Transparencias

de

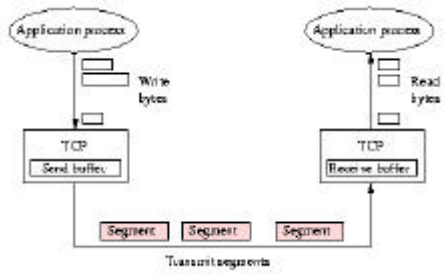
Redes de Ordenadores

Tema 10

Nivel de Transporte: TCP
1ª Parte – TCP

TCP: Características

- *Orientado a la conexión*.
  - Utiliza "3-way handshake" para el establecimiento y liberación de la conexión.
- Servicio de transmisión de una *cadena de bytes*.
  - La aplicación emisora escribe un número de bytes.
  - TCP los divide en segmentos y los envía mediante IP.
  - La aplicación receptora lee la secuencia de bytes

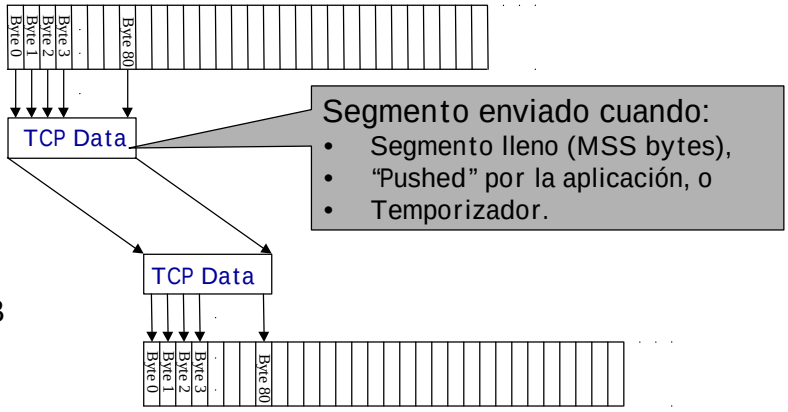


TCP: Características (II)

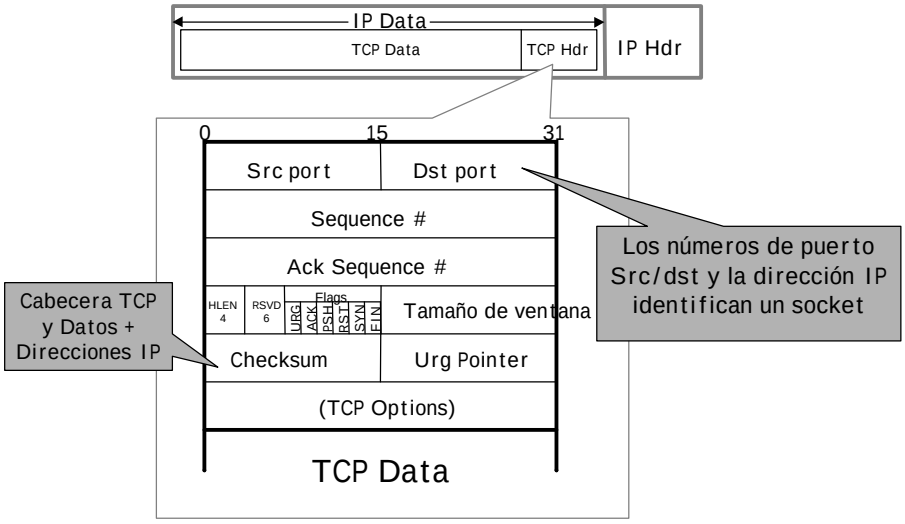
- Servicio fiable:
  - Uso de reconocimientos de datos recibidos.
  - Uso de Checksums para detectar datos corrompidos.
  - Nº de secuencia para detectar pérdidas (temporizadores de retransmisión) o desorden.
  - Control de flujo de ventana deslizante para evita el desbordamiento del receptor.
- *Control de congestion* para repartir la capacidad de la red entre los usuarios.

