LINTI - UNLP

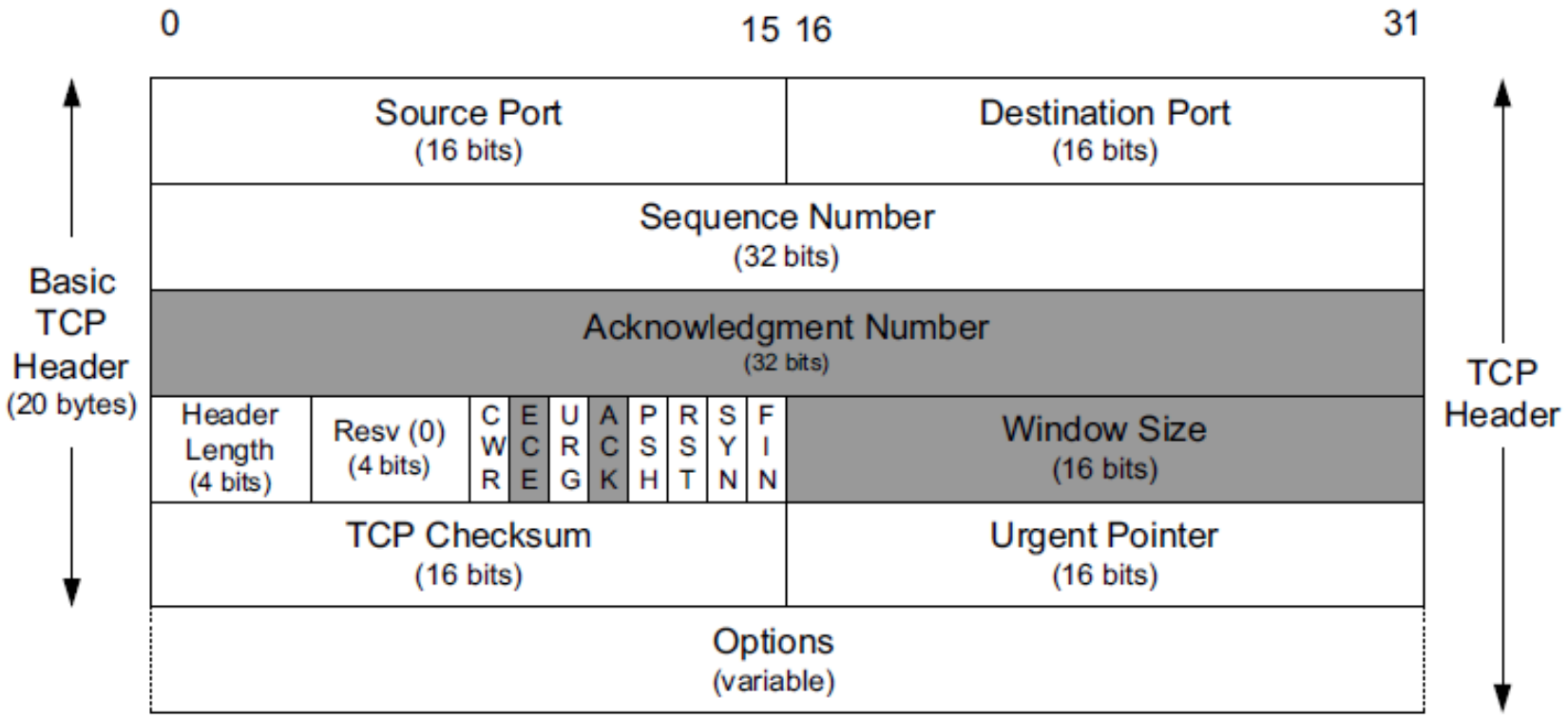
Redes
TCP

luis marrone

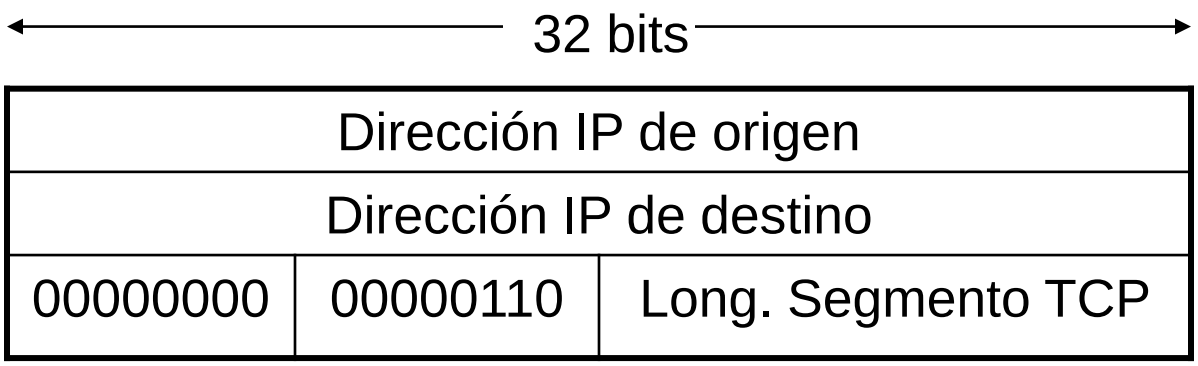
TCP

- Orientado a conexión.
- Confiable.
- Ordenado.
- Byte Stream.
- Ventana.
- Control de flujo y congestión.

TCP – Estructura



La pseudocabecera TCP



Se añade al principio del segmento solo para el cálculo del Checksum, **no se envía**. Permite a TCP comprobar que IP no se ha equivocado (ni le ha engañado) en la entrega del segmento.

El valor $110_2 = 6_{10}$ indica que el protocolo de transporte es TCP

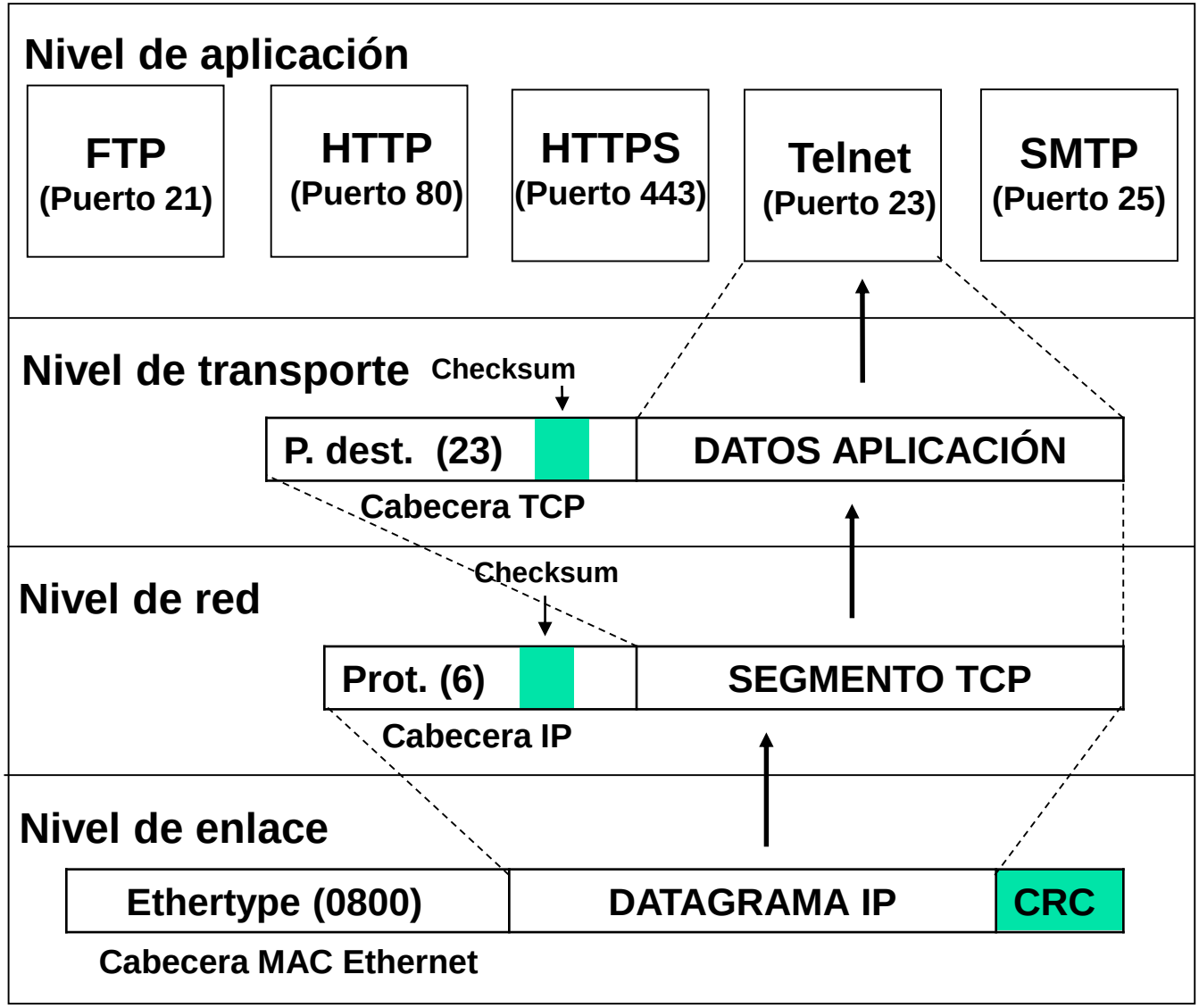
Multiplexación

Múltiples instancias (una o varias por protocolo)

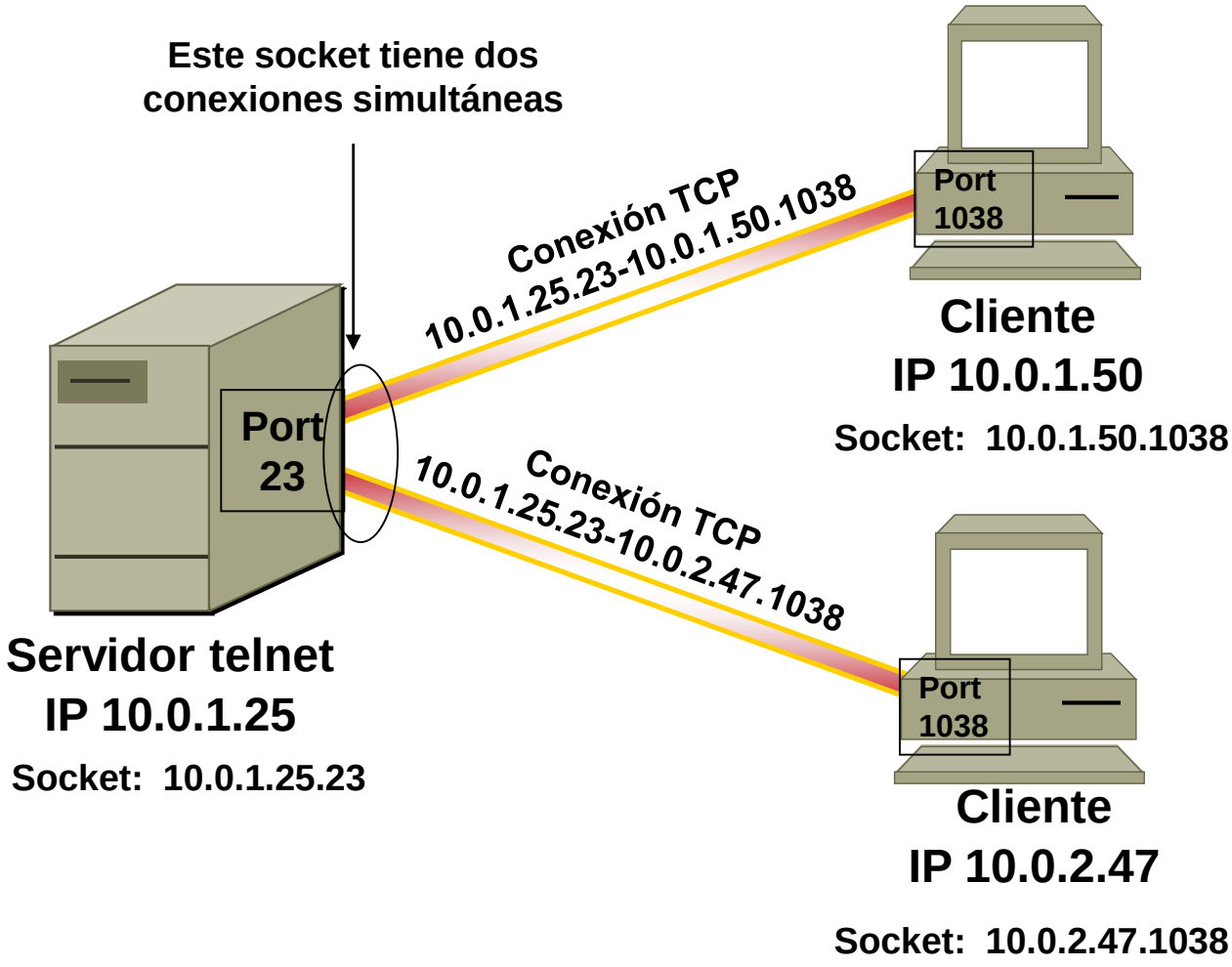
Dos instancias (TCP y UDP)

