

# Trabalho Prático Nº1 – Protocolos da Camada de Transporte (Grupo 6)

André Carvalho<sup>100818</sup>, Flávio Sousa<sup>100715</sup>, and João Longo<sup>95536</sup>

University of Minho

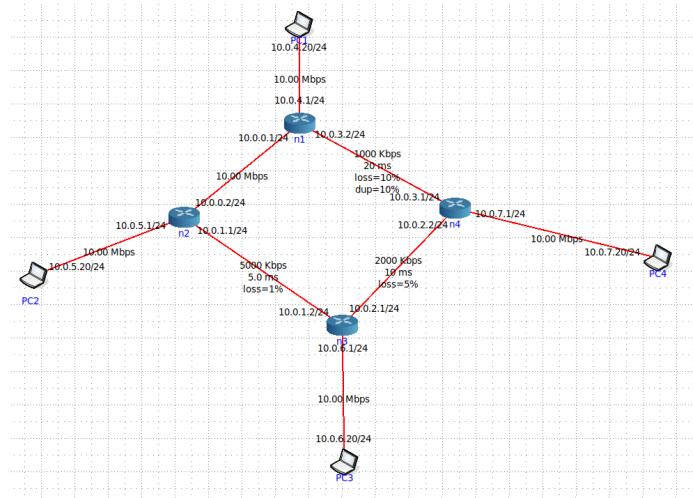
**Abstract.** Este documento descreve a utilização das ferramentas CORE e Wireshark para testar e analisar redes. Foram realizados testes de conectividade usando os comandos ping, traceroute, e iperf para avaliar o desempenho das ligações em termos de latência, largura de banda e perdas de pacotes. Além disso, foram feitas transferências de ficheiros entre dispositivos para observar o comportamento de diferentes protocolos de transporte em condições variadas de rede.

**Keywords:** CORE · Wireshark · TCP · UDP · TFTP · HTTP · FTP

## 1 Resposta às Questões

### 1.1 Parte I: Instalação, configuração e validação da rede de testes

**Pergunta 1.1** A Figura 1. ilustra a topologia analisada neste trabalho. Para a sua implementação, foi empregada a ferramenta CORE, seguindo rigorosamente todas as regras e restrições especificadas no enunciado.



**Fig. 1.** Topologia de Rede

**Pergunta 1.2** Para verificar a conectividade entre todos os hosts, foram empregadas três ferramentas, utilizando o PC1 como base para testar a comunicação com os demais hosts.

- **Ping** Este primeiro comando vai verificar a conectividade entre dois hosts em uma rede, medindo o tempo que os pacotes levam para se transmitirem do remetente ao destinatário. Assim, na Figura 2., obtemos os respetivos valores:

```

root@PC1:/tmp/pycore_4927/PC1.conf# ping 10.0.5.20
PING 10.0.5.20 (10.0.5.20) 56(84) bytes of data.
64 bytes from 10.0.5.20: icmp_seq=1 ttl=62 time=2.57 ms
64 bytes from 10.0.5.20: icmp_seq=2 ttl=62 time=2.06 ms
64 bytes from 10.0.5.20: icmp_seq=3 ttl=62 time=1.58 ms
c
--- 10.0.5.20 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 1.57/2.06/2.57/0.407 ms
root@PC1:/tmp/pycore_4927/PC1.conf# ping 10.0.6.20
PING 10.0.6.20 (10.0.6.20) 56(84) bytes of data.
64 bytes from 10.0.6.20: icmp_seq=1 ttl=61 time=13.1 ms
64 bytes from 10.0.6.20: icmp_seq=2 ttl=61 time=12.7 ms
64 bytes from 10.0.6.20: icmp_seq=3 ttl=61 time=12.3 ms
c
--- 10.0.6.20 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 12.34/12.73/13.14/20.524 ms
root@PC1:/tmp/pycore_4927/PC1.conf# ping 10.0.7.20
PING 10.0.7.20 (10.0.7.20) 56(84) bytes of data.
64 bytes from 10.0.7.20: icmp_seq=1 ttl=62 time=45.4 ms
64 bytes from 10.0.7.20: icmp_seq=2 ttl=62 time=45.3 ms
64 bytes from 10.0.7.20: icmp_seq=3 ttl=62 time=45.4 ms
c
--- 10.0.7.20 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms

```

Fig. 2. Ping

Sendo assim, podemos verificar de forma imediata que todos os hosts comunicam entre si.

- **Traceroute** Este segundo comando é uma ferramenta de diagnóstico de rede usada para determinar o caminho que os pacotes percorrem ao viajar de um host até outro. Na Figura 3. obtemos os caminhos que os pacotes percorrem ao longo do seu percurso:

<pre>traceroute to 10.0.5.20 (10.0.5.20), 30 hops max, 60 byte packets  1  10.0.4.1 (10.0.4.1)  0.497 ms  0.496 ms  1.971 ms  2  10.0.0.2 (10.0.0.2)  4.105 ms  4.090 ms  5.580 ms  3  10.0.5.20 (10.0.5.20)  6.891 ms  6.979 ms  7.177 ms root@PC1:/tmp/pycore.45727/PC1.conf# traceroute 10.0.6.20</pre>	TRACEROUTE do PC1 para o PC2
<pre>traceroute to 10.0.6.20 (10.0.6.20), 30 hops max, 60 byte packets  1  10.0.4.1 (10.0.4.1)  0.605 ms  1.123 ms  0.707 ms  2  10.0.0.2 (10.0.0.2)  2.984 ms  2.973 ms  2.964 ms  3  10.0.1.2 (10.0.1.2)  13.879 ms  13.868 ms  13.953 ms  4  10.0.6.20 (10.0.6.20)  12.776 ms  12.725 ms *</pre>	TRACEROUTE do PC1 para o PC3
<pre>root@PC1:/tmp/pycore.45727/PC1.conf# traceroute 10.0.7.20 traceroute to 10.0.7.20 (10.0.7.20), 30 hops max, 60 byte packets  1  10.0.4.1 (10.0.4.1)  0.961 ms  1.039 ms  0.492 ms  2  10.0.3.1 (10.0.3.1)  43.937 ms  43.925 ms  45.048 ms  3  10.0.7.20 (10.0.7.20)  45.498 ms * *</pre>	TRACEROUTE do PC1 para o PC4

**Fig. 3.** Traceroute

- **Iperf** Por último, este comando tem como principal objetivo medir a largura de banda efetiva entre dois hosts em uma rede e testar a estabilidade e qualidade da conexão. Na Figura 4. obtemos os valores para os diferentes hosts na nossa topologia: **Nota:** Para a execução deste comando, foi inicializado um servidor no PC1 e feita a solicitação nos restantes hosts para o PC1.