Flujo continuo de octetos

Nodo A



El formato de los segmentos



El formato de los segmentos

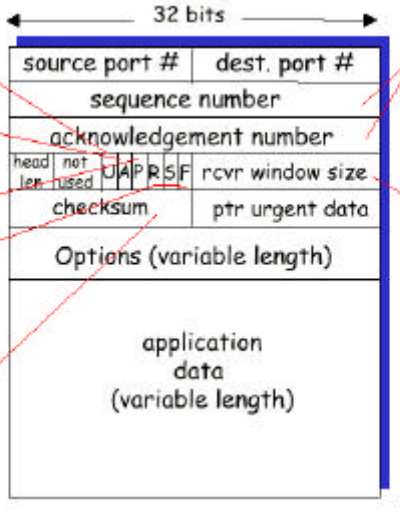
URG: datos urgentes (No usado en general)

ACK: Nº ACK válido

PSH: "Push" de datos (No usado en general)

RST, SYN, FIN: Establecimiento de conexión

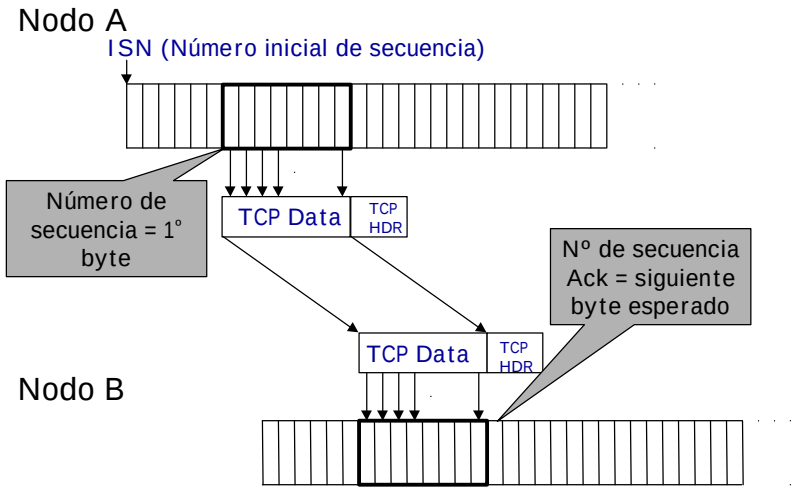
Checksum Internet (como en UDP)



Cuenta de bytes de datos (no segmentos)

Número de bytes que el receptor está dispuesto a aceptar

Números de secuencia



Números de secuencia (II)

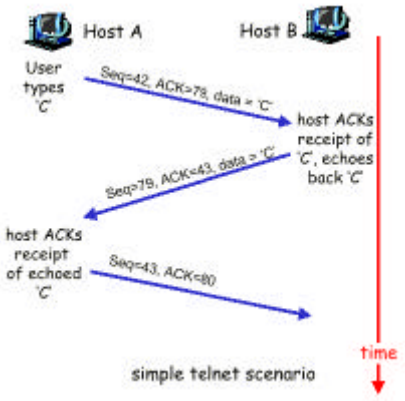
Números de secuencia

- Número de secuencia del primer byte en el segmento de datos

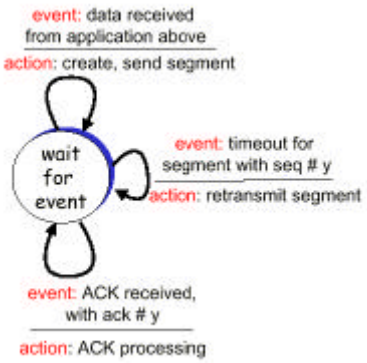
ACKs

- Número de secuencia del siguiente byte esperado del otro extremo
- Acumulativos

- ¿Cómo gestiona el receptor los segmentos desordenados).
  - TCP no lo especifica, depende del implementador