Una instancia IP (puede haber otros protocolos)

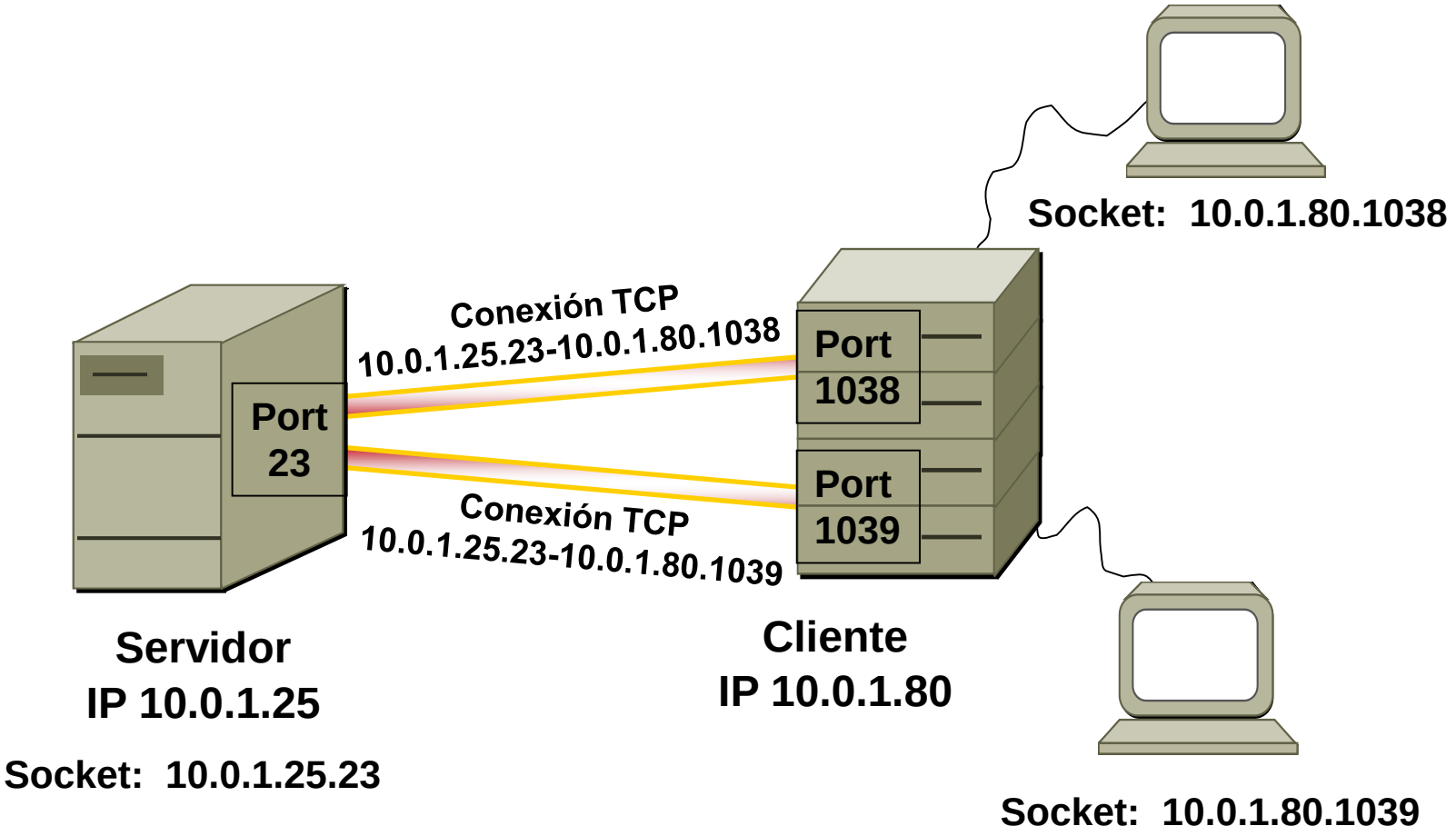
Múltiples instancias (una por interfaz)



Dos conexiones TCP a un mismo socket desde dos sockets con el mismo número de puerto

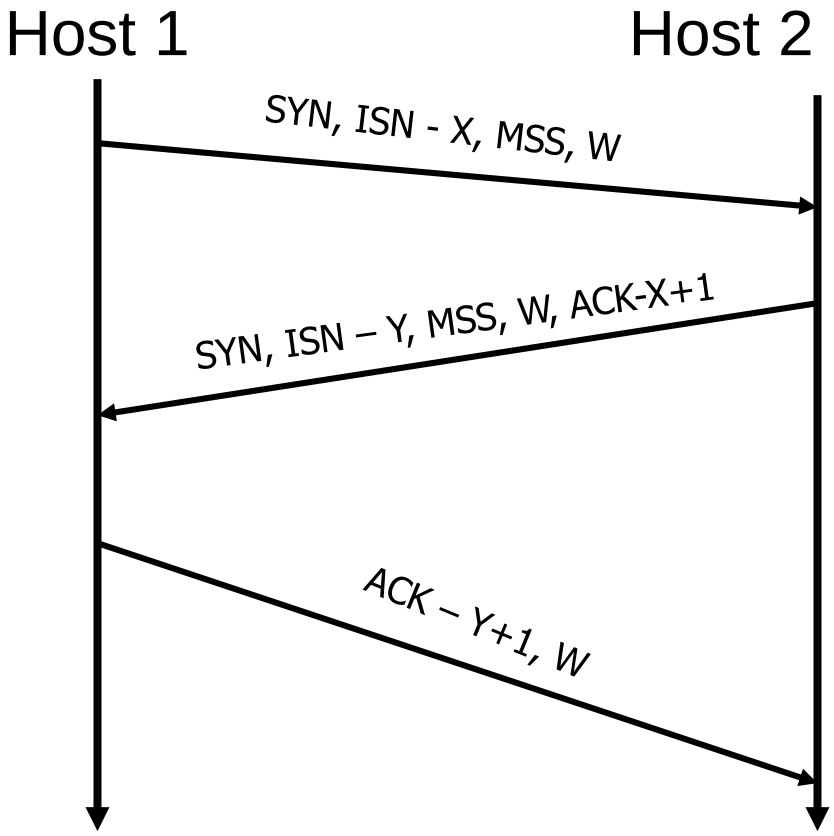


Dos conexiones TCP a un mismo socket desde dos sockets con la misma dirección IP



TCP – Establecimiento de la conexión

- Se intercambian 3 segmentos.
- Se envía el MSS.

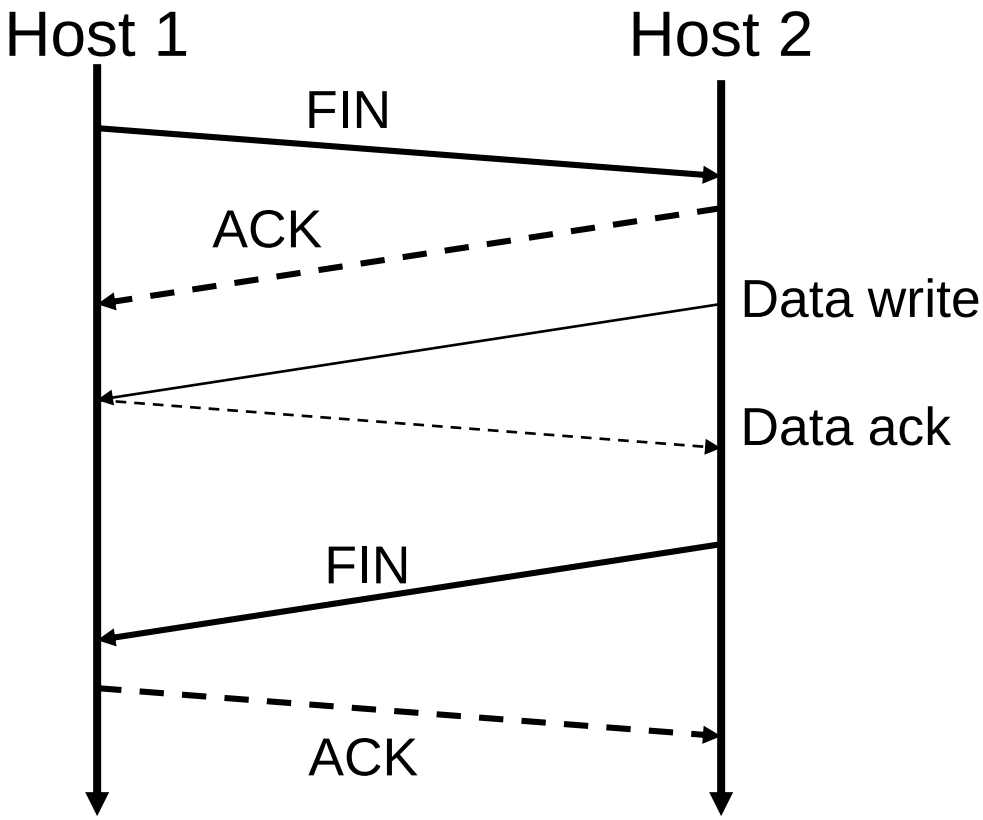


MSS

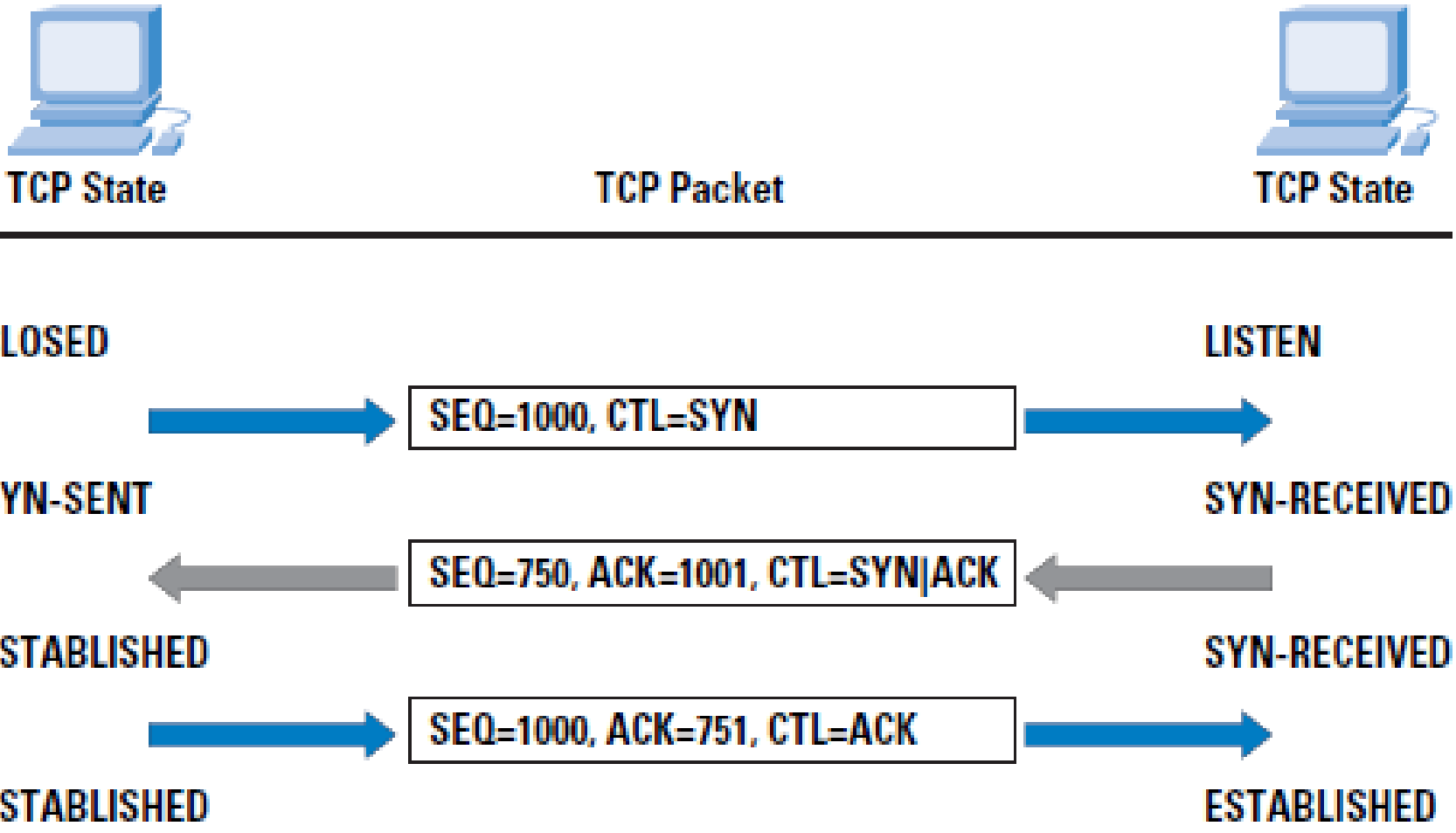
- Máxima longitud del segmento.
- Default = 536 bytes.
- Diferente para cada sentido de la conexión.
- Vinculado al MTU.

