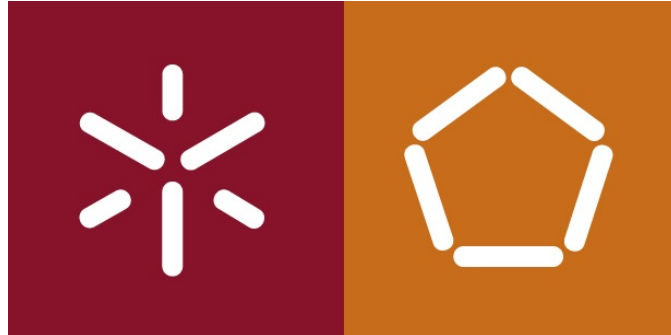


UNIVERSIDADE DO MINHO

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA



---

# Comunicações por Computadores

---

## RELATÓRIO DO TRABALHO PRÁTICO 1

### PROTOCOLOS DA CAMADA DE TRANSPORTE

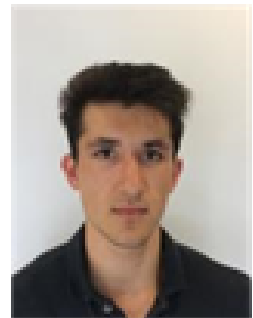
PL 6 GRUPO 3



Pedro Freitas  
A80975



Nuno Silva  
A78156



Shahzod Yusupov  
A82617

February 27, 2019

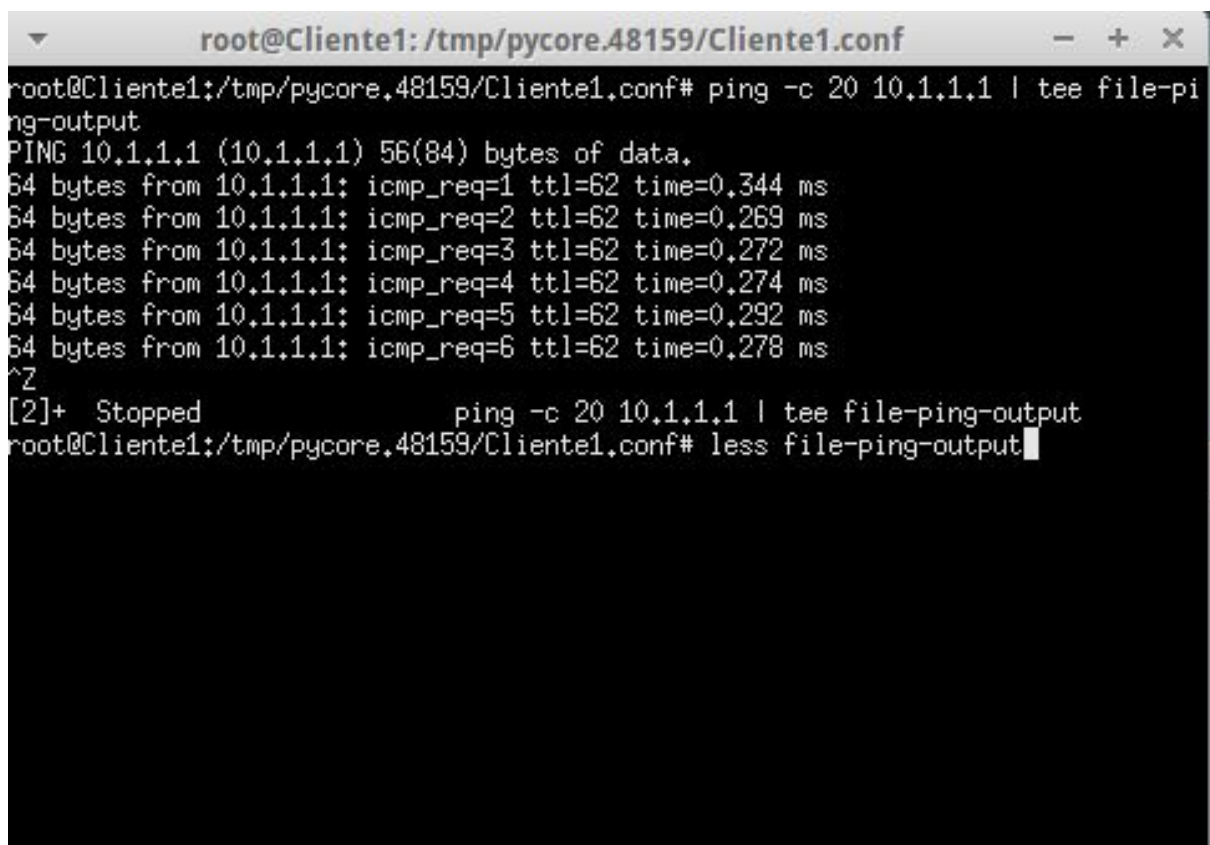
# Preparação

Para podermos realizar todo o guião foi necessário instalar a máquina virtual *core*, tal como na Unidade Curricular Redes de Computadores, assim como o Wireshark.

Numa primeira etapa foi necessário instalar alguns packages de softwares necessários para podermos concretizar o pedido como por exemplo SSH, FTP, TFTP e mini-http.