Transferencia de datos fiable

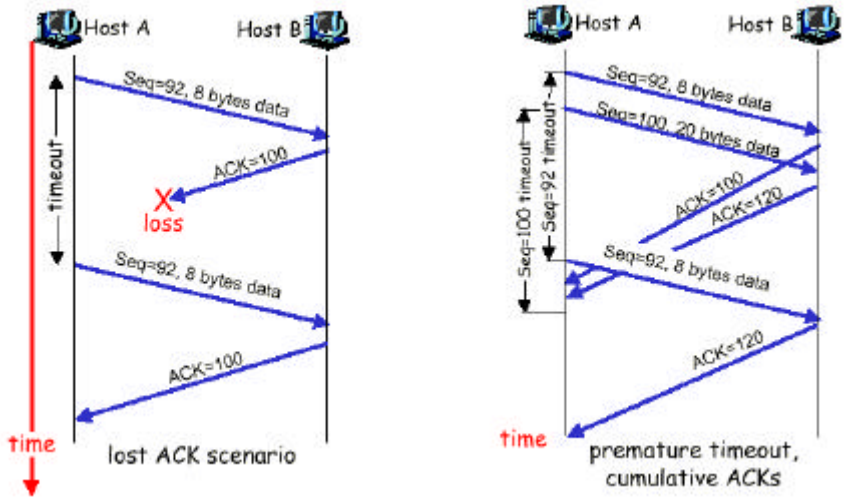



```
00  sendbase = initial_sequence number
01  nextseqnum = initial_sequence number
02
03  loop (forever) {
04    switch(event)
05    event: data received from application above
06       create TCP segment with sequence number nextseqnum
07       start timer for segment nextseqnum
08       pass segment to IP
09       nextseqnum = nextseqnum + length(data)
10    event: timeout for segment with sequence number y
11       retransmit segment with sequence number y
12       compute new timeout interval for segment y
13       restart timer for sequence number y
14    event: ACK received, with ack number y
15       if (y > sendbase) { /* cumulative ACK of all data up thru y-1 */
16          cancel all timers for segments with sequence numbers < y
17          sendbase = y
18          }
19       else { /* a duplicate ACK for ACKed segment */
20          increment number of duplicate ACKs received for y
21          if (number of duplicate ACKS received for y == 3) {
22             /* TCP fast retransmit */
23             resend segment with sequence number y
24             restart timer for segment y
25          }
26  } /* end of loop forever */
```

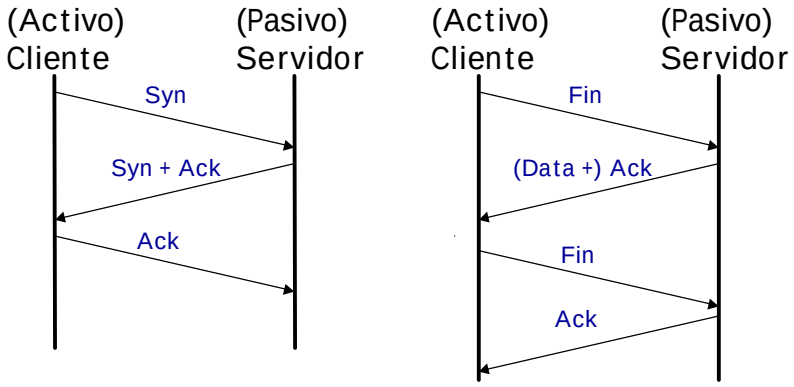
TCP: generación de ACK [RFC 1122, RFC 2581]

Evento	Acción receptor TCP
Llegada de segmento en orden, sin saltos, todo lo demás ya reconocido	ACK retrasado. Espera hasta 500 ms al siguiente segmento. Si no hay próximo segmento envía ACK
Llegada de segmento en orden, sin saltos, un ACK retrasado pendiente	Envía un segmento ACK acumulativo inmediatamente
Llegada de segmento no en secuencia, con número de secuencia mayor de lo esperado, salto detectado	Envía un ACK acumulativo, indicando el número de secuencia del siguiente byte esperado
Llegada de un segmento que llena el salto parcial o completamento	ACK inmediato si el segmento empieza en el extremo inferior del hueco

