Transporte: Ventana Deslizante

2019

Contenidos

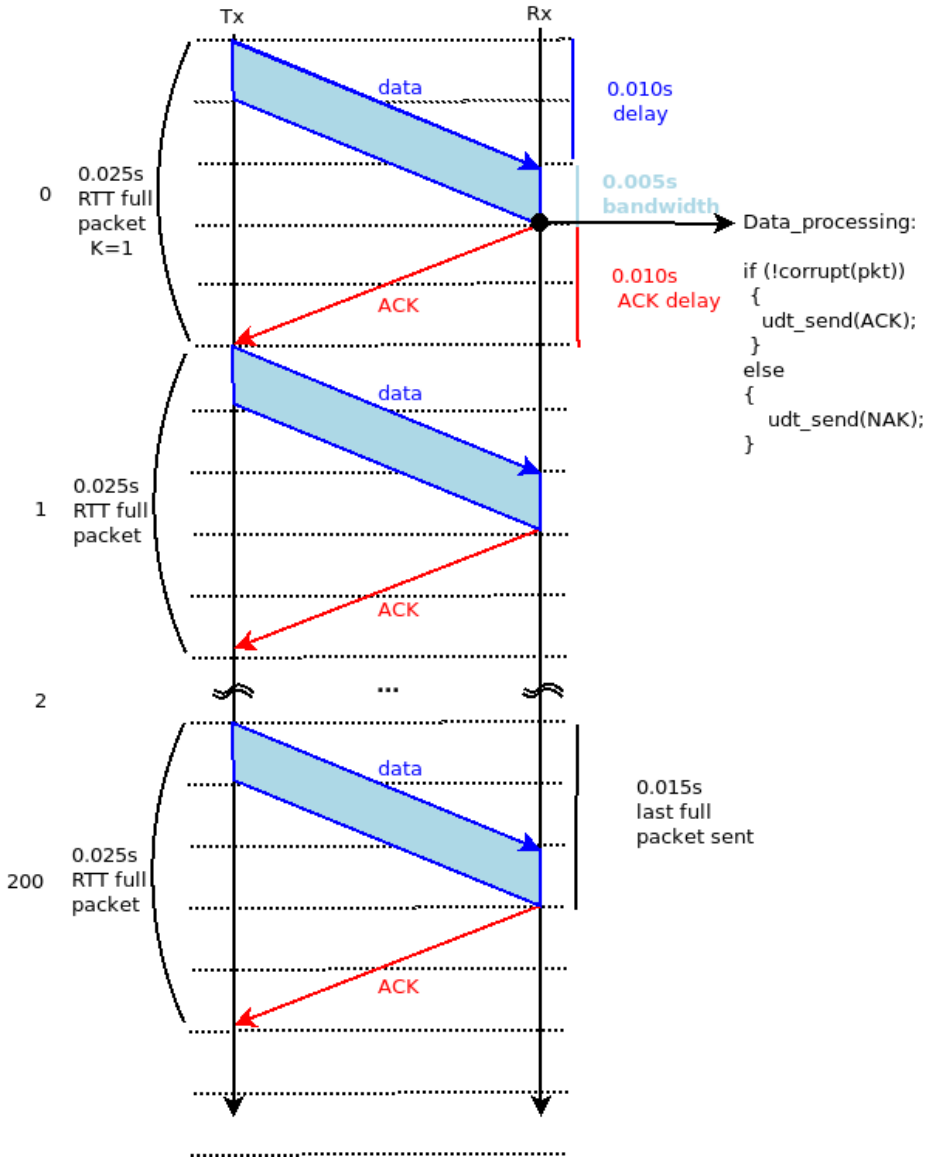
Contenidos

Contenidos

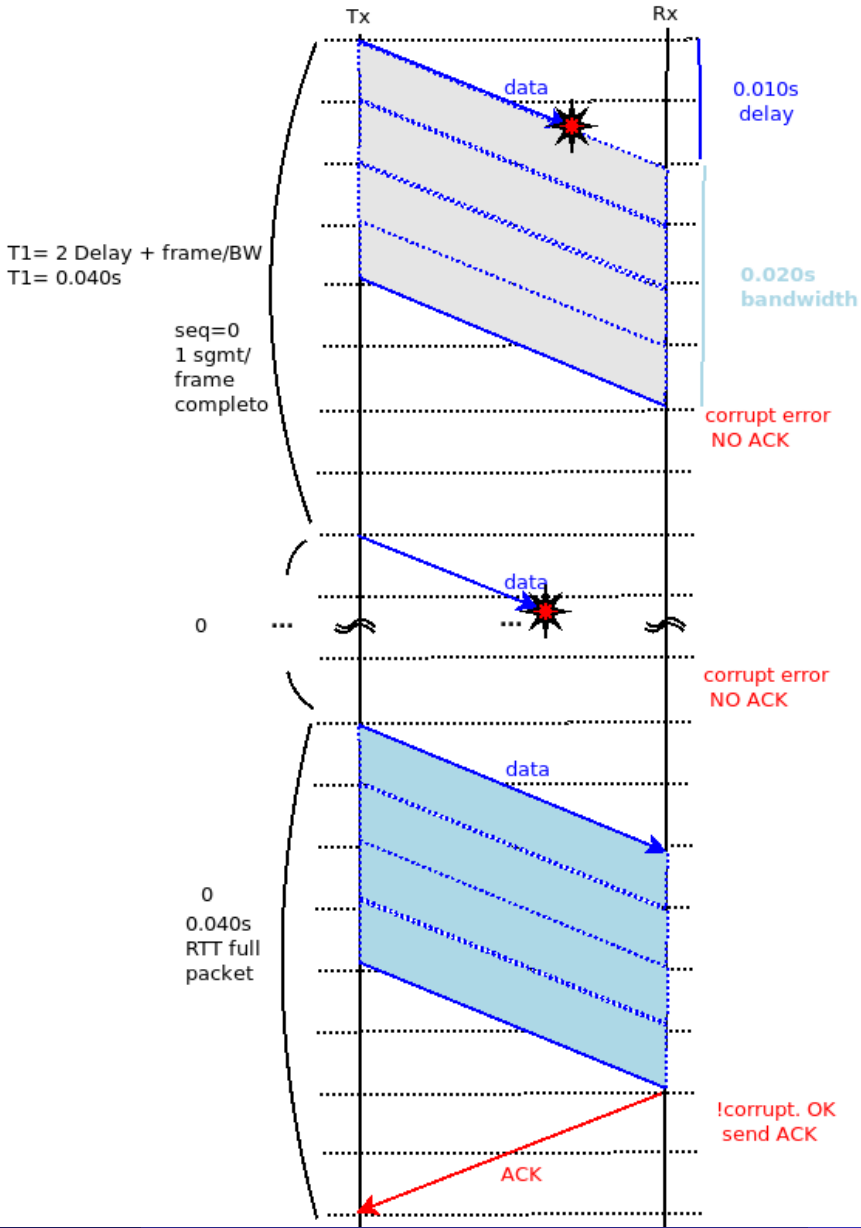
Control de Errores S&W

- El sistema S&W es ineficiente, envía un dato por vez, ventana de transmisión/recepción: $K = 1$ (también llamada W).
- No se envía el próximo mensaje hasta que no se confirma el que se envió.
- Sistema simple y poco eficiente: no optimiza producto: Delay, Bandwidth: $BDP = D \times B$, $D = RTT$ o $D = RTT + L$.
- Cada vez que envía un segmento requiere arrancar un timer: RTO o $T1$.
- Si no recibe confirmación se vence el timer y retransmite.
- Se optimizaría enviando la mayor cantidad de datos posibles en un segmento/frame, pero se torna aún más deficiente si hay muchas retransmisiones. Limitación del tamaño del segmento/frame con respecto al código de detección de errores.