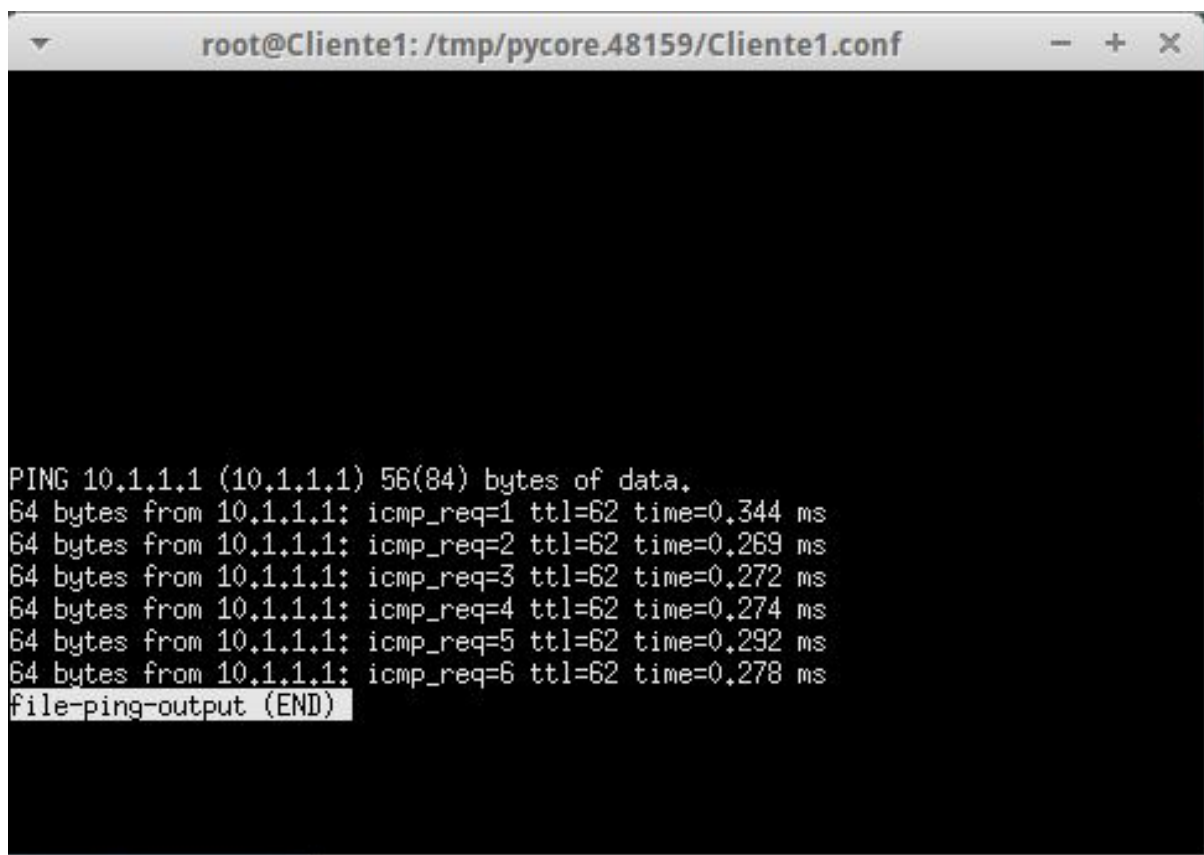
Numa segunda etapa tivemos de preparar uma pasta com dois ficheiros para serem disponibilizados pelos vários servidores.

Depois disso foi hora de emular o ficheiro fornecido no *core*. Começando por testar a conectividade com o Servidor1 com as Figuras 1,2 e 3 podemos ver que tanto o Cliente1 como o Alfa têm conectividade com ele.



```
root@Cliente1: /tmp/pycore.48159/Cliente1.conf
root@Cliente1:/tmp/pycore.48159/Cliente1.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=62 time=0.344 ms
64 bytes from 10.1.1.1: icmp_req=2 ttl=62 time=0.269 ms
64 bytes from 10.1.1.1: icmp_req=3 ttl=62 time=0.272 ms
64 bytes from 10.1.1.1: icmp_req=4 ttl=62 time=0.274 ms
64 bytes from 10.1.1.1: icmp_req=5 ttl=62 time=0.292 ms
64 bytes from 10.1.1.1: icmp_req=6 ttl=62 time=0.278 ms
^Z
[2]+  Stopped                  ping -c 20 10.1.1.1 | tee file-ping-output
root@Cliente1:/tmp/pycore.48159/Cliente1.conf# less file-ping-output
```

Figure 1: Cliente1 - Ping ao Servidor1

A terminal window with a title bar that reads 'root@Cliente1: /tmp/pycore.48159/Cliente1.conf'. The window contains a series of ping command outputs. The first line is 'PING 10.1.1.1 (10.1.1.1) 56(84) bytes of data.'. This is followed by six lines of successful ping responses, each showing '64 bytes from 10.1.1.1: icmp\_req=X ttl=62 time=Y ms' where X ranges from 1 to 6 and Y shows varying millisecond times. The final line is 'file-ping-output (END)' which is highlighted with a light blue background.

```
root@Cliente1: /tmp/pycore.48159/Cliente1.conf

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=0.344 ms
64 bytes from 10.1.1.1: icmp_req=2 ttl=62 time=0.269 ms
64 bytes from 10.1.1.1: icmp_req=3 ttl=62 time=0.272 ms
64 bytes from 10.1.1.1: icmp_req=4 ttl=62 time=0.274 ms
64 bytes from 10.1.1.1: icmp_req=5 ttl=62 time=0.292 ms
64 bytes from 10.1.1.1: icmp_req=6 ttl=62 time=0.278 ms
file-ping-output (END)
```

Figure 2: Cliente1 - Less ao output do ping

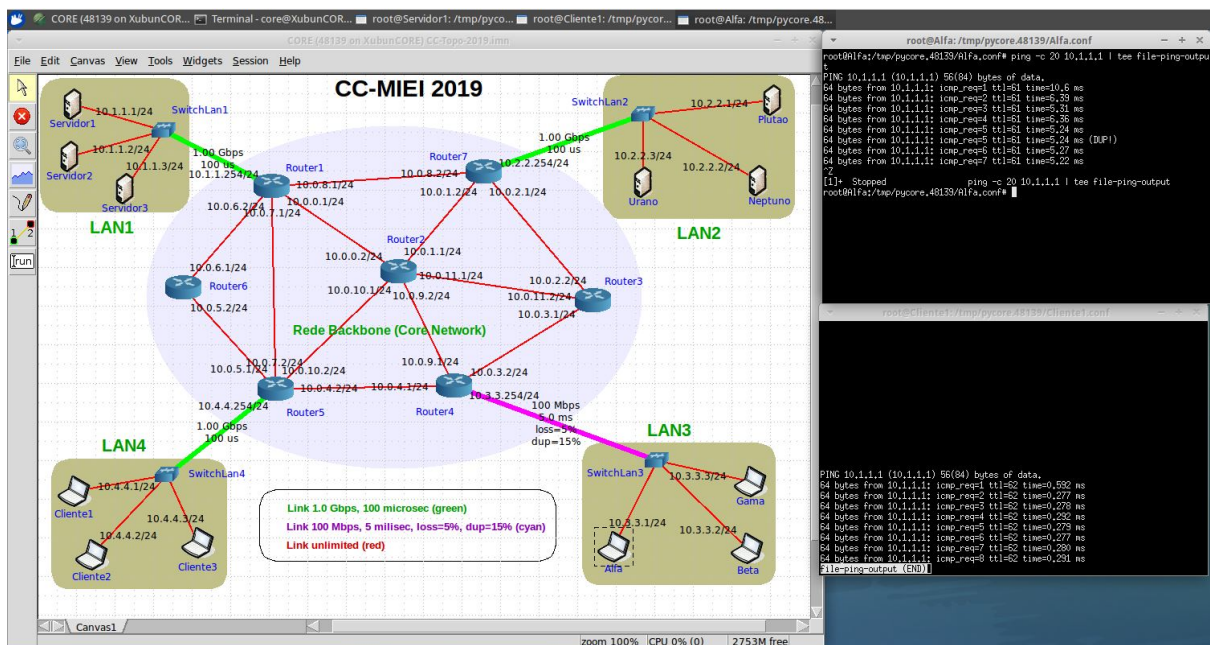


Figure 3: Alpha - Ping ao Servidor1

Depois foi hora de verificar se o servidor SSH já estaria em execução no Servidor1. Para isso com o comando *ps* verificaríamos se algum processo *sshd* estaria em execução e com o comando *netstat* se alguém estaria à escuta na porta 22. Através da Figura 4 podemos verificar que o servidor estava em execução e que havia protocolos à escuta.