TCP: retransmisión



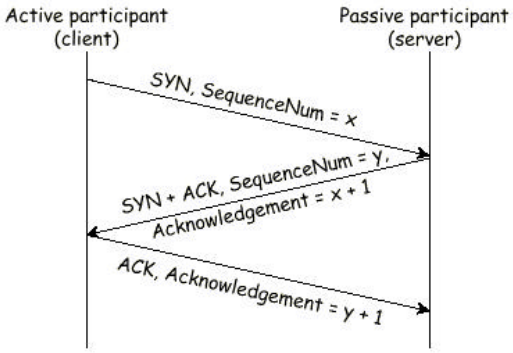
TCP: Gestión de la Conexión



Establecimiento de conexión
3-way handshake

Liberación de conexión
2 x 2-way handshake

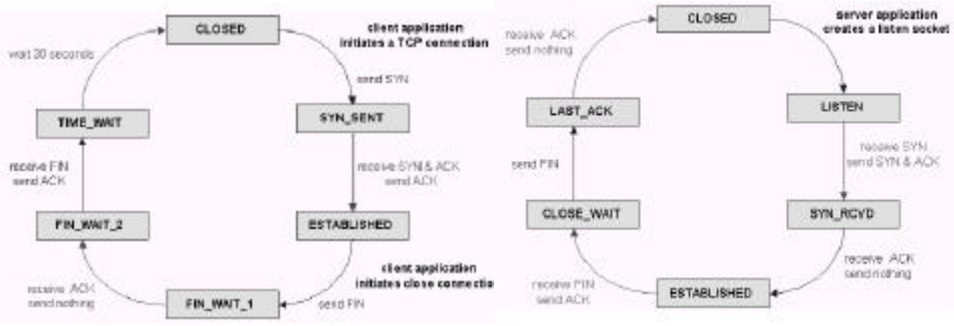
Establecimiento de la conexión



3-way handshake

- **Paso 1**: el cliente envía un segmento TCP SYN al servidor (número inicial de secuencia elegido aleatoriamente)
- **Paso 2**: el servidor recibe el SYN, responde con SYN+ACK
  - Asigna buffers
  - Elige aleatoriamente su nº inicial de secuencia
- **Paso 3**: el cliente responde con ack.

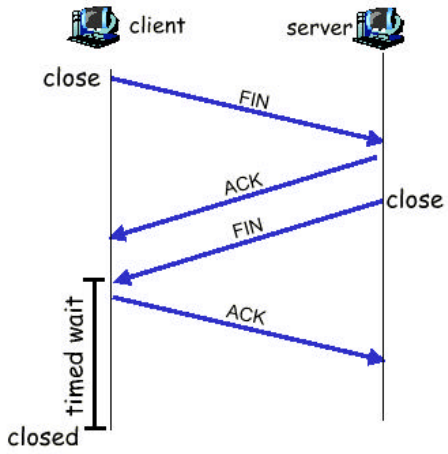
Establecimiento de la conexión (II)



TCP A TCP B

	SEQ	ACK	CTL	
1. CLOSED				LISTEN
2. SYN-SENT	⇒ <100>		<SYN>	⇒ SYN-RECEIVED
3. ESTABLISHED	⇐ <300>	<101>	<SYN,ACK>	⇐ SYN-RECEIVED
4. ESTABLISHED	⇒ <101>	<301>	<ACK>	⇒ ESTABLISHED
5. ESTABLISHED	⇒ <101>	<301>	<ACK> DATA	⇒ ESTABLISHED

Liberación de la conexión

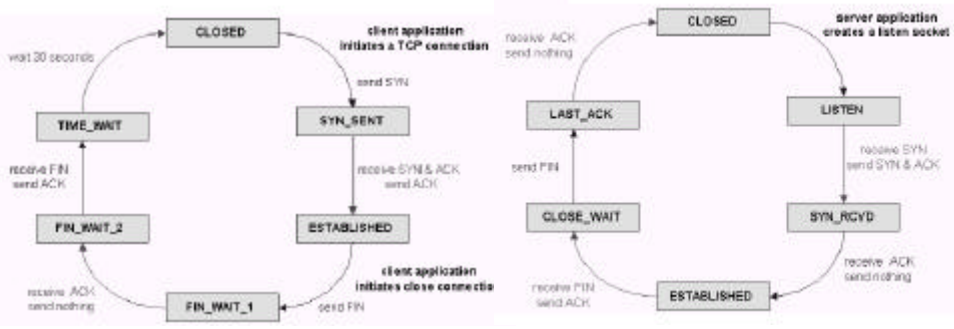


2x2-way handshake

- **Paso 1**: el cliente envía un segmento TCP FIN al servidor
- **Paso 2**: el servidor recibe el FIN, responde con ACK. Después cierra la conexión y envía FIN.
- **Paso 3**: el cliente recibe FIN y responde con ACK.
  - Entra en "tiempo de espera" … responderá con ACK a FIN retransmitidos
- **Paso 4**: el servidor recibe ACK. Cierre de la conexión

- El protocolo contempla FIN simultáneos

Liberación de la conexión (II)



	TCP A	SEQ	ACK	CTL	TCP B
1.	ESTABLISHED				ESTABLISHED
2.	(Close) FIN-WAIT-1	⇒ <100>		<FIN,ACK>	⇒ CLOSE-WAIT
3.	FIN-WAIT-2	⇐ <300>		<ACK>	⇐ CLOSE-WAIT
4.	TIME-WAIT	⇐ <300>		<FIN,ACK>	(Close) ⇐ LAST-ACK
5.	TIME-WAIT	⇒ <101>		<ACK>	⇒ CLOSED
6.	(2 MSL) - CLOSED				

