 TSecr=1325288
11	6.997056	10.4.4.1	10.1.1.1	TCP	66	38100 > http [FIN, ACK] Seq=114 Ack=418 Win=16752 Len=0 TSval=1325288 TSecr=1325288
12	6.997511	10.1.1.1	10.4.4.1	TCP	66	http > 38100 [ACK] Seq=418 Ack=115 Win=14480 Len=0 TSval=1325289 TSecr=1325288
13	10.000179	10.1.1.254	224.0.0.5	OSPF	78	Hello Packet
14	10.393376	fe80::200:ff:feaa:1ff02::5	224.0.0.5	OSPF	90	Hello Packet
15	20.000335	10.1.1.254	224.0.0.5	OSPF	78	Hello Packet
16	20.346413	fe80::200:ff:feaa:1ff02::5	224.0.0.5	OSPF	90	Hello Packet
17	30.000562	10.1.1.254	224.0.0.5	OSPF	78	Hello Packet
18	30.359543	fe80::200:ff:feaa:1ff02::5	224.0.0.5	OSPF	90	Hello Packet

▶ Frame 5: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
 ▶ Ethernet II, Src: 00:00:00:aa:00:12 (00:00:00:aa:00:12), Dst: 00:00:00:aa:00:16 (00:00:00:aa:00:16)
 ▶ Internet Protocol Version 4, Src: 10.4.4.1 (10.4.4.1), Dst: 10.1.1.1 (10.1.1.1)
 ▼ Transmission Control Protocol, Src Port: 38100 (38100), Dst Port: http (80), Seq: 1, Ack: 1, Len: 0
 Source port: 38100 (38100)
 Destination port: http (80)
 [Stream index: 0]
 Sequence number: 1 (relative sequence number)
 Acknowledgement number: 1 (relative ack number)
 Header length: 32 bytes
 ▶ Flags: 0x010 (ACK)
 Window size value: 913

Figura 2 – Captura no wireshark usando o comando HTTP

No.	Time	Source	Destination	Protocol	Length	Info
25	9.273052	10.0.2.15	193.136.9.183	FTP	67	Request: USER gr2018
26	9.273359	193.136.9.183	10.0.2.15	TCP	60	ftp > 44382 [ACK] Seq=1 Ack=14 Win=65535 Len=0
27	9.278044	193.136.9.183	10.0.2.15	FTP	94	Response: 530 This FTP server is anonymous only.
28	9.278107	10.0.2.15	193.136.9.183	TCP	54	44382 > ftp [ACK] Seq=14 Ack=41 Win=14600 Len=0
29	9.278289	10.0.2.15	193.136.9.183	FTP	60	Request: SYST
30	9.278500	193.136.9.183	10.0.2.15	TCP	60	ftp > 44382 [ACK] Seq=41 Ack=20 Win=65535 Len=0
31	9.281960	193.136.9.183	10.0.2.15	FTP	92	Response: 530 Please login with USER and PASS.
32	9.319216	10.0.2.15	193.136.9.183	TCP	54	44382 > ftp [ACK] Seq=20 Ack=79 Win=14600 Len=0

▶ Frame 25: 67 bytes on wire (536 bits), 67 bytes captured (536 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)
 ▼ Transmission Control Protocol, Src Port: 44382 (44382), Dst Port: ftp (21), Seq: 1, Ack: 1, Len: 13
 Source port: 44382 (44382)
 Destination port: ftp (21)
 [Stream index: 24]
 Sequence number: 1 (relative sequence number)
 [Next sequence number: 14 (relative sequence number)]
 Acknowledgement number: 1 (relative ack number)
 Header length: 20 bytes
 ▶ Flags: 0x018 (PSH, ACK)
 Window size value: 14600
 [Calculated window size: 14600]
 [Window size scaling factor: -1 (unknown)]
 ▶ Checksum: 0xd775 [validation disabled]
 ▶ [SEQ/ACK analysis]
 ▶ File Transfer Protocol (FTP)

Figura 3 – Captura no wireshark usando o comando FTP

No.	Time	Source	Destination	Protocol	Length	Info
27	122.679914	10.4.4.1	10.1.1.1	TFTP	56	Read Request, File: file1, Transfer type: octet
28	122.680378	10.1.1.1	10.4.4.1	TFTP	239	Data Packet, Block: 1 (last)
29	122.680681	10.4.4.1	10.1.1.1	TFTP	46	Acknowledgement, Block: 1

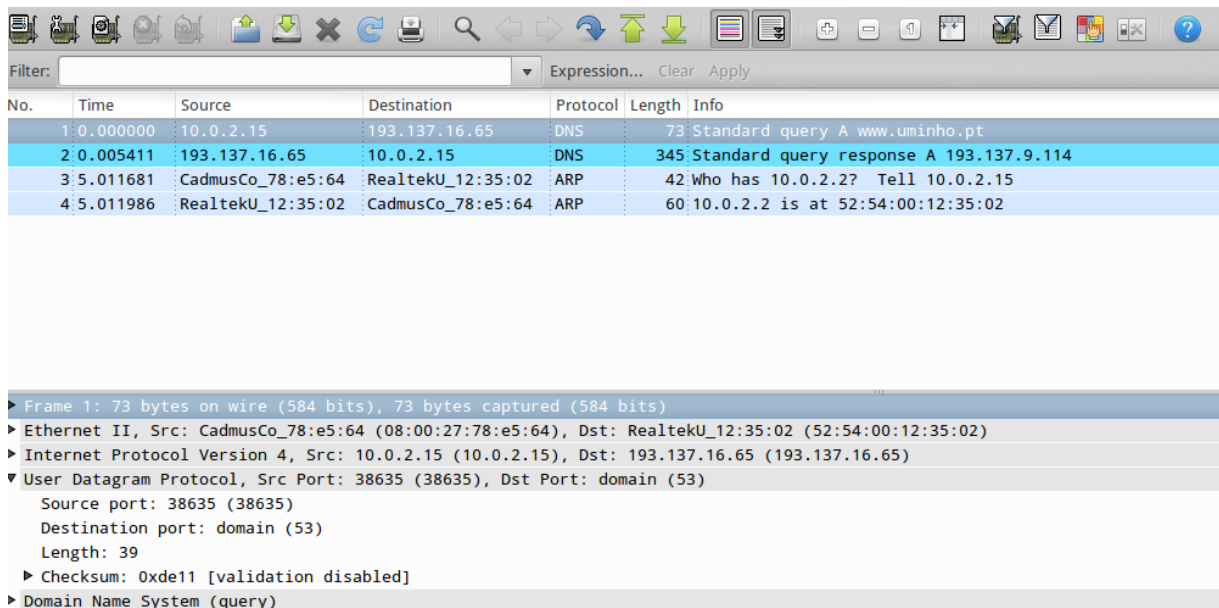
▶ Frame 27: 56 bytes on wire (448 bits), 56 bytes captured (448 bits)
 ▶ Ethernet II, Src: 00:00:00_aa:00:12 (00:00:00:aa:00:12), Dst: 00:00:00_aa:00:16 (00:00:00:aa:00:16)
 ▶ Internet Protocol Version 4, Src: 10.4.4.1 (10.4.4.1), Dst: 10.1.1.1 (10.1.1.1)
 ▼ User Datagram Protocol, Src Port: 44298 (44298), Dst Port: tftp (69)
 Source port: 44298 (44298)
 Destination port: tftp (69)
 Length: 22
 ▶ Checksum: 0x192e [validation disabled]
 ▶ Trivial File Transfer Protocol