Stop & Wait


Tx
Rx
data
0.010s
delay
0.005s
bandwidth
0
0.025s
RTT full
packet
K=1
Data_processing:
if (!corrupt(pkt))
{
udt_send(ACK);
}
else
{
udt_send(NAK);
}
0.010s
ACK delay
ACK
data
1
0.025s
RTT full
packet
ACK
2
...
data
0.015s
last full
packet sent
200
0.025s
RTT full
packet
ACK

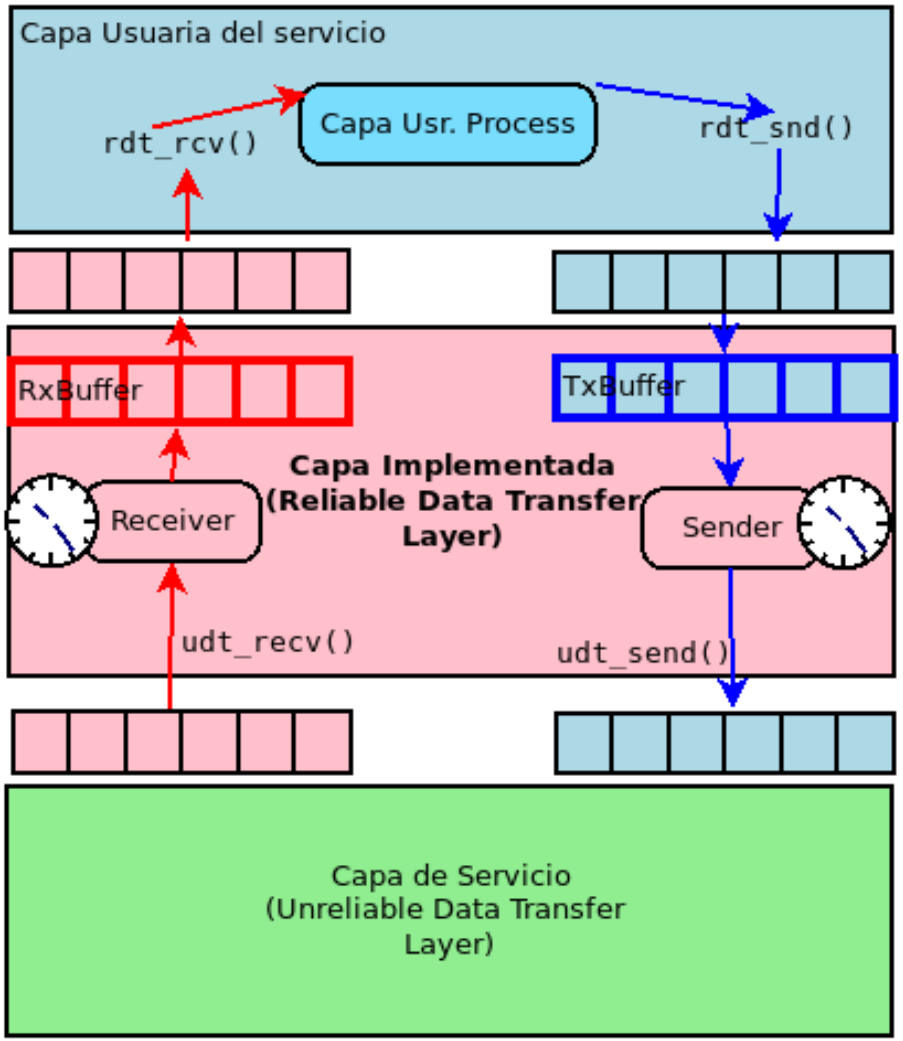
Stop & Wait - Big Frame/Segmento



Pipelining/Sliding Window

- Pipelining: permitir enviar múltiples segmentos/paquetes sin aún haber recibido confirmaciones, paquetes "in-flight".
- La cantidad de segmentos que se puede enviar sin aún recibir confirmación se llama **Ventana**, notado como K o W, $K = n$, donde $n > 1$.
- Requiere ampliar los números de segmentos y de las confirmaciones además del buffering entre capa usuaria-RDT en ambos procesos: Sender, Receiver.
- Por cada mensaje enviado se inicia un timer de retransmisión: $T1$ o RTO (ver más adelante la simplificación).
- Por cada confirmación se descarta/re-inicia el timer RTO. Si no se recibe confirmación vence RTO (timeout) y se retransmite, nuevo timer.

