Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de Comunicações por Computador

Ano Letivo de 2024/2025

Trabalho Prático Nº 1
Grupo 5
Protocolos da Camada de Transporte

Tomás Henrique Alves Melo (A104529)
José Pedro Torres Vasconcelos (A100763)
João Gustavo Serrão (A104444)

11 de outubro de 2024

CC

Índice

Lista de Figuras

1 Parte 1

1.1 Questão 1

Defina em modo de edição uma topologia com quatro roteadores. Faça uma ligação do nó n1 para o nó n2, deste para o nó n3, e deste para o nó n4, resultando numa topologia em anel. Em cada um desses roteadores, ligue um host. Renomeie os hosts como PCx, onde x é o mesmo dígito que identifica o roteador a que está ligado. Por exemplo, PC1 é o host ligado ao roteador n1.

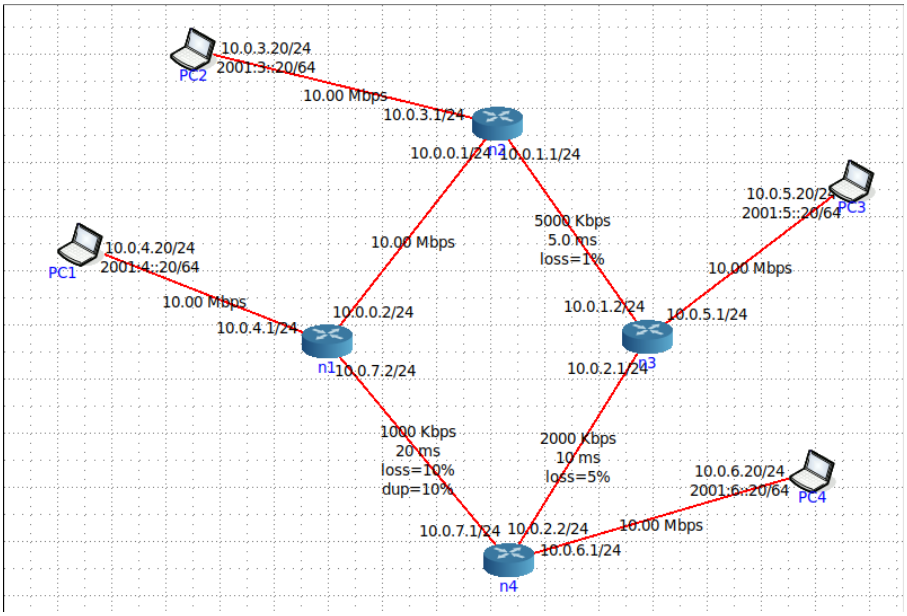


Figura 1.1: Topologia CORE criada

Verifique que são atribuídos automaticamente endereços de rede IPv4 e IPv6 aos vários nós. Apague os endereços IPv6 e deixe apenas os IPv4.

Inspecione as ligações que interligam os nós. Configure o débito das ligações entre os roteadores e hosts a 10 Mbps. Configure as demais ligações, entre os roteadores, da seguinte maneira:

- **Entre os nós 1 e 2: Utilize um débito de 10 Mbps, atraso de 0 ms e perdas de 0%.**
- **Entre os nós 2 e 3: Utilize um débito de 5 Mbps, atraso de 5 ms e perdas de 1%.**
- **Entre os nós 3 e 4: Utilize um débito de 2 Mbps, atraso de 10 ms e perdas de 5%.**
- **Entre os nós 4 e 1: Utilize um débito de 1 Mbps, atraso de 20 ms, perdas de 10% e 10% de duplicações.**

1.2 Questão 2

Verifique que todas as rotas foram configuradas com sucesso e demonstre que os hosts possuem ligação entre si. Utilize-se das ferramentas traceroute, ping e iperf para verificar as rotas entre hosts, as estimativas de perdas de pacotes, atrasos e débito fim-a-fim.

```
root@PC1:/tmp/pycore.36049/PC1.conf# ping -c 3 10.0.3.20
PING 10.0.3.20 (10.0.3.20) 56(84) bytes of data.
64 bytes from 10.0.3.20: icmp_seq=1 ttl=62 time=1.65 ms
64 bytes from 10.0.3.20: icmp_seq=2 ttl=62 time=1.09 ms
64 bytes from 10.0.3.20: icmp_seq=3 ttl=62 time=1.61 ms

--- 10.0.3.20 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 1.094/1.452/1.650/0.253 ms
```

Figura 1.2: Ping de PC1 para PC2

```
root@PC2:/tmp/pycore.36049/PC2.conf# traceroute 10.0.4.20
traceroute to 10.0.4.20 (10.0.4.20), 30 hops max, 60 byte packets
 1  10.0.3.1 (10.0.3.1)  0.325 ms  0.433 ms  0.426 ms
 2  10.0.0.2 (10.0.0.2)  0.700 ms  0.930 ms  1.932 ms
 3  10.0.4.20 (10.0.4.20)  2.427 ms  2.187 ms  2.418 ms
root@PC2:/tmp/pycore.36049/PC2.conf# traceroute 10.0.5.20
traceroute to 10.0.5.20 (10.0.5.20), 30 hops max, 60 byte packets
 1  10.0.3.1 (10.0.3.1)  0.544 ms  1.062 ms  1.057 ms
 2  10.0.1.2 (10.0.1.2)  11.270 ms  11.675 ms  11.990 ms
 3  10.0.5.20 (10.0.5.20)  11.986 ms  12.541 ms *
root@PC2:/tmp/pycore.36049/PC2.conf# traceroute 10.0.6.20
traceroute to 10.0.6.20 (10.0.6.20), 30 hops max, 60 byte packets
 1  10.0.3.1 (10.0.3.1)  0.852 ms  1.605 ms  1.600 ms
 2  10.0.0.2 (10.0.0.2)  2.484 ms  2.479 ms  2.473 ms
 3  10.0.2.2 (10.0.2.2)  38.270 ms  38.779 ms  39.196 ms
 4  10.0.6.20 (10.0.6.20)  40.079 ms * *
```

Figura 1.3: Traceroute - PC2

```
root@PC1:/tmp/pycore.36049/PC1.conf# iperf -c 10.0.3.20
------------------------------------------------------------
Client connecting to 10.0.3.20, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 10.0.4.20 port 46138 connected with 10.0.3.20 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.8 sec  14.6 MBytes  11.4 Mbits/sec
root@PC1:/tmp/pycore.36049/PC1.conf#
```

```
root@PC2:/tmp/pycore.36049/PC2.conf# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size:  128 KByte (default)
------------------------------------------------------------
[  4] local 10.0.3.20 port 5001 connected with 10.0.4.20 port 46138
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-12.8 sec  14.6 MBytes  9.55 Mbits/sec
```

Figura 1.4: Iperf entre PC1 e PC2 - Host servidor: PC2 Host cliente: PC1

1.3 Questão 3

As configurações das rotas foram realizadas dinamicamente pelo protocolo OSPF.