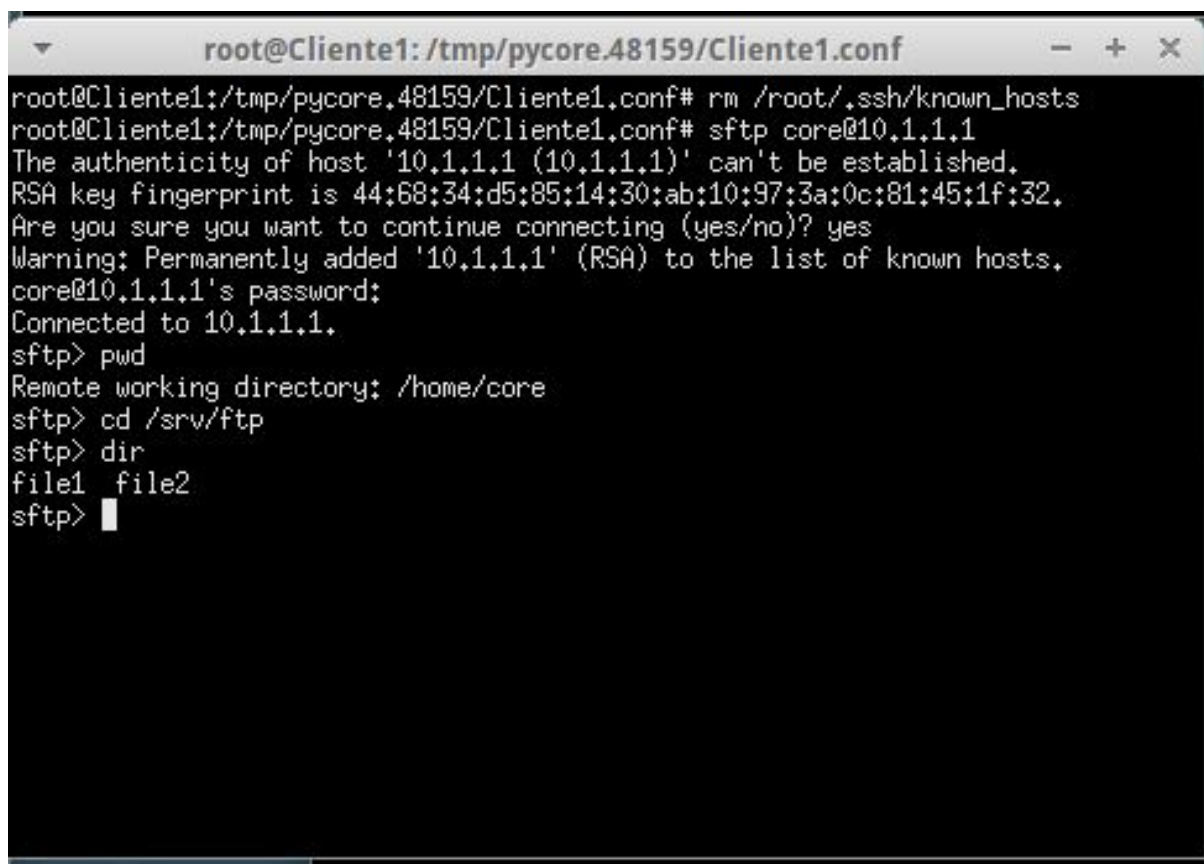
```

root@Servidor1: /tmp/pycore.48159/Servidor1.conf# ps -ef | grep ssh
root      32      1  0 11:58 ?        00:00:00 /usr/sbin/sshd -f /etc/ssh/sshd_config
root     114     33  0 12:27 pts/3    00:00:00 grep --color=auto ssh
root@Servidor1: /tmp/pycore.48159/Servidor1.conf# netstat -n -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:21              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp6       0      0 :::80                   :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
udp        0      0 10.1.1.1:69             0.0.0.0:*
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State       I-Node   Path
unix    2      [ ]       DGRAM     52027
root@Servidor1: /tmp/pycore.48159/Servidor1.conf#

```

Figure 4: Servidor1- ps e netstat

Para transferir o ficheiro por *sftp* já foi necessário uma série de comandos, que podem ser observados pelas Figuras 5 e 6.

A terminal window titled 'root@Cliente1: /tmp/pycore.48159/Cliente1.conf'. The terminal shows the following commands and output: 

```
root@Cliente1:/tmp/pycore.48159/Cliente1.conf# rm /root/.ssh/known_hosts
root@Cliente1:/tmp/pycore.48159/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 44:68:34:d5:85:14:30:ab:10:97:3a:0c:81:45:1f:32.
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> 
```

Figure 5: Conexão com SFTP para permitir transferência do ficheiro

```
root@Cliente1: /tmp/pycore.48139/Cliente1.conf
64 bytes from 10.1.1.1: icmp_req=6 ttl=62 time=0.277 ms
64 bytes from 10.1.1.1: icmp_req=7 ttl=62 time=0.280 ms
64 bytes from 10.1.1.1: icmp_req=8 ttl=62 time=0.291 ms
^Z
[1]+  Stopped                  ping -c 20 10.1.1.1 | tee file-ping-output
root@Cliente1:/tmp/pycore.48139/Cliente1.conf# less file-ping-output

[2]+  Stopped                  less file-ping-output
root@Cliente1:/tmp/pycore.48139/Cliente1.conf# rm /root/.ssh/known_hosts
rm: cannot remove '/root/.ssh/known_hosts': No such file or directory
root@Cliente1:/tmp/pycore.48139/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 7c:4f:8a:af:e1:50:cc:96:20:0c:98:90:89:3b:e8:63.
Are you sure you want to continue connecting (yes/no)? y
Please type 'yes' or '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 file1
```

Figure 6: Transferência do ficheiro

Quanto ao FTP foi necessário executar o servidor manualmente na sua bash com os comandos `chmod a-w /srv/ftp` e `vsftpd /etc/vsftpd.conf -o secure_chroot_dir = /srv/ftp -o anon_root = /srv/ftp`

Depois na bash do cliente foram executados os comandos que podem ser vistos nas Figuras 7 e 8:

```
root@Cliente1:~# ftp 10.1.1.1
Connected to 10.1.1.1.
220 (vsFTPd 2.3.5)
Name (10.1.1.1:root): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> status
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
Nmap: off
Hash mark printing: off; Use of PORT cmds: on
Tick counter printing: off
ftp> █
```

