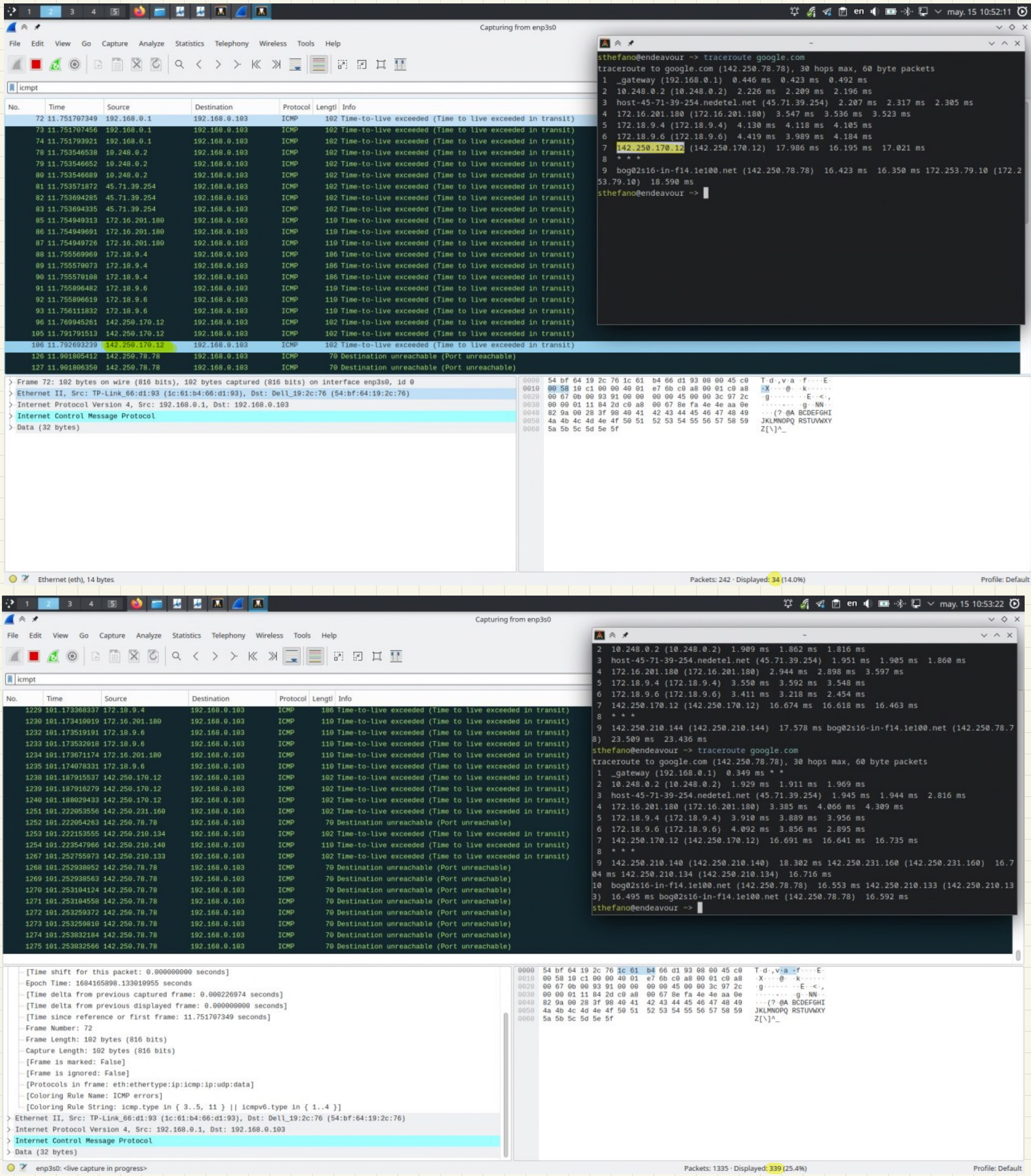
Sthefano Ulloa 00215689 Mayo 14, 2023

Tarea 3

1) Read the following Wireshark tutorial, and use it to capture traffic from the following scenarios. Use screenshots to show your results.

   a) Run 10 traceroute commands against google.com
   b) Watch a video from youtube.com. Capture the TCP handshake, and the congestion window.