1.3.1 a)

Para obter melhores resultados de débito, atraso e perdas de pacotes, quais rotas alteraria? Justifique

R: Para obtermos melhores resultados nesses critérios, devemos evitar o uso do link entre n1 e n4 sempre que possível, devido à alta taxa de perda de pacotes (10%), assim como evitar o link entre n3 e n4, devido à sua perda de 5%, preferindo rotas alternativas com menor perda. Ou seja, na nossa topologia, devemos modificar as rotas: entre PC1 e PC4, pois usa o link n1-n4, optando pelo caminho n1-n2-n3-n4; entre PC4 e PC1, pois usa o link n4-n1, sugerindo-se o caminho n4-n3-n2-n1; e entre PC2 e PC4, pois utiliza o link n1-n4, sendo recomendável o caminho n2-n3-n4.

1.3.2 b)

Caso se desejasse manter o uso do OSPF, seria possível melhorar as rotas definidas dinamicamente? Como?

R: Sim, seria possível melhorar as rotas dinâmicas definidas pelo OSPF. Isso pode ser feito ajustando o custo OSPF para dar preferência aos links com alta largura de banda (10 Mbps) e baixa perda de pacotes. Além disso, podemos recalcular manualmente os custos para levar em consideração não apenas a largura de banda, como ocorre no OSPF padrão, mas também outros fatores importantes, como o atraso e a perda de pacotes. Dessa forma, o OSPF poderá selecionar rotas mais eficientes, melhorando o desempenho geral da rede.

2 Parte 2

2.1 Questão 1

Com base no trabalho realizado, identifique para cada aplicação executada, qual o protocolo de aplicação, o protocolo de transporte, porta de atendimento e overhead de transporte.

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	40*[SYN](1) + 24*[SYN, ACK](1) + 20*[ACK](9) + 20*[FIN, ACK](2) -> **Total: 464**
ssh, sftp	SSH	TCP	22	40*1 + 32*[SYN, ACK](1) + 20*15 + 20*[PSH, ACK] + 32 [RE ACK](1)+ 20*[FIN]*2 -> **Total: 424**
ftp	FTP	TCP	21	40*1 + 24*1 + 20*16 + 20*3 -> **Total: 424**
Tftp	TFTP	UDP	69	8(Length-UDP payload) * 45 -> **Total: 492**
telnet	TELNET	TCP	23	40*1 + 24*1 + 20*23 + 20*[PSH, ACK](2) + 20*[FIN, PSH, ACK] + 20*1 -> **Total: 604**
nslookup ou dig	DNS	UDP	53	8(Lenght-UDP payload) * 6 -> **Total: 48**
Ping	-	-	-	total: 0
Traceroute	-	UDP	34247 (neste caso em específico)	8(Lenght-UDP payload) * 17 total: 136

Tabela 2.1: Protocolos utilizados em cada aplicação.

3 Parte 3

Neste exercício pretende-se transferir ficheiros utilizando os protocolos TFTP, FTP e HTTP no ambiente do CORE. Para tal, deve-se criar um ficheiro para as transferências e correr os servidores/clientes, conforme as instruções no Anexo I. Utilize a topologia criada na Parte I para responder as questões a seguir. O PC1 deverá ser utilizado como servidor, o host que possui o ficheiro a ser partilhado com os demais. Deve-se analisar apenas os protocolos TFTP, FTP e HTTP.

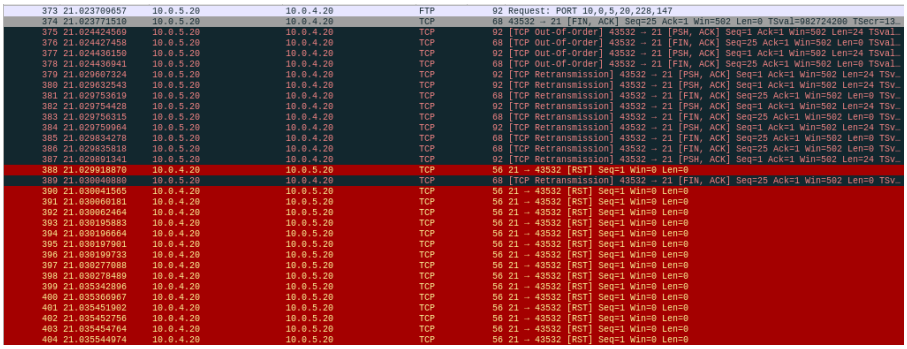
3.1 Questão 1

Descarregue os ficheiros a partir do PC3 com os protocolos TFTP e FTP e responda:

3.1.1 a)

De que forma as perdas de pacotes afetaram o desempenho das aplicações? Que camada lidou com as perdas: transporte ou aplicação? Responda com base nas experiências feitas e nos resultados observados.

R: As perdas de pacotes tiveram um impacto direto e considerável no desempenho das aplicações, sendo mais visíveis nas retransmissões e *resets* de conexão, como observado na figura 3.1. No protocolo FTP, observam-se retransmissões TCP (*TCP Retransmission*) ao tentar enviar pacotes, ou seja, a perda de pacotes forçou a retransmissão. Isso aumenta o tempo total gasto na transferência, impactando o desempenho. Os *resets* TCP (*RST*) também indicam um término abrupto das sessões, possivelmente devido a falhas no cumprimento dos tempos limite ou erros de conexão. A camada responsável pela perda de pacotes é a camada de transporte, mediante o uso de protocolos fiáveis como o TCP, orientado à ligação, uma vez que garante que todos os pacotes são entregues ao destino e na ordem correta, pois para todos os pacotes recebidos é enviado um pacote ACK (Acknowledgment). Recorrendo ao protocolo UDP, apesar deste ser mais "rápido" e de fácil utilização, poderá ser a própria aplicação responsável pela garantia de entrega dos pacotes, no caso de estes não serem entregues, visto que o protocolo de transporte UDP não é orientado à ligação, não possuindo capacidade de verificação de entrega de pacotes, o que poderá levar a um eventual atraso na aplicação. Resumindo, enquanto o FTP opera na camada de aplicação e depende da confiabilidade do TCP, que é responsável por detectar e lidar com as perdas de pacotes, o TFTP utiliza UDP, e por isso a camada de aplicação (TFTP) é quem deve lidar com as perdas, já que o UDP não oferece mecanismos de controle de erros ou retransmissão.