Figura 4 – Captura no wireshark usando o comando TFTP

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.2.15	193.136.9.33	TELNET	55	Telnet Data ...
2	0.000327	193.136.9.33	10.0.2.15	TCP	60	telnet > 35280 [ACK] Seq=1 Ack=2 Win=65535 Len=0
3	0.492822	10.0.2.15	193.136.9.33	TELNET	55	Telnet Data ...
4	0.493048	193.136.9.33	10.0.2.15	TCP	60	telnet > 35280 [ACK] Seq=1 Ack=3 Win=65535 Len=0
5	0.702539	10.0.2.15	193.136.9.33	TELNET	55	Telnet Data ...
6	0.702826	193.136.9.33	10.0.2.15	TCP	60	telnet > 35280 [ACK] Seq=1 Ack=4 Win=65535 Len=0
7	1.030458	10.0.2.15	193.136.9.33	TELNET	55	Telnet Data ...
8	1.030731	193.136.9.33	10.0.2.15	TCP	60	telnet > 35280 [ACK] Seq=1 Ack=5 Win=65535 Len=0
9	1.256293	10.0.2.15	193.136.9.33	TELNET	55	Telnet Data ...
10	1.256571	193.136.9.33	10.0.2.15	TCP	60	telnet > 35280 [ACK] Seq=1 Ack=6 Win=65535 Len=0
11	1.299313	193.136.9.33	10.0.2.15	TELNET	97	Telnet Data ...
12	1.299341	10.0.2.15	193.136.9.33	TCP	54	35280 > telnet [ACK] Seq=6 Ack=44 Win=15544 Len=0
13	1.391655	10.0.2.15	193.136.9.33	TELNET	55	Telnet Data ...
14	1.391874	193.136.9.33	10.0.2.15	TCP	60	telnet > 35280 [ACK] Seq=44 Ack=7 Win=65535 Len=0
15	1.873551	10.0.2.15	193.136.9.33	TELNET	56	Telnet Data ...

▶ Frame 3: 55 bytes on wire (440 bits), 55 bytes captured (440 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: 193.136.9.33 (193.136.9.33)
 ▼ Transmission Control Protocol, Src Port: 35280 (35280), Dst Port: telnet (23), Seq: 2, Ack: 1, Len: 1
 Source port: 35280 (35280)
 Destination port: telnet (23)
 [Stream index: 0]
 Sequence number: 2 (relative sequence number)
 [Next sequence number: 3 (relative sequence number)]
 Acknowledgement number: 1 (relative ack number)
 Header length: 20 bytes
 ▶ Flags: 0x018 (PSH, ACK)
 Window size value: 15544
 [Calculated window size: 15544]
 [Window size scaling factor: -1 (unknown)]
 ▶ Checksum: 0xd6d3 [validation disabled]

Figura 5 – Captura no wireshark usando o comando Telnet

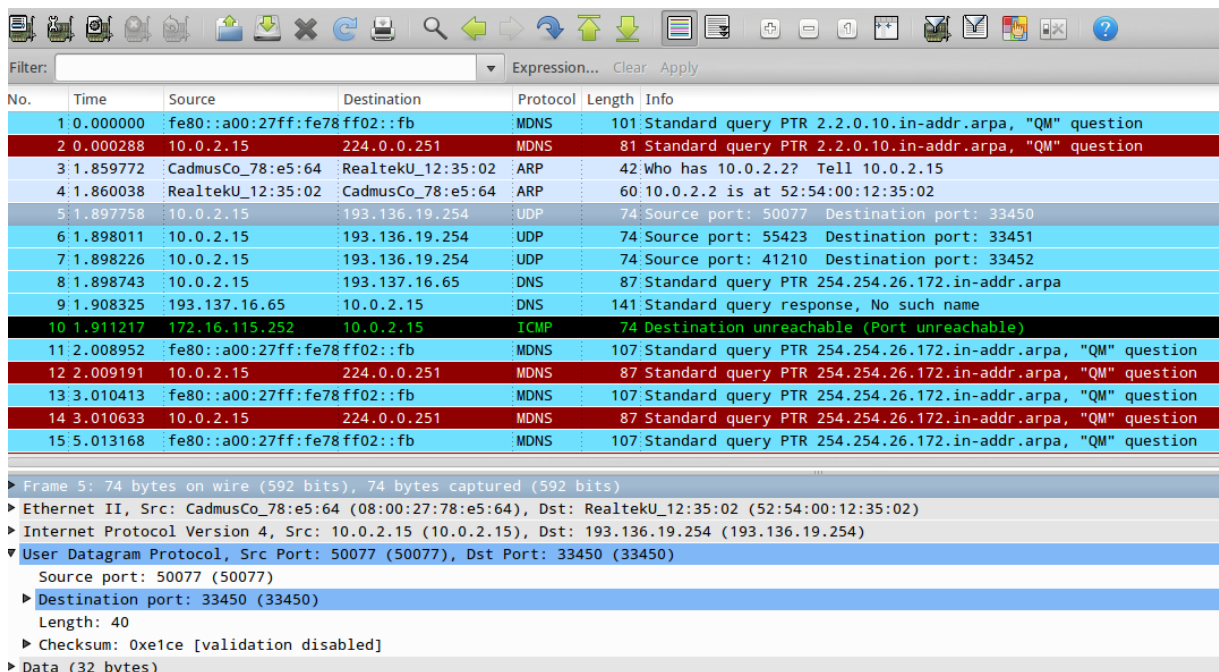


Filter: Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.2.15	193.137.16.65	DNS	73	Standard query A www.uminho.pt
2	0.005411	193.137.16.65	10.0.2.15	DNS	345	Standard query response A 193.137.9.114
3	5.011681	CadmusCo_78:e5:64	RealtekU_12:35:02	ARP	42	Who has 10.0.2.2? Tell 10.0.2.15
4	5.011986	RealtekU_12:35:02	CadmusCo_78:e5:64	ARP	60	10.0.2.2 is at 52:54:00:12:35:02

▶ Frame 1: 73 bytes on wire (584 bits), 73 bytes captured (584 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: 193.137.16.65 (193.137.16.65)
 ▼ User Datagram Protocol, Src Port: 38635 (38635), Dst Port: domain (53)
 Source port: 38635 (38635)
 Destination port: domain (53)
 Length: 39
 ▶ Checksum: 0xde11 [validation disabled]
 ▶ Domain Name System (query)