<pre>root@PC2:/tmp/pycore.45727/PC2.conf# iperf -c 10.0.4.20 Client connecting to 10.0.4.20, TCP port 5001 TCP window size: 110 KByte (default) [ 3] local 10.0.5.20 port 41678 connected with 10.0.4.20 port 5001 [ ID] Interval Transfer Bandwidth [ 3] 0.0-10.9 sec 14.2 MBytes 11.0 Mbits/sec root@PC2:/tmp/pycore.45727/PC2.conf#</pre>	IPERF do PC2 para o PC1
<pre>root@PC3:/tmp/pycore.45727/PC3.conf# iperf -c 10.0.4.20 Client connecting to 10.0.4.20, TCP port 5001 TCP window size: 85.0 KByte (default) [ 3] local 10.0.6.20 port 52832 connected with 10.0.4.20 port 5001 [ ID] Interval Transfer Bandwidth [ 3] 0.0-10.0 sec 2.25 MBytes 1.88 Mbits/sec root@PC3:/tmp/pycore.45727/PC3.conf#</pre>	IPERF do PC3 para o PC1
<pre>root@PC4:/tmp/pycore.45727/PC4.conf# iperf -c 10.0.4.20 Client connecting to 10.0.4.20, TCP port 5001 TCP window size: 85.0 KByte (default) [ 3] local 10.0.7.20 port 33566 connected with 10.0.4.20 port 5001 [ ID] Interval Transfer Bandwidth [ 3] 0.0-12.2 sec 768 KBytes 515 Kbits/sec</pre>	IPERF do PC4 para o PC1

**Fig. 4.** Iperf

Com todos estes dados, a Tabela 1. representa de forma simplificada os dados organizados e as respetivas conclusões:

Os resultados obtidos e a perda média de pacotes variam devido à natureza probabilística do loss em cada caminho, que é definido por uma percentagem. Isso significa que, para cada pacote enviado, existe uma chance específica de

	PC1 -> PC2	PC1 -> PC3	PC1 -> PC4
Estimativa de Perda de Pacotes	0	5 %	10 %
Atrasos	0.04645 ms	13.45263 ms	44.36842 ms
Débitos	11 Mbits/s	1.88 Mbits/s	516 Kbits/s

**Table 1.** Conclusões

ser perdido. Dessa forma, em múltiplas execuções do mesmo comando, podemos tanto não registrar perdas quanto observar uma quantidade significativa de pacotes perdidos. Para calcular a perda média de pacotes, dividimos o número total de pacotes perdidos pelo número esperado de pacotes enviados (neste caso, 20) e multiplicamos o resultado por 100 para expressá-lo em termos percentuais.

O atraso foi determinado somando o valor do campo time de cada iteração do comando ping e dividindo esse total pelo número de pacotes recebidos, obtendo assim o valor médio do atraso.

Por fim, o débito foi diretamente obtido a partir do comando iperf, onde ele é apresentado no campo Bandwidth, indicando a taxa de transmissão de dados medida durante a execução.

Por último falta apenas concluir as rotas utilizadas na comunicação entre os diferentes hosts. Com a análise dos resultados com o comando traceroute podemos concluir as seguintes rotas:

- PC1 (10.0.4.20) -> N1 (10.0.4.1) -> N2 (10.0.0.2) -> PC2 (10.0.5.20)
- PC1 (10.0.4.20) -> N1 (10.0.4.1) -> N2 (10.0.0.2) -> N3 (10.0.1.2) -> PC2 (10.0.6.20)
- PC1 (10.0.4.20) -> N1 (10.0.4.1) -> N4 (10.0.3.1) -> PC4 (10.0.7.20)

### Pergunta 1.3

a) Para otimizar o desempenho da rede e melhorar o débito, reduzir o atraso e minimizar as perdas de pacotes, é fundamental ajustar as rotas, evitando os caminhos com maior atraso, perdas e menor largura de banda. De acordo com a configuração apresentada, a ligação entre os nós 4 e 1 é a mais problemática, devido ao elevado atraso, à baixa largura de banda e à significativa perda de pacotes.

A ligação direta entre n4 e n1 apresenta uma perda elevada (10%), um grande atraso (44.36842 ms), um baixo débito (1 Mbps) e duplicação de pacotes (10%). Assim, recomenda-se evitar este caminho e considerá-lo apenas como uma opção de recurso, no caso de falha das restantes rotas.

A solução ideal seria utilizar a rota n1 → n2 → n3 → n4. Embora esta rota apresente um desempenho degradado nos segmentos entre n2 e n3 (5 Mbps e 1% de perda) e n3 e n4 (2 Mbps e 5% de perda), é significativamente mais eficiente do que a ligação direta entre n4 e n1. Este caminho alternativo acumula um atraso total de 15 ms e uma perda global de 5.95%, com um débito médio de 2 Mbps, o que é consideravelmente superior à ligação entre n4 e n1.

Em suma, para melhorar o desempenho da rede, a estratégia mais eficiente é evitar o caminho direto entre n4 e n1, que apresenta as condições mais desfavoráveis. A rota alternativa  $n1 \rightarrow n2 \rightarrow n3 \rightarrow n4$  oferece um desempenho mais estável, com menor perda de pacotes, menor latência e um débito superior.

**b)** Na topologia apresentada é possível melhorar as rotas definidas dinamicamente pelo OSPF (Open Shortest Path First) através de algumas configurações específicas. O objetivo é maximizar o desempenho, minimizando as perdas de pacotes e o atraso, enquanto evita-se a utilização de ligações menos eficientes.

O OSPF determina as rotas com base no custo das ligações, que está normalmente associado à largura de banda. Dado que as ligações entre n1 e n2, n2 e n3, e n3 e n4 têm um desempenho significativamente melhor do que a ligação direta entre n4 e n1, será favorável refletir esse custo para evidenciar esta diferença de desempenho.

As ligações entre n4 e n1, com um débito de 1 Mbps, 20 ms de atraso, 10% de perdas e duplicação de pacotes, deve ter um custo muito elevado, desencorajando a sua utilização exceto em casos de falha de outras ligações. Já as ligações com maior largura de banda e menor atraso, como n1 para n2 (10 Mbps, 0 ms de atraso e 0% de perdas), devem ter custos mais baixos para que o OSPF os privilegie.

Por exemplo, o custo das ligações poderia ser ajustado da seguinte forma:

- **Ligação n1 → n2:** custo baixo (ex: 1)
- **Ligação n2 → n3:** custo moderado (ex: 2)
- **Ligação n3 → n4:** custo mais alto (ex: 5)
- **Ligação n4 → n1:** custo muito elevado (ex: 10)

Isto garante preferência das rotas  $n1 \rightarrow n2 \rightarrow n3 \rightarrow n4$ , evitando o caminho direto  $n4 \rightarrow n1$ .

Para melhorar a capacidade de resposta do OSPF em caso de falhas ou alterações nas condições da rede, pode-se ajustar os temporizadores *hello interval* e *dead interval*. A redução destes intervalos permite que o OSPF reaja mais rapidamente a alterações na rede, assegurando que as rotas são recalculadas de forma eficiente e sem atrasos desnecessários.

Por exemplo, definir um *hello interval* de 10 segundos e um *dead interval* de 40 segundos pode melhorar a convergência da rede, permitindo uma resposta mais rápida em caso de falhas nas ligações.

Em síntese, para melhorar as rotas dinamicamente definidas pelo OSPF na topologia em anel, o custo das ligações deve ser ajustado de acordo com o desempenho real de cada ligação. A ligação  $n4 \rightarrow n1$ , devido às suas condições desfavoráveis, deve ser atribuído um custo elevado para que o OSPF evite-o. Além disso, ajustar os temporizadores OSPF permitirá uma resposta mais rápida a falhas na rede. Estas mudanças ajudarão a melhorar a eficiência e o desempenho da rede, garantindo menor perda de pacotes e menor latência.