373	21.023709657	10.0.5.20	10.0.4.20	FTP	92 Request: PORT 10,0,5,20,228,147
374	21.023771510	10.0.5.20	10.0.4.20	TCP	68 43532 → 21 [FIN, ACK] Seq=25 Ack=1 Win=502 Len=0 TSval=983724200 TSecr=13…
375	21.024424569	10.0.5.20	10.0.4.20	TCP	92 [TCP Out-Of-Order] 43532 → 21 [PSH, ACK] Seq=1 Ack=1 Win=502 Len=24 TSval…
376	21.024427458	10.0.5.20	10.0.4.20	TCP	68 [TCP Out-Of-Order] 43532 → 21 [FIN, ACK] Seq=25 Ack=1 Win=502 Len=0 TSval…
377	21.024436150	10.0.5.20	10.0.4.20	TCP	92 [TCP Out-Of-Order] 43532 → 21 [PSH, ACK] Seq=1 Ack=1 Win=502 Len=24 TSval…
378	21.024436941	10.0.5.20	10.0.4.20	TCP	68 [TCP Out-Of-Order] 43532 → 21 [FIN, ACK] Seq=25 Ack=1 Win=502 Len=0 TSval…
379	21.029607324	10.0.5.20	10.0.4.20	TCP	92 [TCP Retransmission] 43532 → 21 [PSH, ACK] Seq=1 Ack=1 Win=502 Len=24 TSv…
380	21.029632543	10.0.5.20	10.0.4.20	TCP	92 [TCP Retransmission] 43532 → 21 [PSH, ACK] Seq=1 Ack=1 Win=502 Len=24 TSv…
381	21.029753619	10.0.5.20	10.0.4.20	TCP	68 [TCP Retransmission] 43532 → 21 [FIN, ACK] Seq=25 Ack=1 Win=502 Len=0 TSv…
382	21.029754428	10.0.5.20	10.0.4.20	TCP	92 [TCP Retransmission] 43532 → 21 [PSH, ACK] Seq=1 Ack=1 Win=502 Len=24 TSv…
383	21.029756315	10.0.5.20	10.0.4.20	TCP	68 [TCP Retransmission] 43532 → 21 [FIN, ACK] Seq=25 Ack=1 Win=502 Len=0 TSv…
384	21.029759964	10.0.5.20	10.0.4.20	TCP	92 [TCP Retransmission] 43532 → 21 [PSH, ACK] Seq=1 Ack=1 Win=502 Len=24 TSv…
385	21.029834278	10.0.5.20	10.0.4.20	TCP	68 [TCP Retransmission] 43532 → 21 [FIN, ACK] Seq=25 Ack=1 Win=502 Len=0 TSv…
386	21.029835818	10.0.5.20	10.0.4.20	TCP	68 [TCP Retransmission] 43532 → 21 [FIN, ACK] Seq=25 Ack=1 Win=502 Len=0 TSv…
387	21.029891341	10.0.5.20	10.0.4.20	TCP	92 [TCP Retransmission] 43532 → 21 [PSH, ACK] Seq=1 Ack=1 Win=502 Len=24 TSv…
388	21.029918870	10.0.4.20	10.0.5.20	TCP	56 21 → 43532 [RST] Seq=1 Win=0 Len=0
389	21.030040080	10.0.5.20	10.0.4.20	TCP	68 [TCP Retransmission] 43532 → 21 [FIN, ACK] Seq=25 Ack=1 Win=502 Len=0 TSv…
390	21.030041565	10.0.4.20	10.0.5.20	TCP	56 21 → 43532 [RST] Seq=1 Win=0 Len=0
391	21.030060181	10.0.4.20	10.0.5.20	TCP	56 21 → 43532 [RST] Seq=1 Win=0 Len=0
392	21.030062464	10.0.4.20	10.0.5.20	TCP	56 21 → 43532 [RST] Seq=1 Win=0 Len=0
393	21.030195883	10.0.4.20	10.0.5.20	TCP	56 21 → 43532 [RST] Seq=1 Win=0 Len=0
394	21.030196664	10.0.4.20	10.0.5.20	TCP	56 21 → 43532 [RST] Seq=1 Win=0 Len=0
395	21.030197901	10.0.4.20	10.0.5.20	TCP	56 21 → 43532 [RST] Seq=1 Win=0 Len=0
396	21.030199733	10.0.4.20	10.0.5.20	TCP	56 21 → 43532 [RST] Seq=1 Win=0 Len=0
397	21.030277088	10.0.4.20	10.0.5.20	TCP	56 21 → 43532 [RST] Seq=1 Win=0 Len=0
398	21.030278489	10.0.4.20	10.0.5.20	TCP	56 21 → 43532 [RST] Seq=1 Win=0 Len=0
399	21.035342896	10.0.4.20	10.0.5.20	TCP	56 21 → 43532 [RST] Seq=1 Win=0 Len=0
400	21.035366967	10.0.4.20	10.0.5.20	TCP	56 21 → 43532 [RST] Seq=1 Win=0 Len=0
401	21.035451902	10.0.4.20	10.0.5.20	TCP	56 21 → 43532 [RST] Seq=1 Win=0 Len=0
402	21.035452756	10.0.4.20	10.0.5.20	TCP	56 21 → 43532 [RST] Seq=1 Win=0 Len=0
403	21.035454764	10.0.4.20	10.0.5.20	TCP	56 21 → 43532 [RST] Seq=1 Win=0 Len=0
404	21.035544974	10.0.4.20	10.0.5.20	TCP	56 21 → 43532 [RST] Seq=1 Win=0 Len=0

Figura 3.1: PC3 a realizar o download de File1 recorrendo a FTP

3.1.2 b)

Apresente um diagrama temporal para a transferência de file1 por FTP. Foque-se apenas na transferência de dados [ftp-data] e não na conexão de controlo, pois o FTP usa mais que uma conexão em simultâneo. Identifique, se aplicável, as fases de início de conexão, transferência de dados e fim de conexão.