≈ 34 packets por traceroute.

≈ 340 en 10 traceroute

Hay 2 handshakes, pero ambos son a un dominio de google (Ver ip y host del request)

*enp3s0

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

not udp and not arp and not tls

No.	Time	Source	Destination	Protocol	Lengtl	Info
358	1.246503140	192.168.0.103	142.250.78.161	TCP	74	41922 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=22789240 TSecr=0 WS=128
684	1.264197136	142.250.78.161	192.168.0.103	TCP	74	443 → 41922 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1412 SACK_PERM TSval=3967042352 TSecr=2278924
685	1.264212734	192.168.0.103	142.250.78.161	TCP	66	41922 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=22789258 TSecr=3967042352
845	1.284002050	142.250.78.161	192.168.0.103	TCP	66	443 → 41922 [ACK] Seq=1 Ack=518 Win=66816 Len=0 TSval=3967042373 TSecr=22789261
964	1.353367336	192.168.0.103	142.250.78.161	TCP	66	41922 → 443 [ACK] Seq=518 Ack=2801 Win=63616 Len=0 TSval=22789347 TSecr=3967042442
966	1.353384725	192.168.0.103	142.250.78.161	TCP	66	41922 → 443 [ACK] Seq=518 Ack=4311 Win=62208 Len=0 TSval=22789347 TSecr=3967042442
969	1.373111612	192.168.0.103	142.250.78.3	TCP	74	48296 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=3889218208 TSecr=0 WS=128
970	1.390138939	142.250.78.3	192.168.0.103	TCP	74	80 → 48296 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1412 SACK_PERM TSval=3130964062 TSecr=38892182
971	1.390169722	192.168.0.103	142.250.78.3	TCP	66	48296 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3889218225 TSecr=3130964062
972	1.390276878	192.168.0.103	142.250.78.3	OCSP	485	Request
973	1.406941528	142.250.78.3	192.168.0.103	TCP	66	80 → 48296 [ACK] Seq=1 Ack=420 Win=66816 Len=0 TSval=3130964078 TSecr=3889218226
974	1.457314629	142.250.78.3	192.168.0.103	OCSP	768	Response
975	1.457341067	192.168.0.103	142.250.78.3	TCP	66	48296 → 80 [ACK] Seq=420 Ack=703 Win=64128 Len=0 TSval=3889218293 TSecr=3130964129
979	1.476956863	142.250.78.161	192.168.0.103	TCP	66	443 → 41922 [ACK] Seq=4311 Ack=582 Win=66816 Len=0 TSval=3967042566 TSecr=22789454
981	1.477874520	192.168.0.103	142.250.78.161	TCP	66	41922 → 443 [ACK] Seq=1013 Ack=4925 Win=64128 Len=0 TSval=22789471 TSecr=3967042567
983	1.478719733	142.250.78.161	192.168.0.103	TCP	66	443 → 41922 [ACK] Seq=4925 Ack=1013 Win=68864 Len=0 TSval=3967042568 TSecr=22789455
986	1.481873298	192.168.0.103	142.250.78.161	TCP	66	41922 → 443 [ACK] Seq=1044 Ack=5441 Win=64128 Len=0 TSval=22789475 TSecr=3967042568
991	1.481998812	192.168.0.103	142.250.78.161	TCP	66	41922 → 443 [ACK] Seq=1044 Ack=11041 Win=61312 Len=0 TSval=22789475 TSecr=3967042571
994	1.482821800	192.168.0.103	142.250.78.161	TCP	66	41922 → 443 [ACK] Seq=1044 Ack=13841 Win=63488 Len=0 TSval=22789476 TSecr=3967042572
997	1.483722811	192.168.0.103	142.250.78.161	TCP	66	41922 → 443 [ACK] Seq=1044 Ack=16641 Win=63488 Len=0 TSval=22789477 TSecr=3967042573
1000	1.484612916	192.168.0.103	142.250.78.161	TCP	66	41922 → 443 [ACK] Seq=1044 Ack=19441 Win=63488 Len=0 TSval=22789478 TSecr=3967042574
1002	1.485631798	192.168.0.103	142.250.78.161	TCP	66	41922 → 443 [ACK] Seq=1044 Ack=20986 Win=64000 Len=0 TSval=22789479 TSecr=3967042575
1004	1.497628742	142.250.78.161	192.168.0.103	TCP	66	443 → 41922 [ACK] Seq=20986 Ack=1044 Win=68864 Len=0 TSval=3967042587 TSecr=22789472