TCP – Fin de la conexión

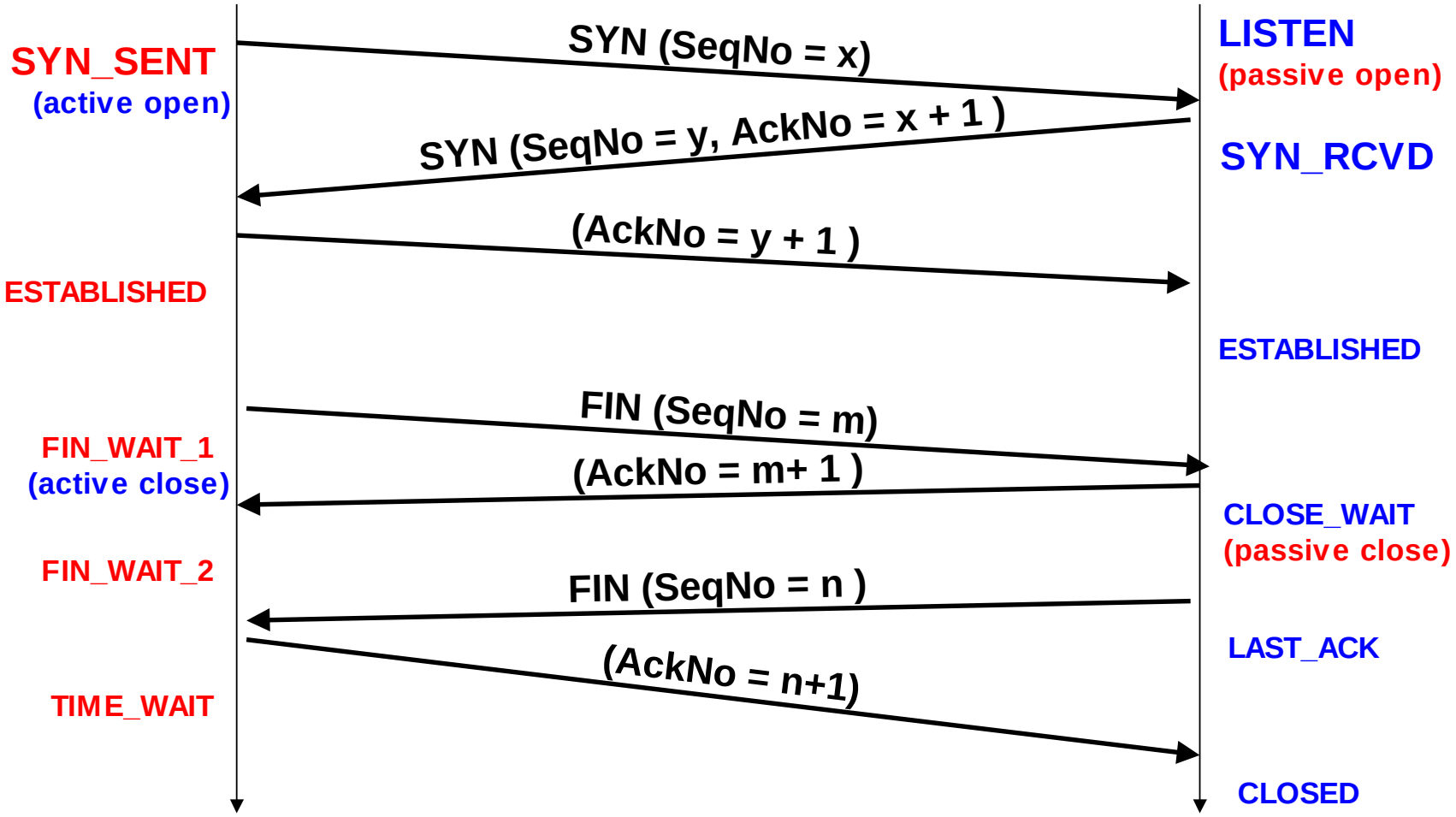
- Se intercambian 4 segmentos



SYN


TCP State
TCP Packet
TCP State
CLOSED
LISTEN
SEQ=1000, CTL=SYN
SYN-SENT
SYN-RECEIVED
SEQ=750, ACK=1001, CTL=SYN|ACK
ESTABLISHED
SYN-RECEIVED
SEQ=1000, ACK=751, CTL=ACK
ESTABLISHED
ESTABLISHED

SYN – FIN


SYN_SENT
(active open)
SYN (SeqNo = x)
LISTEN
(passive open)
SYN (SeqNo = y, AckNo = x + 1)
SYN_RCVD
(AckNo = y + 1)
ESTABLISHED
ESTABLISHED
FIN (SeqNo = m)
FIN_WAIT_1
(active close)
(AckNo = m+ 1)
CLOSE_WAIT
(passive close)
FIN_WAIT_2
FIN (SeqNo = n)
LAST_ACK
(AckNo = n+1)
TIME_WAIT
CLOSED

Netstat

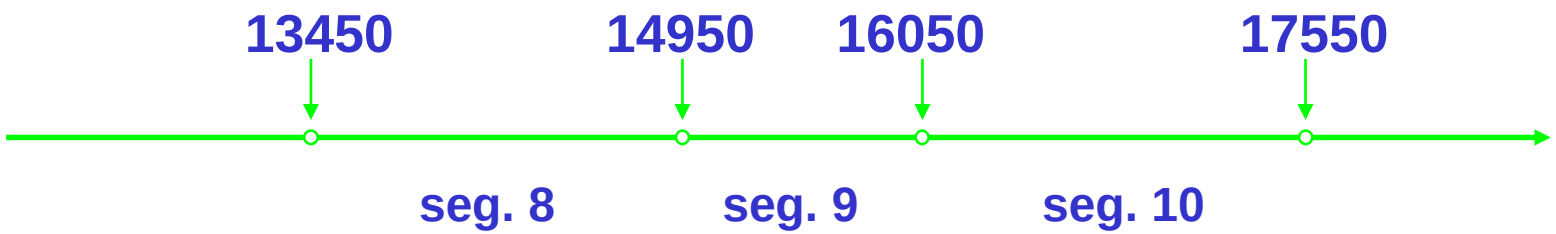
U:\>netstat -a -n -p TCP

Active Connections

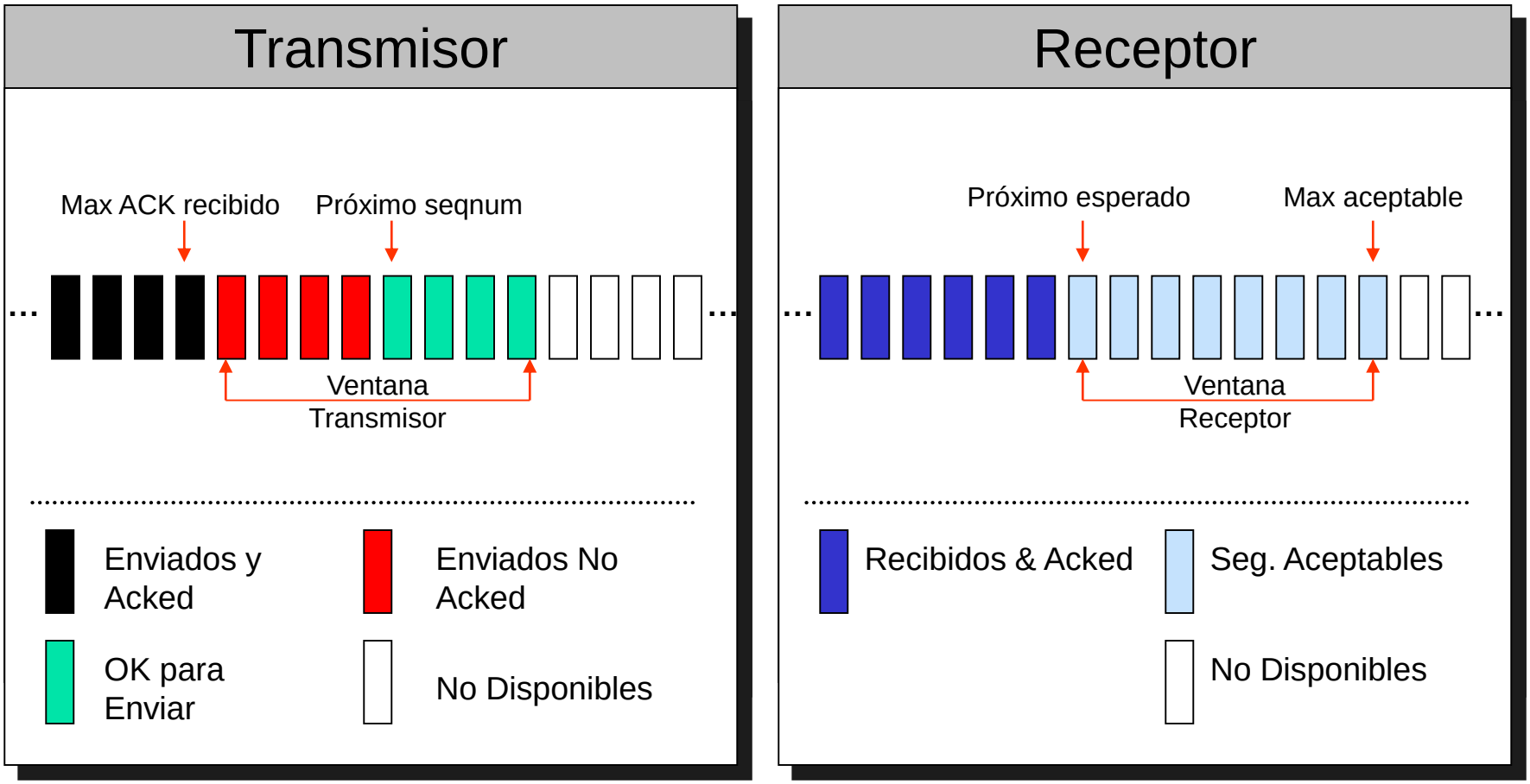
Proto	Local Address	Foreign Address	State
TCP	0.0.0.0:135	0.0.0.0:0	LISTENING
TCP	0.0.0.0:12345	0.0.0.0:0	LISTENING
TCP	10.10.2.38:139	0.0.0.0:0	LISTENING
TCP	10.10.2.38:1132	0.0.0.0:0	LISTENING
TCP	10.10.2.38:1132	10.10.6.170:139	ESTABLISHED
TCP	10.10.2.38:1200	10.10.6.99:1027	ESTABLISHED
TCP	10.10.2.38:1203	10.10.6.105:1306	ESTABLISHED
TCP	10.10.2.38:1207	0.0.0.0:0	LISTENING
TCP	10.10.2.38:1207	10.10.3.228:139	ESTABLISHED