Figura 6 – Captura no wireshark usando o comando nslookup



Filter: Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fe80::a00:27ff:fe78:ff02::fb	ff02::fb	MDNS	101	Standard query PTR 2.2.0.10.in-addr.arpa, "QM" question
2	0.000288	10.0.2.15	224.0.0.251	MDNS	81	Standard query PTR 2.2.0.10.in-addr.arpa, "QM" question
3	1.859772	CadmusCo_78:e5:64	RealtekU_12:35:02	ARP	42	Who has 10.0.2.2? Tell 10.0.2.15
4	1.860038	RealtekU_12:35:02	CadmusCo_78:e5:64	ARP	60	10.0.2.2 is at 52:54:00:12:35:02
5	1.897758	10.0.2.15	193.136.19.254	UDP	74	Source port: 50077 Destination port: 33450
6	1.898011	10.0.2.15	193.136.19.254	UDP	74	Source port: 55423 Destination port: 33451
7	1.898226	10.0.2.15	193.136.19.254	UDP	74	Source port: 41210 Destination port: 33452
8	1.898743	10.0.2.15	193.137.16.65	DNS	87	Standard query PTR 254.254.26.172.in-addr.arpa
9	1.908325	193.137.16.65	10.0.2.15	DNS	141	Standard query response, No such name
10	1.911217	172.16.115.252	10.0.2.15	ICMP	74	Destination unreachable (Port unreachable)
11	2.008952	fe80::a00:27ff:fe78:ff02::fb	ff02::fb	MDNS	107	Standard query PTR 254.254.26.172.in-addr.arpa, "QM" question
12	2.009191	10.0.2.15	224.0.0.251	MDNS	87	Standard query PTR 254.254.26.172.in-addr.arpa, "QM" question
13	3.010413	fe80::a00:27ff:fe78:ff02::fb	ff02::fb	MDNS	107	Standard query PTR 254.254.26.172.in-addr.arpa, "QM" question
14	3.010633	10.0.2.15	224.0.0.251	MDNS	87	Standard query PTR 254.254.26.172.in-addr.arpa, "QM" question
15	5.013168	fe80::a00:27ff:fe78:ff02::fb	ff02::fb	MDNS	107	Standard query PTR 254.254.26.172.in-addr.arpa, "QM" question

▶ Frame 5: 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)
 ▶ Internet Protocol Version 4, Src: 10.0.2.15 (10.0.2.15), Dst: 193.136.19.254 (193.136.19.254)
 ▼ User Datagram Protocol, Src Port: 50077 (50077), Dst Port: 33450 (33450)
 Source port: 50077 (50077)
 ▶ Destination port: 33450 (33450)
 Length: 40
 ▶ Checksum: 0xe1ce [validation disabled]
 ▶ Data (32 bytes)

Figura 7 – Captura no wireshark usando o comando traceroute

Questão 2

FTP – File Transfer Protocol

Inicialmente, não entendemos que apenas seria para representar a transferência para a conexão de dados, daí termos representado todo o diagrama temporal da transferência incluindo todas as mensagens de controlo.

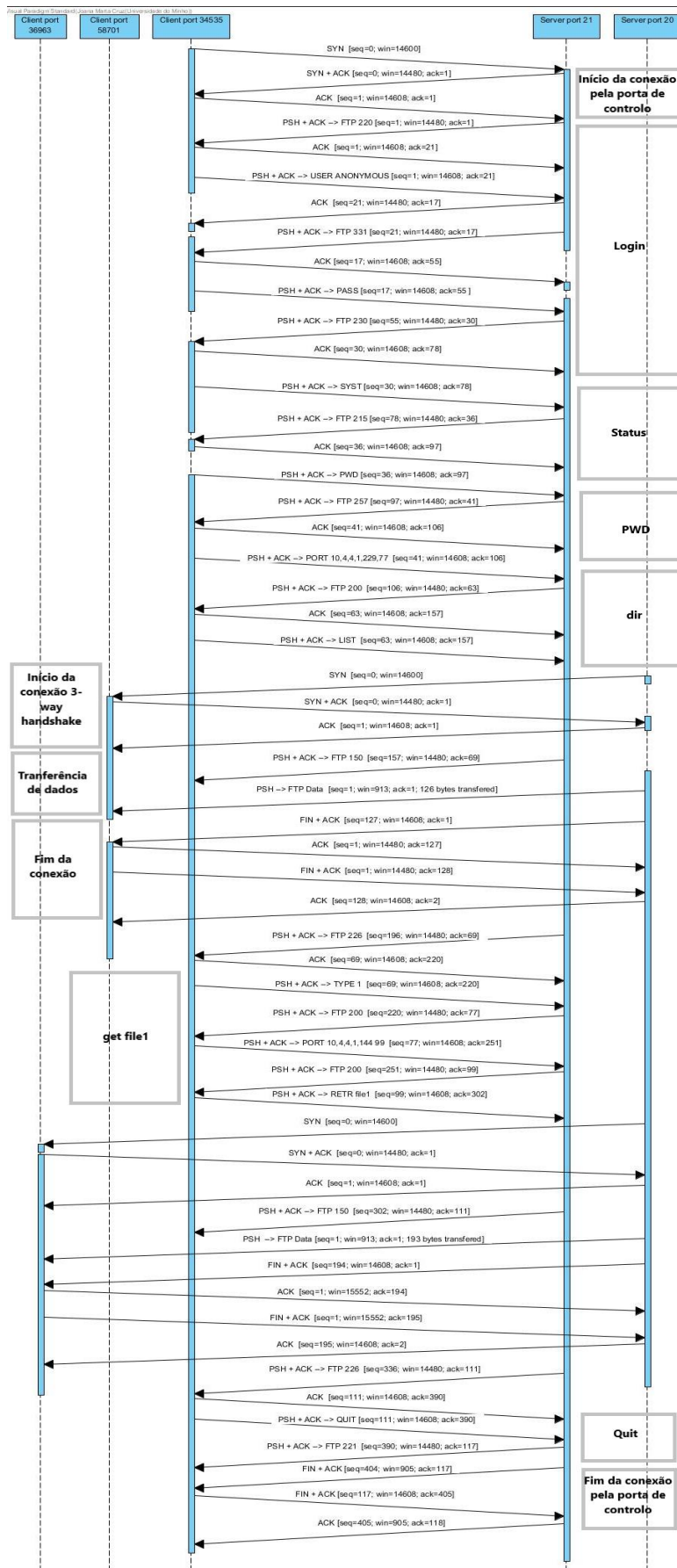


Figura 8 – Diagrama temporal da transferência por FTP

De seguida, apresentamos uma lista dos segmentos trocados pelo cliente e pelo servidor e o seu significado:

- **Cliente para Servidor**

1. **USER ANONYMOUS** – utilizador informa que pretende entrar como anónimo;
2. **PASS** – password de acesso;
3. **SYST** – comando status inserido na bash;
4. **PWD** – comando pwd inserido na bash;
5. **PORT** – o cliente FTP para estabelecer uma segunda conexão(de dados) indica a porta e o endereço;
6. **LIST** – utilizador requer a listagem de uma diretoria;
7. **TYPE 1** – indica que pretende o ficheiro 1;
8. **RETR** – utilizador insere na bash o comando get, para obter um certo ficheiro;
9. **QUIT** – cliente FTP pretende terminar a conexão;

- **Servidor para Cliente**

1. **FTP 220** - o servidor FTP está pronto para aceitar um login;
2. **FTP 331** - o servidor FTP indica que o login como anónimo é permitido;
3. **FTP 230** - mensagem de welcome após ser efetuado o login;
4. **FTP 215** - resposta dada ao comando SYST;
5. **FTP 257** - resposta dada ao comando PWD;
6. **FTP 200** - o servidor FTP server dá conhecimento positivo sobre o comando PORT;
7. **FTP 150** - o servidor FTP notifica o cliente que está pronto a transferir a lista pedida;
8. **FTP 226** - indica que está a transferência pedida foi efetuada;
9. **FTP 221** - mensagem de obrigado pela visita.