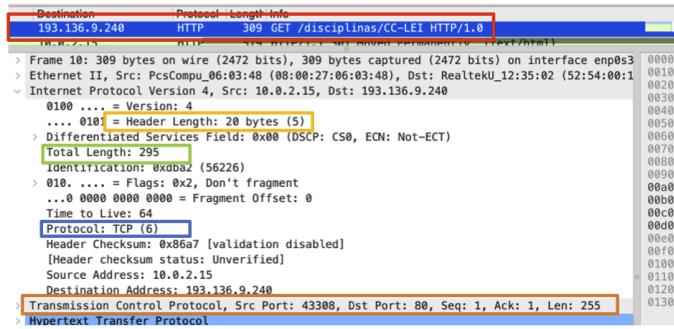
Comando usado (Aplicação)	Protocolo de Aplicação (Se aplicável)	Protocolo de Transporte (Se aplicável)	Porta de Atendimento (Se aplicável)	Overhead de Transporte em Bytes (Se aplicável)
WGET, LYNX OU VIA BROWSER	HTTP	TCP	80	20
SSH OU TFTP		TCP	21	20
FTP	FTP	TCP	69	8
TFTP	TFTP	UDP	-	-
TELNET	TELNET	TCP	23	20
NSLOOKUP OU DIG	DNS	UDP	53	8
PING	-	-	-	-
TRACEROUTE	-	UDP	-	8

**Table 2.** Uso da camada de transporte por parte das aplicações

## 1.2 Parte II: Uso da camada de transporte por parte das aplicações

O overhead do TCP varia entre 20 e 60 bytes, sendo 20 bytes o tamanho mínimo do cabeçalho, com o valor adicional decorrente de flags e opções extras. Um exemplo é o pacote SYN, usado na fase de estabelecimento de uma conexão, que contém apenas o cabeçalho com a flag SYN ativada e sem dados de aplicação, sendo considerado overhead por não transmitir carga útil.

**WGET** O pacote mostrado na Figura 5 é resultado da execução do comando ‘wget’, utilizando o protocolo HTTP, como indicado na primeira linha da captura. A linha 12 revela que o protocolo de transporte é TCP, e na linha 5 observa-se que o cabeçalho tem 20 bytes. O tamanho total do pacote é 295 bytes (linha 7), logo, o pacote TCP encapsulado possui 275 bytes (295 - 20). A penúltima linha mostra que a porta de destino é 80 e o payload TCP é de 255 bytes, resultando em um overhead de transporte de 20 bytes (275 - 255).

**Fig. 5.** WGET

**SSH** O pacote exibido na Figura 6 resulta da execução do comando ‘ssh’, utilizando o protocolo SSH, como indicado na primeira linha da captura. A linha 12 confirma que o protocolo de transporte é TCP. Na linha 5, observa-se que o cabeçalho tem 20 bytes, enquanto o tamanho total do pacote é 81 bytes (linha 7), resultando em um pacote TCP de 61 bytes (81 - 20). A penúltima linha indica que a porta de destino é 22 e que o payload TCP tem 41 bytes, o que leva a um overhead de transporte de 20 bytes (61 - 41).

No.	Time	Source	Destination	Protocol	Length	Info
1	8.0.102455945	193.136.9.201	193.136.9.201	SSHv2	95	Client: Protocol
	10.0.160327798	193.136.9.201	193.136.9.201	SSHv2	1175	Server: Protocol (SSH-)
>	Frame 8: 95 bytes on wire (760 bits), 95 bytes captured (760 bits) on interface enp0s3, id 0000					
>	Ethernet II, Src: PcsCompu_06:03:48 (08:00:27:06:03:48), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)					
>	Internet Protocol Version 4, Src: 10.0.2.15, Dst: 193.136.9.201					
>	0100 .... = Version: 4					
>	.... 0101 = Header Length: 20 bytes (5)					
>	Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)					
>	Total Length: 81					
>	Identification: 0x2386 (9094)					
>	010 .... = Flags: 0x2, Don't fragment					
>	...0 0000 0000 0000 = Fragment Offset: 0					
>	Time to Live: 64					
>	Protocol: TCP (6)					
>	Header Checksum: 0x3fc1 [validation disabled]					
>	[Header checksum status: Unverified]					
>	Source Address: 10.0.2.15					
>	Destination Address: 193.136.9.201					
>	Transmission Control Protocol, Src Port: 38102, Dst Port: 22, Seq: 1, Ack: 1, Len: 41					
>	SSH Protocol					

Fig. 6. SSH

**FTP** O pacote mostrado na Figura 7 resulta da execução do comando ‘-p’, utilizando o protocolo FTP, como indicado na primeira linha da captura. A linha 12 confirma que o protocolo de transporte é TCP. Na linha 5, observa-se que o cabeçalho tem 20 bytes, enquanto o tamanho total do pacote é 60 bytes (linha 7), resultando em um pacote TCP de 40 bytes (60 - 20). A penúltima linha mostra que a porta de origem é 21 e que o payload TCP tem 20 bytes, resultando em um overhead de transporte de 20 bytes (40 - 20).

No.	Time	Source	Destination	Protocol	Length	Info
1	6.0.027269578	193.137.214.36	10.0.2.15	FTP	74	Response: 220 (v)
	8.3.757186851	10.0.2.15	193.137.214.36	FTP	64	Request: USER ftp
>	Frame 6: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface enp0s3, id 0000					
>	Ethernet II, Src: RealtekU_12:35:02 (52:54:00:12:35:02), Dst: PcsCompu_06:03:48 (08:00:27:06:03:48)					
>	Internet Protocol Version 4, Src: 193.137.214.36, Dst: 10.0.2.15					
>	0100 .... = Version: 4					
>	.... 0101 = Header Length: 20 bytes (5)					
>	Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)					
>	Total Length: 60					
>	Identification: 0x023e (574)					
>	000 .... = Flags: 0x0					
>	...0 0000 0000 0000 = Fragment Offset: 0					
>	Time to Live: 64					
>	Protocol: TCP (6)					
>	Header Checksum: 0xd4c1 [validation disabled]					
>	[Header checksum status: Unverified]					
>	Source Address: 193.137.214.36					
>	Destination Address: 10.0.2.15					
>	Transmission Control Protocol, Src Port: 21, Dst Port: 52958, Seq: 1, Ack: 1, Len: 20					
>	File Transfer Protocol (FTP)					
>	220 (vsFTPd 3.0.3)\r\n					

Fig. 7. FTP

**TFTP** O pacote exibido na Figura 8 resulta da execução do comando ‘T-p‘, utilizando o protocolo TFTP, como indicado na primeira linha da captura. A linha 12 confirma que o protocolo de transporte é UDP. Na linha 5, observa-se que o cabeçalho tem 20 bytes, enquanto o tamanho total do pacote é 72 bytes (linha 7), resultando em um pacote UDP de 52 bytes (72 - 20). A linha 19 mostra que a porta de destino é 69 e, na linha 25, é informado que o payload UDP é de 44 bytes. Portanto, o overhead de transporte é de 8 bytes (52 - 44).