*enp3s0

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

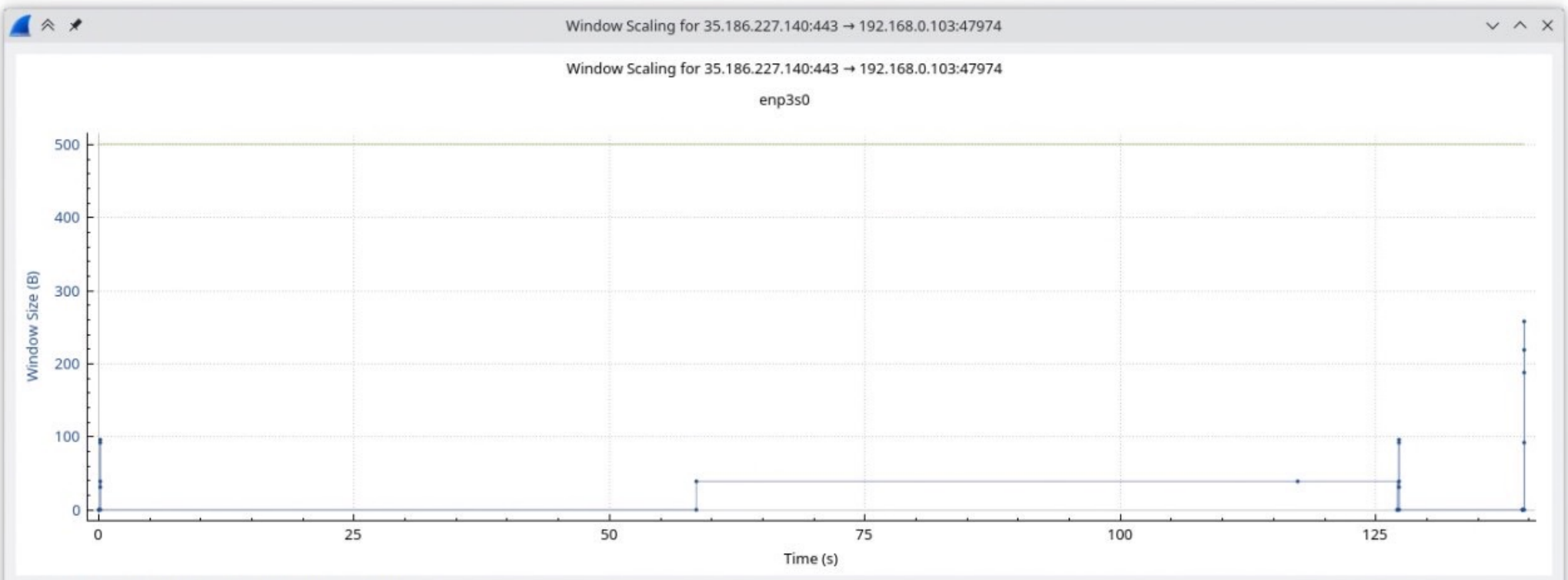
not udp and not arp and not tls

No.	Time	Source	Destination	Protocol	Lengtl	Info
358	1.246503140	192.168.0.103	142.250.78.161	TCP	74	41922 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=22789240 TSecr=0
684	1.264197136	142.250.78.161	192.168.0.103	TCP	74	443 → 41922 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1412 SACK_PERM TSval=396704
685	1.264212734	192.168.0.103	142.250.78.161	TCP	66	41922 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=22789258 TSecr=3967042352
845	1.284002050	142.250.78.161	192.168.0.103	TCP	66	443 → 41922 [ACK] Seq=1 Ack=518 Win=66816 Len=0 TSval=3967042373 TSecr=22789261
964	1.353367336	192.168.0.103	142.250.78.161	TCP	66	41922 → 443 [ACK] Seq=518 Ack=2801 Win=63616 Len=0 TSval=22789347 TSecr=396704244
966	1.353384725	192.168.0.103	142.250.78.161	TCP	66	41922 → 443 [ACK] Seq=518 Ack=4311 Win=62208 Len=0 TSval=22789347 TSecr=396704244
969	1.373111612	192.168.0.103	142.250.78.3	TCP	74	48296 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=3889218208 TSecr=0
970	1.390138939	142.250.78.3	192.168.0.103	TCP	74	80 → 48296 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1412 SACK_PERM TSval=313096
971	1.390169722	192.168.0.103	142.250.78.3	TCP	66	48296 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3889218225 TSecr=3130964062
972	1.390276878	192.168.0.103	142.250.78.3	OCSP	485	Request
973	1.406941528	142.250.78.3	192.168.0.103	TCP	66	80 → 48296 [ACK] Seq=1 Ack=420 Win=66816 Len=0 TSval=3130964078 TSecr=3889218226