TCP – Número de secuencia

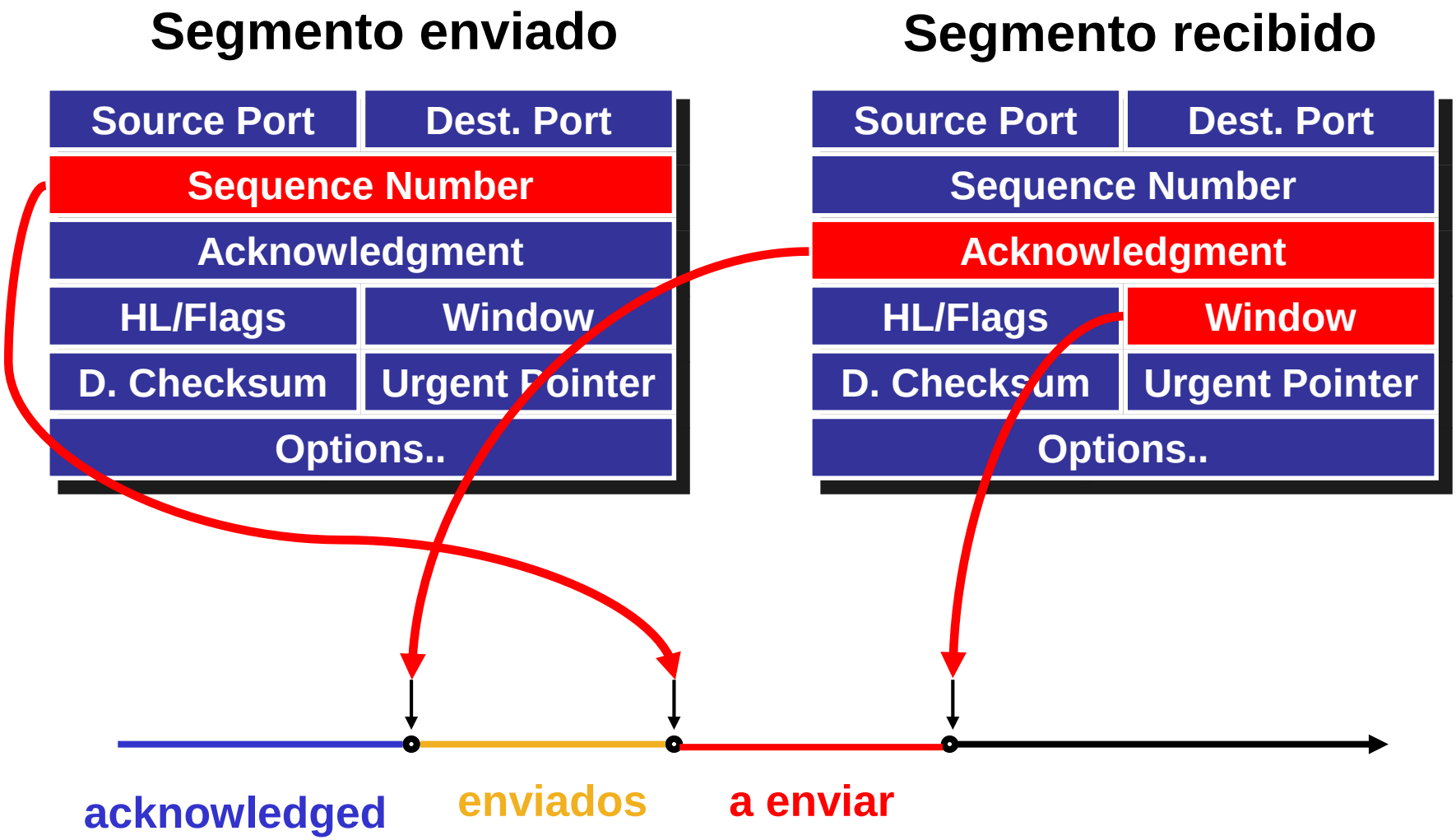
- Número de 32 bits.
- Valor inicial fijado en el establecimientro de la conexión, (ISN).
- TCP desagrega el stream de bytes en segmentos.
  - Cada segmento tiene asociado el número de secuencia.
  - Indica su ubicación en el stream de bytes.



TCP – Ventana deslizante



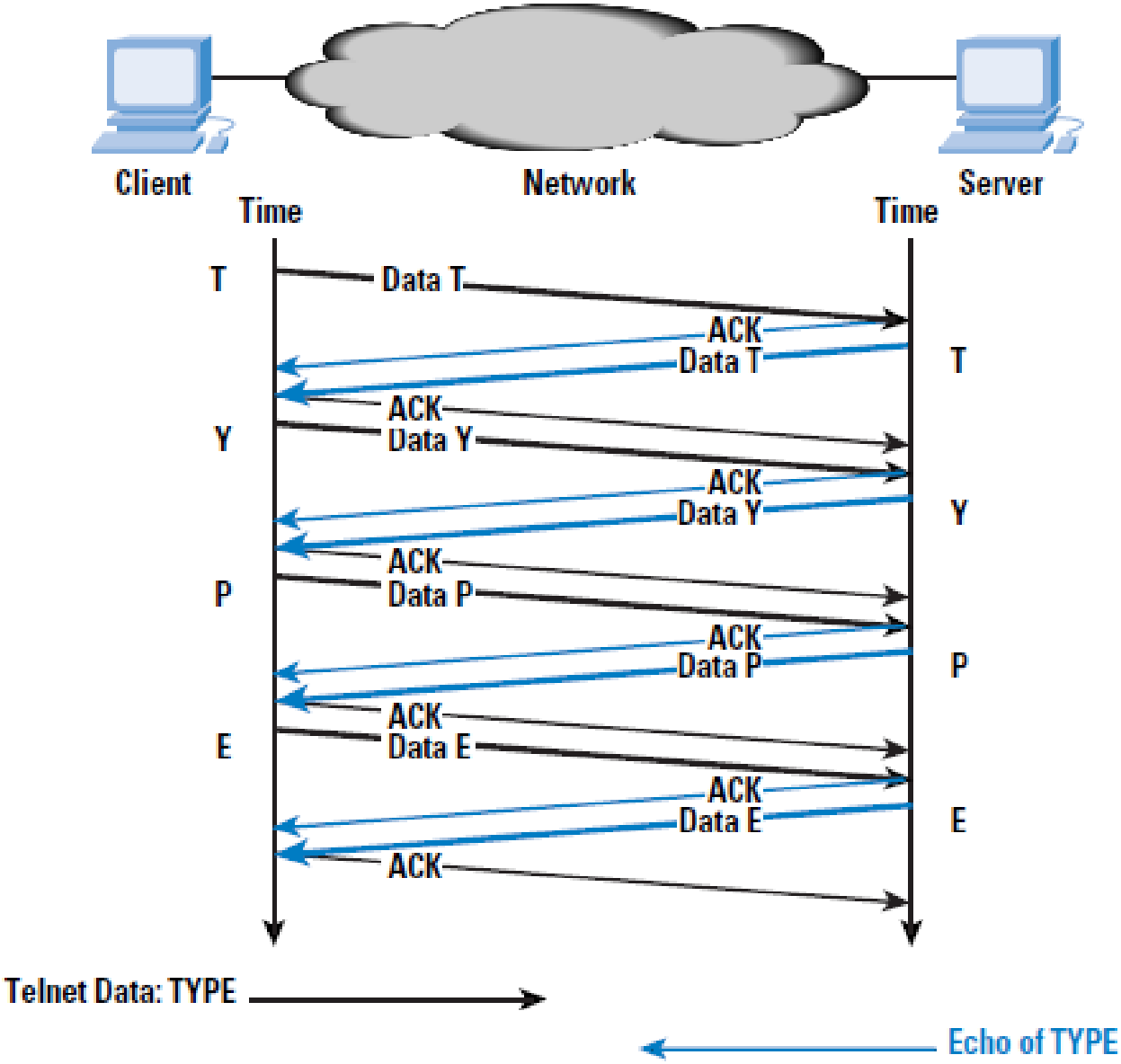
TCP – Ventana deslizante



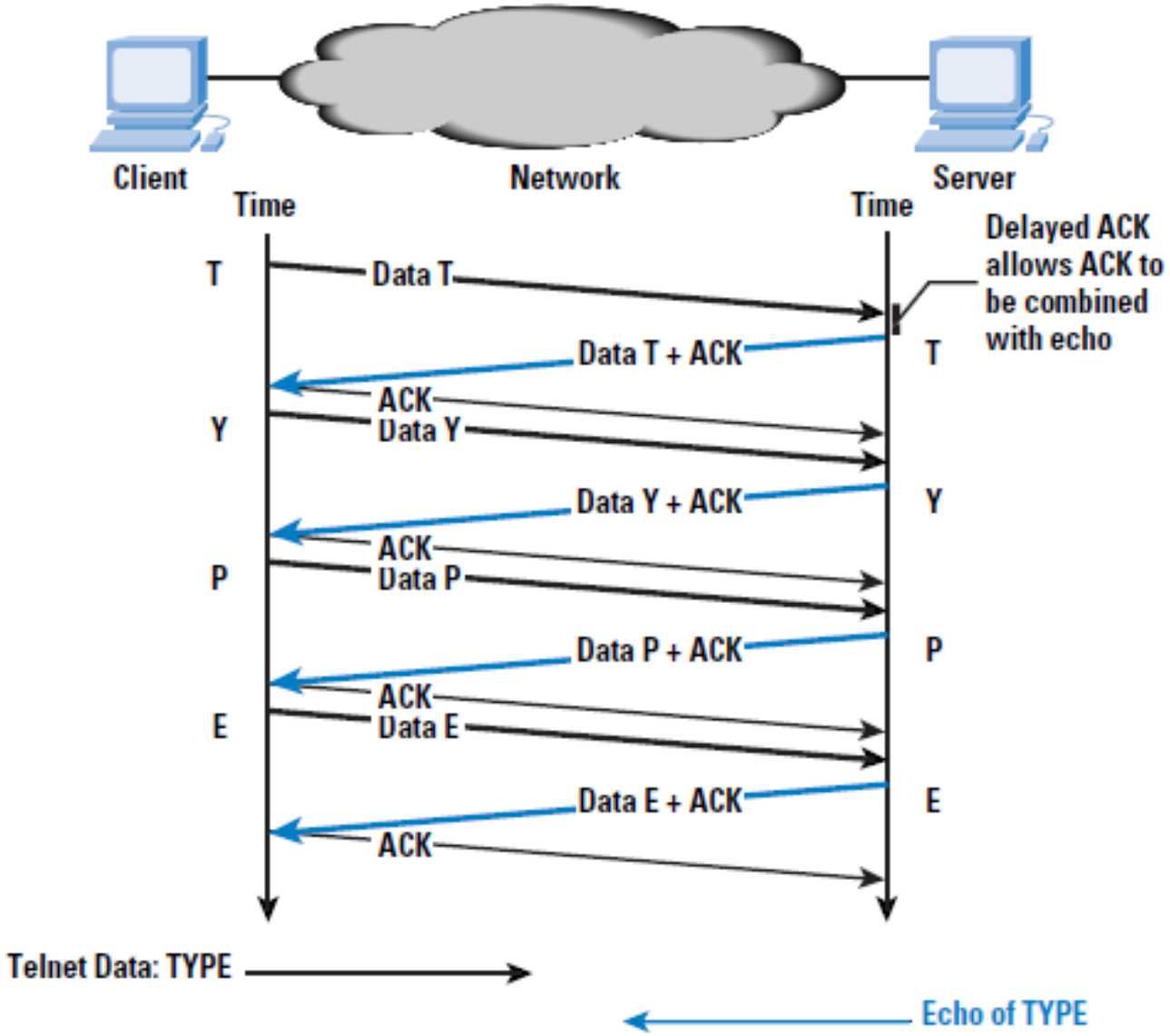
Síndrome “silly window”

- Problema (1982)
  - Si el receptor publica pequeños incrementos en la ventana el transmisor pierde tiempo enviando segmentos pequeños.
- Solución
  - El receptor no debe publicar pequeños incrementos de la ventana
  - Incrementar la ventana en min(MSS,RecvBuffer/2)