No.	Time	Source	Destination	Protocol	Length	Info
169	406.261105	10.4.4.1	10.1.1.1	TCP	66	34536 > ftp [ACK] Seq=63 Ack=157 Win=14608 Len=0 TSval=997557 TSecr=997557
170	406.261106	10.4.4.1	10.1.1.1	FTP	72	Request: LIST
171	406.261418	10.1.1.1	10.4.4.1	TCP	74	ftp-data > 58701 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=997557 TSecr=0 WS=16
172	406.261620	10.4.4.1	10.1.1.1	TCP	74	58701 > ftp-data [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PERM=1 TSval=997557 TSecr=997557
173	406.261813	10.1.1.1	10.4.4.1	TCP	66	ftp-data > 58701 [ACK] Seq=1 Ack=1 Win=14608 Len=0 TSval=997557 TSecr=997557
174	406.261813	10.1.1.1	10.4.4.1	FTP	105	Response: 150 Here comes the directory listing.
175	406.262522	10.1.1.1	10.4.4.1	FTP-DAT	192	FTP Data: 126 bytes
176	406.262573	10.1.1.1	10.4.4.1	TCP	66	ftp-data > 58701 [FIN, ACK] Seq=127 Ack=1 Win=14608 Len=0 TSval=997557 TSecr=997557
177	406.262964	10.4.4.1	10.1.1.1	TCP	66	58701 > ftp-data [ACK] Seq=1 Ack=127 Win=14480 Len=0 TSval=997558 TSecr=997557
178	406.263059	10.4.4.1	10.1.1.1	TCP	66	58701 > ftp-data [FIN, ACK] Seq=1 Ack=128 Win=14480 Len=0 TSval=997558 TSecr=997557
179	406.263646	10.1.1.1	10.4.4.1	TCP	66	ftp-data > 58701 [ACK] Seq=128 Ack=2 Win=14608 Len=0 TSval=997558 TSecr=997558

▶ Frame 171: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
 ▶ Ethernet II, Src: 00:00:00:aa:00:16 (00:00:00:aa:00:16), Dst: 00:00:00:aa:00:12 (00:00:00:aa:00:12)
 ▶ Internet Protocol Version 4, Src: 10.1.1.1 (10.1.1.1), Dst: 10.4.4.1 (10.4.4.1)
 ▼ Transmission Control Protocol, Src Port: ftp-data (20), Dst Port: 58701 (58701), Seq: 0, Len: 0
 Source port: ftp-data (20)
 Destination port: 58701 (58701)
 [Stream index: 5]
 Sequence number: 0 (relative sequence number)
 Header length: 40 bytes
 Flags: 0x002 (SYN)
 Window size value: 14600
 [Calculated window size: 14600]
 Checksum: 0x1935 [validation disabled]
 Options: (20 bytes)

Figura 9 – Captura wireshark da transferência do ficheiro 1 por FTP

Por último, apresentamos o diagrama apenas para a conexão de transferência de dados do ficheiro mais pequeno. Para a conceção deste diagrama guiarmo-nos pela troca de segmentos obtida apenas na figura acima apresentada.

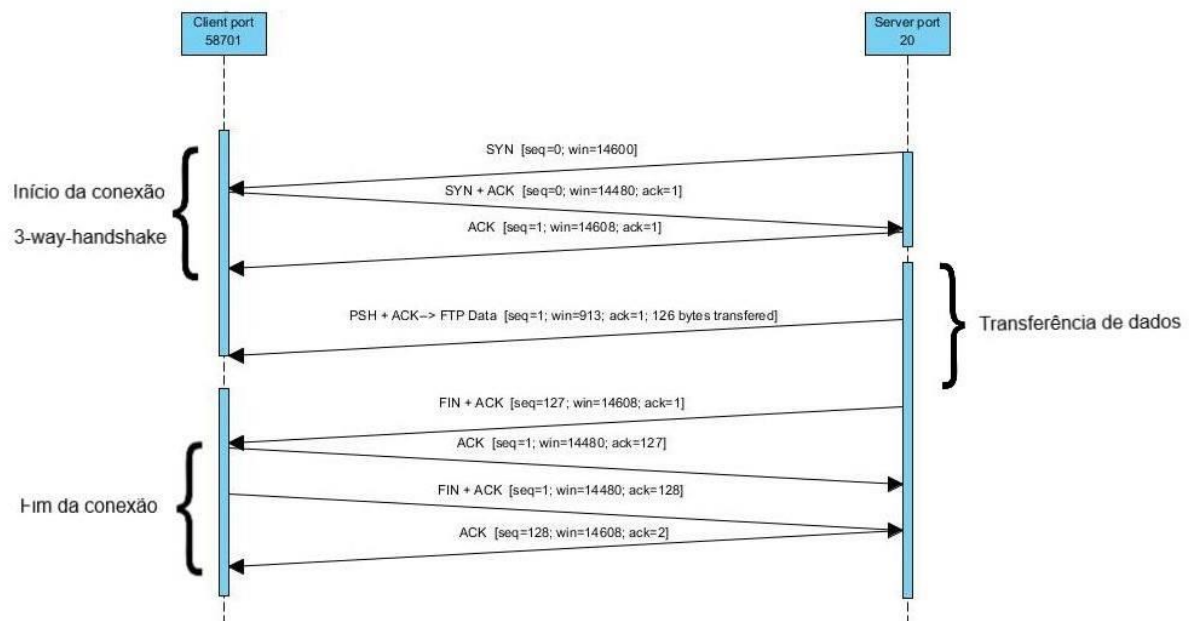


Figura 10 – Diagrama temporal para a transferência por FTP

TFTP – Trivial File Transfer Protocol

No.	Time	Source	Destination	Protocol	Length	Info
27	122.679914	10.4.4.1	10.1.1.1	TFTP	56	Read Request, File: file1, Transfer type: octet
28	122.680378	10.1.1.1	10.4.4.1	TFTP	239	Data Packet, Block: 1 (last)
29	122.680681	10.4.4.1	10.1.1.1	TFTP	46	Acknowledgement, Block: 1

<p>▶ Frame 27: 56 bytes on wire (448 bits), 56 bytes captured (448 bits)</p> <p>▶ Ethernet II, Src: 00:00:00_aa:00:12 (00:00:00:aa:00:12), Dst: 00:00:00_aa:00:16 (00:00:00:aa:00:16)</p> <p>▶ Internet Protocol Version 4, Src: 10.4.4.1 (10.4.4.1), Dst: 10.1.1.1 (10.1.1.1)</p> <p>▼ User Datagram Protocol, Src Port: 44298 (44298), Dst Port: tftp (69)</p> <p>Source port: 44298 (44298)</p> <p>Destination port: tftp (69)</p> <p>Length: 22</p> <p>▶ Checksum: 0x192e [validation disabled]</p> <p>▶ Trivial File Transfer Protocol</p>

Figura 11 – Captura wireshark da transferência do ficheiro 1 por TFTP

Através dos dados que conseguimos capturar na imagem acima, obtemos o seguinte diagrama temporal.