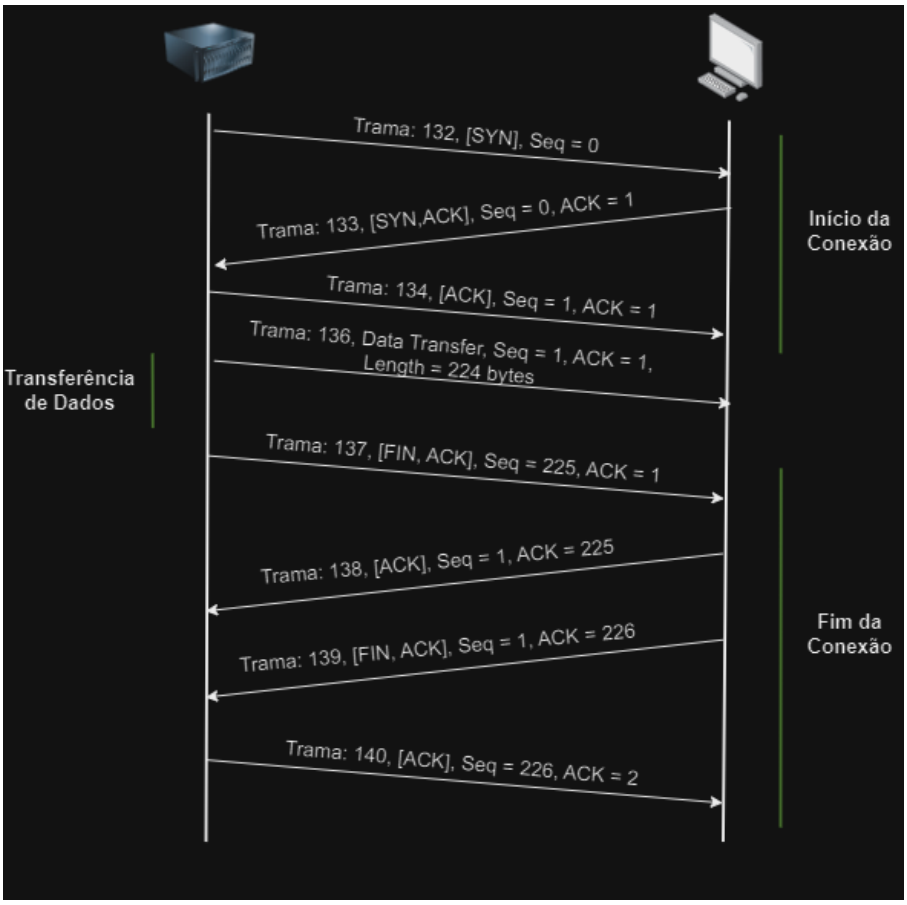


Figura 3.2: Diagrama temporal da transferência de *file1* por FTP

3.1.3 c)

Apresente um diagrama temporal para a transferência de file1 por TFTP. Identifique, se aplicável, as fases de início de conexão, transferência de dados e fim de conexão. Identifique também os tipos de segmentos trocados e os números de sequência usados quer nos dados como nas confirmações.

R: Com base na figura 3.3, observa-se que o cliente faz uma solicitação de leitura (*Read Request*) ao servidor, que então responde com o envio de um pacote de dados (*Data Packet*). Para garantir que a transferência foi bem-sucedida, o cliente envia um pacote de confirmação (*ACK - Acknowledge*).

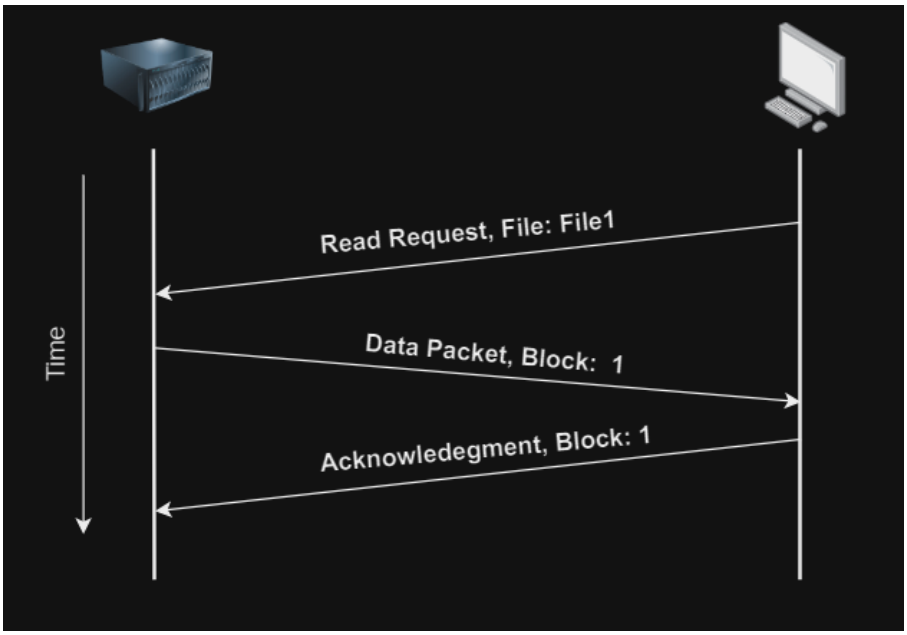


Figura 3.3: Diagrama temporal da transferência de *file1* por TFTP

3.2 Questão 2

Descarregue os ficheiros a partir do PC2 com os protocolos TFTP, FTP e HTTP e responda:

3.2.1 a)

Na transferência HTTP:

- **Identifique o início e o fim da sessão TCP e analise como os números de sequência e ACKs são usados na conexão.**

  R: O início do TCP geralmente ocorre com uma sequência de pacotes SYN, SYN-ACK e ACK. É possível verificar isto observando a frame 345 na figura 3.4, com o pacote SYN enviado pelo cliente. Em seguida, na frame 351, vemos o pacote SYN-ACK como resposta a partir do servidor e, finalmente, na frame 357, o ACK enviado pelo cliente para confirmar a conexão.

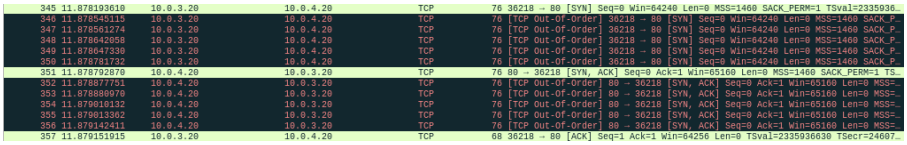


Figura 3.4: Início da conexão TCP

O encerramento da sessão TCP geralmente ocorre com pacotes FIN ou FIN-ACK, seguidos por uma resposta com um pacote ACK. É possível verificar isso observando a frame 465 na figura 3.5, na qual o cliente envia um pacote FIN-ACK, seguido pelo servidor que, na frame 471, responde com um pacote ACK para confirmar o fim da sessão.

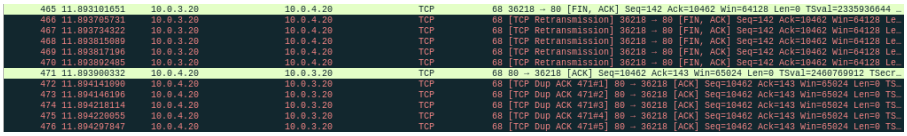


Figura 3.5: Fim da conexão TCP

Os números de sequência são usados ao longo da conexão para garantir que os pacotes são enviados e recebidos na ordem correta em que devem ser lidos. Por outro lado, os pacotes ACK são utilizados para confirmar o sucesso das operações de recebimento e solicitação de um pacote pelo segmento seguinte.

- **Identifique o número de sequência inicial e como ele é incrementado com cada pacote tanto pelo cliente quanto pelo servidor.**

  R: O número de sequência inicial é 0, começando com o primeiro pacote SYN, e em seguida passa para 1 no primeiro ACK. A partir desse ponto, ele aumenta de acordo com os bytes transmitidos a cada vez, que neste caso são 1448, como pode ser visto na imagem abaixo.

