Deber N°3 CMP-4005

Realizado por: Esteban Alvarado 215138

1. **Lea el siguiente tutorial de *Wireshark*, y úselo para capturar el tráfico en los siguientes escenarios:**

   a) Ejecute 10 comandos de *traceroute* sobre www.google.com.

1636	19:32:13.078333	10.224.41.54	192.168.100.47	ICMP	70	34646	33442	Time-to-live exceeded (Time to live exceeded in transit)
1638	19:32:13.083955	10.224.41.54	192.168.100.47	ICMP	70	34646	33443	Time-to-live exceeded (Time to live exceeded in transit)
1640	19:32:13.092889	192.168.0.41	192.168.100.47	ICMP	70	34646	33444	Time-to-live exceeded (Time to live exceeded in transit)
1651	19:32:13.127592	192.168.0.41	192.168.100.47	ICMP	70	34646	33445	Time-to-live exceeded (Time to live exceeded in transit)
1654	19:32:13.138482	192.168.0.41	192.168.100.47	ICMP	70	34646	33446	Time-to-live exceeded (Time to live exceeded in transit)
1659	19:32:13.144236	192.168.0.42	192.168.100.47	ICMP	94	34646	33447	Time-to-live exceeded (Time to live exceeded in transit)
1669	19:32:13.173657	192.168.0.42	192.168.100.47	ICMP	94	34646	33448	Time-to-live exceeded (Time to live exceeded in transit)
1671	19:32:13.180023	192.168.0.42	192.168.100.47	ICMP	94	34646	33449	Time-to-live exceeded (Time to live exceeded in transit)
1673	19:32:13.187682	181.39.98.21	192.168.100.47	ICMP	70	34646	33450	Time-to-live exceeded (Time to live exceeded in transit)
1684	19:32:13.223015	181.39.98.21	192.168.100.47	ICMP	70	34646	33451	Time-to-live exceeded (Time to live exceeded in transit)
1686	19:32:13.230431	181.39.98.21	192.168.100.47	ICMP	70	34646	33452	Time-to-live exceeded (Time to live exceeded in transit)
1689	19:32:13.236944	74.125.146.39	192.168.100.47	ICMP	70	34646	33453	Time-to-live exceeded (Time to live exceeded in transit)
1707	19:32:13.329801	142.250.163.94	192.168.100.47	ICMP	70	34646	33454	Time-to-live exceeded (Time to live exceeded in transit)
1719	19:32:13.357270	186.3.125.46	192.168.100.47	ICMP	70	34646	33455	Time-to-live exceeded (Time to live exceeded in transit)
1730	19:32:13.401256	142.250.163.95	192.168.100.47	ICMP	94	34646	33456	Time-to-live exceeded (Time to live exceeded in transit)
1745	19:32:13.508518	142.250.163.95	192.168.100.47	ICMP	94	34646	33457	Time-to-live exceeded (Time to live exceeded in transit)
1748	19:32:13.527439	186.3.125.47	192.168.100.47	ICMP	94	34646	33458	Time-to-live exceeded (Time to live exceeded in transit)
1960	19:32:28.646469	108.170.253.1	192.168.100.47	ICMP	110	34646	33462	Time-to-live exceeded (Time to live exceeded in transit)
1971	19:32:28.750797	216.239.56.234	192.168.100.47	ICMP	110	34646	33463	Time-to-live exceeded (Time to live exceeded in transit)
1982	19:32:28.868728	142.250.210.130	192.168.100.47	ICMP	94	34646	33464	Time-to-live exceeded (Time to live exceeded in transit)
1993	19:32:28.975718	142.250.231.163	192.168.100.47	ICMP	110	34646	33465	Time-to-live exceeded (Time to live exceeded in transit)
2003	19:32:29.023857	172.217.173.196	192.168.100.47	ICMP	70	34646	33466	Destination unreachable (Port unreachable)
2015	19:32:29.068864	142.250.231.163	192.168.100.47	ICMP	110	34646	33467	Time-to-live exceeded (Time to live exceeded in transit)

   Si chequeamos la ubicación de la última IP: **142.250.231.163**, veremos lo siguiente:

País	**Estados Unidos**
Ciudad	
Latitud	**37.751**
Longitud	**-97.822**
ISP	**Google**

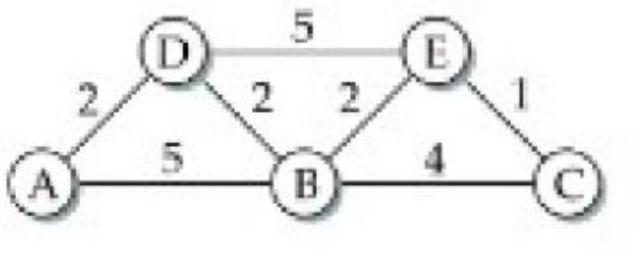
   Traceroute lo que hace es determinar la ruta que siguen los paquetes hacia su destino, en este caso, hacia *google.com*. La dirección IP **192.168.100.47** es la dirección desde donde se ejecutó el comando, y el resto de las direcciones que se muestran en la pantalla de Wireshark, son los routers por donde pasaron los paquetes para llegar a Google.com

   b) Observe un video de *youtube.com*. Capture el TCP *handshake*, y la ventana de congestión.