Figure 7: Conexão FTP ao Servidor1 e status



```
root@Cliente1: ~
Store unique: off; Receive unique: off
Case: off; CR stripping: on
Quote control characters: on
Ntrans: off
Nmap: 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.
-rw-r--r--  1 0      0      193 Feb 27 11:09 file1
-rwxr-xr-x  1 0      0    104508 Feb 27 11:10 file2
226 Directory send OK.
ftp> get file1
local: file1 remote: file1
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for file1 (193 bytes).
226 Transfer complete.
193 bytes received in 0.00 secs (5711.4 kB/s)
ftp> quit
221 Goodbye.
root@Cliente1:~#
```

Figure 8: FTP - transferência de ficheiro

*Quanto ao TFTP foi necessário preparar a diretoria através da bash do servidor e transferir através da bash do cliente, tal como está presente na Figura 9:*



```
root@Servidor1:/tmp/pycore.48159/Servidor1.conf
root@Servidor1:/tmp/pycore.48159/Servidor1.conf# chmod -R 777 /srv/ftp
No command 'chmod' found, did you mean:
  Command 'chmod' from package 'coreutils' (main)
chmod: command not found
root@Servidor1:/tmp/pycore.48159/Servidor1.conf# chmod -R 777 /srv/ftp
root@Servidor1:/tmp/pycore.48159/Servidor1.conf# touch atftpd.log
root@Servidor1:/tmp/pycore.48159/Servidor1.conf# atftpd --verbose=3 --user root
ftp --logfile atftpd.log --bind-address 10.1.1.1 --daemon --no-fork /srv/ftp/
atftpd: unrecognized option '--daemon'
root@Servidor1:/tmp/pycore.48159/Servidor1.conf# atftpd --verbose=3 --user root
ftp --logfile atftpd.log --bind-address 10.1.1.1 --daewon --no-fork /srv/ftp/
[]

root@Cliente1:/tmp/pycore.48159/Cliente1.conf
root@Cliente1:/tmp/pycore.48159/Cliente1.conf# atftp 10.1.1.1
ftp> status
connected: 10.1.1.1 port 69
mode:      octet
verbose:   off
trace:     off
options
  tsize:   disabled
  blksize: disabled
  timeout: disabled
  multicast: disabled
tftp variables
  client-port: 76
  mcast-ip:    0.0.0.0
  listen-delay: 2
  timeout-delay: 2
  last command: ---
ftp> []
```

Figure 9: TFTP- Preparação da diretoria e transferência do ficheiro

# Questões e Respostas

1.

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				
tracert	DNS	UDP	53	8
telnet	TELNET	TCP	23	40*
ftp	FTP	TCP	21	32*
Tftp	TFTP	UDP	69	8
wget	http	TCP	80	32*
nslookup	DNS	UDP	53	8
ssh	SSH	TCP	22	32*

\* O overhead típico de pacotes que usam protocolo de transporte TCP é 20, porém, se estes tiverem flags, o valor é maior.

Esta informação foi obtida das capturas de dados através do wireshark:

No.	Time	Source	Destination	Protocol	Length	Info
4	1.319581968	192.168.100.183	192.168.100.254	DNS	78	Standard query 0x4a17 A cisco.di.uminho.pt
5	1.319817521	fd24:b53e:c57c:0:b8...	fd24:b53e:c57c::1	DNS	98	Standard query 0x4a17 A cisco.di.uminho.pt
6	1.320022269	192.168.100.183	192.168.100.254	DNS	78	Standard query 0x6110 AAAA cisco.di.uminho.pt
7	1.320583673	192.168.100.254	192.168.100.183	DNS	541	Standard query response 0x4a17 A cisco.di.uminho.pt A 193.136.19.2...
8	1.320619535	192.168.100.254	192.168.100.183	DNS	127	Standard query response 0x6110 AAAA cisco.di.uminho.pt SOA dns.di...
9	1.321111919	192.168.100.183	193.136.19.254	UDP	74	54518 → 33434 Len=32
10	1.321173661	192.168.100.183	193.136.19.254	UDP	74	48882 → 33435 Len=32
11	1.321234607	192.168.100.183	193.136.19.254	UDP	74	43357 → 33436 Len=32
12	1.321288878	192.168.100.183	193.136.19.254	UDP	74	58171 → 33437 Len=32
13	1.321342693	192.168.100.183	193.136.19.254	UDP	74	48754 → 33438 Len=32

▶ Frame 4: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface 0  
 ▶ Ethernet II, Src: Asustekc\_34:ce:58 (38:d5:47:34:ce:58), Dst: Vmware\_d2:19:f0 (00:0c:29:d2:19:f0)  
 ▶ Internet Protocol Version 4, Src: 192.168.100.183, Dst: 192.168.100.254  
 ▶ User Datagram Protocol, Src Port: 55971, Dst Port: 53  
   Source Port: 55971  
   Destination Port: 53  
   Length: 44  
   Checksum: 0x9330 [unverified]  
   [Checksum Status: Unverified]  
   [Stream index: 0]  
 ▶ Domain Name System (query)

Figure 10: Captura de traceroute

8	1.551920684	192.168.100.183	193.136.9.183	TCP	74	60488 → 23 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK PERM=1 TSval=38159...
11	2.572587777	192.168.100.183	193.136.9.183	TCP	74	[TCP Retransmission] 60488 → 23 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SA...
35	4.588093207	192.168.100.183	193.136.9.183	TCP	74	[TCP Retransmission] 60488 → 23 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SA...
55	8.652476037	192.168.100.183	193.136.9.183	TCP	74	[TCP Retransmission] 60488 → 23 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SA...
170	16.844623619	192.168.100.183	193.136.9.183	TCP	74	[TCP Retransmission] 60488 → 23 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SA...
264	32.972614423	192.168.100.183	193.136.9.183	TCP	74	[TCP Retransmission] 60488 → 23 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SA...