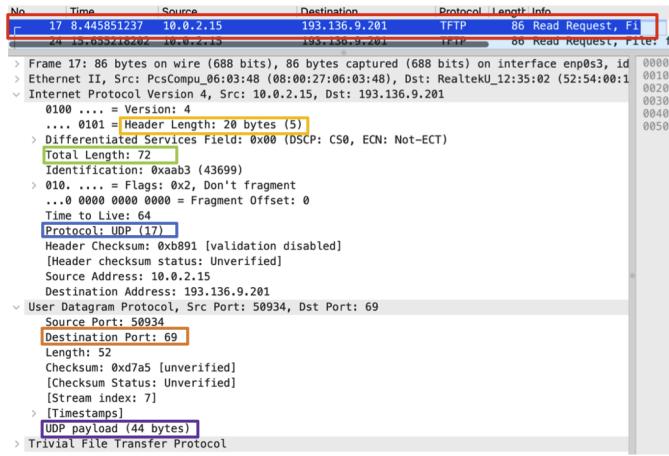


Fig. 8. TFTP

**TELNET** O pacote mostrado na Figura 9 resulta da execução do comando ‘telnet’, utilizando o protocolo TELNET, como indicado na primeira linha da captura. A linha 12 confirma que o protocolo de transporte é TCP. Na linha 5, observa-se que o cabeçalho tem 20 bytes, e o tamanho total do pacote é 67 bytes (linha 7), resultando em um pacote TCP de 47 bytes (67 - 20). A penúltima linha indica que a porta de destino é 23, e o campo “Len: 27” informa que o payload TCP tem 27 bytes. Assim, o overhead de transporte é de 20 bytes (47 - 27).

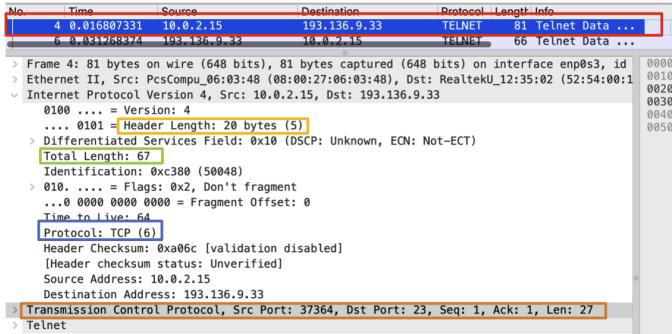
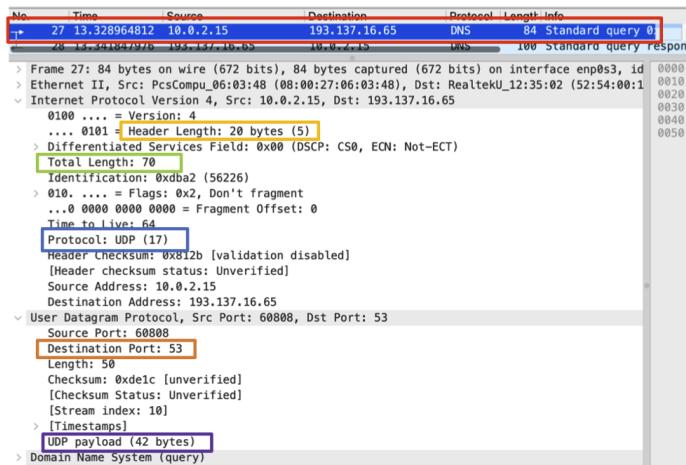


Fig. 9. TELNET

**NSLOOKUP** O pacote mostrado na Figura 10 resulta da execução do comando ‘nslookup’, utilizando o protocolo DNS, como indicado na primeira linha da captura. A linha 12 confirma que o protocolo de transporte é UDP. Na linha

5, observa-se que o cabeçalho tem 20 bytes, e o tamanho total do pacote é 70 bytes (linha 7), resultando em um pacote UDP de 50 bytes ( $70 - 20$ ). A linha 19 indica que a porta de destino é 53, e a linha 25 informa que o payload UDP tem 42 bytes. Assim, o overhead de transporte é de 8 bytes ( $50 - 42$ ).



**Fig. 10.** NSLOOKUP

**PING** A captura do Wireshark apresentada na Figura 11 mostra o tráfego da rede após a execução do comando ‘ping’, filtrando os resultados com o filtro “`icmp.type==8 || icmp.type==0`”. Este filtro é utilizado porque, quando o campo type do pacote ICMP é 8 ou 0, o pacote corresponde a um Echo Request ou a um Echo Reply, respectivamente, os tipos de pacotes gerados pelo comando ‘ping’. Como a execução do comando ‘ping’ resulta exclusivamente em pacotes ICMP, que é um protocolo da camada de rede destinado a fornecer informações sobre o estado e a conectividade da rede, não há dados para preencher a tabela, pois esta não se aplica ao presente caso.

No.	Time	Source	Destination	Protocol	Length	Info
7	0.067751506	10.0.2.15	172.217.168.163	ICMP	98	Echo (ping) request
8	0.098535087	172.217.168.163	10.0.2.15	ICMP	98	Echo (ping) reply
13	1.068815094	10.0.2.15	172.217.168.163	ICMP	98	Echo (ping) request
14	1.092758248	172.217.168.163	10.0.2.15	ICMP	98	Echo (ping) reply
15	2.069836868	10.0.2.15	172.217.168.163	ICMP	98	Echo (ping) request
16	2.092872733	172.217.168.163	10.0.2.15	ICMP	98	Echo (ping) reply
17	3.071375979	10.0.2.15	172.217.168.163	ICMP	98	Echo (ping) request
18	3.090737716	172.217.168.163	10.0.2.15	ICMP	98	Echo (ping) reply
21	4.071981928	10.0.2.15	172.217.168.163	ICMP	98	Echo (ping) request
22	4.088670821	172.217.168.163	10.0.2.15	ICMP	98	Echo (ping) reply
24	5.073756459	10.0.2.15	172.217.168.163	ICMP	98	Echo (ping) request
25	5.099926584	172.217.168.163	10.0.2.15	ICMP	98	Echo (ping) reply
29	6.075003937	10.0.2.15	172.217.168.163	ICMP	98	Echo (ping) request
30	6.092517954	172.217.168.163	10.0.2.15	ICMP	98	Echo (ping) reply
31	7.076561173	10.0.2.15	172.217.168.163	ICMP	98	Echo (ping) request
32	7.090560927	172.217.168.163	10.0.2.15	ICMP	98	Echo (ping) reply
33	8.076823699	10.0.2.15	172.217.168.163	ICMP	98	Echo (ping) request
34	8.094918545	172.217.168.163	10.0.2.15	ICMP	98	Echo (ping) reply
35	9.078059724	10.0.2.15	172.217.168.163	ICMP	98	Echo (ping) request
36	9.094096664	172.217.168.163	10.0.2.15	ICMP	98	Echo (ping) reply
37	10.079611950	10.0.2.15	172.217.168.163	ICMP	98	Echo (ping) request
38	10.098192551	172.217.168.163	10.0.2.15	ICMP	98	Echo (ping) reply
39	11.081384960	10.0.2.15	172.217.168.163	ICMP	98	Echo (ping) request
40	11.186167579	172.217.168.163	10.0.2.15	ICMP	98	Echo (ping) reply
41	12.083395098	10.0.2.15	172.217.168.163	ICMP	98	Echo (ping) request
42	12.087542399	172.217.168.163	10.0.2.15	ICMP	98	Echo (ping) reply
47	13.084530572	10.0.2.15	172.217.168.163	ICMP	98	Echo (ping) request
48	13.082424911	172.217.168.163	10.0.2.15	ICMP	98	Echo (ping) reply
49	14.086915415	10.0.2.15	172.217.168.163	ICMP	98	Echo (ping) request
50	14.108920578	172.217.168.163	10.0.2.15	ICMP	98	Echo (ping) reply