974	1.457314629	142.250.78.3	192.168.0.103	OCSP	768	Response
975	1.457341067	192.168.0.103	142.250.78.3	TCP	66	48296 → 80 [ACK] Seq=420 Ack=703 Win=64128 Len=0 TSval=3889218293 TSecr=313096412
979	1.476956863	142.250.78.161	192.168.0.103	TCP	66	443 → 41922 [ACK] Seq=4311 Ack=582 Win=66816 Len=0 TSval=3967042566 TSecr=2278945
981	1.477874520	192.168.0.103	142.250.78.161	TCP	66	41922 → 443 [ACK] Seq=1013 Ack=4925 Win=64128 Len=0 TSval=22789471 TSecr=396704256
983	1.478719733	142.250.78.161	192.168.0.103	TCP	66	443 → 41922 [ACK] Seq=4925 Ack=1013 Win=68864 Len=0 TSval=3967042568 TSecr=2278945
986	1.481873298	192.168.0.103	142.250.78.161	TCP	66	41922 → 443 [ACK] Seq=1044 Ack=5441 Win=64128 Len=0 TSval=22789475 TSecr=396704256
991	1.481998812	192.168.0.103	142.250.78.161	TCP	66	41922 → 443 [ACK] Seq=1044 Ack=11041 Win=61312 Len=0 TSval=22789475 TSecr=396704257
994	1.482821800	192.168.0.103	142.250.78.161	TCP	66	41922 → 443 [ACK] Seq=1044 Ack=13841 Win=63488 Len=0 TSval=22789476 TSecr=396704257
997	1.483722811	192.168.0.103	142.250.78.161	TCP	66	41922 → 443 [ACK] Seq=1044 Ack=16641 Win=63488 Len=0 TSval=22789477 TSecr=396704257
1000	1.484612916	192.168.0.103	142.250.78.161	TCP	66	41922 → 443 [ACK] Seq=1044 Ack=19441 Win=63488 Len=0 TSval=22789478 TSecr=396704257
1002	1.485631798	192.168.0.103	142.250.78.161	TCP	66	41922 → 443 [ACK] Seq=1044 Ack=20986 Win=64000 Len=0 TSval=22789479 TSecr=396704257
1004	1.497628742	142.250.78.161	192.168.0.103	TCP	66	443 → 41922 [ACK] Seq=20986 Ack=1044 Win=68864 Len=0 TSval=3967042587 TSecr=227894