▶ Frame 0: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0  
 ▶ Ethernet II, Src: AsustekC\_34:ce:58 (38:d5:47:34:ce:58), Dst: Vmware\_d2:19:f0 (00:0c:29:d2:19:f0)  
 ▶ Internet Protocol Version 4, Src: 192.168.100.183, Dst: 193.136.9.183  
 ▼ Transmission Control Protocol, Src Port: 60488, Dst Port: 23, Seq: 0, Len: 0  
   Source Port: 60488  
   Destination Port: 23  
   [Stream index: 0]  
   [TCP Segment Len: 0]  
   Sequence number: 0 (relative sequence number)  
   [Next sequence number: 0 (relative sequence number)]  
   Acknowledgment number: 0  
   1010 .... = Header Length: 40 bytes (10)  
   ▶ Flags: 0x002 (SYN)  
   Window size value: 29200  
   [Calculated window size: 29200]  
   Checksum: 0xcbb9 [unverified]

Figure 11: Captura de telnet

No.	Time	Source	Destination	Protocol	Length	Info
22	2.365729583	193.136.9.183	192.168.100.183	FTP	86	Response: 220 (vsFTPD 2.3.5)
102	13.784527308	192.168.100.183	193.136.9.183	FTP	82	Request: USER anonymous
104	13.786940070	193.136.9.183	192.168.100.183	FTP	100	Response: 331 Please specify the password.
116	15.618252105	192.168.100.183	193.136.9.183	FTP	74	Request: PASS p
117	15.645447093	193.136.9.183	192.168.100.183	FTP	89	Response: 230 Login successful.
119	15.645686978	192.168.100.183	193.136.9.183	FTP	72	Request: SYST
120	15.647431360	193.136.9.183	192.168.100.183	FTP	85	Response: 215 UNIX Type: L8

▶ Frame 102: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface 0  
 ▶ Ethernet II, Src: AsustekC\_34:ce:58 (38:d5:47:34:ce:58), Dst: Vmware\_d2:19:f0 (00:0c:29:d2:19:f0)  
 ▶ Internet Protocol Version 4, Src: 192.168.100.183, Dst: 193.136.9.183  
 ▼ Transmission Control Protocol, Src Port: 57188, Dst Port: 21, Seq: 1, Ack: 21, Len: 16  
   Source Port: 57188  
   Destination Port: 21  
   [Stream index: 0]  
   [TCP Segment Len: 16]  
   Sequence number: 1 (relative sequence number)  
   [Next sequence number: 17 (relative sequence number)]  
   Acknowledgment number: 21 (relative ack number)  
   1000 .... = Header Length: 32 bytes (8)  
   ▶ Flags: 0x018 (PSH, ACK)  
   Window size value: 229  
   [Calculated window size: 29312]  
   [Window size scaling factor: 128]

Figure 12: Captura de ftp

No.	Time	Source	Destination	Protocol	Length	Info
32	2.188462950	192.168.100.183	193.136.9.183	TFTP	86	Read Request, File: file1, Transfer type: octet, tsize=0, blksize=512, ...
68	10.390456790	192.168.100.183	193.136.9.183	TFTP	86	Read Request, File: file1, Transfer type: octet, tsize=0, blksize=512, ...
72	17.402235117	192.168.100.183	193.136.9.183	TFTP	86	Read Request, File: file1, Transfer type: octet, tsize=0, blksize=512, ...
82	24.410258822	192.168.100.183	193.136.9.183	TFTP	86	Read Request, File: file1, Transfer type: octet, tsize=0, blksize=512, ...

▶ Frame 32: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface 0 ▶ Ethernet II, Src: AsustekC_34:ce:58 (38:d5:47:34:ce:58), Dst: Vmware_d2:19:f0 (00:0c:29:d2:19:f0) ▶ Internet Protocol Version 4, Src: 192.168.100.183, Dst: 193.136.9.183 ▶ User Datagram Protocol, Src Port: 58003, Dst Port: 69 Source Port: 58003 Destination Port: 69 Length: 52 Checksum: 0x5eb5 [unverified] [Checksum Status: Unverified] [Stream index: 4] ▶ Trivial File Transfer Protocol
--

Figure 13: Captura de Tftp

14	2.338864820	192.168.100.183	193.136.9.240	HTTP	228	GET /disciplinas/CC-MIEI/ HTTP/1.1
15	2.339824717	193.136.9.240	192.168.100.183	TCP	66	80 → 54094 [ACK] Seq=1 Ack=163 Win=30080 Len=0 TSval=419534884 TSe...
16	2.341143280	193.136.9.240	192.168.100.183	TCP	5858	80 → 54094 [ACK] Seq=1 Ack=163 Win=30080 Len=5792 TSval=419534884 ...
17	2.341188083	192.168.100.183	193.136.9.240	TCP	66	54094 → 80 [ACK] Seq=163 Ack=5793 Win=40832 Len=0 TSval=2598579681...
18	2.341362138	192.168.100.183	193.136.9.240	HTTP	2597	HTTP/1.1 200 OK (text/html)

▶ Frame 14: 228 bytes on wire (1824 bits), 228 bytes captured (1824 bits) on interface 0 ▶ Ethernet II, Src: AsustekC_34:ce:58 (38:d5:47:34:ce:58), Dst: Vmware_d2:19:f0 (00:0c:29:d2:19:f0) ▶ Internet Protocol Version 4, Src: 192.168.100.183, Dst: 193.136.9.240 ▶ Transmission Control Protocol, Src Port: 54094, Dst Port: 80, Seq: 1, Ack: 1, Len: 162 Source Port: 54094 Destination Port: 80 [Stream index: 0] [TCP Segment Len: 162] Sequence number: 1 (relative sequence number) [Next sequence number: 163 (relative sequence number)] Acknowledgment number: 1 (relative ack number) 1680 ..... = Header Length: 32 bytes (8) ▶ Flags: 0x018 (PSH, ACK) Window size value: 229 [Calculated window size: 29312] [Window size scaling factor: 128] Checksum: 0xa335 [unverified] [Checksum Status: Unverified] Urgent pointer: 0 Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
---

Figure 14: Captura de browser/http

10	2.801384609	192.168.100.183	192.168.100.254	DNS	73	Standard query 0x03cb A www.uminho.pt
11	2.801921612	192.168.100.254	192.168.100.183	DNS	317	Standard query response 0x03cb A www.uminho.pt A 193.137.9.114 NS ...
12	3.037460492	Vmware_d2:19:f0	Broadcast	ARP	60	Who has 192.168.100.154? Tell 192.168.100.254
13	3.398759529	192.168.100.162	192.168.100.255	NBNS	110	Registration NB DESKTOP-A6JF4NO<20>
14	3.467664315	192.168.100.162	192.168.100.255	NBNS	110	Registration NB WORKGROUP<00>
15	3.467697395	192.168.100.162	192.168.100.255	NBNS	110	Registration NB DESKTOP-A6JF4NO<00>
16	4.037857756	Vmware_d2:19:f0	Broadcast	ARP	60	Who has 192.168.100.154? Tell 192.168.100.254
17	4.169535478	192.168.100.162	192.168.100.255	NBNS	110	Registration NB DESKTOP-A6JF4NO<20>

▶ Frame 10: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface 0 ▶ Ethernet II, Src: AsustekC_34:ce:58 (38:d5:47:34:ce:58), Dst: Vmware_d2:19:f0 (00:0c:29:d2:19:f0) ▶ Internet Protocol Version 4, Src: 192.168.100.183, Dst: 192.168.100.254 ▶ User Datagram Protocol, Src Port: 55971, Dst Port: 53 Source Port: 55971 Destination Port: 53 Length: 39 Checksum: 0x9648 [unverified] [Checksum Status: Unverified] [Stream index: 1] ▶ Domain Name System (query)
--