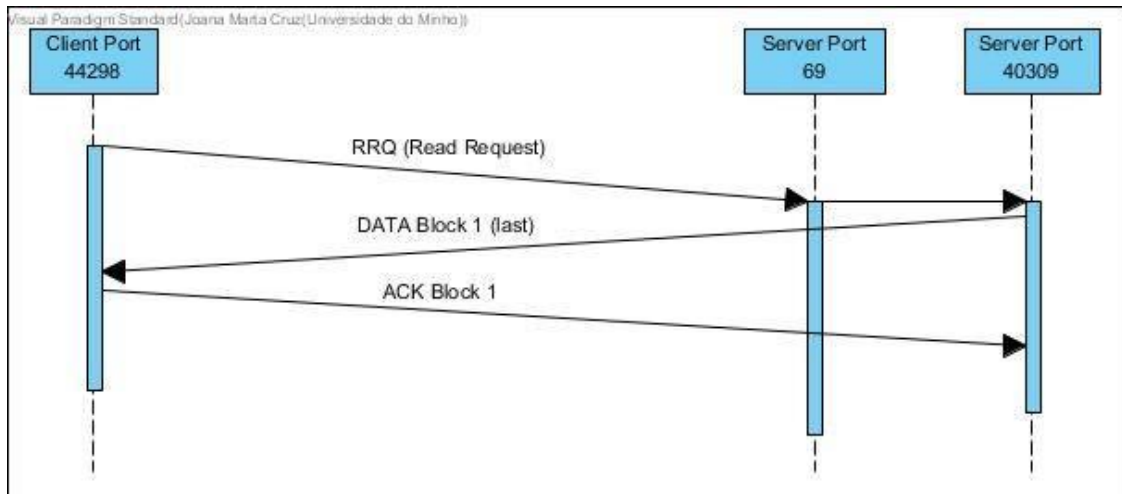


Figura 12 – Diagrama temporal da transferência por TFTP

O servidor escuta sempre na porta 69 (porta padrão do TFTP), mas encaminha para outra porta todas as respostas (DATA, ACK) para libertar a porta 69 para outros possíveis pedidos (o servidor gera um novo socket UDP para manipular o TFTP pedido). Daí a nossa representação deste diagrama temporal da transferência por TFTP.

Mensagens TFTP usadas

RRQ – Read Request Message, usada para notificar que pretende ler dados;

DATA – mensagem usada pelo cliente ou o servidor para o envio de um bloco de dados;

ACK - mensagem de reconhecimento da recepção do bloco de dados;

Questão 3

Para esta questão e a próxima já achamos necessário inserir os diferentes tempos das transferências do ficheiro 2 entre o Cliente 1 e o Servidor 1, usando diferentes protocolos.

Protocolo	Cliente 1(tempo em s)
FTP	0.63
TFTP	0.079
HTTP	3.9
SFTP	5

Estes tempos foram obtidos pelos diversos comandos, em que tínhamos que executar manualmente o servidor, ou apenas verificar se já se encontrava à escuta.

```
root@Cliente1:/tmp/pycore.41562/Cliente1.conf
Connected to 10.1.1.1.
No proxy connection.
Connecting using address family: any.
Mode: stream; Type: binary; Form: non-print; Structure: file
Verbose: on; Bell: off; Prompting: on; Globbing: on
Store unique: off; Receive unique: off
Case: off; CR stripping: on
Quote control characters: on
Ntrans: off
Wmap: off
Hash mark printing: off; Use of PORT cmds: on
Tick counter printing: off
ftp> pwd
257 "/"
ftp> dir
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rwxrwxrwx  1.0  0      193 Feb 19 11:38 file1
-rwxrwxrwx  1.0  0      104508 Feb 19 11:38 file2
226 Directory send OK.
ftp> get file2
local: file2 remote: file2
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for file2 (104508 bytes).
226 Transfer complete.
104508 bytes received in 0,63 secs (162,5 kB/s)
ftp> quit
221 Goodbye.
root@Cliente1:/tmp/pycore.41562/Cliente1.conf#

root@Alfa:/tmp/pycore.41562/Alfa.conf
Connected to 10.1.1.1.
No proxy connection.
Connecting using address family: any.
Mode: stream; Type: binary; Form: non-print; Structure: file
Verbose: on; Bell: off; Prompting: on; Globbing: on
Store unique: off; Receive unique: off
Case: off; CR stripping: on
Quote control characters: on
Ntrans: off
Wmap: off
Hash mark printing: off; Use of PORT cmds: on
Tick counter printing: off
ftp> pwd
257 "/"
ftp> dir
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rwxrwxrwx  1.0  0      193 Feb 19 11:38 file1
-rwxrwxrwx  1.0  0      104508 Feb 19 11:38 file2
226 Directory send OK.
ftp> get file2
local: file2 remote: file2
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for file2 (104508 bytes).
226 Transfer complete.
104508 bytes received in 2,10 secs (48,6 kB/s)
ftp> quit
221 Goodbye.
root@Alfa:/tmp/pycore.41562/Alfa.conf#
```

Figura 13 - Transferência do ficheiro 2 por FTP

```
root@Cliente1:/tmp/pycore.41562/Cliente1.conf
client-port: 76
mcast-ip: 0,0,0,0
listen-delay: 2
timeout-delay: 2
Last command: quit
tftp> get file2
Overwrite local file [y/n]? y
tftp> status
Connected: 10.1.1.1 port 69
Mode: octet
Verbose: off
Trace: off
Options
  tsize: disabled
  blksize: disabled
  timeout: disabled
  multicast: disabled
mtftp variables
  client-port: 76
  mcast-ip: 0,0,0,0
  listen-delay: 2
  timeout-delay: 2
Last command: get file2
Last file: file2
  Bytes transferred: 105,328kB
  Time of transfer: 0,079s
  Throughput: 1,337MB/s
tftp> quit
root@Cliente1:/tmp/pycore.41562/Cliente1.conf#

root@Alfa:/tmp/pycore.41562/Alfa.conf
mtftp variables
  client-port: 76
  mcast-ip: 0,0,0,0
  listen-delay: 2
  timeout-delay: 2
Last command: quit
tftp> get file2
Overwrite local file [y/n]? y
tftp> status
Connected: 10.1.1.1 port 69
Mode: octet
Verbose: off
Trace: off
Options
  tsize: disabled
  blksize: disabled
  timeout: disabled
  multicast: disabled
mtftp variables
  client-port: 76
  mcast-ip: 0,0,0,0
  listen-delay: 2
  timeout-delay: 2
Last command: get file2
Last file: file2
  Bytes transferred: 19,274MB
  Time of transfer: 1,928s
  Throughput: 9,996MB/s
tftp>
```

Figura 14 - Transferência do ficheiro 2 por TFTP

```
root@Cliente1:/tmp/pycore.41562/Cliente1.conf - + x
root@Cliente1:/tmp/pycore.41562/Cliente1.conf# wget http://10.1.1.1/file2
--2019-02-26 19:16:16-- http://10.1.1.1/file2
Connecting to 10.1.1.1:80... connected.
HTTP request sent, awaiting response... 200 Ok
Length: 104508 (102K) [text/plain]
Saving to: 'file2.1'

100%[=====] 104,508 26.3K/s in 3.9s
2019-02-26 19:16:20 (26.3 KB/s) - 'file2.1' saved [104508/104508]
root@Cliente1:/tmp/pycore.41562/Cliente1.conf#

root@Alfa:/tmp/pycore.41562/Alfa.conf - + x
root@Alfa:/tmp/pycore.41562/Alfa.conf# wget http://10.1.1.1/file2
--2019-02-26 19:16:23-- http://10.1.1.1/file2
Connecting to 10.1.1.1:80... connected.
HTTP request sent, awaiting response... 200 Ok
Length: 104508 (102K) [text/plain]
Saving to: 'file2.1'

100%[=====] 104,508 43.9K/s in 2.3s
2019-02-26 19:16:25 (43.9 KB/s) - 'file2.1' saved [104508/104508]
root@Alfa:/tmp/pycore.41562/Alfa.conf#
```