Fig. 11. PING

**TRACEROUTE** Na Figura 12, observa-se uma captura do Wireshark após a utilização do comando ‘traceroute’. Além dos esperados pacotes ICMP, que, como já mencionado no caso do ‘ping’, não se enquadram na tabela a ser preenchida, também foram registrados pacotes UDP, que transportam informações relacionadas ao ‘traceroute’, como o TTL (Time to Live). É com base em um desses pacotes que as informações da tabela foram preenchidas. O pacote selecionado corresponde à linha destacada na Figura 20 (primeira linha da captura). A linha 11 informa que o "Time to Live: 1" indica que o TTL é 1, o que está de acordo com o fato de que este é o primeiro pacote gerado pelo comando ‘traceroute’, sendo essencial para o seu funcionamento. Na linha 12, a informação "Protocol: UDP (17)" revela que o protocolo de transporte é UDP. A linha 5 mostra que o comprimento do cabeçalho do pacote é de 20 bytes, e na linha 7, que o tamanho total do pacote é 60 bytes, resultando em um pacote UDP de 40 bytes (60 - 20). Na linha 19, a "Destination Port: 33434" indica que o número da porta de destino é 33434. Este número corresponde ao primeiro pacote enviado como resultado do comando ‘traceroute’, podendo ser considerado uma porta de destino não no sentido tradicional, mas como um mecanismo auxiliar para que o comando alcance seu objetivo de identificar uma rota. Esses números permitem distinguir os diferentes saltos executados em uma determinada rota. Além disso, na linha 25, a informação "UDP payload (32 bytes)" indica que o payload do pacote UDP é de 32 bytes, resultando em um overhead de transporte de 8 bytes (40 - 32). Assim, em relação à tabela abaixo, para este caso, é possível evidenciar apenas dois dados significativos: o protocolo de transporte, que é UDP, e o overhead de transporte, que é de 8 bytes, sendo as demais informações não aplicáveis.

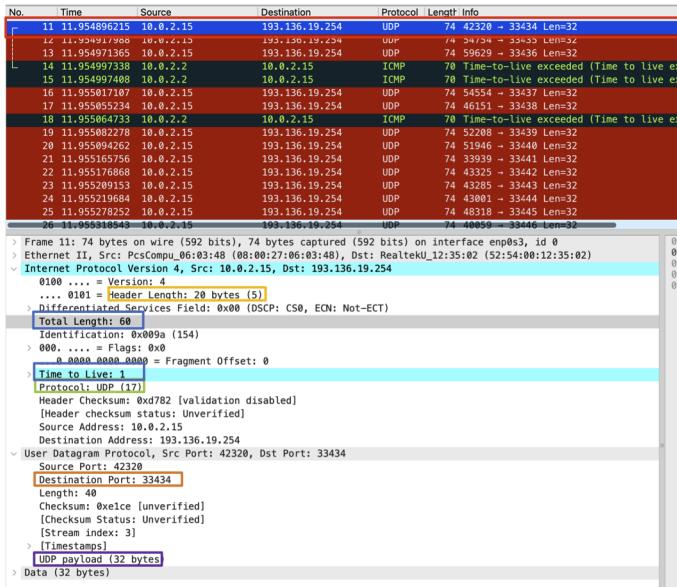


Fig. 12. TRACEROUTE

### 1.3 Parte III: Utilização de serviços de transferência de ficheiro no ambiente CORE

#### Pergunta 3.1

a) O TFTP é um protocolo simples de transferência de ficheiros que utiliza o UDP na camada de transporte. Ao contrário do TCP, o UDP não implementa controlo de fluxo, verificação de erros ou retransmissão automática de pacotes perdidos, o que torna o TFTP mais vulnerável a perdas de pacotes.

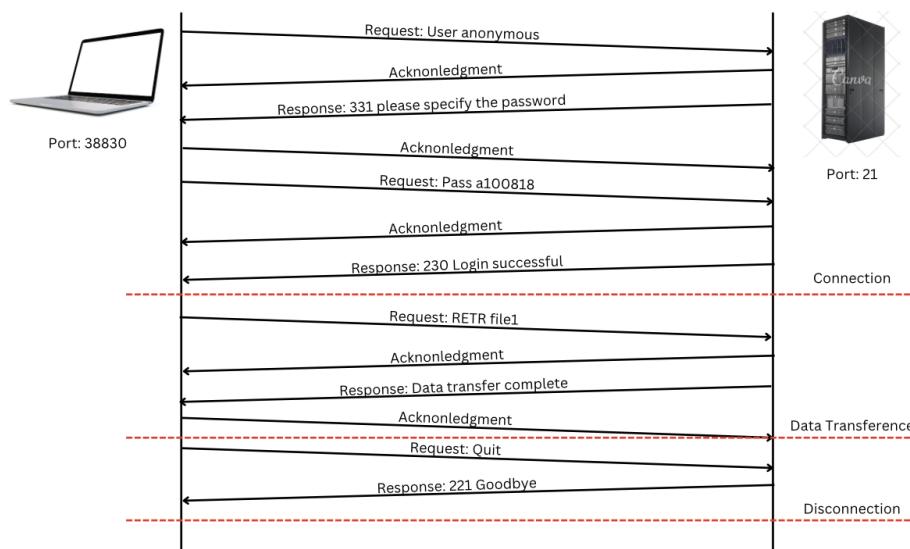
O FTP utiliza o TCP na camada de transporte, que implementa mecanismos fiáveis para lidar com as perdas de pacotes, como o controlo de fluxo, verificação de erros e retransmissão automática de pacotes perdidos.

Com base nas experiências realizadas, conclui-se que: - O TFTP sofre consideravelmente mais com as perdas de pacotes devido ao uso do UDP, que não oferece mecanismos de fiabilidade. As perdas de pacotes causam maiores atrasos e podem resultar em falhas na transferência. - O FTP, por outro lado, lida melhor com as perdas de pacotes, uma vez que o TCP implementa retransmissões automáticas e controlo de erros, o que proporciona uma transferência mais fiável, ainda que com algum impacto no tempo de transferência devido às retransmissões necessárias.

Em suma, as perdas de pacotes afetam de forma mais severa o TFTP do que o FTP. No TFTP, é a camada de aplicação que lida com as perdas, resultando em maior vulnerabilidade e ineficiência. No FTP, a camada de transporte (TCP)

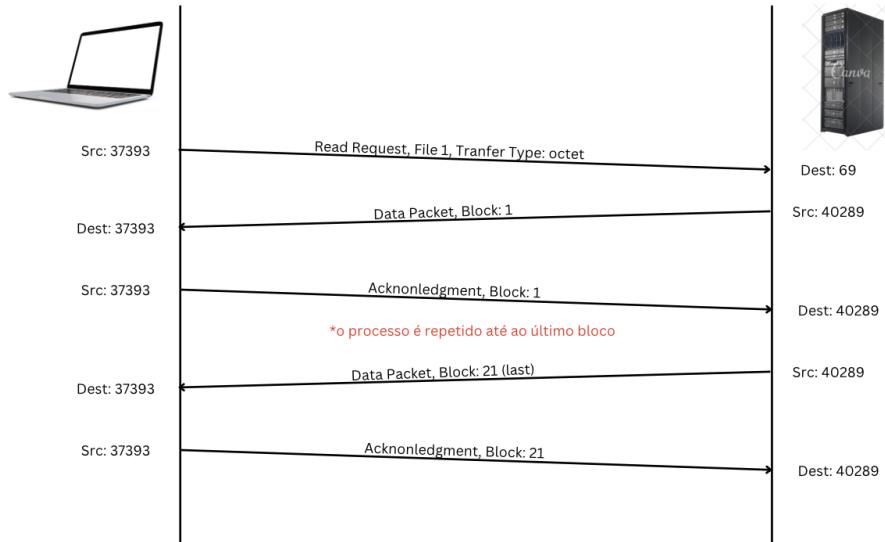
gere as perdas de forma eficaz, garantindo uma transferência mais fiável, apesar de possíveis atrasos adicionais provocados pelas retransmissões. Deste modo, o FTP é mais adequado para ambientes com perdas de pacotes, enquanto o TFTP apresenta maior degradação do desempenho nessas condições

b) .