Figure 15: Captura de nslookup

No.	Time	Source	Destination	Protocol	Length	Info
81	10.972416619	192.168.100.183	193.136.9.183	SSHv2	107	Client: Protocol (SSH-2.0-openssh_7.2p2 Ubuntu-4ubuntu2.6)
83	11.066929819	193.136.9.183	192.168.100.183	SSHv2	197	Server: Protocol (SSH-2.0-OpenSSH_5.9p1 Debian-Subuntul.4)
85	11.067471194	192.168.100.183	193.136.9.183	SSHv2	1402	Client: Key Exchange Init
86	11.067663470	193.136.9.183	192.168.100.183	SSHv2	1050	Server: Key Exchange Init
88	11.046455957	192.168.100.183	193.136.9.183	SSHv2	146	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
90	11.059679969	193.136.9.183	192.168.100.183	SSHv2	378	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys
91	11.063734983	192.168.100.183	193.136.9.183	SSHv2	82	Client: New Keys
93	11.103543838	192.168.100.183	193.136.9.183	SSHv2	106	Client: Encrypted packet (len=40)
95	11.105295689	193.136.9.183	192.168.100.183	SSHv2	106	Server: Encrypted packet (len=40)
96	11.195537264	192.168.100.183	193.136.9.183	SSHv2	122	Client: Encrypted packet (len=56)
97	11.112785829	193.136.9.183	192.168.100.183	SSHv2	122	Server: Encrypted packet (len=56)
98	11.113143063	192.168.100.183	193.136.9.183	SSHv2	426	Client: Encrypted packet (len=360)
99	11.126683385	193.136.9.183	192.168.100.183	SSHv2	122	Server: Encrypted packet (len=56)
151	16.511840140	192.168.100.183	193.136.9.183	SSHv2	202	Client: Encrypted packet (len=136)
153	16.579211219	193.136.9.183	192.168.100.183	SSHv2	90	Server: Encrypted packet (len=24)
155	16.579641465	192.168.100.183	193.136.9.183	SSHv2	178	Client: Encrypted packet (len=112)

Frame 81: 107 bytes on wire (856 bits), 107 bytes captured (856 bits) on interface 0						
Ethernet II, Src: AsusTek-80:14:ee:50:00:00, Dst: VMware-d2:19:f0:00:0c:29:00:00:00						
Internet Protocol Version 4, Src: 192.168.100.183, Dst: 193.136.9.183						
Transmission Control Protocol, Src Port: 43978, Dst Port: 22, Seq: 1, Ack: 1, Len: 41						
Source Port: 43978 Destination Port: 22 [Stream index: 0] [TCP Segment Len: 41] Sequence number: 1 (relative sequence number) [Next sequence number: 42 (relative sequence number)] Acknowledgment number: 1 (relative ack number) 1000 ..... = Header Length: 32 bytes (8) Flags: 0x018 (PSH, ACK) Window size value: 229 [Calculated window size: 29312] [Window size scaling factor: 128]						

Figure 16: Captura de ssh

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. (Nota: a transferência por FTP envolve mais que uma conexão FTP, nomeadamente uma de controlo [ftp] e outra de dados [ftp-data]). Faça o diagrama apenas para a conexão de transferência de dados do ficheiro mais pequeno). Para podermos responder a esta questão foi necessário recorreremos ao Wireshark enquanto executávamos as transferências a partir dos comandos da bash.

Primeiramente fizemos a transferência via TFP obtendo os seguintes pacotes de dados:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.1.1.254	224.0.0.5	OSPF	78	Hello Packet
2	0.001495	fe80::200:ff:fe02::5	ff02::5	OSPF	90	Hello Packet
3	4.397265	10.4.4.1	10.1.1.1	FTP	74	Request: TYPE I
4	4.397423	10.1.1.1	10.4.4.1	FTP	97	Response: 200 Switching to Binary mode.
5	4.397601	10.4.4.1	10.1.1.1	FTP	89	Request: PORT 10,4,4,1,130,162
6	4.397844	10.1.1.1	10.4.4.1	FTP	117	Response: 200 PORT command successful. Consider using PASV.
7	4.398168	10.4.4.1	10.1.1.1	FTP	78	Request: RETR file1
8	4.398287	10.1.1.1	10.4.4.1	TCP	74	ftp-data > 33442 [SYN] Seq=0 Win=14608 Len=0 MSS=1460 SACK_PERM=1 TSval=842447 TSecr=0 WS=16
9	4.398421	10.4.4.1	10.1.1.1	TCP	74	33442 > ftp-data [SYN, ACK] Seq=0 Ack=1 Win=14608 Len=0 MSS=1460 SACK_PERM=1 TSval=842447 TSecr=842447 WS=16
10	4.398547	10.1.1.1	10.4.4.1	TCP	66	ftp-data > 33442 [ACK] Seq=1 Ack=1 Win=14608 Len=0 TSval=842447 TSecr=842447
11	4.398621	10.1.1.1	10.4.4.1	FTP	130	Response: 150 Opening BINARY mode data connection for file1 (193 bytes).
12	4.398624	10.1.1.1	10.4.4.1	FTP-DAT	259	FTP Data: 193 bytes
13	4.398668	10.1.1.1	10.4.4.1	TCP	66	ftp-data > 33442 [FIN, ACK] Seq=194 Ack=1 Win=14608 Len=0 TSval=842447 TSecr=842447
14	4.398993	10.4.4.1	10.1.1.1	TCP	66	33442 > ftp-data [ACK] Seq=1 Ack=194 Win=15552 Len=0 TSval=842447 TSecr=842447
15	4.398998	10.4.4.1	10.1.1.1	TCP	66	33442 > ftp-data [FIN, ACK] Seq=1 Ack=194 Win=15552 Len=0 TSval=842447 TSecr=842447
16	4.399005	10.1.1.1	10.4.4.1	TCP	66	ftp-data > 33442 [ACK] Seq=195 Ack=2 Win=14608 Len=0 TSval=842447 TSecr=842447
17	4.399508	10.1.1.1	10.4.4.1	FTP	90	Response: 226 Transfer complete.
18	4.399893	10.4.4.1	10.1.1.1	TCP	66	38394 > ftp [ACK] Seq=44 Ack=171 Win=913 Len=0 TSval=842447 TSecr=842447
19	8.596910	10.4.4.1	10.1.1.1	FTP	72	Request: QUIT
20	8.597488	10.1.1.1	10.4.4.1	FTP	80	Response: 221 Goodbye.
21	8.597668	10.1.1.1	10.4.4.1	TCP	62	38394 > 38394 [RST] Seq=4440000000 Len=0 TSval=842447 TSecr=842447

Figure 17: Captura da transferência via ftp

Através da análise dos vários pacotes de dados podemos concluir que o diagrama temporal da transferência seria o da Figura 18. Temos de tomar em especial atenção porque a captura dos pacotes capturou pacotes não relevantes para a transferência ( que pertenciam à parte do controlo) e que apesar do pacote número 13 ter chegado primeiro, o pacote número 14 foi enviado primeiro. Logo após a transferência o cliente enviou um [ACK] para o servidor, mas o cliente recebeu uma [FIN,ACK] do servidor antes que este tenha recebido o [ACK] dos dados.