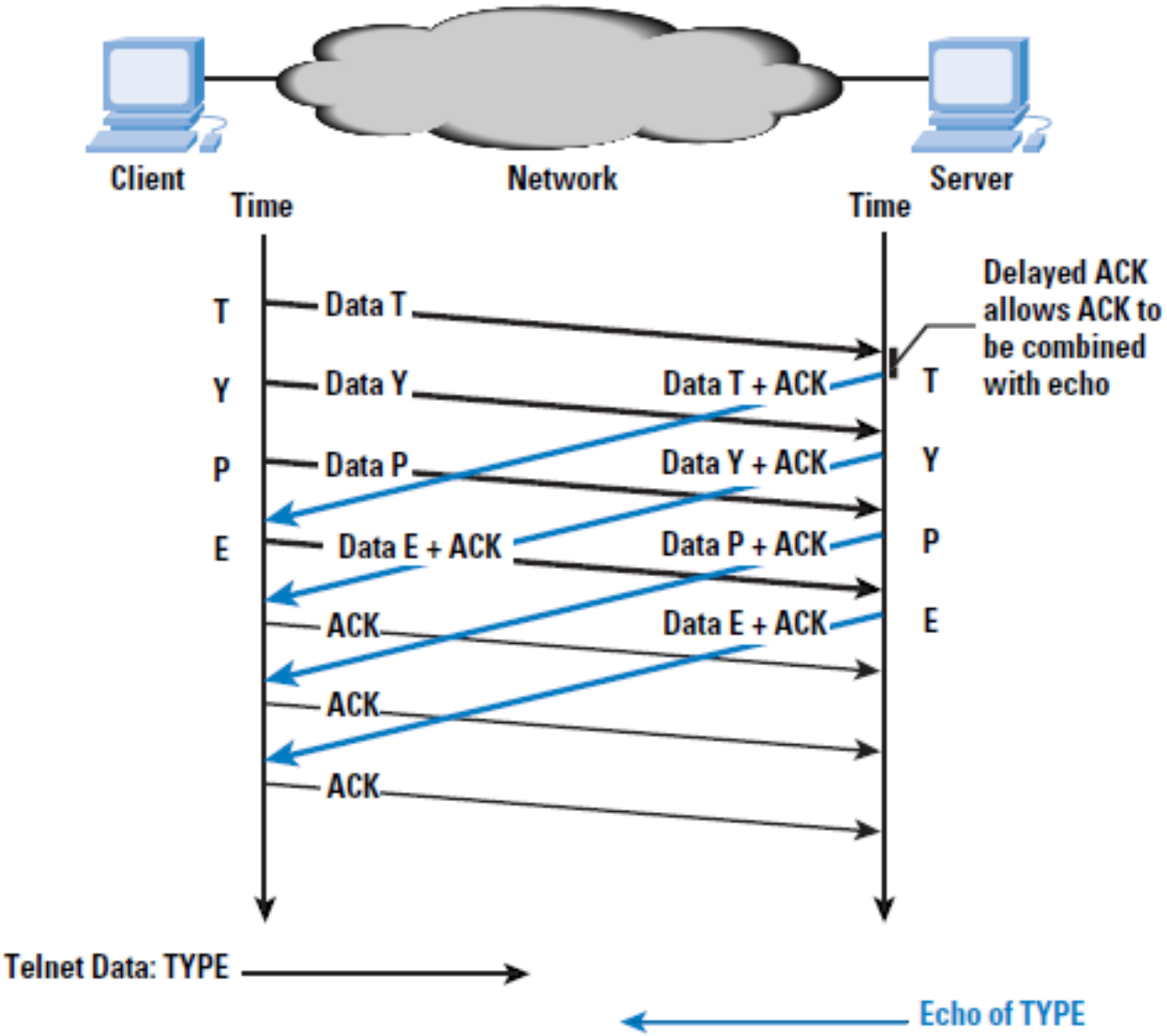
Silly Window



Silly Window (LAN)...



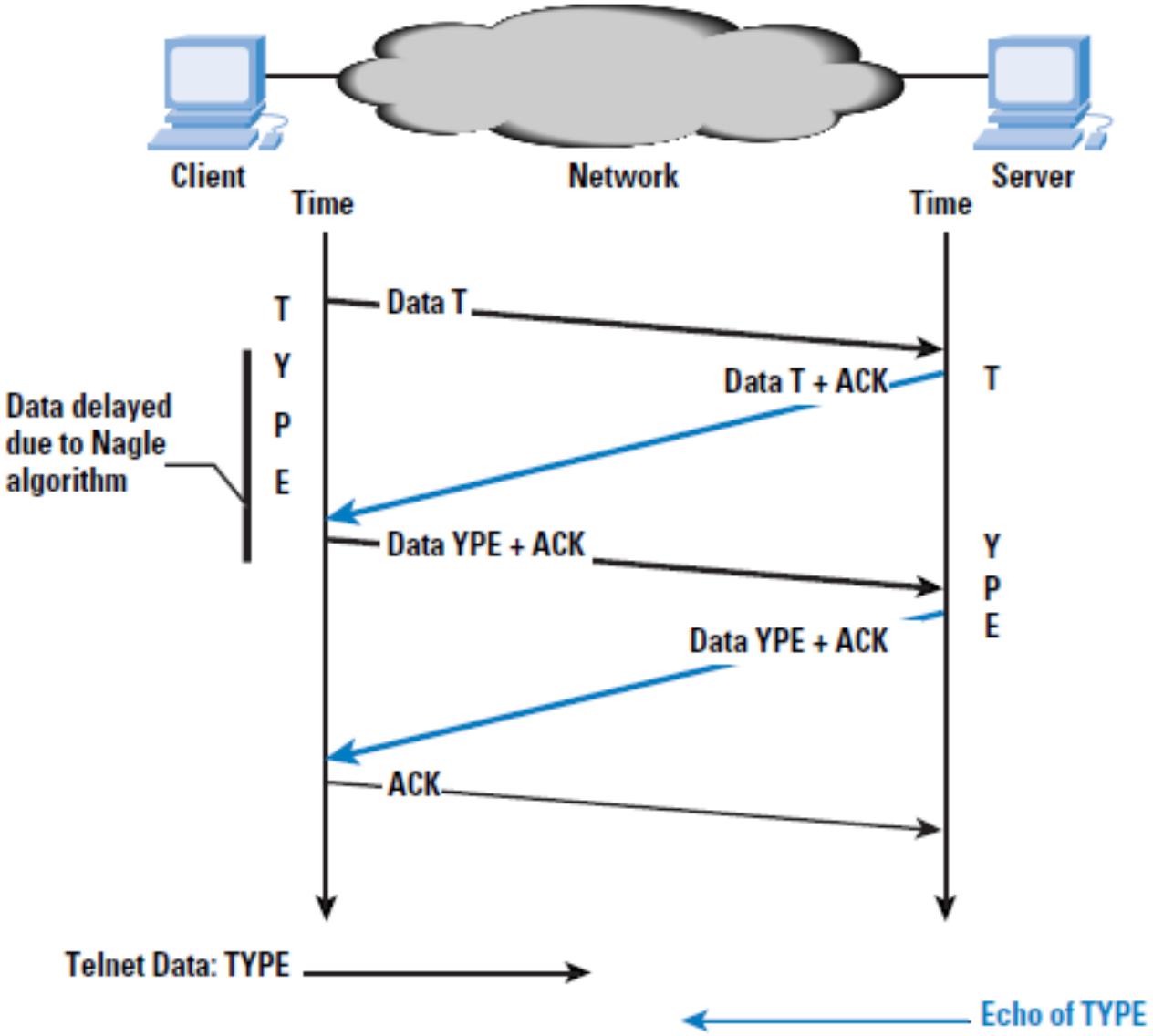
Silly Window (WAN)...



Algoritmo de Nagle

- Problema de segmentos pequeños
  - Aplicaciones que generan un byte por segmento.
  - ¿Esperar por más datos?
- Solución: Algoritmo de Nagle
  - Dejar solamente un segmento pequeño pendiente de confirmación.
- Acknowledges por lotes
  - Retardar el timer del ack
    - Piggyback el ack con el tráfico en sentido inverso
    - Si no disparar el ack en 200ms.

Nagle


Client
Network
Server
Time
Time
T
Y
P
E
Data T
Data T + ACK
T
Data delayed due to Nagle algorithm
Data YPE + ACK
Y
P
E
Data YPE + ACK
ACK
Telnet Data: TYPE
Echo of TYPE

Problemas de Nagle

- Aplicaciones X-Window
- Movimientos del mouse.
- Teclas de función: F1....

Timeout y RTT

- Problema:
  - El RTT varía sustancialmente
  - Demasiado largo =>subutilización
  - Demasiado corto => retransmisiones inútiles.
- Solución:
  - Timeout adaptable, estimando el RTT.

Estimando el RTT

- Chebyshev's Theorem
- Max RTT=Avg RTT+k*Desv
- Avg RTT = (1-x)*Avg RTT + x*Muestra RTT, x = 0.1
- Probabilidad de error < 1/(k**2)
- Válida para toda distribución de muestras.

Timeout = AverageRTT + 4*Deviation

Fijando el Timeout

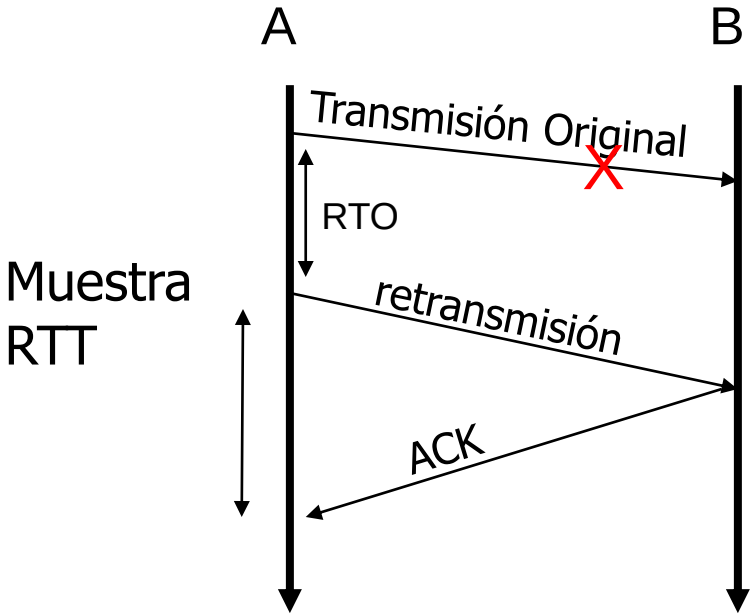
Timeout = Avg RTT + 4*Desv

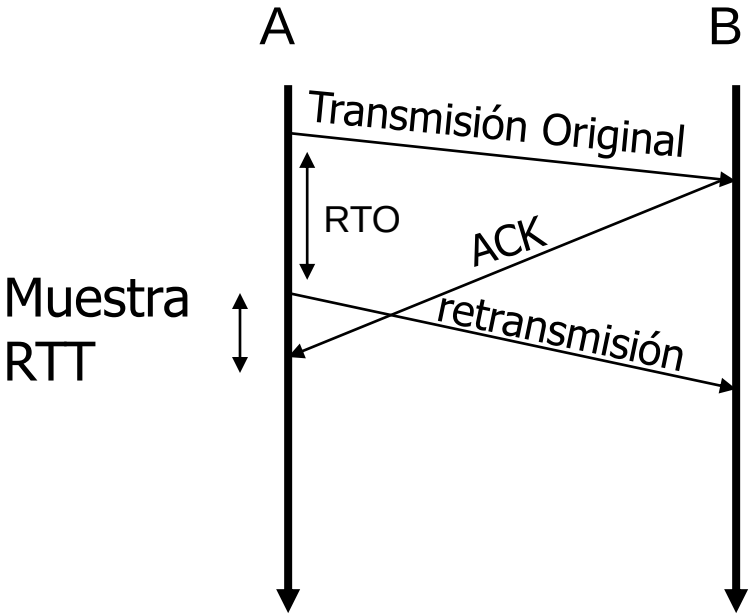
Desv = (1-x)*Desv +

x*|Muestra RTT- Avg RTT|

- Se suele fijar en múltiplos de 200,500,1000mseg.