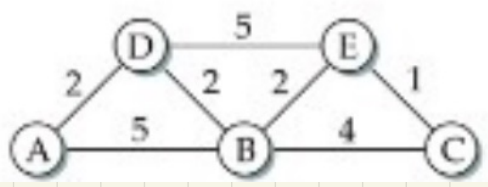
```
> Transmission Control Protocol, Src Port: 48296, Dst Port: 80, Seq: 1, Ack: 1, Len: 419
v Hypertext Transfer Protocol
  v POST /gts1c3 HTTP/1.1\r\n
    > [Expert Info (Chat/Sequence): POST /gts1c3 HTTP/1.1\r\n]
      Request Method: POST
      Request URI: /gts1c3
      Request Version: HTTP/1.1
    Host: ocsp.pki.goog\r\n
    User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/112.0\r\n
    Accept: */*\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
    Content-Type: application/ocsp-request\r\n
  > Content-Length: 84\r\n
    Connection: keep-alive\r\n
    Pragma: no-cache\r\n
    Cache-Control: no-cache\r\n
```

```
0050  54 54 50 2f 31 2e 31 0d  0a 48 6
0060  63 73 70 2e 70 6b 69 2e  67 6f 6
0070  65 72 2d 41 67 65 6e 74  3a 20 4
0080  61 2f 35 2e 30 20 28 58  31 31 3
0090  78 20 78 38 36 5f 36 34  3b 20 7
00a0  2e 30 29 20 47 65 63 6b  6f 2f 3
00b0  30 31 20 46 69 72 65 66  6f 78 2
00c0  0d 0a 41 63 63 65 70 74  3a 20 2
00d0  63 63 65 70 74 2d 4c 61  6e 67 7
00e0  65 6e 2d 55 53 2c 65 6e  3b 71 3
00f0  41 63 63 65 70 74 2d 45  6e 63 6
0100  20 67 7a 69 70 2c 20 64  65 66 6
0110  43 6f 6e 74 65 6e 74 2d  54 79 7
0120  70 6c 69 63 61 74 69 6f  6e 2f 6
0130  65 71 75 65 73 74 0d 0a  43 6f 6
0140  4c 65 6e 67 74 68 3a 20  38 34 0
0150  65 63 74 69 6f 6e 3a 20  6b 65 6
0160  76 65 0d 0a 50 72 61 67  6d 61 3
0170  61 63 68 65 0d 0a 43 61  63 68 6
0180  72 6f 6c 3a 20 6e 6f 2d  63 61 6
0190  0a 30 52 30 50 30 4e 30  4c 30 4
01a0  0e 03 02 1a 05 00 04 14  c7 2e 7
01b0  b3 ba ed 47 42 b8 bb c6  c0 24 0
01c0  7f af 85 cd ee 95 cd 3d  9c d0 e
```

HTTP Host (http.host), 21 bytes

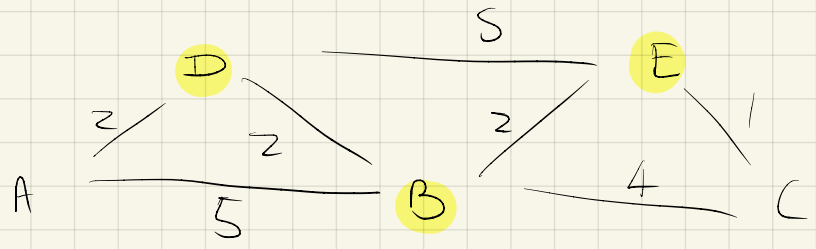


2) Use Dijkstra's to get the routing tables for nodes A, B and E.