Apertura simultánea de una conexión

	TCP A	SEQ	ACK	CTL	TCP B
1.	CLOSED				CLOSED
2.	SYN-SENT	⇒ <100>		<SYN>	. . .
3.	SYN-RECEIVED	⇐ <300>		<SYN>	⇐ SYN-SENT
4.		... <100>		<SYN>	⇒ SYN-RECEIVED
5.	SYN-RECEIVED	⇒ <100>	<301>	<SYN,ACK>	...
6.	ESTABLISHED	⇐ <300>	<101>	<SYN,ACK>	⇐ SYN-RECEIVED
7.	ESTABLISHED	⇒ <101>	<301>	<ACK>	⇒ ESTABLISHED

Recuperación de un SYN antiguo duplicado

	TCP A		SEQ	ACK	CTL		TCP B
1.	CLOSED						LISTEN
2.	SYN-SENT	⇒	<100>		<SYN>	...	
3.	(duplicate)	...	<90>		<SYN>	⇒	SYN-RECEIVED
4.	SYN-SENT	⇐	<300>	<91>	<SYN,ACK>	⇐	SYN-RECEIVED
5.	SYN-SENT	⇒	<91>		<RST>	⇒	LISTEN
6.		...	<100>		<SYN>	⇒	SYN-RECEIVED
7.	SYN-SENT	⇐	<400>	<101>	<SYN,ACK>	⇐	SYN-RECEIVED
8.	ESTABLISHED	⇒	<101>	<401>	<ACK>	⇒	ESTABLISHED

Descubrimiento de una conexión semiabierta

	TCP A		SEQ	ACK	CTL		TCP B
1.	**(Crash)**						(Send 300, receive 100)
2.	CLOSED						ESTABLISHED
3.	SYN-SENT	⇒	<400>		<SYN>	⇒	**(??)**
4.	**(!!)**	⇐	<300>	<100>	<ACK>	⇐	ESTABLISHED
5.	SYN-SENT	⇒	<100>		<RST>	⇒	**(Abort!!)**
6.	SYN-SENT						CLOSED
7.	SYN-SENT	⇒	<400>		<SYN>	⇒	

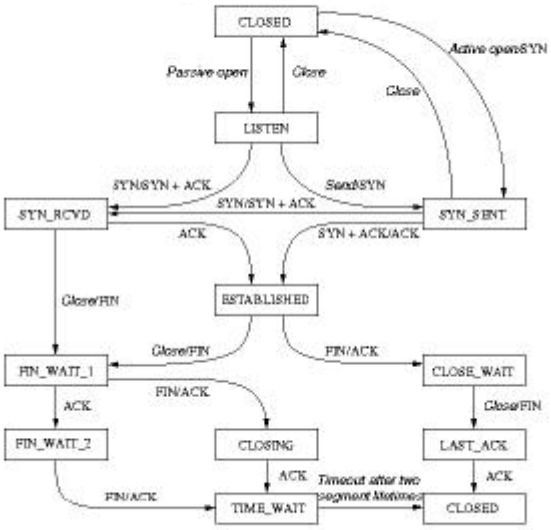
Un SYN antiguo duplicado inicia un Reset en dos conexiones pasivas

TCP A		SEQ	ACK	CTL		TCP B
1. LISTEN						LISTEN
2.	...	< Z >		<SYN>	⇒	SYN-RECEIVED
3. **(??)**	⇐	< X >	<Z+1>	<SYN,ACK>	⇐	SYN-RECEIVED
4.	⇒	<Z+1>		<RST>	⇒	(vuelta a LISTEN)
5. LISTEN						LISTEN

Secuencia de cierre simultáneo

TCP A		SEQ	ACK	CTL		TCP B
1. ESTABLISHED						ESTABLISHED
2. (Close) FIN-WAIT-1	⇒	<100>	<300>	<FIN,ACK>	...	FIN-WAIT-1
	⇐	<300>	<100>	<FIN,ACK>	⇐	
	...	<100>	<300>	<FIN,ACK>	⇒	
3. CLOSING	⇒	<101>	<301>	<ACK>	...	CLOSING
	⇐	<301>	<101>	<ACK>	⇐	
	...	<101>	<301>	<ACK>	⇒	
4. TIME-WAIT (2 MSL) CLOSED						TIME-WAIT (2 MSL) CLOSED