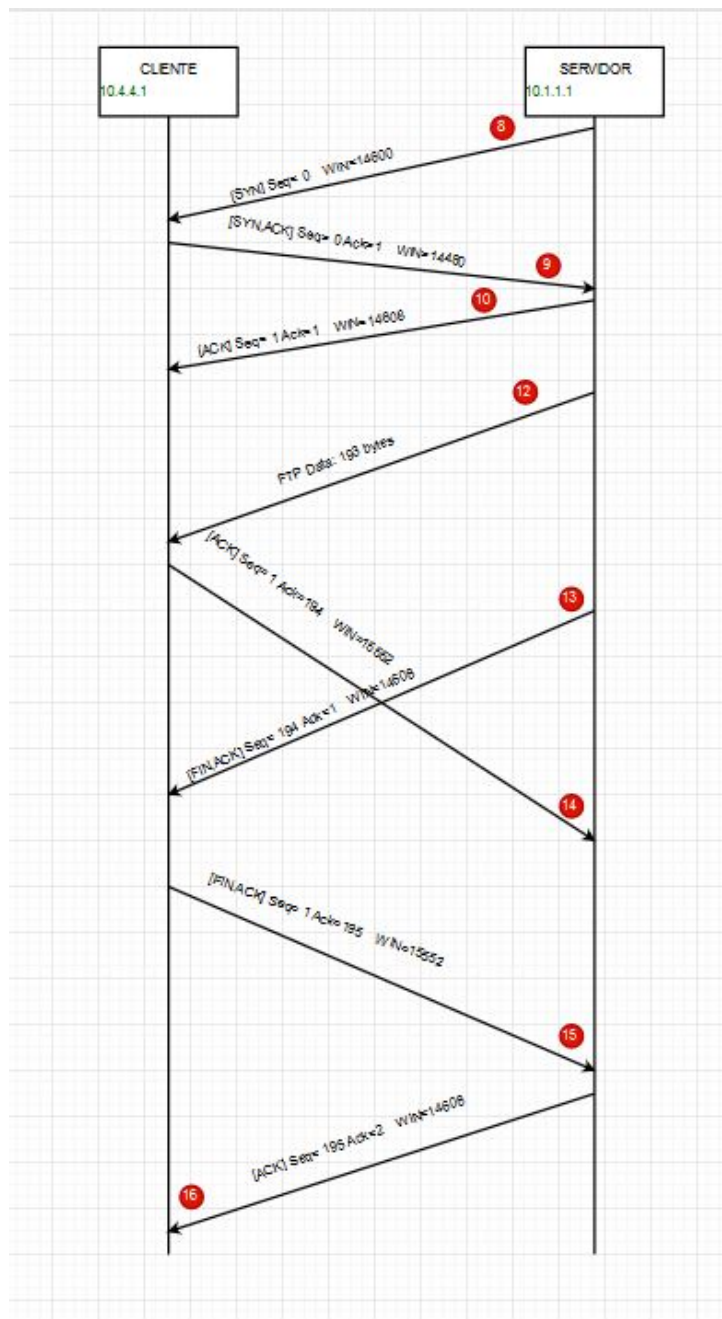


Figure 18: Linha temporal FTP

\* O valor dentro dos círculos vermelhos refere-se ao número do pacote de dados. \*

*Em relação ao TFTP o processo foi análogo, também sido capturado através do Wire-shark os pacotes de dados:*

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fe80::200:ff:feaa:1ff02::5		OSPF	90	Hello Packet
2	0.019277	10.1.1.254	224.0.0.5	OSPF	78	Hello Packet
3	0.019745	10.1.1.254	224.0.0.5	OSPF	78	hello Packet
4	0.030098	fe80::200:ff:feaa:1ff02::5		OSPF	90	Hello Packet
5	14.625956	00:00:00:aa:00:12	Broadcast	ARP	42	Who has 10.1.1.1? Tell 10.1.1.254
6	14.626115	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
7	14.626115	10.4.4.1	10.1.1.1	TFTP	56	Read Request, File: file1, Transfer type: octet
8	14.626441	10.1.1.1	10.4.4.1	TFTP	239	Data Packet, Block: 1 (last)
9	14.626772	10.4.4.1	10.1.1.1	TFTP	46	Acknowledgement, Block: 1
10	19.629285	00:00:00:aa:00:16	00:00:00:aa:00:12	ARP	42	Who has 10.1.1.254? Tell 10.1.1.1
11	19.629286	00:00:00:aa:00:12	00:00:00:aa:00:16	ARP	42	10.1.1.254 is at 00:00:00:aa:00:12
12	20.020180	10.1.1.254	224.0.0.5	OSPF	78	hello Packet
13	20.061254	fe80::200:ff:feaa:1ff02::5		OSPF	90	Hello Packet

Figure 19: Captura da transferência via tftp



*Analisando o tráfego de pacotes concluímos facilmente que o processo de transferência foi muito mais simples que o anterior:*