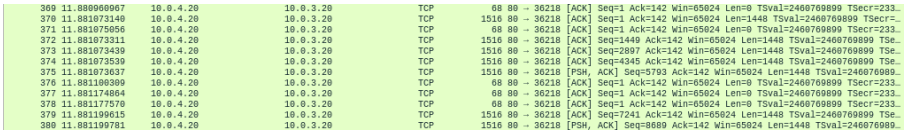


Figura 3.6: Transmissão de dados

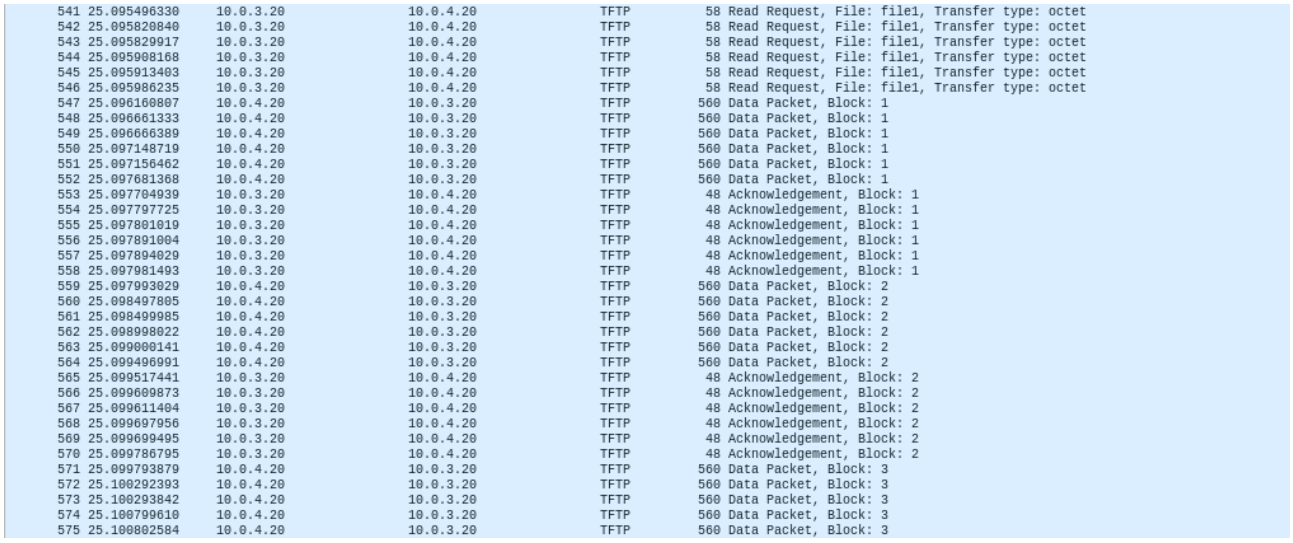
3.2.2 b)

Qual dos protocolos seria o mais adequado para a obtenção dos ficheiros pelo PC2? Justifique.

R: Tendo em conta as opções de protocolos disponíveis, *HTTP, FTP, TFTP*, podemos apontar algumas vantagens e desvantagens de cada um deles:

- *HTTP* é confiável e eficiente, já que foi projetado para baixar arquivos de um servidor *web*. Usa TCP como transporte, tem mecanismos otimizados para recuperação de erros e pode ser mais adequado para redes com perdas de pacotes, pois tem mecanismos de recuperação e controlo de congestionamento eficientes.

- *FTP* usa TCP, dando-lhe benefícios para controle de congestionamento e retransmissão confiável de pactoes. Apesar de ser útil para transferir vários arquivos de forma simultânea, este protocolo é mais pesado em termos de *overhead* e pode nao ser tão eficiente quanto o *HTTP* pois não é tão otimizado para transferências de ficheiros;



541	25.095496330	10.0.3.20	10.0.4.20	TFTP	58 Read Request, File: file1, Transfer type: octet
542	25.095820840	10.0.3.20	10.0.4.20	TFTP	58 Read Request, File: file1, Transfer type: octet
543	25.095829917	10.0.3.20	10.0.4.20	TFTP	58 Read Request, File: file1, Transfer type: octet
544	25.095908168	10.0.3.20	10.0.4.20	TFTP	58 Read Request, File: file1, Transfer type: octet
545	25.095913403	10.0.3.20	10.0.4.20	TFTP	58 Read Request, File: file1, Transfer type: octet
546	25.095986235	10.0.3.20	10.0.4.20	TFTP	58 Read Request, File: file1, Transfer type: octet
547	25.096160807	10.0.4.20	10.0.3.20	TFTP	560 Data Packet, Block: 1
548	25.096661333	10.0.4.20	10.0.3.20	TFTP	560 Data Packet, Block: 1
549	25.096666389	10.0.4.20	10.0.3.20	TFTP	560 Data Packet, Block: 1
550	25.097148719	10.0.4.20	10.0.3.20	TFTP	560 Data Packet, Block: 1
551	25.097156462	10.0.4.20	10.0.3.20	TFTP	560 Data Packet, Block: 1
552	25.097681368	10.0.4.20	10.0.3.20	TFTP	560 Data Packet, Block: 1
553	25.097704939	10.0.3.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 1
554	25.097797725	10.0.3.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 1
555	25.097801019	10.0.3.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 1
556	25.097891004	10.0.3.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 1
557	25.097894029	10.0.3.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 1
558	25.097981493	10.0.3.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 1
559	25.097993029	10.0.4.20	10.0.3.20	TFTP	560 Data Packet, Block: 2
560	25.098497805	10.0.4.20	10.0.3.20	TFTP	560 Data Packet, Block: 2
561	25.098499985	10.0.4.20	10.0.3.20	TFTP	560 Data Packet, Block: 2
562	25.098998022	10.0.4.20	10.0.3.20	TFTP	560 Data Packet, Block: 2
563	25.099000141	10.0.4.20	10.0.3.20	TFTP	560 Data Packet, Block: 2
564	25.099496991	10.0.4.20	10.0.3.20	TFTP	560 Data Packet, Block: 2
565	25.099517441	10.0.3.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 2
566	25.099609873	10.0.3.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 2
567	25.099611404	10.0.3.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 2
568	25.099697956	10.0.3.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 2
569	25.099699495	10.0.3.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 2
570	25.099786795	10.0.3.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 2
571	25.099793879	10.0.4.20	10.0.3.20	TFTP	560 Data Packet, Block: 3
572	25.100292393	10.0.4.20	10.0.3.20	TFTP	560 Data Packet, Block: 3
573	25.100293842	10.0.4.20	10.0.3.20	TFTP	560 Data Packet, Block: 3
574	25.100799610	10.0.4.20	10.0.3.20	TFTP	560 Data Packet, Block: 3
575	25.100802584	10.0.4.20	10.0.3.20	TFTP	560 Data Packet, Block: 3

Figura 3.7: Download do file1 com recurso ao protocolo TFTP

- *TFTP* usa UDP, que não tem controlo de congestionamento ou retransmissão automática, como consta na figura 3.7, tornando este protocolo vulnerável em redes com alta perda de pacotes, levando a transferências corrompidas ou incompletas.