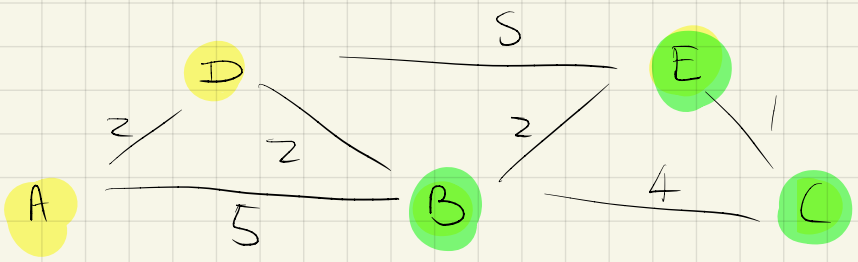
- A

Step	Confirmed	Tentative
1	(A, 0,)	
2	(A, 0, -)	(D, 2, D) (B, 5, B)
3	(A, 0, -) (D, 2, D)	(B, 5, B)
	(A, 0, -) (D, 2, D)	(B, 4, D) (E, 7, D)
4	(A, 0, -) (D, 2, D)	(E, 7, D)
	(B, 4, D)	
5	(A, 0, -) (D, 2, D)	(E, 6, D) (C, 9, B)
	(B, 4, D)	
6	(A, 0, -) (D, 2, D)	(C, 9, B)
	(B, 4, D) (E, 6, D)	
7	(A, 0, -) (D, 2, D)	(C, 7, D)
	(B, 4, D) (E, 6, D)	
8	(A, 0, -) (D, 2, D)	
	(B, 4, D) (E, 6, D)	
	(C, 7, D)	



• B

Step	Confirmed	Tentative
1	(B, 0, -)	
2	(B, 0, -)	(A, 5, A) (D, 2, D)
		(E, 2, E) (C, 4, C)
3	(B, 0, -) (D, 2, D)	(A, 5, A) (C, 4, C)
	(E, 2, E)	
4	(B, 0, -) (D, 2, D)	(A, 4, D) (C, 3, E)
	(E, 2, E)	
5	(B, 0, -) (D, 2, D)	(A, 4, D)
	(E, 2, E) (C, 3, E)	
6	(B, 0, -) (D, 2, D)	
	(E, 2, E) (C, 3, E)	
	(A, 4, D)	



- E

Step	Confirmed	Tentative
1	(E, 0, -)	
2	(E, 0, -)	(C, 1, C) (B, 2, B) (D, 5, D)
3	(E, 0, -) (C, 1, C)	(B, 2, B) (D, 5, D)
4	(E, 0, -) (C, 1, C) (B, 2, B)	(D, 5, D)
5	(E, 0, -) (C, 1, C) (B, 2, B)	(D, 4, B) (A, 7, B)
6	(E, 0, -) (C, 1, C) (B, 2, B) (D, 4, B)	(A, 7, B)
7	(E, 0, -) (C, 1, C) (B, 2, B) (D, 4, B)	(A, 6, B)
8	(E, 0, -) (C, 1, C) (B, 2, B) (D, 4, B) (A, 6, B)	

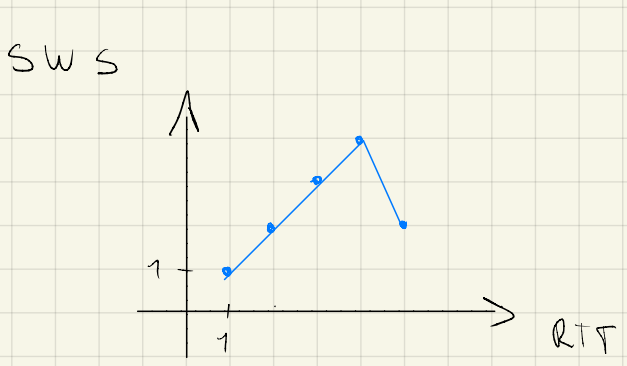
3) Suppose a host wants to establish the reliability of a link by sending packets and measuring the percentage that are received; routers, for example, do this. Explain the difficulty of doing this over a TCP connection.