Figura 15 - Transferência do ficheiro 2 por HTTP

```
root@Cliente1:/tmp/pycore.41562/Cliente1.conf - + x
root@Cliente1:/tmp/pycore.41562/Cliente1.conf# rm /root/.ssh/known_hosts
root@Cliente1:/tmp/pycore.41562/Cliente1.conf# sftp core@10.1.1.1
The authenticity of host '10.1.1.1 (10.1.1.1)' can't be established.
RSA key fingerprint is b1:01:17:d1:b4:7d:c4:1a:32:7e:27:0f:3d:7c:80:50.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.1.1.1' (RSA) to the list of known hosts.
core@10.1.1.1's password:
Connected to 10.1.1.1.
sftp> pwd
Remote working directory: /home/core
sftp> cd /srv/ftp
sftp> dir
file1 file2
sftp> get file2
Fetching /srv/ftp/file2 to file2
/srv/ftp/file2 100% 102KB 20.4KB/s 00:05
sftp> quit
root@Cliente1:/tmp/pycore.41562/Cliente1.conf#

root@Alfa:/tmp/pycore.41562/Alfa.conf - + x
root@Alfa:/tmp/pycore.41562/Alfa.conf# sftp core@10.1.1.1
core@10.1.1.1's password:
Connected to 10.1.1.1.
sftp> pwd
Remote working directory: /home/core
sftp> cd /srv/ftp
sftp> dir
file1 file2
sftp> get file2
Fetching /srv/ftp/file2 to file2
/srv/ftp/file2 100% 102KB 25.5KB/s 00:04
sftp> quit
root@Alfa:/tmp/pycore.41562/Alfa.conf#
```

Figura 16 - Transferência do ficheiro 2 por SFTP

A tabela a seguir apresentada, inclui certas conclusões que retiramos ao longo das experiências efetuadas, ao mesmo tempo que obtemos alguma informação extra para obtermos uma análise crítica dos diferentes parâmetros.

Protocolo	Uso da camada de transporte	Eficiência na transferência (tempo)	Complexidade	Segurança
FTP – File Transfer Protocol	O FTP usa o TCP, com duas conexões distintas, uma para controlo e outra para dados. O canal de controlo é usado na porta 21 no servidor, e é responsável pela troca de comandos simples entre o cliente e o servidor. Enquanto o canal de dados é usado na porta 20 para troca de dados na forma de listagens de diretorias e transferências de dados.	Transferência por FTP envolve o cliente enviar comandos para os quais o servidor responde. Uma única transferência pode envolver uma série de comandos. Isso obviamente tem um impacto negativo, já que há um atraso de ida e volta para cada comando.	Dado o FTP usar o TCP que é um protocolo fiável e orientado à conexão, o software usado utiliza mais memória que por exemplo o do TFTP, necessita sempre de estabelecer duas conexões (uma de controlo e uma de dados), e tem bastantes comandos disponíveis, é considerado um protocolo com algum nível de complexidade.	Para uma transferência FTP entre cliente e servidor é necessário uma autenticação com username e password, apesar disso, é grande risco quanto à segurança porque a transferência de arquivos é feita sem encriptação, ou seja, se alguém intercetar o segmento TCP durante a transmissão, terá acesso a todas informações dos seus arquivos e credenciais pondo em risco o seu servidor local e computador local.

Figura – Tabela para comparação dos diferentes protocolos conforme os parâmetros pedidos

Protocolo	Uso da camada de transporte	Eficiência na transferência (tempo)	Complexidade	Segurança
TFTP – Trivial File Transfer Protocol	O TFTP usa o UDP como protocolo de transporte na porta 69, mas encaminha para outra porta, possibilitando outros pedidos.	Dado o TDTP usar o UDP, este acaba por ganhar em termos de tempo de transferência, pois não ter que efetuar uma conexão antes da transferência.	Dado o TFTP usar o UDP que é um protocolo de transporte não orientado à conexão e não fiável(não existe garantia na entrega de segmentos), o software usado utiliza pouca memória, e tem apenas cinco mensagens disponíveis, é considerado um protocolo com baixo nível de complexidade.	Comparando TFTP com os outros protocolos este é considerado o menos seguro, pois nem mecanismos de autenticação ou de encriptação fornece.

Figura – Tabela para comparação dos diferentes protocolos conforme os parâmetros pedidos(continuação)

Protocolo	Uso da camada de transporte	Eficiência na transferência (tempo)	Complexidade	Segurança
HTTP – HyperText Transfer Protocol	O HTTP usa o TCP na porta 80.	Dado que o HTTP usa uma única conexão TCP, contrariamente ao FTP, consegue alcançar um melhor tempo de resposta mesmo para o primeiro arquivo solicitado. Apesar de que existem algumas ineficiências pois o TCP estabelece uma conexão antes de transferir qualquer dados, ou seja, esse mecanismo é reiniciado para cada pedido de um ficheiro, resultando numa possível sobrecarga.	Dado o HTTP usar o TCP, e necessitar de estabelecer apenas uma conexão (de dados), é considerado um protocolo com algum nível de complexidade.	O HTTP tem vulnerabilidades que podem prejudicar os utilizadores, da mesma forma que o FTP. Para se conseguir uma conexão segura usando este protocolo deverá se usar o HTTPS.

Figura – Tabela para comparação dos diferentes protocolos conforme os parâmetros pedidos(continuação)

Protocolo	Uso da camada de transporte	Eficiência na transferência (tempo)	Complexidade	Segurança
SFTP – Secure File Transfer Protocol	O SFTP utiliza o TCP na porta 22.	Devido ao uso de encriptação, este protocolo acaba por ser o único devidamente seguro, mas por outro lado perde a nível de eficiência de transferência, pois o seu tempo de transmissão aumenta.	Dado o SFTP ser baseado no FTP, sendo as únicas diferenças que no SFTP apenas existe um canal, e a informação circula encriptada devido ao protocolo SSH, consideramos este protocolo também com algum nível de complexidade.	Como o SFTP é executado sobre o SSH, ele é inerentemente seguro. Ao reutilizar a conexão principal, nenhuma outra conexão é aberta entre o cliente e o servidor, resultando em uma única conexão segura e eficiente por meio de firewalls. Isto significa que mesmo que alguém intercepte os segmentos TCP estará impossibilitado de decifrar os dados transmitidos.

Figura – Tabela para comparação dos diferentes protocolos conforme os parâmetros pedidos(continuação)

Questão 4

Para esta questão achamos necessário incluir a mesma tabela inserida na questão anterior mas com a particularidade que agora comparamos os tempos entre as transferências do ficheiro 2 entre o Cliente 1 e o Servidor 1, e entre o Alfa e o Servidor 1. Eles estão situados em diferentes LANs, com ligações com diferentes larguras de banda, sendo que a ligação entre o Alfa e o Cliente 1 funciona com perdas, atrasos e duplicações.