Ambigüedades

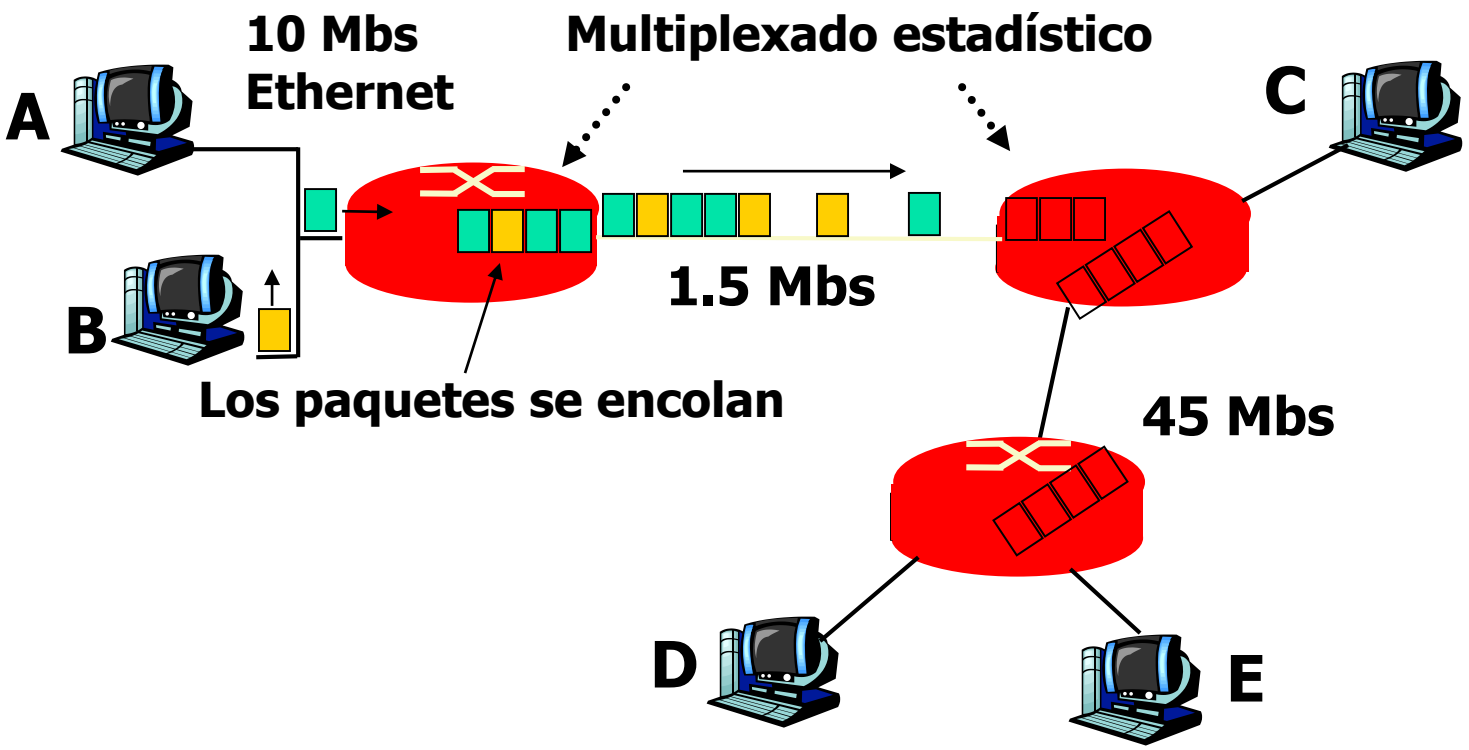




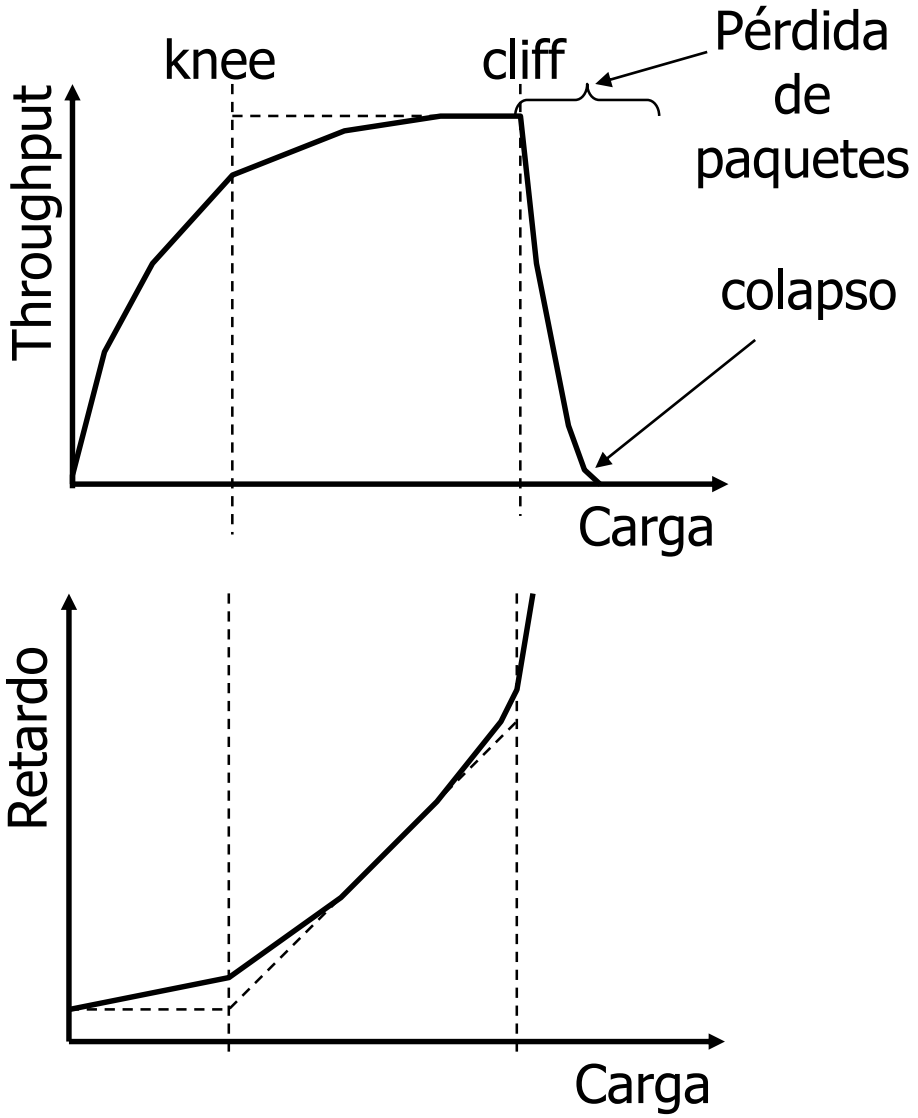
Estimador de Karn

- Ante una retransmisión:
  - No actualizar los RTT
  - Timer backoff
    - RTO = 2*RTO
- Volver a estimar después de una transmisión exitosa.
- Las muestras del RTT se toman en base a la opción de Time Stamp de TCP.

Congestión

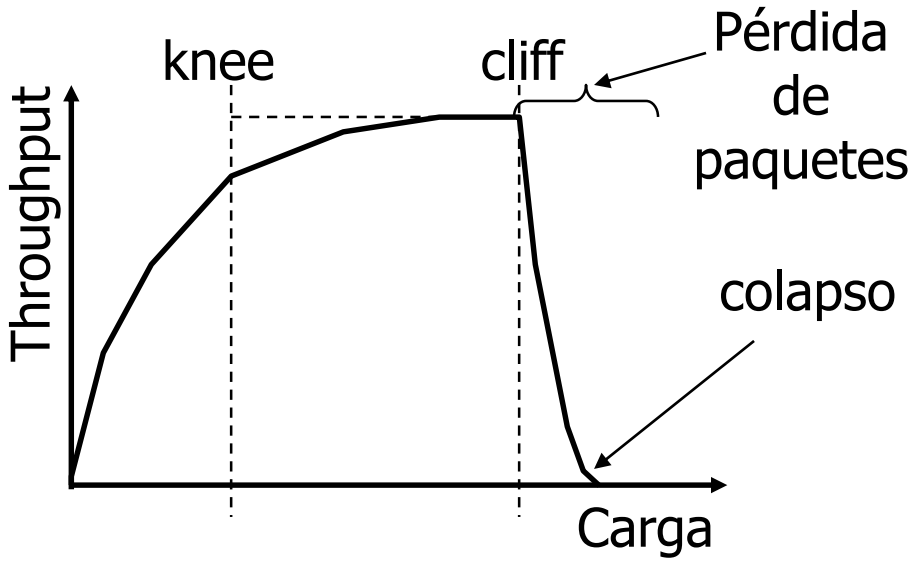

10 Mbs
Ethernet
Multiplexado estadístico
A
B
C
D
E
1.5 Mbs
45 Mbs
Los paquetes se encolan

Congestión...


knee
cliff
Pérdida
de
paquetes
Throughput
colapso
Carga
Retardo
Carga

Congestión: controlar vs. evitar

- Controlar
  - Permanecer a la izquierda del cliff
- Evitar
  - Permanecer a la izquierda del knee



Control de Congestión

- Modelo end-to-end
  - Los extremos son la fuente de la demanda.
  - Los extremos deben estimar los tiempos y grado de congestión y reducir la demanda.
  - Los nodos intermedios deben monitorear el estado de la red.

Control de Congestión...

- Modelo basado en la red
  - Los extremos no son confiables.
  - El nodo de la red tiene control sobre el tráfico.
  - Acciones más rápidas.

TCP – Control de Congestión

- Utiliza tres variables:
  - cwnd: ventana de congestión.
  - rcv_win: ventana del receptor. Publicada en el segmento.
  - ssthresh: valor del umbral. Actualiza cwnd.
- Para el envío
  - win = min(rcv_win,cwnd)

TCP – Slow Start

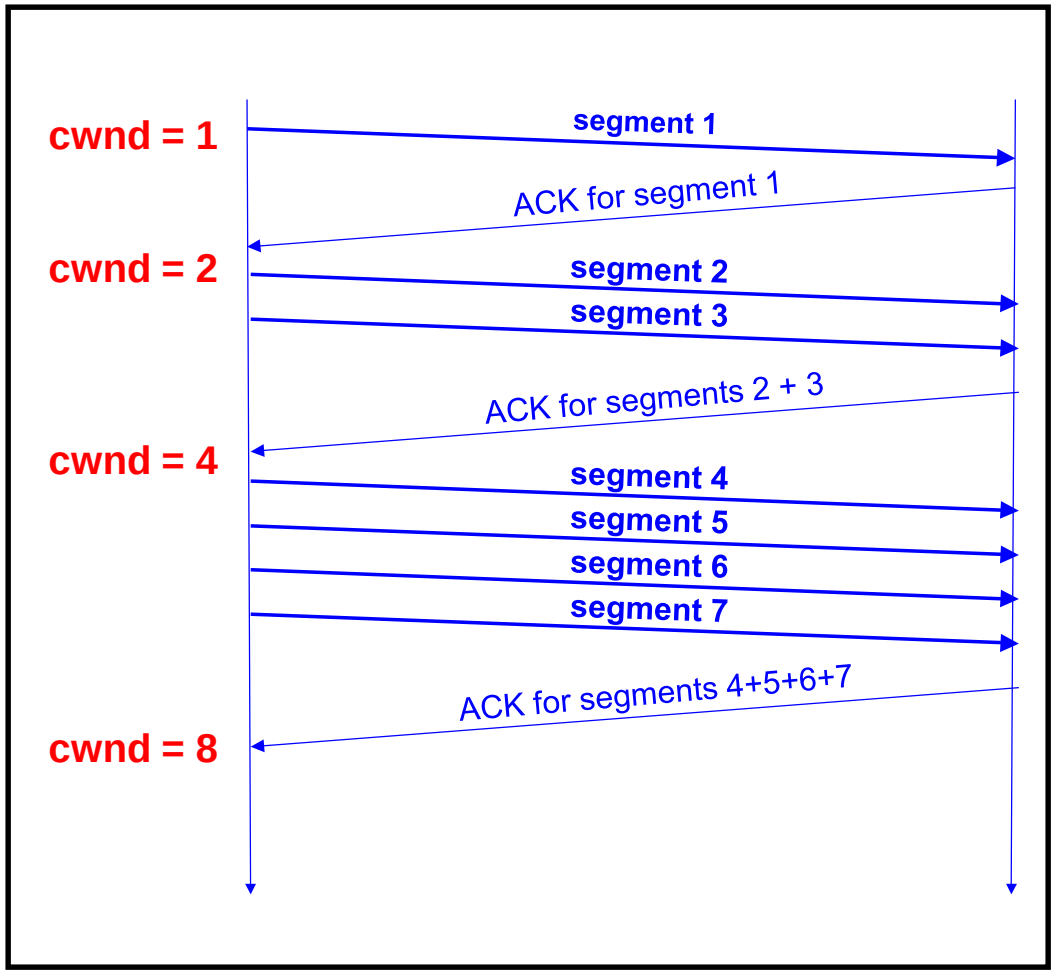
- Inicializa el sistema y descubre la congestión rápidamente.
- Incrementa cwnd hasta la congestión → estima el óptimo cwnd.
- Detecta congestión por pérdida de segmentos.
- Desventajas
  - Detección tardía
  - Enlaces de alta velocidad →ventanas mayores→ mayor pérdida.
  - Interacción con el algoritmo de retransmisión y tiemouts.

TCP – Slow Start...

- En el comienzo o después de congestión:
  - cwnd = 1
  - Después de cada ACK:
    - cwnd ← cwnd+1
- Pese al incremento unitario el crecimiento es exponencial.

Slow Start: Ejemplo

- TCP detiene el crecimiento de cwnd cuando:
  - cwnd >= ssthresh



Evitando la Congestión

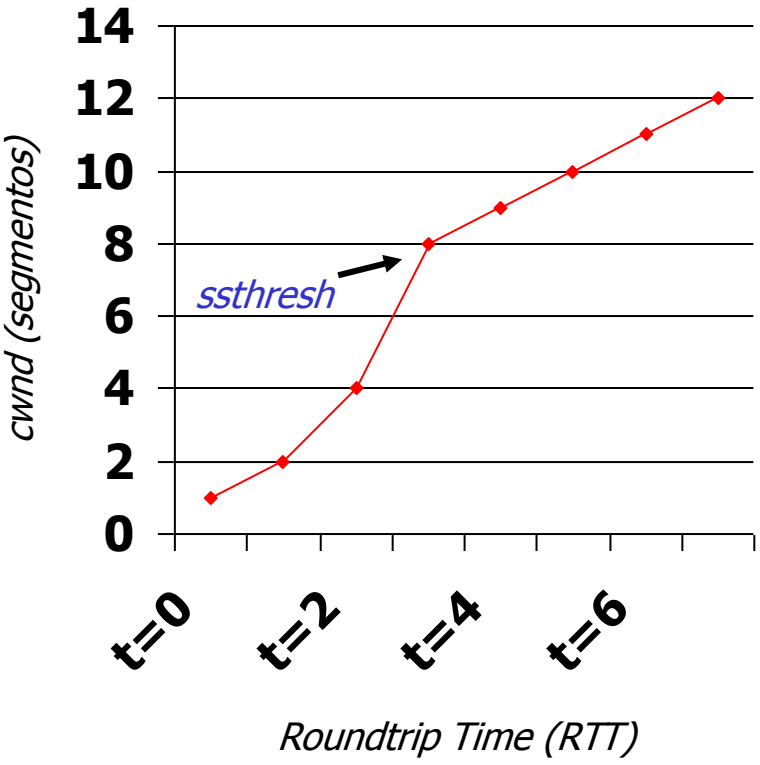
- Mantener la operación a la izquierda del knee.
- Incremento aditivo, comenzar con ssthresh, incrementar cwnd lentamente.
- Disminución multiplicativa: cortar la ventana de congestión drásticamente si se detecta una pérdida.