Diagrama de Transición de Estados

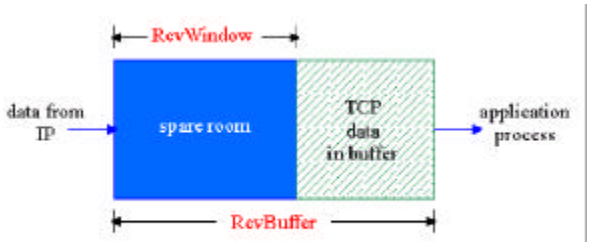


TCP: control de flujo

- Se basa en el mecanismo de ventana deslizante pero la situación es diferente al nivel de enlace de datos.
- Puede conectar muchos nodos diferentes
  - Necesita establecimiento y liberación explícita de la conexión
- Diferentes RTT potenciales
  - Necesita un mecanismo adaptativo de timeout
- Retraso elevado en la red
  - Necesita estar preparado para la llegada de segmentos muy antiguos
- Diferente capacidad en el destino
  - Necesita acomodar diferentes capacidades de almacenamiento
- Diferente capacidad de red
  - Necesita estar preparado para la congestión de la red

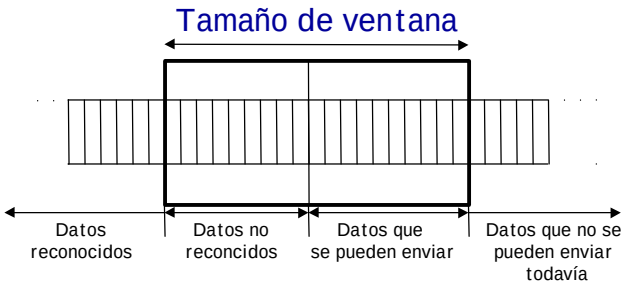
TCP: Control de flujo (II)

- Mecanismo de ventana deslizante.
- Receptor: informa "explícitamente" al emisor del espacio libre en el buffer (dinámico)
  - *Rcvr window Size* en el segmento TCP
- Emisor: cantidad de datos transmitidos no reconocidos, menor que *Rcvr window Size* más reciente.



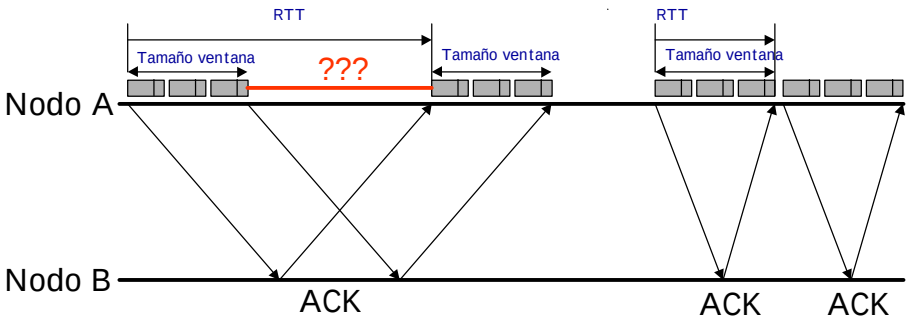
receiver buffering

Ventana deslizante TCP



- La retransmisión es con Retroceso a N
- El receptor notifica el tamaño actual de la ventana (generalmente 4k – 8kbytes al iniciar la conexión).

Ventana deslizante TCP (II)



(1) RTT > Tamaño de ventana

(2) RTT = Tamaño de ventana

¿Cuándo acepta un segmento el receptor?