Protocolo	Cliente 1(tempo em s)	Alfa(tempo em s)
FTP	0.63	2.10
TFTP	0.079	1.928
HTTP	3.9	2.3
SFTP	5	4

```
root@Cliente1:/tmp/pycore.41562/Cliente1.conf
ping-output
PING 10.1.1.1 (10.1.1.1) 56(84) bytes of data:
64 bytes from 10.1.1.1: icmp_req=1 ttl=62 time=0.656 ms
64 bytes from 10.1.1.1: icmp_req=2 ttl=62 time=0.327 ms
64 bytes from 10.1.1.1: icmp_req=3 ttl=62 time=0.335 ms
64 bytes from 10.1.1.1: icmp_req=4 ttl=62 time=0.914 ms
64 bytes from 10.1.1.1: icmp_req=5 ttl=62 time=0.371 ms
64 bytes from 10.1.1.1: icmp_req=6 ttl=62 time=0.371 ms
64 bytes from 10.1.1.1: icmp_req=7 ttl=62 time=0.424 ms
64 bytes from 10.1.1.1: icmp_req=8 ttl=62 time=0.423 ms
64 bytes from 10.1.1.1: icmp_req=9 ttl=62 time=0.452 ms
64 bytes from 10.1.1.1: icmp_req=10 ttl=62 time=0.497 ms
64 bytes from 10.1.1.1: icmp_req=11 ttl=62 time=0.391 ms
64 bytes from 10.1.1.1: icmp_req=12 ttl=62 time=0.408 ms
64 bytes from 10.1.1.1: icmp_req=13 ttl=62 time=0.408 ms
64 bytes from 10.1.1.1: icmp_req=14 ttl=62 time=1.05 ms
64 bytes from 10.1.1.1: icmp_req=15 ttl=62 time=0.369 ms
64 bytes from 10.1.1.1: icmp_req=16 ttl=62 time=0.411 ms
64 bytes from 10.1.1.1: icmp_req=17 ttl=62 time=0.689 ms
64 bytes from 10.1.1.1: icmp_req=18 ttl=62 time=0.379 ms
64 bytes from 10.1.1.1: icmp_req=19 ttl=62 time=0.399 ms
64 bytes from 10.1.1.1: icmp_req=20 ttl=62 time=0.320 ms
--- 10.1.1.1 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 18996ms
rtt min/avg/max/mdev = 0.320/0.479/1.052/0.194 ms
root@Cliente1:/tmp/pycore.41562/Cliente1.conf#

root@Alfa:/tmp/pycore.41562/Alfa.conf# ping -c 20 10.1.1.1 | tee file-ping-output
PING 10.1.1.1 (10.1.1.1) 56(84) bytes of data:
64 bytes from 10.1.1.1: icmp_req=1 ttl=61 time=11.4 ms
64 bytes from 10.1.1.1: icmp_req=2 ttl=61 time=6.60 ms
64 bytes from 10.1.1.1: icmp_req=3 ttl=61 time=5.43 ms
64 bytes from 10.1.1.1: icmp_req=4 ttl=61 time=5.38 ms
64 bytes from 10.1.1.1: icmp_req=5 ttl=61 time=6.23 ms
64 bytes from 10.1.1.1: icmp_req=6 ttl=61 time=6.13 ms
64 bytes from 10.1.1.1: icmp_req=7 ttl=61 time=5.40 ms
64 bytes from 10.1.1.1: icmp_req=8 ttl=61 time=5.39 ms
64 bytes from 10.1.1.1: icmp_req=9 ttl=61 time=6.25 ms
64 bytes from 10.1.1.1: icmp_req=10 ttl=61 time=5.42 ms
64 bytes from 10.1.1.1: icmp_req=11 ttl=61 time=6.69 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=13 ttl=61 time=5.33 ms
64 bytes from 10.1.1.1: icmp_req=14 ttl=61 time=6.21 ms
64 bytes from 10.1.1.1: icmp_req=15 ttl=61 time=5.33 ms
64 bytes from 10.1.1.1: icmp_req=16 ttl=61 time=6.53 ms
64 bytes from 10.1.1.1: icmp_req=17 ttl=61 time=5.46 ms
64 bytes from 10.1.1.1: icmp_req=18 ttl=61 time=5.46 ms (DUPL)
64 bytes from 10.1.1.1: icmp_req=19 ttl=61 time=5.30 ms
64 bytes from 10.1.1.1: icmp_req=20 ttl=61 time=5.42 ms
--- 10.1.1.1 ping statistics ---
20 packets transmitted, 19 received, +1 duplicates, 5% packet loss, time 19038ms
rtt min/avg/max/mdev = 5.304/6.105/11.465/1.333 ms
root@Alfa:/tmp/pycore.41562/Alfa.conf#
```

Figura 17 – Ping do Cliente1 para o Servidor1 e do Alfa para o Servidor1

Primeiro analisando quanto ao tempo de transferência, podemos retirar da tabela acima, que o tempo que mais aumentou, foi o da transferência por TFTP. A causa deve-se ao uso do UDP como transporte, fazendo o controlo de erros a nível da aplicação. Enquanto que a transferência por FTP, por utilizar o TCP como transporte, dado a ser fiável, utiliza internamente o mecanismo ACK para detetar informação em falta e automaticamente retransmite essa informação. Quanto à diminuição dos tempos com HTTP e SFTP apenas deve-se a fatores externos.

130	290.251494	10.3.3.1	10.1.1.1	TCP	74	49136 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=12158303 TSecr=0 WS=16
131	290.251748	10.1.1.1	10.3.3.1	TCP	74	http > 49136 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PERM=1 TSval=12158304 TSecr=12158303 WS=1
132	290.251753	10.1.1.1	10.3.3.1	TCP	74	http > 49136 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PERM=1 TSval=12158304 TSecr=12158303 WS=1
133	290.257657	10.3.3.1	10.1.1.1	TCP	66	49136 > http [ACK] Seq=1 Ack=1 Win=14608 Len=0 TSval=12158304 TSecr=12158304
134	290.257662	10.3.3.1	10.1.1.1	TCP	66	[TCP Dup ACK 133#1] 49136 > http [ACK] Seq=1 Ack=1 Win=14608 Len=0 TSval=12158304 TSecr=12158304
135	290.257766	10.3.3.1	10.1.1.1	TCP	66	[TCP Dup ACK 133#2] 49136 > http [ACK] Seq=1 Ack=1 Win=14608 Len=0 TSval=12158304 TSecr=12158304
136	290.257784	10.3.3.1	10.1.1.1	HTTP	179	GET /file2 HTTP/1.1
137	290.258913	10.1.1.1	10.3.3.1	TCP	66	http > 49136 [ACK] Seq=1 Ack=114 Win=14480 Len=0 TSval=12158305 TSecr=12158304
138	290.258913	10.1.1.1	10.3.3.1	TCP	534	[TCP segment of a reassembled PDU]
139	290.259084	10.1.1.1	10.3.3.1	HTTP	66	HTTP/1.1 404 Not Found (text/html)
140	290.265079	10.3.3.1	10.1.1.1	TCP	66	49136 > http [ACK] Seq=114 Ack=469 Win=15680 Len=0 TSval=12158306 TSecr=12158306
141	290.265088	10.3.3.1	10.1.1.1	TCP	66	49136 > http [RST, ACK] Seq=114 Ack=470 Win=15680 Len=0 TSval=12158306 TSecr=12158306
142	295.257678	00:00:00:aa:00:12	00:00:00:aa:00:16	ARP	42	Who has 10.1.1.1? Tell 10.1.1.254
143	295.257889	00:00:00:aa:00:16	00:00:00:aa:00:12	ARP	42	10.1.1.1 is at 00:00:00:aa:00:16
144	300.035846	10.1.1.254	224.0.0.5	OSPF	78	Hello Packet
145	300.114848	fe80::200:ff:feaa:1ff02::5	224.0.0.5	OSPF	90	Hello Packet
146	310.036862	10.1.1.254	224.0.0.5	OSPF	78	Hello Packet
147	310.130673	fe80::200:ff:feaa:1ff02::5	224.0.0.5	OSPF	90	Hello Packet
148	320.036735	10.1.1.254	224.0.0.5	OSPF	78	Hello Packet
149	320.145228	fe80::200:ff:feaa:1ff02::5	224.0.0.5	OSPF	90	Hello Packet