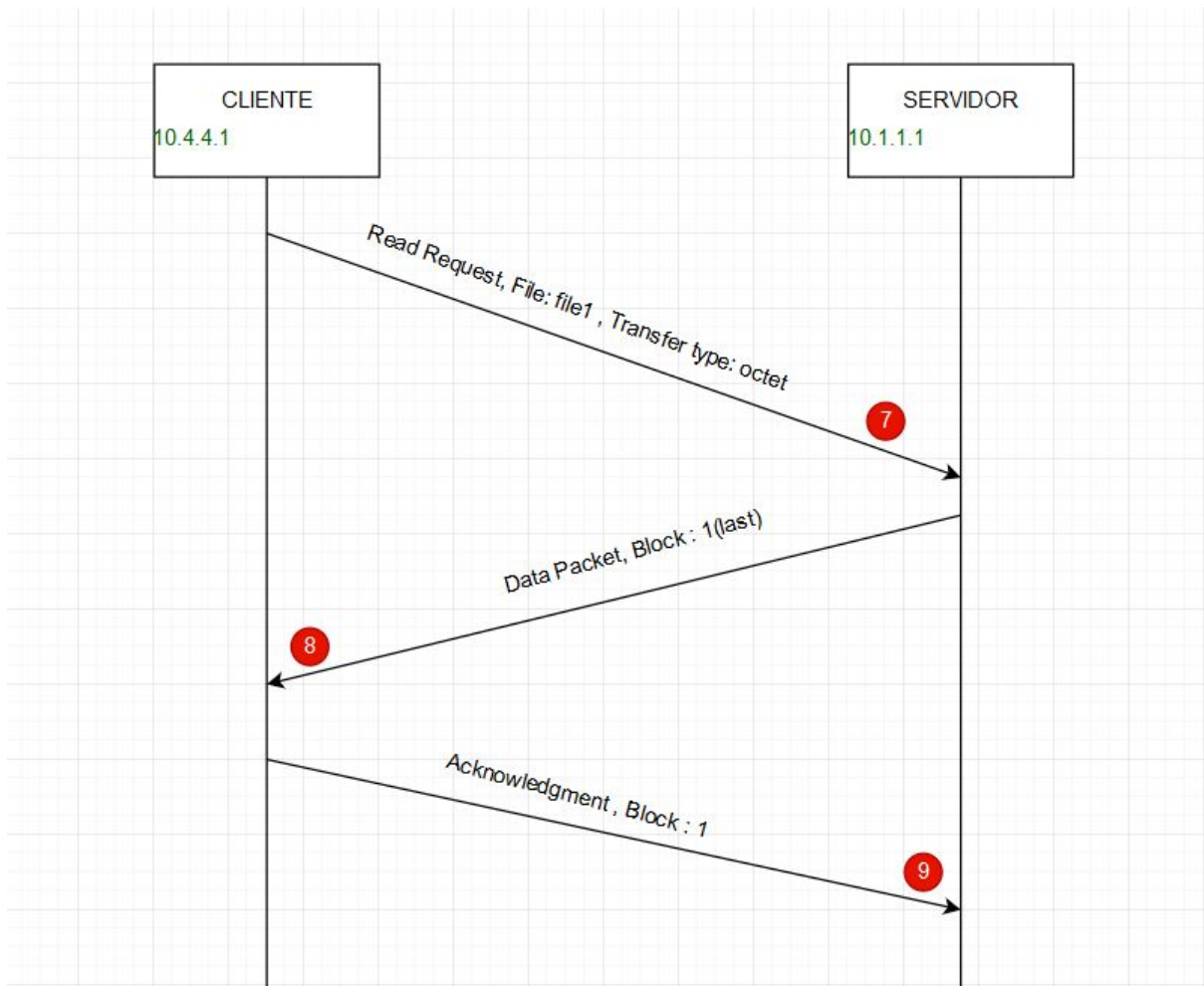


Figure 20: Linha temporal TFTP

*\* O valor dentro dos círculos vermelhos refere-se ao número do pacote de dados. \**

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*

- **FTP** Protocolo de transporte : TCP .
- **TFTP** Protocolo de transporte : UDP .
- **HTTP** Protocolo de transporte : TCP .
- **SFTP** Protocolo de transporte : TCP .

*ii) Eficiência na transferência* Para este tópico nós interpretamos eficiência como a junção de menor tempo de transferência e fiabilidade máxima (garantia que existe a transferência)

- **FTP** Garante a transferência porém é mais lento que os restantes.
- **TFTP** Mais rápido porém não há garantia da receção de dados
- **HTTP** Escolha ideal visto que este é fiável e usa pipelining o que o torna bastante rápido
- **SFTP** Garante a transferência porém é mais lento que os restantes, exceto do FTP.

*iii) Complexidade*

- **FTP** Devido à fiabilidade na transferência de dados torna-se bastante complexo.
- **TFTP** Semelhante ao FTP mas mais simples e como é baseado em UDP torna-se menos complexo.
- **HTTP** De complexidade moderada
- **SFTP** Para além da fiabilidade, permite ainda acesso, transferência e gestão dos ficheiros, resultando numa elevada complexidade

*iv) Segurança* Por segurança nós entendemos a fiabilidade da transferência e também a forma como ela é enviada, de forma a não ser possível ser lida antes de chegar ao destino.

- **FTP** Pouco seguro, uma vez que a informação não é encriptada mas passada como texto.
- **TFTP** Não fornece qualquer tipo de segurança
- **HTTP** O HTTP não é encriptado e é vulnerável a ataques "man-in-the-middle" e espionagem, o que permite que invasores tenham acesso a contas de sites e informações confidenciais e modifiquem páginas da Web para injetar malware ou anúncios
- **SFTP** Garante elevada segurança, uma vez que assegura encriptação de dados através do ssh e, para além disso, possui autenticação do utilizador e do servidor.

*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).*

*Quando pretendemos realizar uma transferência de dados, o protocolo de transporte que é usado, é escolhido pelo protocolo de aplicação tentando fazer a decisão conforme o protocolo de transporte que se adequa mais à transferência em específico.*

*Se o protocolo de transporte utilizado for o UDP temos de estar conscientes que pode haver perdas de dados, porém, estas perdas não afeta drasticamente o funcionamento da aplicação. A possível perda de dados é porque não existe confirmação pela parte do recetor que recebeu de facto esses dados, sendo que se o recetor não receber o pacote esse pacote deixa simplesmente de existir. Por isso não existe pedido de reenvio não havendo assim influência no desempenho nem a existência de pacotes duplicados. Sendo também este protocolo orientado ao datagrama (não há conexão entre emissor e recetor) não existe congestão de dados não sendo afetado também o desempenho.*

*Se o protocolo usado for TCP já existe uma conexão entre emissor e recetor e garantia que o recetor recebe todos os dados transmitidos pelo emissor. Quando o processo de transferência de dados está a decorrer existe a probabilidade de haver congestão de dados provocando assim atrasos na transferência de pacotes. Esses atrasos muitas vezes são mal interpretados e confundidos com perdas de dados o que leva a um reenvio de dados não necessário, o que por si só afeta o desempenho do processo, assim como leva à duplicação de pacotes. Se o pacote de facto perder-se é necessário um reenvio do pacote que afeta diretamente o desempenho negativamente.*

# Conclusão

*Com este trabalho pudemos observar os diferentes comportamentos de diferentes protocolos, de aplicação e transporte, de forma a compreendê-los melhor.*

*Para tal, foram realizados vários testes que foram possíveis com a topologia core e juntamente com o Wireshark pudemos acompanhar todo o processo de transferência de dados. Assim foi-nos permitido diferenciar os protocolos TCP e UDP, verificando as suas diferenças.*

*Desta forma, podemos concluir que se quisermos optar por uma transferência de dados fiável com garantias que os dados são recebidos temos de optar por uma aplicação que utilize o protocolo TCP, porém não podemos exigir a maior rapidez e eficiência. Dentro das aplicações vistas que usam este protocolo devemos optar pela HTTP visto que esta utiliza pipelining tornando-a mais rápida e eficiente. Enquanto que se quisermos aproveitar a velocidade máxima possível e a perda de dados não for tão significativa assim devemos optar por aplicações que usam o protocolo de transporte UDP.*