Al usar TCP hay que considerar que este protocolo incluye acknowledments por diseño, considera timeouts para reenviar paquetes perdidos. Asegurandose de que los paquetes lleguen en orden al host. Entonces resulta evidente que si se reenvian los paquetes perdidos se complican las mediciones pues dependiendo de nuestra implementación, los paquetes reenviados enmascaran a los perdidos.
Además, configuraciones y caracteristicas de dispositivos intermediarios pueden afectar las mediciones ya que pueden afectar el rate de paquetes recibidos, aumentar el delay, etc. (Aunque creo que esto aplica para otros tipos de conexiones que no son directas)

4) Consider a simple congestion control algorithm that uses linear increase and multiplicative decrease (no slow start). Assume the congestion window size is in units of packets rather than bytes, and it is one packet initially.

a) Give a detailed sketch of this algorithm.
b) Assume the delay is latency only, and that when a group of packets is sent, only a single ACK is returned.
c) Plot the congestion window as a function of RTT for the situation in which the following packets are lost: 9, 25, 30, 38 and 50. For simplicity, assume a perfect timeout mechanism that detects a lost packet exactly 1 RTT after it is transmitted.

a) Linear increase significa que cada vez que un grupo de paquetes envian todos el ACK, el SWS (Sender Window size) se incrementa, sumando una cantidad fija.
Multiplicative decrease significa que cuando haya un timeout (paquete se perdio). El SWS se reduce por un factor.
Entonces asumiendo que el linear increase es de 1 y el multiplicative decrease es de 2 (dividir a la mitad). Funcionaria de la siguiente manera:

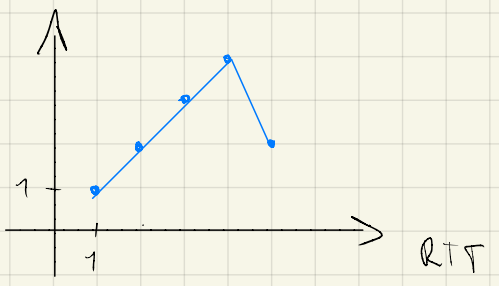


- Envio P1 y recibo ACK. SWS aumenta a 2
- Envio P2, P3 y recibo sus ACK. SWS aumenta a 3
- Envio P4, P5, P6 y recibo ACKs. SWS aumenta a 4
- Envio P7, P8, P9, P10. Si se pierde alguno el SWS se divide y ahora es 2. Asi que se reenvia el P perdido y el siguiente (2)

b) Si esque solo se envia un ACK por grupo, no se podria reenviar solo el paquete perdido y los que sigan (dependiendo del SWS) sino que se reenvia todo el grupo desde el inicio:

- Envio P1 y recibo ACK. SWS aumenta a 2
- Envio P2, P3 y recibo sus ACK. SWS aumenta a 3
- Envio P4, P5, P6 y recibo ACKs. SWS aumenta a 4
- Envio P7 - P10 y supongamos se pierde P9. El SWS baja 2 y se reenvia P7 y P8 ya que no se sabe especificamente cual/es se perdieron pues se cuenta un solo ACK cuando todo el grupo llega.

SWS



c) Si se pierden 9, 25, 30, 36 y 50

Asumiendo ACKs individuales para reenviar desde los perdidas en vez de todo el grupo de nuevo.

RTT	1	2	3	4	5	6	7	8	9
Paquetes	1	2-3	4-6	7-10	9-10	11-13	14-17	18-22	23-28
SWS	1	2	3	4	2	3	4	5	6

(9) (25)

RTT	10	11	12	13	14	15	16	17
Paquetes	25-27	28-31	30-31	32-34	35-36	36-39	40-42	43-46
SWS	3	4	2	3	4	2	3	4

(30) (36)

RTT	18	19
Paquetes	47-51	50-52
SWS	5	2.5 → 3

(50)