▶ Frame 134: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
 ▶ Ethernet II, Src: 00:00:00:aa:00:12 (00:00:00:aa:00:12), Dst: 00:00:00:aa:00:16 (00:00:00:aa:00:16)
 ▶ Internet Protocol Version 4, Src: 10.3.3.1 (10.3.3.1), Dst: 10.1.1.1 (10.1.1.1)
 ▶ Transmission Control Protocol, Src Port: 49136 (49136), Dst Port: http (80), Seq: 1, Ack: 1, Len: 0

Figura 18 – Captura no wireshark da transferência por HTTP usando a ligação não segura

No.	Time	Source	Destination	Protocol	Length	Info
35	75.605633	10.3.3.1	10.1.1.1	SSHv2	82	[TCP Retransmission] Encrypted request packet len=16
36	75.606556	10.1.1.1	10.3.3.1	TCP	78	ssh > 56450 [ACK] Seq=1650 Ack=1410 Win=17376 Len=0 TSval=12104642 TSecr=12104641 SLE=1394 SRE=1410
37	75.611867	10.3.3.1	10.1.1.1	SSHv2	114	Encrypted request packet len=48
38	75.612033	10.1.1.1	10.3.3.1	TCP	66	ssh > 56450 [ACK] Seq=1650 Ack=1458 Win=17376 Len=0 TSval=12104644 TSecr=12104643
39	75.612100	10.1.1.1	10.3.3.1	TCP	114	[TCP segment of a reassembled PDU]
40	75.617241	10.3.3.1	10.1.1.1	TCP	66	56450 > ssh [ACK] Seq=1458 Ack=1698 Win=19408 Len=0 TSval=12104644 TSecr=12104644
41	75.617241	10.3.3.1	10.1.1.1	TCP	66	[TCP Dup ACK 40#1] 56450 > ssh [ACK] Seq=1458 Ack=1698 Win=19408 Len=0 TSval=12104644 TSecr=12104644
42	75.617274	10.3.3.1	10.1.1.1	SSHv2	130	Encrypted request packet len=64
43	75.618340	10.1.1.1	10.3.3.1	TCP	130	[TCP segment of a reassembled PDU]
44	75.664032	10.3.3.1	10.1.1.1	TCP	66	56450 > ssh [ACK] Seq=1522 Ack=1762 Win=19408 Len=0 TSval=12104656 TSecr=12104645
45	78.864010	10.3.3.1	10.1.1.1	SSHv2	210	Encrypted request packet len=144
46	78.901836	10.1.1.1	10.3.3.1	TCP	66	ssh > 56450 [ACK] Seq=1762 Ack=1666 Win=19920 Len=0 TSval=12105467 TSecr=12105456
47	80.009997	10.1.1.254	224.0.0.5	OSPF	78	Hello Packet
48	80.172494	fe80::200:ff:feaa:1ff02::5	224.0.0.5	OSPF	90	Hello Packet
49	80.594383	10.1.1.1	10.3.3.1	TCP	130	[TCP segment of a reassembled PDU]
50	80.595943	10.3.3.1	10.1.1.1	TCP	66	56450 > ssh [ACK] Seq=1666 Ack=1826 Win=19408 Len=0 TSval=12105890 TSecr=12105890
51	83.024213	10.3.3.1	10.1.1.1	SSHv2	210	Encrypted request packet len=144
52	83.024213	10.3.3.1	10.1.1.1	SSHv2	210	[TCP Retransmission] Encrypted request packet len=144
53	83.033524	10.1.1.1	10.3.3.1	TCP	66	ssh > 56450 [ACK] Seq=1826 Ack=1810 Win=22464 Len=0 TSval=12106497 TSecr=12106496
54	83.033572	10.1.1.1	10.3.3.1	TCP	78	[TCP Dup ACK 53#1] ssh > 56450 [ACK] Seq=1826 Ack=1810 Win=22464 Len=0 TSval=12106497 TSecr=12106496 SLE=16

▶ Frame 1: 90 bytes on wire (720 bits), 90 bytes captured (720 bits)
 ▶ Ethernet II, Src: 00:00:00:aa:00:12 (00:00:00:aa:00:12), Dst: IPv6mcst_00:00:00:05 (33:33:00:00:00:05)
 ▶ Internet Protocol Version 6, Src: fe80::200:ff:feaa:12 (fe80::200:ff:feaa:12), Dst: ff02::5 (ff02::5)
 ▶ Open Shortest Path First

Figura 19 – Captura no wireshark da transferência por SFTP usando a ligação não segura

Pelas capturas efetuadas nesta ligação não segura retiramos as seguintes informações:

- Perda e duplicação de pacotes são problemas relacionados com protocolos orientados à mensagem como o UDP. No UDP quando uma mensagem é enviada ela pode chegar duplicada, fora de ordem ou nem chegar. Não existe nenhum mecanismo para resolver situações de perda de mensagem.
- Relativamente à perda de pacotes utilizando TCP, este deteta o segmento em falta e automaticamente retransmite esse segmento. Quanto à duplicação, pode acontecer de os dois pacotes chegarem ao destino, o TCP resolve esse problema de pacotes de IP duplicados. Podemos ver as mensagens de erro de duplicação e retransmissão nas figuras acima.

Conclusão

Quanto ao TCP concluímos que:

- orientado à conexão pois para conseguir transmitir os dados para um destino, é preciso, antes, estabelecer uma conexão com este. O protocolo TCP tem 3 fases durante uma conexão (estabelecimento da ligação, transferência de dados, e encerramento da ligação);
- bastante confiável pois há sempre garantia da entrega dos segmentos, devido aos seguintes fatores:
 1. Retransmissão de um segmento pois o TCP, ao enviar um segmento, inicializa um temporizador (mecanismo de *timeout*) para receber a confirmação do receção. Se, após o *timeout*, a confirmação não for recebida, o segmento é retransmitido;
 2. Controle de fluxo para que o recetor não receba mais segmentos do que a sua capacidade permite, evitando, assim, a perda de segmentos por parte do recetor (ex.: um transmissor rápido enviando segmentos para um recetor lento);
 3. Controle de congestionamento, ou seja, o transmissor não pode enviar mais segmentos do que a rede pode suportar, evitando, assim, a perda de segmentos durante a transmissão.

Quanto ao UDP concluímos que é não orientado à conexão, e não fiável. Sendo a aplicação em si que tem que lidar com perdas de segmentos, duplicação e atraso.

Concluímos que a maioria das aplicações utiliza o TCP por existir a certeza que os dados são entregues, e na transferência de ficheiros isso é muito importante. Contudo, para uma aplicação em que a velocidade na entrega dos dados é mais útil do que a confiabilidade (como, por exemplo, streaming de vídeo e áudio), o protocolo UDP é mais adequado.