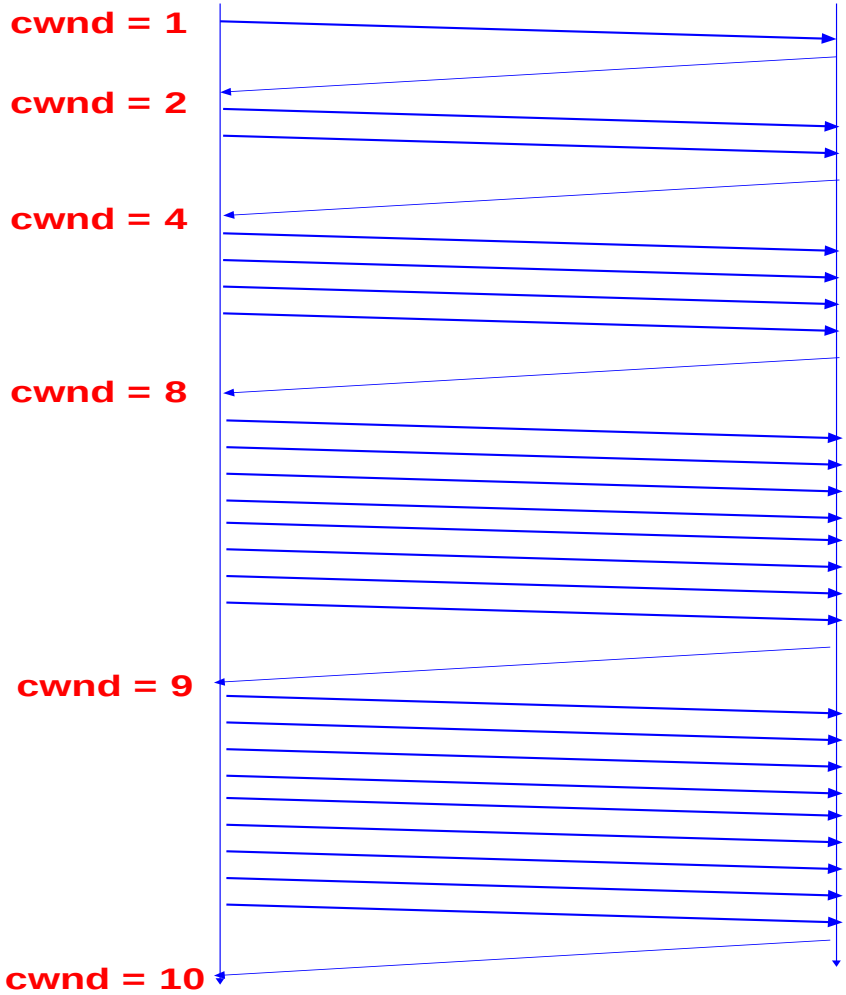
Evitando la Congestión...

- Disminuir la velocidad del “Slow Start”.
- Si cwnd > ssthresh entonces
  - Por cada ACK,
    - cwnd ← cwnd+1/cwnd
- Cwnd se incrementa en 1 si todos los segmentos recibieron su ACK.

Combinando....

- *ssthresh = 8*





Armando el rompecabezas

Inicialmente:

```
cwnd = 1;
ssthresh = infinito;
```

ack recibido:

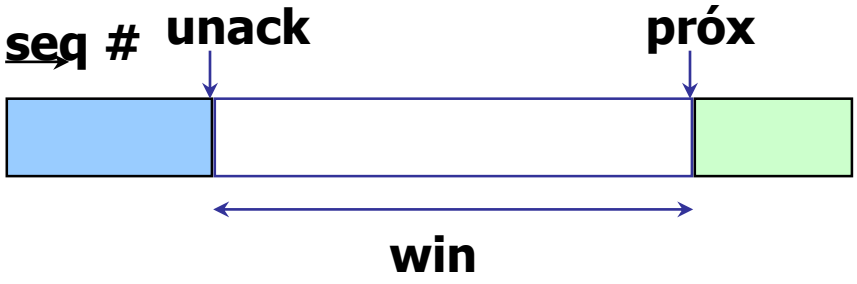
```
si (cwnd < ssthresh)
      /* Slow Start*/
      cwnd = cwnd + 1;
si no
      /* Congestion Avoidance */
      cwnd = cwnd + 1/cwnd;
```

Timeout: (detección de pérdida)

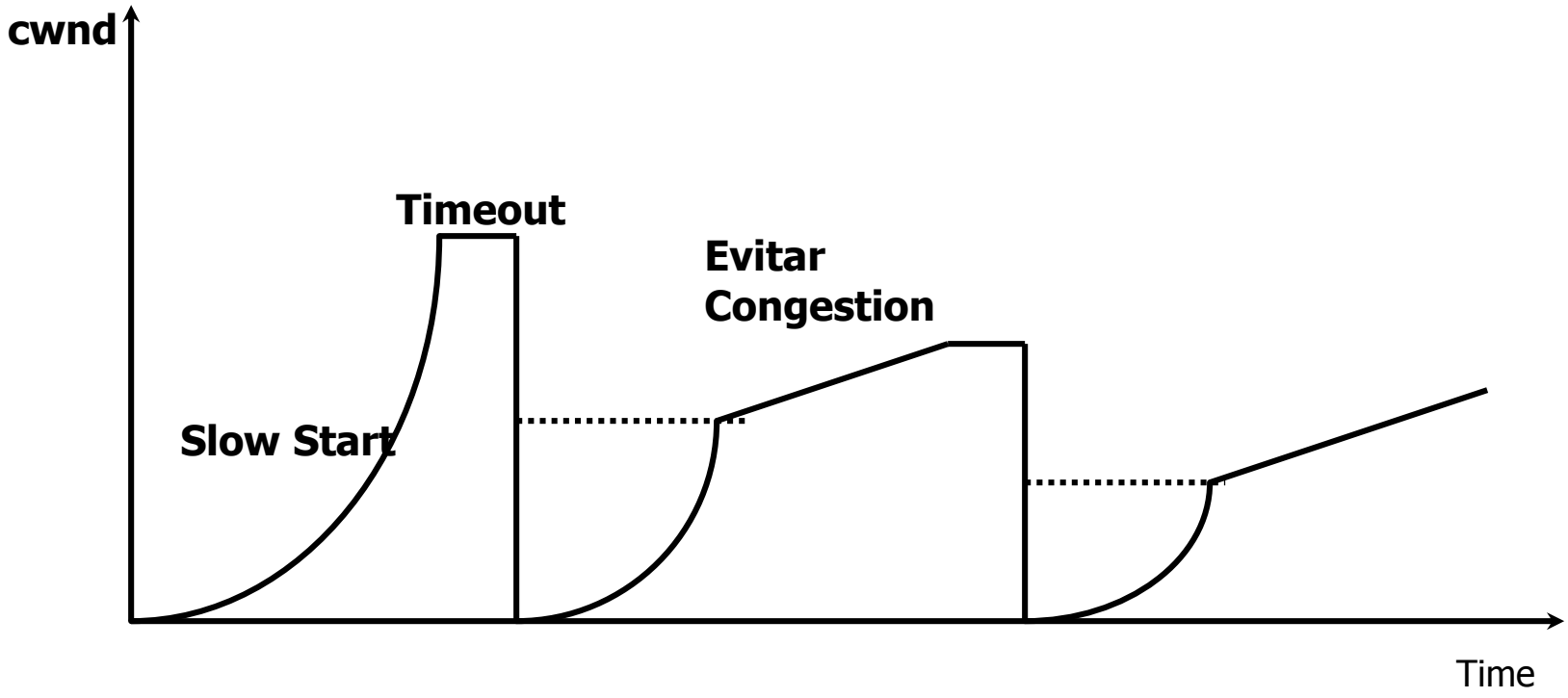
```
/* Multiplicative decrease */
ssthresh = win/2;
cwnd = 1;
```

```
mientras (próx < unack + win)
    transmitir próx segmento;
```

dónde win = min(cwnd, flow_win);



Finalmente....



Detección de paquetes perdidos

- Esperar RTO (Retransmission timeout).
- RTO es usualmente dos veces RTT.
- Degradación de performance.
- No esperar RTO.
  - Utilizar mecanismos alternativos.
  - Utilizar RTO si fallan los anteriores.

Fast Retransmit y Fast Recovery

- Frente a un segmento fuera de orden se debe enviar un ACK.
- Provoca duplicación de ACKs.
- Esta duplicación se ve como debida a:
  - Paquetes perdidos
  - Reordenamiento de paquetes.
- No se puede discriminar.
- Si se reciben 3 ACKs duplicados se considera que se debe a un paquete perdido.

Fast Retransmit y Fast Recovery

- Al recibir el tercer ACK repetido se retransmite sin esperar el RTO.
- Eso es Fast Retransmit
- Luego se ejecuta “congestion avoidance”, no slow start.
- Eso es Fast Recovery.

Integrando....

- Slow Start.
- Congestion Avoidance.
- Si aparecen ACKs duplicados
  - Fast Retransmit y Fast Recovery.
  - Congestion Avoidance.
- Si RTO
  - Slow Start.
- Resumiendo, TCP Reno.

TCP Vegas

- 1994
- Crecimiento más lento que el slow start.
- Nuevo mecanismo de retransmisión.
  - Se chequea el TO al recibir el primer ACK duplicado.
- Nuevo algoritmo de congestion avoidance.
  - Evita las oscilaciones de Reno.
- Monitorea la diferencia entre el throughput estimado y el real.
- Trata de reducir a cero los paquetes almacenados en los buffers de los routers.

Ventana TCP Vegas

$$w_s(t+1) = \begin{cases} w_s(t) + \frac{1}{D_s(t)}, \text{ si } \frac{w_s(t)}{ds} - \frac{w_s(t)}{D_s(t)} < \alpha_s \\ w_s(t) - \frac{1}{D_s(t)}, \text{ si } \frac{w_s(t)}{ds} - \frac{w_s(t)}{D_s(t)} > \beta_s \\ w_s(t), \ \alpha_s < \frac{w_s(t)}{ds} - \frac{w_s(t)}{D_s(t)} < \beta_s \end{cases}$$

$D_s(t) = RTT \ de \ la \ \ fuente \ s$

$d_s = mínimo \ RTT \ de \ la \ \ fuente \ s$

$\alpha_s \ y \ \beta_s, \ parámetros$

Ventana TCP Vegas...

- Ajusta la ventana manteniendo Diff entre α_s y β_s con $\alpha_s < \beta_s$.