**Fig. 13.** Diagrama Temporal - FTP

c) .

**Fig. 14.** Diagrama Temporal - TFTP**Pergunta 3.2**

- a) Primeiro, o cliente inicia a conexão enviando um segmento **SYN** com número de sequência 0 para o servidor. Nesta etapa, o **ACK** não é aplicável, pois é o início da comunicação. Em seguida, o servidor responde com um segmento **SYN-ACK**, onde o número de sequência do servidor é 0 e o número de **ACK** é 1, confirmando o recebimento do **SYN** enviado pelo cliente. Finalmente, o cliente envia um segmento **ACK** para o servidor, com número de sequência 1 e **ACK** 1, confirmando o recebimento do **SYN-ACK** do servidor. Neste processo, podemos observar que o número de **ACK** é sempre um incremento de 1 em relação ao último número de sequência enviado. Durante o período de transferência de dados, o número de **ACK** do cliente mantém-se constante em 142, enquanto o valor da sequência de resposta é igual a esse **ACK** e o número de sequência varia conforme os dados são trocados. Por fim, o cliente envia um segmento **FIN** com número de sequência 142 e um **ACK** 10462. O servidor, por sua vez, responde com um **ACK** igual a 143 e um número de sequência de 10462 igual ao **ACK**.

Op	Time	Source IP	Dest IP	Protocol	Port	Sequence Number	ACK Number	Data Length	Flags	Information
25	37.58578548	20.0.4.20	10.0.5.20	TCP	74	80 - 37436	[SYN, ACK] Seq=9 Ack=1 Win=5160 Len=0 MSS=1460 SACK_Rew=1 Tsvcl=2669593334 Tscr=3458410700 Ws=128			
26	37.58578548	20.0.4.20	10.0.5.20	TCP	74	80 - 37436	[SYN, ACK] Seq=9 Ack=1 Win=5160 Len=0 MSS=1460 SACK_Rew=1 Tsvcl=2669593334 Tscr=3458410700 Ws=128			
27	37.58615711	20.0.4.20	10.0.5.20	HTTP	207	GET /file1 HTTP/1.1				
28	37.58615711	20.0.4.20	10.0.5.20	TCP	1514	80 - 37436	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=2669593334 Tscr=3458410700 [TCP segment of a reassembled PDU]			
29	37.58137951	20.0.4.20	10.0.5.20	TCP	1514	80 - 37436	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=2669593334 Tscr=3458410700 [TCP segment of a reassembled PDU]			
30	37.58137951	20.0.4.20	10.0.5.20	TCP	1514	80 - 37436	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=2669593334 Tscr=3458410700 [TCP segment of a reassembled PDU]			
31	37.578027150	20.0.4.20	10.0.5.20	TCP	1514	80 - 37436	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=2669593334 Tscr=3458410700 [TCP segment of a reassembled PDU]			
32	37.578027150	20.0.4.20	10.0.5.20	TCP	1514	80 - 37436	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=2669593334 Tscr=3458410700 [TCP segment of a reassembled PDU]			
33	37.574497850	20.0.4.20	10.0.5.20	TCP	1514	80 - 37436	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=2669593334 Tscr=3458410700 [TCP segment of a reassembled PDU]			
34	37.574497850	20.0.4.20	10.0.5.20	TCP	1514	80 - 37436	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=2669593334 Tscr=3458410700 [TCP segment of a reassembled PDU]			
35	37.57572691	20.0.4.20	10.0.5.20	TCP	1514	80 - 37436	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=2669593334 Tscr=3458410700 [TCP segment of a reassembled PDU]			
36	37.57572691	20.0.4.20	10.0.5.20	TCP	1514	80 - 37436	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=2669593334 Tscr=3458410700 [TCP segment of a reassembled PDU]			
37	37.577087697	20.0.4.20	10.0.5.20	TCP	66	77436	[SYN, ACK] Seq=142 Win=52720 Len=8 Tsvcl=3458410700 Tscr=2669593337			
38	37.577087697	20.0.4.20	10.0.5.20	TCP	66	77436	[SYN, ACK] Seq=142 Win=52720 Len=8 Tsvcl=3458410700 Tscr=2669593337			
39	37.578242350	20.0.4.20	10.0.5.20	TCP	1514	80 - 37436	[ACK] Seq=142 Win=52720 Len=8 Tsvcl=3458410700 Tscr=2669593337			
40	37.578242350	20.0.4.20	10.0.5.20	TCP	1514	80 - 37436	[ACK] Seq=142 Win=52720 Len=8 Tsvcl=3458410700 Tscr=2669593337			
41	37.578242350	20.0.4.20	10.0.5.20	TCP	1514	80 - 37436	[ACK] Seq=142 Win=52720 Len=8 Tsvcl=3458410700 Tscr=2669593337			
42	37.578945140	20.0.4.20	10.0.5.20	TCP	66	77436 - 88 [ACK] Seq=142 Win=52720 Len=8 Tsvcl=3458410700 Tscr=2669593337				
43	37.578945140	20.0.4.20	10.0.5.20	TCP	66	89 - 37436	[SYN, ACK] Seq=142 Win=5024 Len=8 Tsvcl=2669593336 Tscr=3458410722			
44	37.580244744	20.0.4.20	10.0.5.20	TCP	66	77436 - 88 [ACK] Seq=142 Win=5160 Len=0 MSS=1460 SACK_Rew=1 Tsvcl=2669593337 Tscr=3458410723				
45	37.582279021	20.0.4.20	10.0.5.20	TCP	66	89 - 37436	[ACK] Seq=142 Win=5024 Len=8 Tsvcl=2669593336 Tscr=3458410722			

Fig. 15. Transferência HTTP para PC2

b) Considerando que a ligação entre o PC1 e o PC2 apresenta um débito de 10 Mbps, com reduzidos atrasos e sem perdas ou duplicações, como apresentado na Tabela 1, podemos optar pelo TFTP para transferir os dados. O TFTP é um protocolo simples que utiliza o UDP para a transmissão de pacotes, o que o torna mais rápido e eficiente em redes sem perdas e duplicações de pacotes, como é o caso do caminho entre o PC1 e o PC2. A principal vantagem de utilizar o TFTP nesta situação é que ele tem menos sobrecarga comparado a protocolos mais complexos, como o FTP ou HTTP, porque não implementa mecanismos de controlo de fluxo, como o TCP, nem precisa estabelecer uma conexão antes de começar a transferência. Em uma rede confiável, onde os pacotes não se perdem e não há necessidade de retransmissão, o TFTP é ideal, pois pode enviar dados de forma rápida e eficiente, sem desperdiçar tempo com controlo de congestionamento ou garantia de entrega, já que o caminho não apresenta problemas de confiabilidade.

### Pergunta 3.3

a) É possível identificar a duplicação e perda de pacotes devido à existência de pacotes que contém a tag [TCP Spurious Retransmission] no caso de duplicação e [TCP Retransmission] no caso da perda. A perda de pacotes causa atrasos e retransmissões e a duplicação causa um consumo desnecessário da largura de banda, o que afeta negativamente a eficiência da transferência. O TCP utiliza um mecanismo de Timeout e retransmissão para lidar com as perdas, ou seja, se o servidor não receber um ACK a confirmar a receção do pacote, após um período de tempo este é retransmitido.