Source	Destination	Protocol	Lengt	sport	dport	Info
192.168.100.47	142.250.78.3	TCP	78	63212	80	63212 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=856785442 TSecr=0 SACK_PERM
142.250.78.3	192.168.100.47	TCP	74	80	63212	80 → 63212 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1412 SACK_PERM TSval=2790493103 TSecr=856785442 WS=256
192.168.100.47	142.250.78.3	TCP	66	63212	80	63212 → 80 [ACK] Seq=1 Ack=1 Win=131584 Len=0 TSval=856785461 TSecr=2790493103
192.168.100.47	142.250.78.3	HTTP	423	63212	80	GET /gts1c3/ME8wTTBLMEkwRzAHBgUrDgMCGgQUxy55it3%2FYTSzuu1HQri7xsAkB2MEFIp0f6%2BFze6VzT2c0OJGFPNxNR0nAhA7lK5VH

   Cuando se envía por primera vez un paquete al destino, el Source establece la ventana de congestión en base al *máximum sender size* (MSS). Después este incrementará en base a la siguiente fórmula: $CongestionWindow += MSS \times (MSS/CongestionWindow)$. Por tanto, la ventana de congestión es igual a $\boldsymbol{MSS = 1460}$.

2. Use Dijkstra para obtener las tablas de ruteo para los nodos A, B y E.



DIJKSTRA: NODO A

PASO	CONFIRMADO	TENTATIVO
1	(A,0,-)	-
2	(A,0,-)	(D,2,D), (B,5,B)
3	(A,0,-) (D,2,D)	(B,5,B)
4	(A,0,-) (D,2,D)	(B,4,D), (E,7,D)
5	(A,0,-) (D,2,D) (B,4,D)	(E,7,D)
6	(A,0,-) (D,2,D) (B,4,D)	(E,6,B) (C,8,B)
7	(A,0,-) (D,2,D) (B,4,D) (E,6,B)	(C,8,B)
8	(A,0,-) (D,2,D) (B,4,D) (E,6,B)	(C,7,E)
9	(A,0,-) (D,2,D) (B,4,D) (E,6,B) (C,7,E)	-

TABLA DE RUTEO: NODO A

Destino	Costo	Siguiente salto
B	4	D
C	7	E
D	2	D
E	6	B

DIJKSTRA: NODO B

PASO	CONFIRMADO	TENTATIVO
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) (E,2,E) (C,4,C)
4	(B,0,-) (D,2,D)	(A,4,D) (E,2,E) (C,4,C)
5	(B,0,-) (D,2,D) (E,2,E)	(A,4,D) (C,4,C)
6	(B,0,-) (D,2,D) (E,2,E)	(A,4,D) (C,3,E)
7	(B,0,-) (D,2,D) (E,2,E) (C,3,E)	(A,4,D)
8	(B,0,-) (D,2,D) (E,2,E) (C,3,E) (A,4,D)	-

TABLA DE RUTEO: NODO B

Destino	Costo	Siguiente salto
A	4	D
C	3	E
D	2	D
E	2	E

DIJKSTRA: NODO E

PASO	CONFIRMADO	TENTATIVO
1	(E,0,-)	-
2	(E,0,-)	(D,5,D) (B,2,B) (C,1,C)
3	(E,0,-) (C,1,C)	(D,5,D) (B,2,B)
4	(E,0,-) (C,1,C)	(D,5,D) (B,2,B)
5	(E,0,-) (C,1,C) (B,2,B)	(D,5,D)

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,7,B)
8	(E,0,-) (C,1,C) (B,2,B) (D,4,B)	(A,6,D)
9	(E,0,-) (C,1,C) (B,2,B) (D,4,B) (A,6,D)	-

TABLA DE RUTEO: NODO E

Destino	Costo	Siguiente salto
A	6	D
B	2	B
C	1	C
D	4	B

3. **Suponga que un host quiere conocer la fiabilidad de un link al enviar paquetes y midiendo el porcentaje de los que fueron recibidos; los routers, por ejemplo, hacen esto. Explique la dificultad de hacer esto sobre una conexión TCP.**

   Para responder esta pregunta tenemos que tomar en cuenta tres puntos:

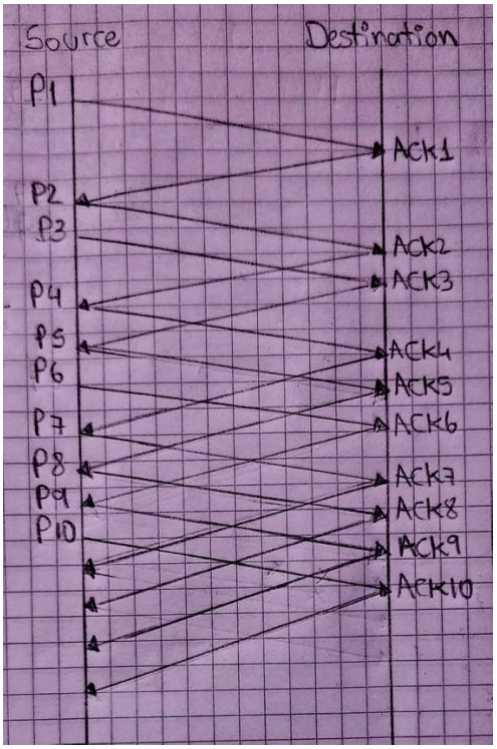
   - Las conexiones TCP tienen tiempos de ida y vuelta (RTTs) que pueden variar por la ubicación entre nodos, o incluso por el momento en que se establece la conexión.
   - TCP garantiza la entrega fiable de los datos, por lo cual retransmite cada segmento si no se recibe un ACK en un tiempo de espera determinado.
   - Dado el rango de posibles RTTs a lo largo del tiempo, elegir un valor de tiempo de espera apropiado no es tan fácil, y por ello TCP emplea mecanismos de retransmisión adaptativa.

   Ahora bien, todos estos mecanismos de retransmisión se basan tiempos de espera de ACKs, lo que quiere decir que si se acaba el tiempo de espera y el host no ha recibido un ACK, esto indicaría que probablemente se ha perdido un segmento. Sin embargo, hay que tener en cuenta que el que se acabe el tiempo de espera no indica al *host* si los segmentos que envió después del segmento perdido se recibieron correctamente. Esto se debe a que los ACKs de TCP son acumulativos; sólo identifican al último segmento que no se recibió fuera de orden.

   Por lo tanto, si se pierden segmentos durante una conexión TCP o simplemente se agota el tiempo de espera, será difícil para el host calcular el porcentaje de paquetes que fueron recibidos correctamente debido a los ACKs acumulativos que se maneja en el algoritmo de Sliding Window de TCP.

4. Considere un algoritmo de control de congestión simple que usa incremento lineal y decremento multiplicativo (sin *slow start*). Asuma que el tamaño de congestión de la ventana es en unidades de paquetes, en lugar de bytes, y que es inicialmente un paquete.

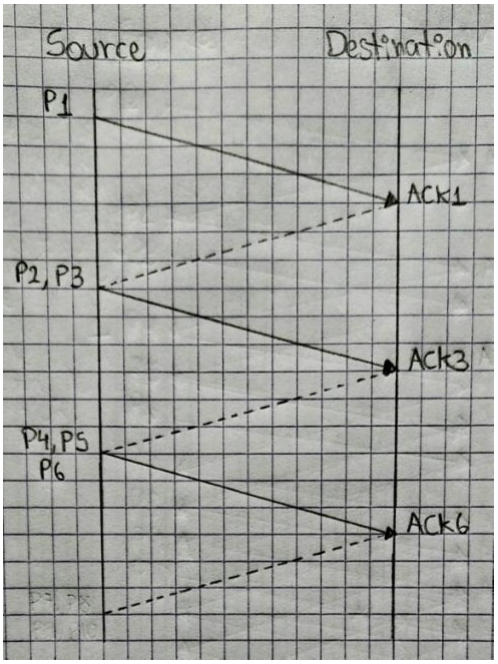
   a) Dé un boceto detallado de este algoritmo.



Cada vez que el Source envía con éxito un paquete de Congestion Window añade el equivalente a 1 paquete a Congestion Window, esto corresponde al “incremento lineal”.

En cambio, cada vez que se produce un timeout, el Source fija Congestion Window a la mitad de su valor anterior. Esta reducción a la mitad de Congestion Window para cada tiempo de espera corresponde a la "disminución multiplicativa".

   b) Asuma que el *delay* es solo *latencia*, y que cuando un grupo de paquetes se envía, solo un ACK se retorna.



c) Dibuje la ventana de congestión como una función de RTT para la situación en donde los siguientes paquetes se pierden: 9, 25, 30, 38 y 50. Para simplificar, asuma un mecanismo de *timeout* perfecto que detecta paquetes perdidos exactamente 1 RTT después de su transmisión.