- Considerando el throughput de la fuente como:

$$x_s(t) = \frac{w_s(t)}{D_s}$$

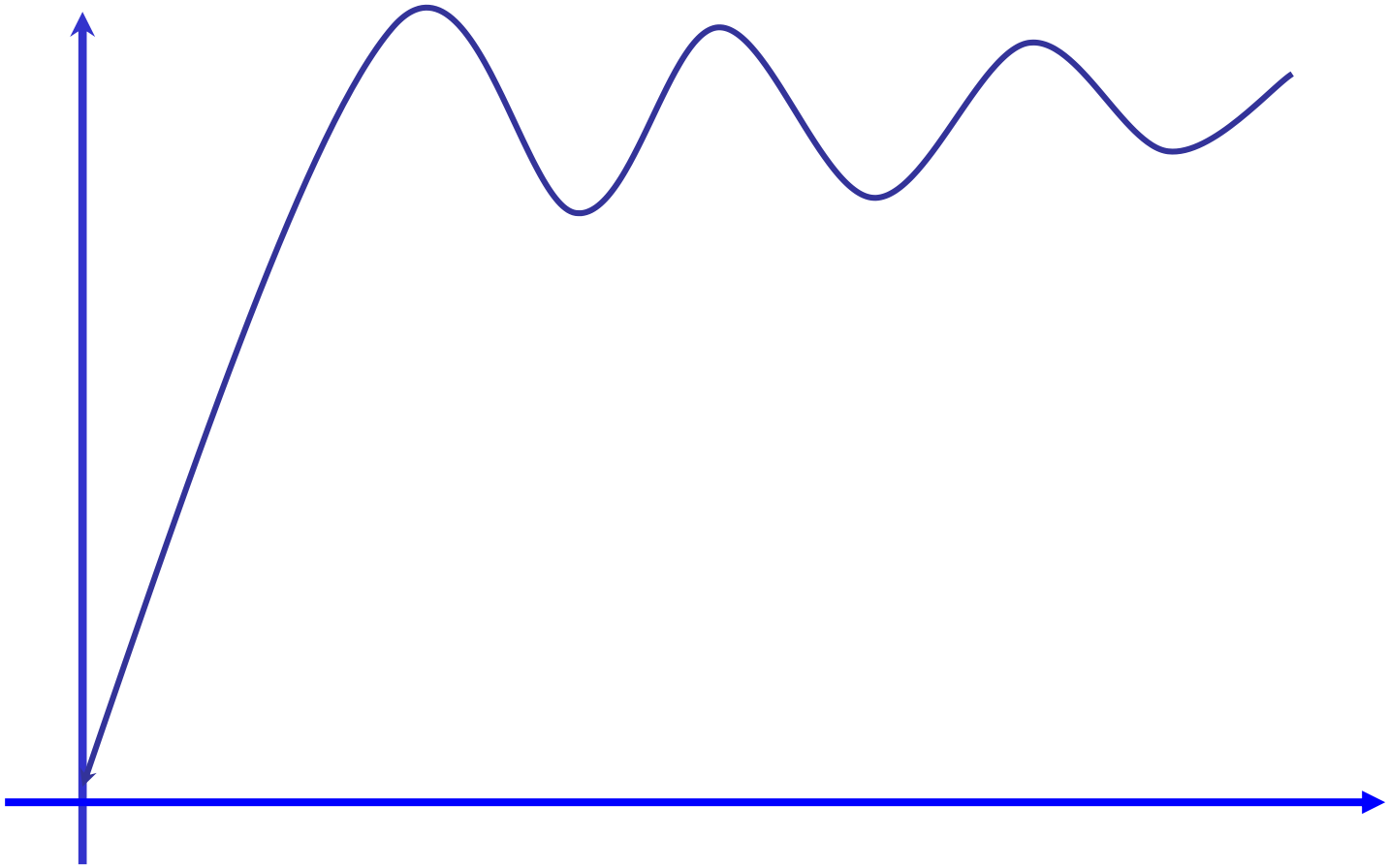
$$w_s(t) - d_s x_s(t)$$

representa el backlog "buffereado" en la red.

Ventana TCP Vegas...

- Si multiplicamos los condicionales anteriores por d_s.
- La fuente incrementa/decrementa la ventana si el backlog es menor/mayor que α_s d_s.

Throughput Vegas



TCP –Persist Timer

- Es necesario hacer una suerte de polling.
- Pueden producirse “deadlocks”.
- Al recibirse una ventana de 0 se activa el persist timer. Normalmente 5 segundos.
- Cuando expira se envía un segmento de 1 byte para verificar el estado del receptor.
- El receptor le contesta acorde con el estado en que se encuentra.

TCP – Persist Timer...

- Si el receptor vuelve a constestar con ACK , wind=0.
  - Entonces el persist timer hace un back-off binario.(10, 20, 40seg...).
  - Al contestar el ACK no validando el byte recibido el emisor continúa enviando este byte de prueba.
- La diferencia con el RTO es que en este caso el emisor envía permanentemente esta prueba hasta que la ventana se incremente o se haga un reset de la conexión.

TCP – Keepalive Timer

- En TCP si no hay intercambio de datos no hay tráfico alguno, pero la conexión persiste.
- Persiste hasta que haya una caída en algún extremo o reboot de alguno de los hosts.
- Este timer se define para interrogar al otro extremo por su estado.
- No es parte de la especificación de TCP.

TCP – Keepalive Timer...

- No se recomienda porque puede provocar caídas en caso que la falla sea transitoria.
- También incrementa el uso del ancho de banda disponible como cualquier otra acción de control.
- Sin embargo en algunos casos es necesario
  - Ej, Telnet.
  - Los servidores necesitan conocer el estado de los clientes dado que están reservando sus recursos para atenderlos.

TCP – Keepalive Timer

- En el extremo en que esté habilitado este timer si no hay actividad en 2 horas envía un segmento de prueba.
- El segmento de prueba es similar al de persist timer.
- Puede también enviarse vacío como un ACK solamente pero con un número de secuencia inesperado.

TCP – Keepalive Timer...

- El receptor se puede encontrar en alguno de estos estados:
  - 1. **Activo**: Responderá al segmento de prueba. El emisor resetea el timer por otras 2 horas. Si en ese intervalo aparece tráfico entonces se vuelve a resetear.
  - 2. **Caído o en proceso de reboot**: El emisor no recibirá respuesta y se genera un timeout a los 75 segundos. Repite este proceso 10 veces en intervalos de 75 seg. Si no recibe respuesta finaliza la conexión.

TCP – Keepalive Timer...

- Estados...
  - 3. **Finalizó el "reboot"**: el emisor recibirá una respuesta que será un reset de la conexión.
  - 4. **Receptor activo pero inalcanzable por el emisor**: el emisor no recibe respuesta y genera un timeout de 75 seg al cabo de los cuales retransmite el byte de prueba. Es similar al caso 2.

Performance

- Depende de :
  - “bandwidth X delay”.
- Los problemas aparecen cuando ese producto es grande.
- Los valores actuales están en:
  - 10^6 bits.

Problemas de performance

- Límite en el tamaño de la ventana
  - 2^16 bytes = 65535 bytes.
- Pérdida de paquetes.
  - Fast retransmit y Fast recovery no son suficientes.
- Medición del Round Trip.
  - Al ser medido por los extremos genera acciones tardías.

Confiablidad de TCP

- En las sesiones de alta velocidad pueden aparecer números de secuencias duplicados.
  - Por “wrap-around” en la sesión corriente.
  - Reencarnación de la sesión.

TCP – "Wrap-around"

- El número de secuencia tiene 32 bits.
- En velocidades altas el espacio de 32 bits puede reciclar dentro del intervalo de tiempo que un segmento es retardado en la red.
- Para evitar el reciclado se requiere una elección adecuada del MSL.

TCP – "Wrap-around"...

- Para conseguir una operación libre de este error:
  - 2^31/B > MSL(seg), donde B es la velocidad efectiva del enlace en bytes/seg.
  - Twrap = 2^31/B.

Red	B (MBps)	Twrap (seg)
Ethernet	1.25	1700
FDDI	12.5	170
Gigabit	125	17

TCP – "Wrap-around"...

- Salidas posibles:
  - Aumentar el número de bits para identificar la secuencia.
  - Mecanismo PAWS (Protect Against Wrapped Sequence numbers).

RTTM

- Estimación ineficiente: una muestra por ventana.
- Enviar un timestamp por cada segmento transmitido.
- El receptor lo responde en cada ACK.
- Por diferencia se obtiene el RTT.

Timestamp

- Se utiliza este mecanismo en ventanas grandes.
- Opción timestamp

kind=8	Length=10	TS value	TS Echo Replay
<1>	<1>	<4>	<4>

- El Echo replay es válido si el flag de ACK está activo. Si no debe ser 0.

Timestamp...

- Si tenemos en cuenta la alineación por palabra:

NOP	NOP	TS option	Length=10
TS value			
TS echo replay			

Timestamp

- Se utiliza el Echo replay recibido si el segmento da acknowledge a datos nuevos.
- Si se recibe más de un timestamp antes de enviar el echo, TCP debe elegir uno sólo de los TS a quien responder.

PAWS

- Definido para rechazar segmentos duplicados y reencarnaciones.
- Utiliza la opción de timestamp.
- Asume que los segmentos de datos y ACK recibidos contienen un valor de TS monótono no-decreciente.
- Un segmento puede ser descartado como duplicado si se recibe con un valor de TS menor que uno anterior.

PAWS

- "Menor que" significa que si *s* y *t* son valores de TS, entonces
  - *s < t* si
    - *0<(t-s)<2^31*
- Los valores de TS enviados en <SYN> y/o <SYN,ACK> inicializan PAWS.
- No requiere sincronización de relojes entre emisor y receptor.

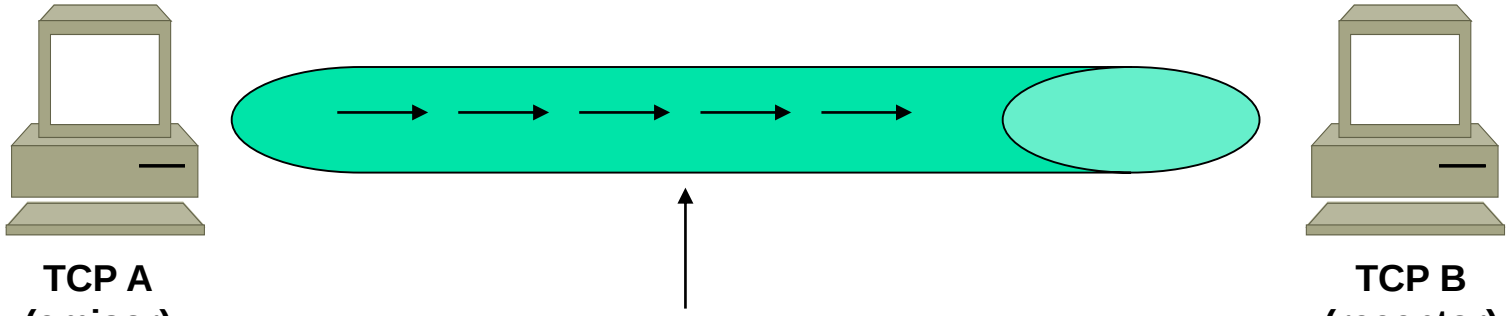
Redes LFN

- Las redes LFN (Long, Fat pipe Networks/Elephant Networks) son las que tienen un elevado ancho de banda y un elevado RTT (retardo).
- El producto de ambos da una idea comparativa de dichas redes.
  - Enlace vía satélite de 2 Mb/s y retardo 500 ms: BW*RTT = 1 Mb
  - Enlace por fibra de larga distancia de 1 Gb/s y RTT = 40 ms: BW*RTT = 40 Mb

TCP en redes LFN

- La ventana de TCP es un campo de 16 bits. Su valor máximo es 65535 bytes.
- En TCP no es posible enviar más de 65535 bytes seguidos sin haber recibido un ACK
- En una red LFN con BW*RTT > 64 Kbytes el rendimiento se puede ver limitado por este motivo. La limitación es tanto mayor cuanto mayor es el BW*RTT de la red

Performance de TCP en LFN



0 ms: TCP A empieza a enviar datos a 2 Mb/s
262 ms: TCP A ha enviado 64 KB y tiene que parar
500 ms: TCP A empieza a recibir los ACK y transmite los siguientes 64 KB
762 ms: TCP A ha enviado el segundo grupo de 64 KB y tiene que parar
1000 ms: TCP A empieza a recibir los ACK del segundo grupo y transmite
1262 ms: TCP A tiene que parar
…

Eficiencia: 262/500 = 52,4 % = 1,048 Mb/s (64 KB/ RTT)

Solución al problema de TCP en redes LFN

- Tener ventanas mayores que 64 KB. Pero el campo es de 16 bytes y no se puede ampliar
- Aplicar un factor de escala al tamaño de ventana.
- Soportado por los dos TCP que establecen la conexión.
- Lo acuerdan al principio de esta y lo mantienen durante toda la conexión

Rendimiento con factor de escala

- Caudal max. = ventana / RTT
- Con RTT = 43 ms y ventana 524280 bits:
  Caudal max. = 524280 / 0,043 = **12,2 Mb/s**

Factor de escala	Tam. Ventana (bits)	Caudal max. (Mb/s)
1	524280	12,2
2	1048560	24,4
4	2097120	48,8
8	4194240	97,6
16	8388480	195,2

Escalaje de ventana

- RFC 1323.
- Extiende la ventana a 32 bits.
- Opción: Window Scale.
- Se envía con SYN y con <SYN,ACK> en el establecimiento de la conexión.
- Representa el valor a escalar su ventana de recepción.

Escalaje de ventana

Kind = 3	Length = 3	Shift.cnt

- Shift.cnt representa la cantidad de bits que se tiene que desplazar a la izquierda el valor publicado de la ventana.