Sendo assim, chegamos à conclusão que o protocolo *HTTP* é o mais adequado, entre as opções disponíveis, para o *PC2* obter o ficheiro.

3.3 Questão 3

Descarregue os ficheiros a partir do PC4 com os protocolos TFTP, FTP e HTTP e responda:

3.3.1 a)

Na transferência HTTP:

- **Identifique a perda de pacotes e a duplicação de pacotes numa sessão TCP.**

  R: O TCP identifica a perda de pacotes, se aplicável, por meio de um *Timeout*, que é o período em que ele espera por um ACK sem recebê-lo. O TCP também detecta ACKs duplicados; ao receber três ACKs solicitando o mesmo segmento, assume que o pacote foi perdido. Além disso, a duplicação de pacotes é identificada através do número de sequência; se dois pacotes têm o mesmo número, um deles é descartado. Podemos observar um exemplo de perda de pacotes na imagem abaixo, nos pacotes identificados como *TCP Retransmission*, e exemplos de pacotes duplicados em "TCP DUP ACK".

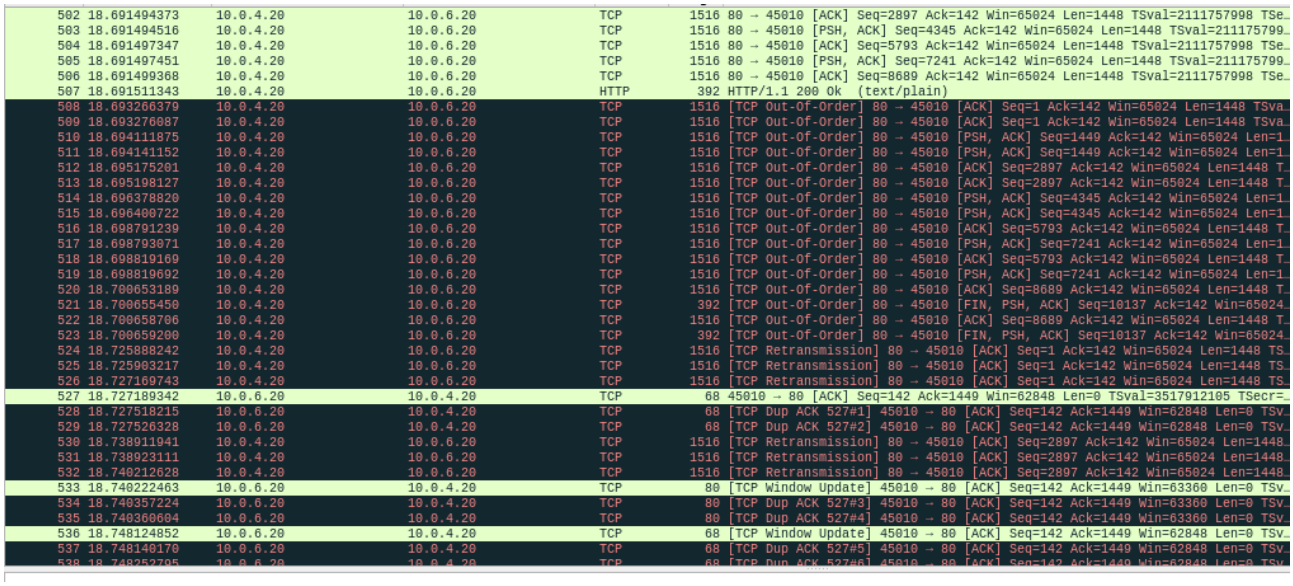


Figura 3.8: Download do ficheiro com recurso ao protocolo HTTP

- **Explique o impacto da perda e duplicação de pacotes numa sessão TCP, bem como os mecanismos usados pelo TCP para lidar com estas situações.**

  R: O impacto que a perda de pacotes pode causar é no tempo total de transmissão, que aumenta, e na diminuição da taxa de transferências. ambos devido aos mecanismos que o TCP usa para resolver o problema. Por outro lado, a duplicação tem um impacto na area do desperdício de recurso, ambos no emissor e na banda de transmissão, pois envia informação repetida.

  Para lidar com a perda de pacotes, o protocolo TCP utiliza o método de retransmissão. Quando um pacote perdido é detectado, o TCP reenvia esse mesmo pacote. Além disso, o TCP aplica o controle de congestionamento para regular a taxa de envio com base na perda de pacotes. A solução para pacotes duplicados é descartar um dos dois no recetor.

3.3.2 b)

Qual dos protocolos seria o mais adequado para a obtenção dos ficheiros pelo PC4? Justifique.

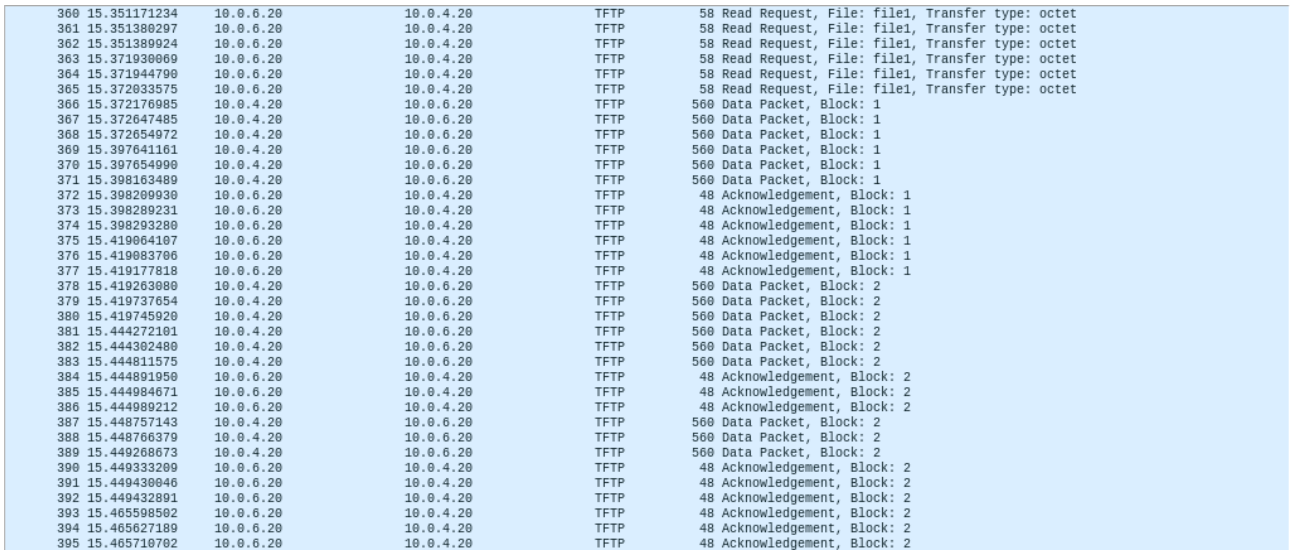
R: Tendo em conta as opções de protocolos disponíveis, *HTTP, FTP, TFTP*, podemos apontar algumas vantagens e desvantagens de cada um deles:

- *HTTP* é confiável e eficiente, já que foi projetado para baixar arquivos de um servidor *web*. Usa TCP como transporte, tem mecanismos otimizados para recuperação de erros e pode ser mais adequado para redes com perdas de pacotes, pois tem mecanismos de recuperação e controlo de congestionamento eficientes.