Op	Time	Source IP	Dest IP	Protocol	Port	Sequence Number	ACK Number	Data Length	Flags	Information
12	16.054172475	20.0.7.20	10.0.4.20	TCP	74	77338 - 38 [SYN, ACK] Seq=9 Ack=1 Win=5160 Len=0 MSS=1460 SACK_Rew=1 Tsvcl=887687231 Tscr=3458410700 Ws=128				
13	16.054172475	20.0.7.20	10.0.4.20	TCP	74	77338 - 38 [SYN, ACK] Seq=9 Ack=1 Win=5160 Len=0 MSS=1460 SACK_Rew=1 Tsvcl=887687231 Tscr=3458410700 Ws=128				
14	16.04908414	20.0.7.20	10.0.4.20	HTTP	207	GET /file1 HTTP/1.1				
15	16.04908414	20.0.7.20	10.0.4.20	TCP	1514	80 - 37436	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=887687231 Tscr=3458410700 [TCP segment of a reassembled PDU]			
16	16.03609339	20.0.7.20	10.0.4.20	TCP	1514	80 - 37436	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=887687231 Tscr=3458410700 [TCP segment of a reassembled PDU]			
17	16.03609339	20.0.7.20	10.0.4.20	TCP	1514	80 - 37436	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=887687231 Tscr=3458410700 [TCP segment of a reassembled PDU]			
18	16.052255960	20.0.7.20	10.0.4.20	TCP	66	77338 - 88 [ACK] Seq=142 Win=52720 Len=8 Tsvcl=887687231 Tscr=3458410700 [TCP segment of a reassembled PDU]				
19	16.052255960	20.0.7.20	10.0.4.20	TCP	66	77338 - 88 [ACK] Seq=142 Win=52720 Len=8 Tsvcl=887687231 Tscr=3458410700 [TCP segment of a reassembled PDU]				
20	16.055142350	20.0.7.20	10.0.4.20	TCP	66	77338 - 88 [ACK] Seq=142 Win=52720 Len=8 Tsvcl=887687231 Tscr=3458410700 [TCP segment of a reassembled PDU]				
21	16.07995945	20.0.7.20	10.0.4.20	TCP	1514	80 - 37338	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=887687231 Tscr=3458410700 [TCP segment of a reassembled PDU]			
22	16.07995945	20.0.7.20	10.0.4.20	TCP	1514	80 - 37338	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=887687231 Tscr=3458410700 [TCP segment of a reassembled PDU]			
23	16.08983838	20.0.7.20	10.0.4.20	TCP	1514	80 - 37338	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=887687231 Tscr=3458410700 [TCP segment of a reassembled PDU]			
24	16.08983838	20.0.7.20	10.0.4.20	TCP	1514	80 - 37338	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=887687231 Tscr=3458410700 [TCP segment of a reassembled PDU]			
25	16.091715124	20.0.7.20	10.0.4.20	TCP	1514	80 - 37338	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=887687231 Tscr=3458410700 [TCP segment of a reassembled PDU]			
26	16.091715124	20.0.7.20	10.0.4.20	TCP	1514	80 - 37338	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=887687231 Tscr=3458410700 [TCP segment of a reassembled PDU]			
27	16.114360265	20.0.7.20	10.0.4.20	TCP	1514	80 - 37338	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=887687231 Tscr=3458410700 [TCP segment of a reassembled PDU]			
28	16.114360265	20.0.7.20	10.0.4.20	TCP	1514	80 - 37338	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=887687231 Tscr=3458410700 [TCP segment of a reassembled PDU]			
29	16.114360265	20.0.7.20	10.0.4.20	TCP	1514	80 - 37338	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=887687231 Tscr=3458410700 [TCP segment of a reassembled PDU]			
30	16.114360265	20.0.7.20	10.0.4.20	TCP	1514	80 - 37338	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=887687231 Tscr=3458410700 [TCP segment of a reassembled PDU]			
31	16.114360265	20.0.7.20	10.0.4.20	TCP	1514	80 - 37338	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=887687231 Tscr=3458410700 [TCP segment of a reassembled PDU]			
32	16.114360265	20.0.7.20	10.0.4.20	TCP	1514	80 - 37338	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=887687231 Tscr=3458410700 [TCP segment of a reassembled PDU]			
33	16.114360265	20.0.7.20	10.0.4.20	TCP	1514	80 - 37338	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=887687231 Tscr=3458410700 [TCP segment of a reassembled PDU]			
34	16.114360265	20.0.7.20	10.0.4.20	TCP	1514	80 - 37338	[ACK] Seq=142 Win=5024 Len=148 Tsvcl=887687231 Tscr=3458410700 [TCP segment of a reassembled PDU]			
35	16.114360265	20.0.7.20	10.0.4.20	TCP	66	77338 - 88 [ACK] Seq=142 Win=5160 Len=0 MSS=1460 SACK_Rew=1 Tsvcl=887687231 Tscr=3458410700 [TCP segment of a reassembled PDU]				
36	16.114360265	20.0.7.20	10.0.4.20	TCP	66	89 - 37338	[ACK] Seq=142 Win=5024 Len=8 Tsvcl=887687231 Tscr=3458410700			

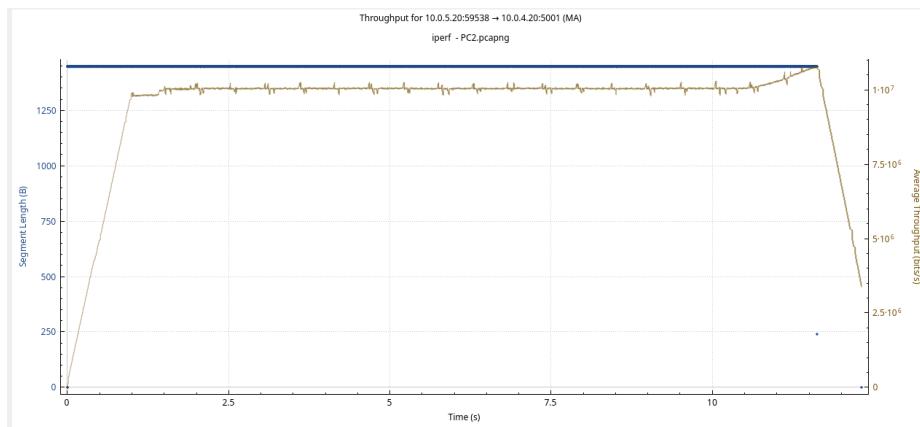
Fig. 16. Transferência HTTP para PC4

**b)** A ligação entre o PC1 e o PC4 é muito mais problemática, apresentando um cenário de 1 Mbps de débito, um atraso de 44 ms, 10% de perdas e 10% de duplicações de pacotes, como apresentado na Tabela 1. Nesta situação, seria crucial utilizar um protocolo mais robusto, que ofereça mecanismos de controlo de erro e garantia de entrega de pacotes, como o FTP, que utiliza o TCP.

O TCP é mais adequado para redes com perdas e duplicações porque ele garante a integridade dos dados, retransmitindo pacotes que perdem-se e garantindo que todos os pacotes sejam entregues na ordem correta. O uso do FTP neste caminho garantiria que os dados enviados do PC1 chegassem ao PC4 de forma íntegra e sem perda de informação, já que o protocolo é capaz de gerenciar congestionamento e garantir a entrega correta dos pacotes mesmo em condições de rede adversas.