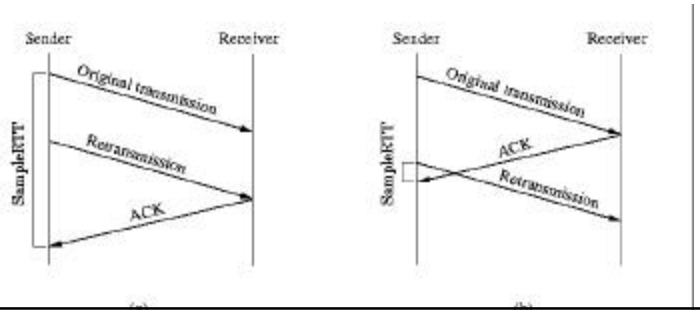
Longitud Segmento	Ventana Recepción	Prueba
0	0	SEG.SEQ = = RCV.NXT
0	>0	RCV.NXT <= SEG.SEQ < RCV.NXT+RCV.WND
>0	0	no aceptable
>0	>0	RCV.NXT <= SEG.SEQ < RCV.NXT+RCV.WND or RCV.NXT <= SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

Cuando llegan segmentos erróneos

- Envía RST (permanece en el mismo estado)
  - Estado "closed" y llega cualquier cosa salvo RST
  - Estados "listen", "syn-sent", "syn-received" y ACKS recibidos acerca de algo no enviado
- Envía ACK – "established" y otros estados sincronizados
- Utiliza seq, números ack que otro lado creerá (p.e., de los ACK entrantes o datos)

Temporizador y RTT

- ¿Cómo ajustar el valor del temporizador?
  - Mayor que RTT (variable)
  - Demasiado corto: expiración prematura y retransmisiones innecesarias
  - Demasiado largo: reacción lenta frente a la pérdida de segmentos
- ¿Cómo estimar RTT?
  - Muestreo de RTT: Medir el tiempo desde la transmisión de un segmento hasta la recepción del ACK (Algoritmo de Karn/Partridge)
    - Ignorar la retransmisiones y ACKs acumulativos de segmentos



Temporizador y RTT (II)

- Algoritmo de Jacobson/Karels
- Las muestras de RTT varían en cada instante, una estimación de RTT menos variable:
  - Utilizar varias mediciones recientes no sólo el último muestreo
  - *RTTestimada =(1-x) * RTTestimada + x * RTTmuestreada*
  - La influencia de un muestreo decrece con rapidez exponencial
    - Un valor típico de x es 0,1
- Para fijar el temporizador se tiene en cuanta la desviación del valor de RTT:
  - RTT más un margen de seguridad
  - A mayor variación de RTTestimada ⇒ margen de seguridad mayor
  - *Timeout = RTTestimado + 4 * Desviación*
  - *Desviación = (1-x) * desviación + x * abs(RTTmuestreada-RTTestimada*)

Rendimiento TCP (sin errores)

- Ventana max TCP = 64KB
  - Rendimiento max TCP
    - internacional - 100ms RTT - 5.2Mbps
    - metropolitano - 1ms RTT - 520Mbps
    - local - 0.1ms RTT - 5.2Gbps
  - El problema es el tamaño máximo de ventana de 64KB
- Solución: Añadir la opción *window scale*
  - Ambos extremos de la conexión TCP acuerdan al establecerla utilizar factores de escala
  - Cada extremo informa al otro del factor de escala que aceptará
  - *Ventana de recepción real = Ventana Recibida Segmento * 2**shift-cnt*
  - Max shift-cnt = 14
  - Ventana max = 64KB * 2**14 = ~8Gb
  - Rendimiento max TCP internacional = 80Gbps

Control de Congestión

- La red no es capaz de gestionar todos los datos cursados.
  - Uno de los problemas principales
  - Diferente del control de flujo
- Manifestaciones:
  - Pérdidas de paquetes (se sobrepasa el buffer de los routers)
  - Retrasos elevados (encolamiento en los routers)
- Costo de la congestión:
  - Cuando un paquete se elimina, la capacidad de transmisión usada por dicho paquete se desperdicia
- Enfoques:
  - Control de congestión extremo a extremo
    - NO es preciso una realimentación explícita de la red
    - La congestión se infiere de la pérdida o retraso observado en los sistemas finales
    - Enfoque adoptado por TCP
  - Control de congestión asistido por la red
    - Los routers proporcionan información a los sistemas finales
    - Un bit indicando la congestión (SNA, DECbit, TCP/IP ECN, ATM)
    - Se indica explícitamente la velocidad al emisor

Control de Congestión (II)

- Control extremo a extremo, sin actuación en la red.
- Velocidad de transmisión limitado por el tamaño de la ventana de congestión (Congwin)
  - WinSize =Min(Congwin,Rcvwinsize) donde RcvWinSize es la ventana notificada por el receptor a efectos de control de flujo.