- *FTP* usa TCP, dando-lhe benefícios para controle de congestionamento e retransmissão confiável de pactoes. Apesar de ser útil para transferir vários arquivos de forma simultânea, este protocolo é mais pesado em termos de *overhead* e pode nao ser tão eficiente quanto o *HTTP* pois não é tão otimizado para transferências de ficheiros;

- *TFTP* usa UDP, que não tem controlo de congestionamento ou retransmissão automática, como

consta na figura 3.9, tornando este protocolo vulnerável em redes com alta perda de pacotes, levando a transferências corrompidas ou incompletas.



360	15.351171234	10.0.6.20	10.0.4.20	TFTP	58 Read Request, File: file1, Transfer type: octet
361	15.351380297	10.0.6.20	10.0.4.20	TFTP	58 Read Request, File: file1, Transfer type: octet
362	15.351389924	10.0.6.20	10.0.4.20	TFTP	58 Read Request, File: file1, Transfer type: octet
363	15.371930069	10.0.6.20	10.0.4.20	TFTP	58 Read Request, File: file1, Transfer type: octet
364	15.371944790	10.0.6.20	10.0.4.20	TFTP	58 Read Request, File: file1, Transfer type: octet
365	15.372033575	10.0.6.20	10.0.4.20	TFTP	58 Read Request, File: file1, Transfer type: octet
366	15.372176985	10.0.4.20	10.0.6.20	TFTP	560 Data Packet, Block: 1
367	15.372647485	10.0.4.20	10.0.6.20	TFTP	560 Data Packet, Block: 1
368	15.372654972	10.0.4.20	10.0.6.20	TFTP	560 Data Packet, Block: 1
369	15.397641161	10.0.4.20	10.0.6.20	TFTP	560 Data Packet, Block: 1
370	15.397654990	10.0.4.20	10.0.6.20	TFTP	560 Data Packet, Block: 1
371	15.398163489	10.0.4.20	10.0.6.20	TFTP	560 Data Packet, Block: 1
372	15.398209930	10.0.6.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 1
373	15.398289231	10.0.6.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 1
374	15.398293280	10.0.6.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 1
375	15.419064107	10.0.6.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 1
376	15.419083706	10.0.6.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 1
377	15.419177818	10.0.6.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 1
378	15.419263080	10.0.4.20	10.0.6.20	TFTP	560 Data Packet, Block: 2
379	15.419737654	10.0.4.20	10.0.6.20	TFTP	560 Data Packet, Block: 2
380	15.419745920	10.0.4.20	10.0.6.20	TFTP	560 Data Packet, Block: 2
381	15.444272101	10.0.4.20	10.0.6.20	TFTP	560 Data Packet, Block: 2
382	15.444302480	10.0.4.20	10.0.6.20	TFTP	560 Data Packet, Block: 2
383	15.444811575	10.0.4.20	10.0.6.20	TFTP	560 Data Packet, Block: 2
384	15.444891950	10.0.6.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 2
385	15.444984671	10.0.6.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 2
386	15.444989212	10.0.6.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 2
387	15.448757143	10.0.4.20	10.0.6.20	TFTP	560 Data Packet, Block: 2
388	15.448766379	10.0.4.20	10.0.6.20	TFTP	560 Data Packet, Block: 2
389	15.449268673	10.0.4.20	10.0.6.20	TFTP	560 Data Packet, Block: 2
390	15.449333209	10.0.6.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 2
391	15.449430046	10.0.6.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 2
392	15.449432891	10.0.6.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 2
393	15.465598502	10.0.6.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 2
394	15.465627189	10.0.6.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 2
395	15.465710702	10.0.6.20	10.0.4.20	TFTP	48 Acknowledgement, Block: 2

Figura 3.9: Download de File1 através do protocolo TFTP

Sendo assim, chegamos à conclusão que o protocolo *HTTP* é o mais adequado, entre as opções disponíveis, para o *PC4* obter o ficheiro. O uso de TCP permite garantir a confiabilidade da transferência.

3.4 Questão 4

Simule uma congestão de rede fazendo o iperf gerar uma taxa de bits por segundo superior à largura de banda do canal (conexão) a partir de um host. Investigue como a janela de congestão muda durante o evento de congestão.

- **Explique como os mecanismos de controle de congestão do TCP (ex.: slow start, congestion avoidance) ajustam o fluxo de dados.**

  R: No início da conexão TCP, o congestion window (cwnd) é pequeno. Como mostra no gráfico de escalonamento da janela (primeiro gráfico, até cerca de 5 segundos), a janela de congestão aumenta exponencialmente durante essa fase, permitindo que o TCP descubra rapidamente a capacidade disponível do canal. Após o slow start, o TCP entra no modo de evitação de congestão. Aqui, o crescimento da janela de congestionamento é linear, como se pode observar entre os 5 e ≈ 10 segundos. Esse comportamento tem o objetivo de evitar sobrecarregar o canal de comunicação à medida que o TCP se aproxima da capacidade máxima do link. Relativamente aos eventos de congestão, o gráfico de escalonamento da janela exibe quedas abruptas (como observado por volta de 17 segundos), o que sugere um evento de congestionamento. Quando o TCP detecta perdas de pacotes, ele assume que a rede está congestionada e reduz a janela de congestionamento, diminuindo o fluxo de dados (fast retransmit e fast recovery). O TCP, então, tenta aumentar a janela de forma conservadora após a recuperação, como visto no gráfico, onde a janela começa a crescer novamente após a queda. Em suma, o TCP usa vários metodos e estados para ajustar a congetão, isto afeta o troughput dos dados enviados ao longo do tempo e eventos de congestão.

- **Apresente a taxa de transferência (throughput) da conexão TCP e compare-a com o débito calculado na Parte I.**

  A taxa de transferência foi variando ao longo do tempo tendo em média mantido 7Mbps (5.6e7 bits / 1000000 / 8), sendo esta menor ao troughput observado anteriormente, pois quando existe um evento de congestão o troughput desce.