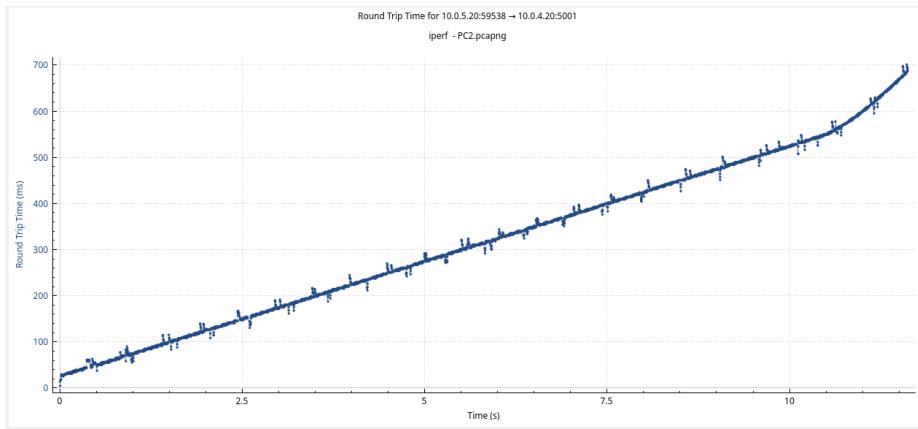
Portanto, para transmitir dados entre o PC1 e o PC4, o FTP, com o suporte do TCP, seria a melhor escolha, pois a sua robustez é necessária para solucionar os problemas de confiabilidade causados pelas perdas e duplicações presentes neste caminho.

**Pergunta 3.4** A Figura 15, mostra a taxa de transferência ou throughput da conexão TCP em bits por segundo (bps) ao longo do tempo, bem como o comprimento dos segmentos TCP. No início, o throughput cresce rapidamente, o que reflete o comportamento do Slow Start. Em torno de 1,5 a 2 segundos, o throughput atinge um valor máximo próximo à largura de banda disponível, e depois ele começa a oscilar de forma mais estável. As pequenas flutuações indicam pequenos ajustes que o TCP faz para lidar com a variação nas condições da rede (por mecanismos de Congestion Avoidance). Por volta de 10 segundos, o throughput diminui drasticamente, indicando que a transmissão foi finalizada ou algum evento de perda de pacotes que causou uma redução significativa.



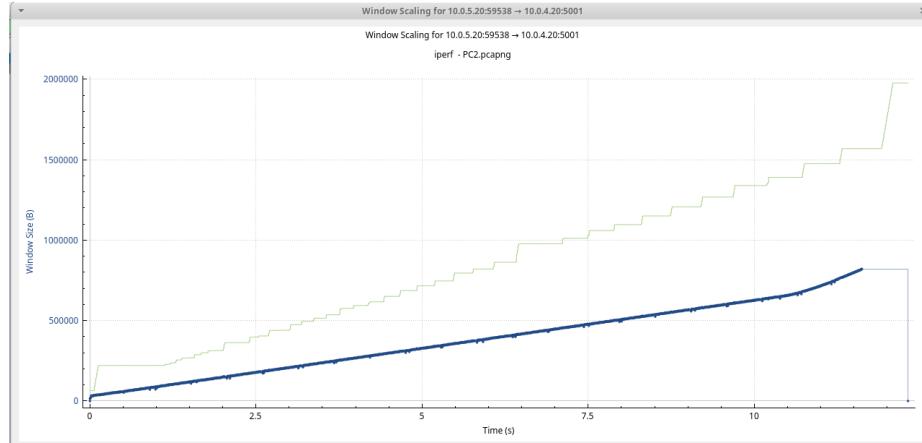
**Fig. 17.** Gráfico de Throughput

Na Figura 16, visualizamos o RTT - tempo que um pacote leva para ir e voltar entre o cliente e o servidor. O RTT começa baixo (em torno de 50 ms) e aumenta linearmente à medida que o tempo avança, indicando que a latência está a crescer com o aumento do congestionamento. Esse aumento no RTT é esperado durante o Slow Start e Congestion Avoidance, pois mais pacotes estão a ser enviados conforme a janela de congestionamento cresce. Próximo ao final, o RTT chega a cerca de 700 ms, indicando que a rede está congestionada e o tempo de resposta é significativamente mais lento.



**Fig. 18.** Gráfico de Round Trip Time

Por último, na Figura 17, o gráfico mostra o tamanho da janela de congestionamento (cwnd) ao longo do tempo em bytes. No início, o gráfico mostra um crescimento linear da janela, o que é típico durante a fase de Slow Start do TCP. Cada vez que um pacote é confirmado, a janela de envio é aumentada. Após cerca de 2 segundos, o crescimento da janela torna-se mais suave, o que indica que o algoritmo TCP entrou na fase de Congestion Avoidance. Aqui, o aumento da janela é mais conservador para evitar congestionamento.



**Fig. 19.** Gráfico de Window Size

## 2 Conclusões

Ao longo deste trabalho, explorámos a complexidade da interconexão de sistemas em redes de computadores, destacando o papel fundamental que os protocolos de transporte desempenham neste ecossistema. A nossa análise proporcionou uma visão abrangente sobre como diferentes protocolos influenciam as operações de rede, bem como os princípios subjacentes que regem estes sistemas.

Uma das lições mais valiosas que retirámos foi a importância de escolher o protocolo de transporte adequado para cada aplicação. Cada protocolo possui características distintas, e uma escolha criteriosa pode impactar significativamente tanto o desempenho como a segurança da comunicação. Por exemplo, protocolos como o SSH e o FTP, que utilizam o TCP como protocolo de transporte, asseguram a fiabilidade nas comunicações, embora introduzam um overhead maior devido aos cabeçalhos e mecanismos de controlo. Em contrapartida, protocolos que fazem uso do UDP, como o TFTP, são mais leves em termos de overhead, o que os torna apropriados para transferências de ficheiros em redes locais, embora sejam menos fiáveis.

Além disso, compreendemos que a complexidade de implementação varia entre os diferentes protocolos. Protocolos como o TFTP são notoriamente simples e leves, projetados para uma fácil implementação em redes locais. Por outro lado, protocolos como o SSH, que incorporam criptografia avançada para garantir a segurança, apresentam uma maior complexidade tanto na sua implementação como na sua manutenção ao longo do tempo.

A segurança emergiu como uma consideração crítica durante o nosso estudo. Verificámos que protocolos como o SSH oferecem um elevado nível de segurança, graças à robustez da criptografia que protege as comunicações. Em contraste, protocolos como o FTP são considerados inseguros por padrão, uma vez que as

informações são transmitidas sem encriptação, tornando-as vulneráveis a ataques de interseção (Man-in-the-Middle).

Em suma, as reflexões que surgiram ao longo deste trabalho expandiram a nossa compreensão das complexidades inerentes às redes de computadores e aos protocolos de transporte. A seleção cuidadosa dos protocolos, considerando fatores como eficiência, complexidade e segurança, é essencial para garantir o correto funcionamento e a segurança das operações de rede em diversos contextos.

## References

1. David, G., Brian, T., Marjorie, S., Sailu, R., Anshu, A.: HTTP The Definitive Guide. 1nd edn. O'REILLY, United States os America (2002)
2. LNCS Homepage, [https://link.springer.com/chapter/10.1007/978-0-387-74390-5<sub>3</sub>](https://link.springer.com/chapter/10.1007/978-0-387-74390-5_3), last accessed 2024/09/20