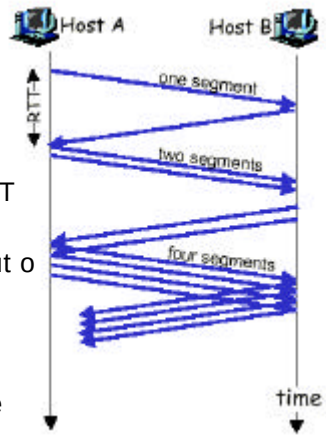
  - W segmentos, cada uno con MSS bytes enviados en un RTT:
  - Rendimiento = W * MSS / RTT (bytes / seg)

Control de congestión (III)

- TCP utiliza una ventana que superpone a la ventana notificada por el receptor:
  - *Window = min (Rcvr window Size, Cong window)*
  - Transmitir tan rápido como se pueda sin pérdidas (Congwin tan grande como sea posible)
  - Incrementar Congwin hasta que se produzcan pérdidas (congestión)
  - Pérdidas: decrementar Congwin, y comenzar el incremento de nuevo
- Dos fases:
  - Arranque lento
  - Anulación de Congestión ("Congestion avoidance")
- Variables importantes
  - Congwin
  - Threshold: ½ del valor de Congwin previo a la congestión
  - Congwin cuenta bytes en TCP (aquí por comodidad lo haremos en segmentos

Arranque lento

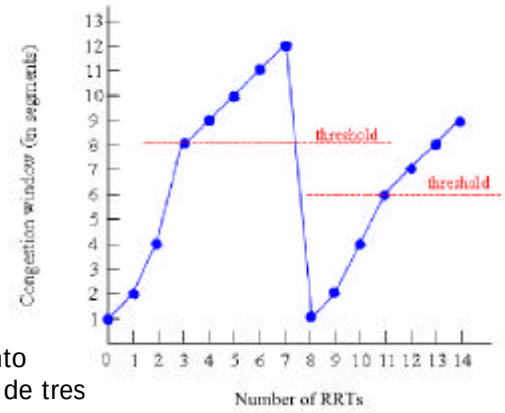
- Algoritm0:
  - Inicializar: *Congwin=1*
  - Con cada segmento reconocido *Congwin=Congwin+1*
  - Hasta (pérdida o Congwin>Threshold)



- El tamaño de ventana se duplica por cada RTT (no demasido lento)
- Evento Pérdida: timeout (Tahoe TCP), timeout o tres ACKs duplicados (Reno TCP)
- El arranque lento sigue hasta que se produce una pérdida desde el inicio de la conéxión
- Problema: Pérdida de datos hasta la mitad de Congwin

Anulación de Congestión

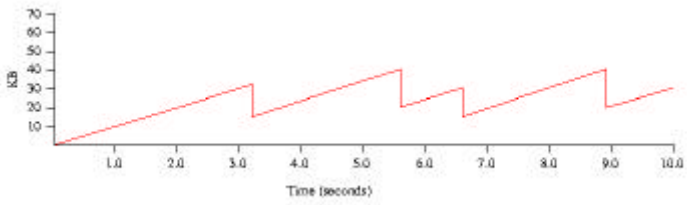
- Algoritmo
  - Finaliza arranque lento
  - Congwin>Threshold
  - Hasta evento pérdida
    - Cada segmento Congwin reconocido Congwin ++
  - Threshold = Congwin/2
  - *Congwin=1*
  - Arranque lento



- Reno TCP evita el arranque lento (recuperación rápida) después de tres ACKs duplicados

Aumento aditivo/Disminución multiplicativa

- Incrementar Congwin cuando la congestión disminuye
- Decrementar Congwin cuando la congestión aumenta
- ¿Cómo determina el emisor si la red está o no congestionada?
- Se produce un timeout
  - Un timeout señala la pérdida de un paquete
  - Los paquetes se pierden raras veces por errores de transmisión
  - La pérdida de un paquete indica una congestión
- Algoritmo:
  - Incrementar Congwin en 1 paquete por RTT (aumento lineal)
  - Dividir Congwin entre 2 cuando se produce una pérdida (disminución multiplicativa)



Retransmisión y recuperación rápida

- Problema: Los temporizadores TCP conducen a periodos desocupados
- Retransmisión rápida: utilizar ACKs duplicados para disparar la retransmisión
- Resultado
- Recuperación rápida: elimina la fase de arranque lento; ir directamente a la mitad del últimc Congwin con éxito