- **Forneça imagens das capturas de tráfego para suportar suas observações.**

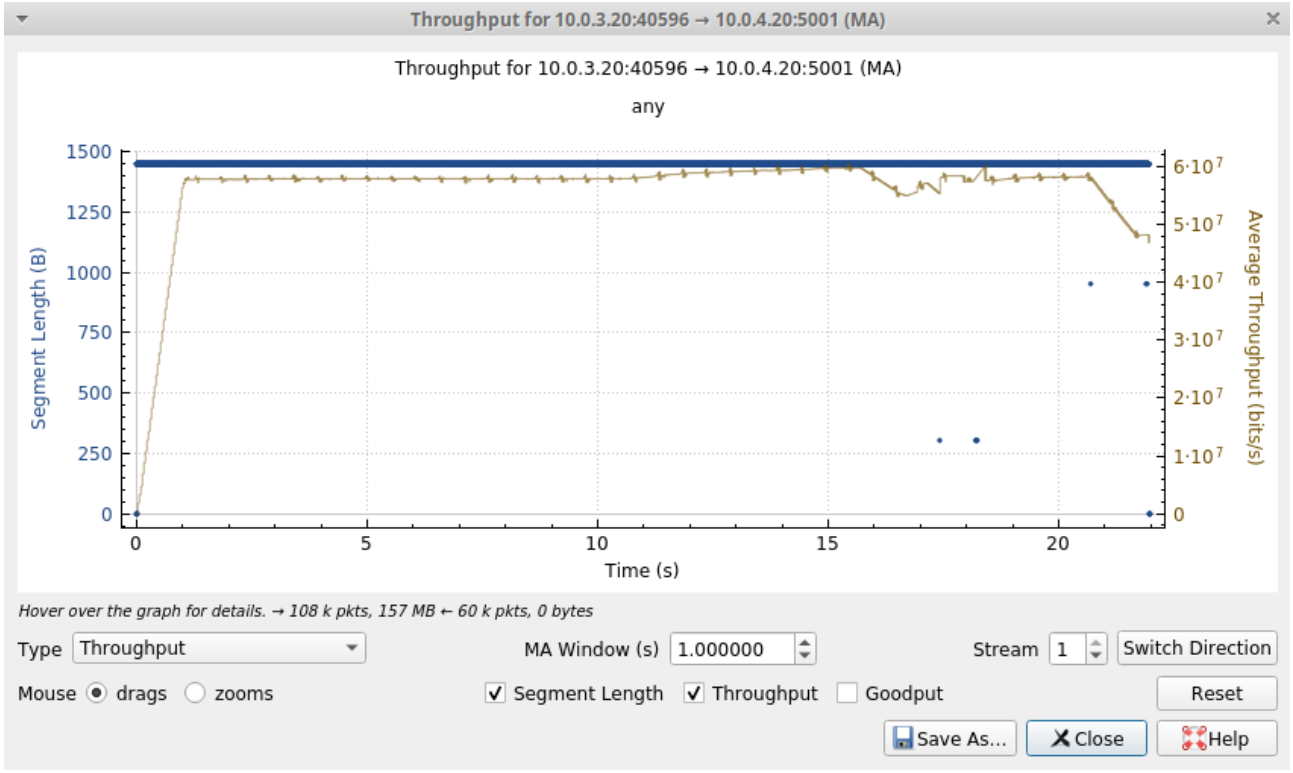


Figura 3.10: Gráfico do Throughput

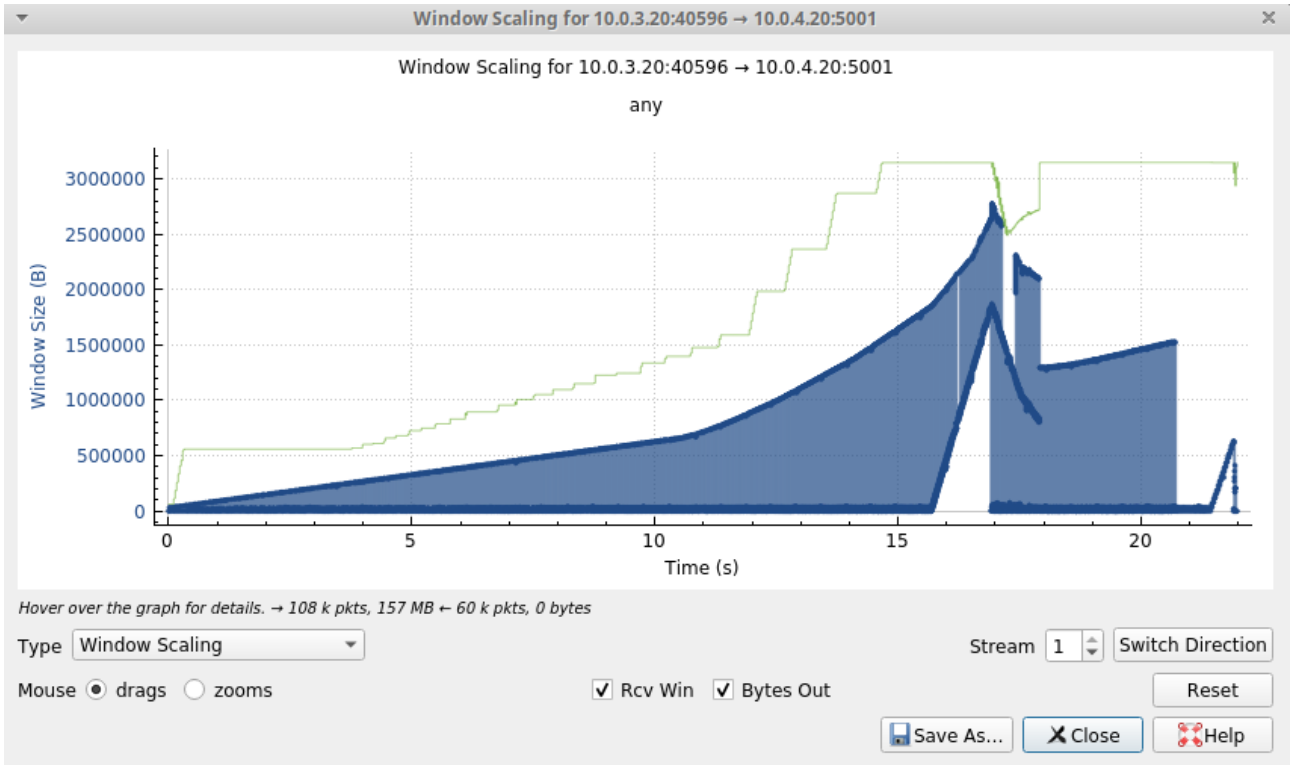


Figura 3.11: Gráfico da Window Size

4 Conclusão

Ao realizar este trabalho prático, foi possível observar em detalhes o funcionamento de diferentes serviços de transferência de arquivos numa rede, com foco nos protocolos de transporte envolvidos, o seu desempenho, a maneira como estabelecem conexões com o servidor, além da sua complexidade e aspectos de segurança. Analisamos ainda o impacto de perdas de pacotes em diversos protocolos, destacando as diferenças no gerenciamento de confiabilidade entre TCP e UDP. Através das atividades realizadas, acreditamos ter alcançado todos os objetivos estabelecidos, obtendo uma compreensão sólida sobre os protocolos da camada de transporte.