RxBuffer, TxBuffer, Timers en Pipelining

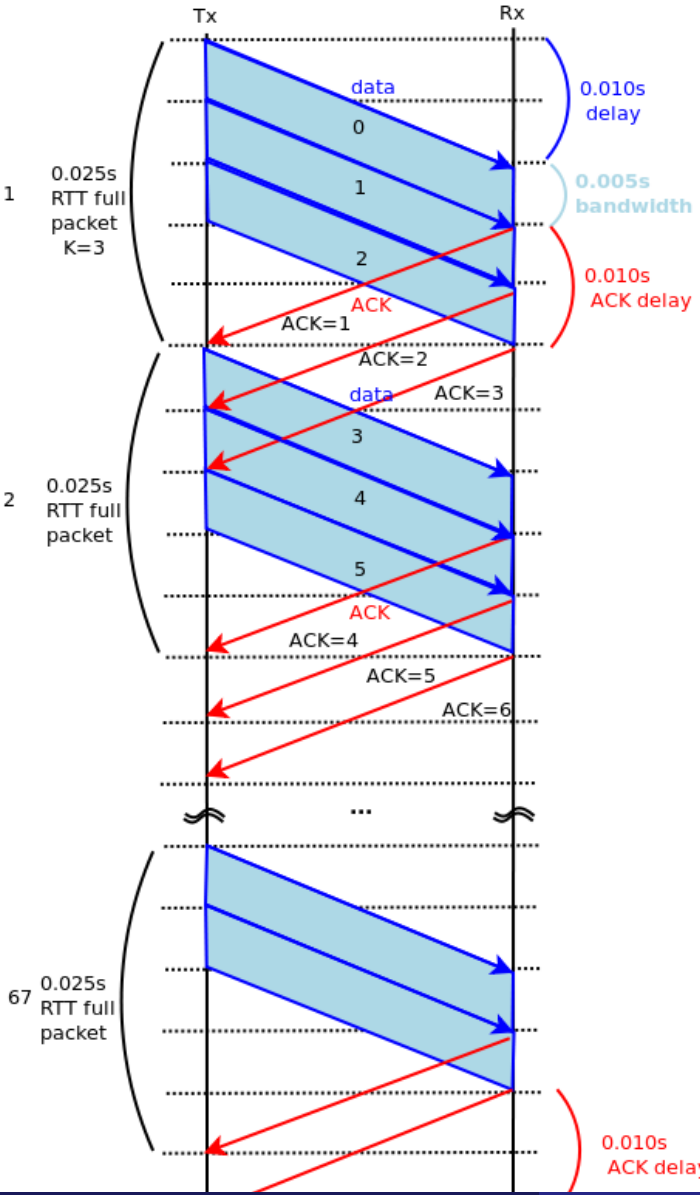


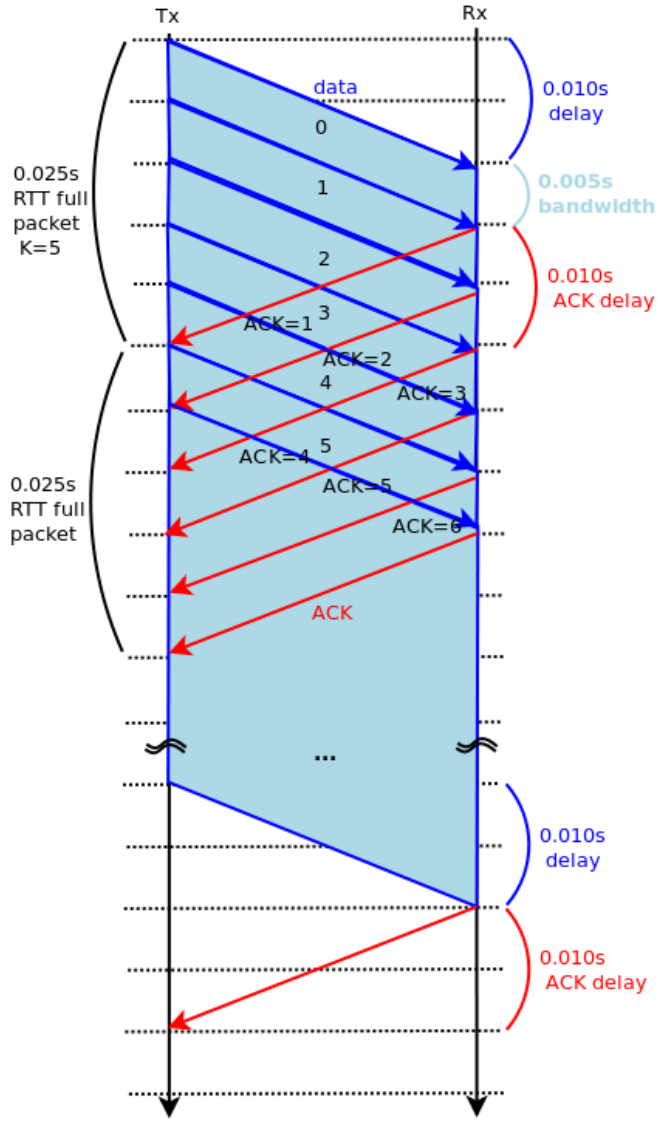
Pipelining/Sliding Window (Cont.)

- Podría generarse Confirmaciones Negativas: NAK (NO Acknowledge), implícitas o explícitas.
- Algoritmo más eficiente, óptimo, si llena el pipe.
- Los números de secuencia, $0..M-1$, si se numeran en módulo, no necesariamente son $M = K$, $K < M$.
- Recordar que aumentar M permite ser tolerante a fallas de ACK delayed.
- Alternativas:
  - Go-back-N.
  - Selective-Repeat.

Nota: Los ejemplos se muestran con confirmaciones por segmento, no por byte e indicando el que se espera

Pipelining


Tx
Rx
data
0
1
2
ACK
ACK=1
ACK=2
data
ACK=3
3
4
5
ACK
ACK=4
ACK=5
ACK=6
...
0.010s
delay
0.005s
bandwidth
0.010s
ACK delay
1
0.025s
RTT full
packet
K=3
2
0.025s
RTT full
packet
67
0.025s
RTT full
packet
0.010s
ACK delay


Tx
Rx
data
0
1
2
3
ACK=1
ACK=2
4
ACK=3
5
ACK=4
ACK=5
ACK=6
ACK
...
0.010s
delay
0.005s
bandwidth
0.010s
ACK delay
0.025s
RTT full
packet
K=5
0.025s
RTT full
packet
0.010s
delay
0.010s
ACK delay

Análisis de Rendimiento

$$K = Window = 3, S = MaxSgmt_{bytes} = 1500B$$

$$RTT = Latencia_{seg} = 0.020s = 0.010s + 0.010s$$

$$L = DelayTransf_{seg} = 0.005s$$

$$R = \frac{1500 \times 8}{0.005} = 2400000bps = 2.4Mbps$$

$$U = \frac{K \times \frac{L}{R}}{RTT + \frac{L}{R}} =$$

$$(3 \times 0.005)/(0.020 + 0.005) = 0.6(60\%)$$

- Se obtiene: $2.4Mbps \times 0.6 = 1.44Mbps$
- Si $K = 5$ se obtiene el 100%, si el tráfico es constante.

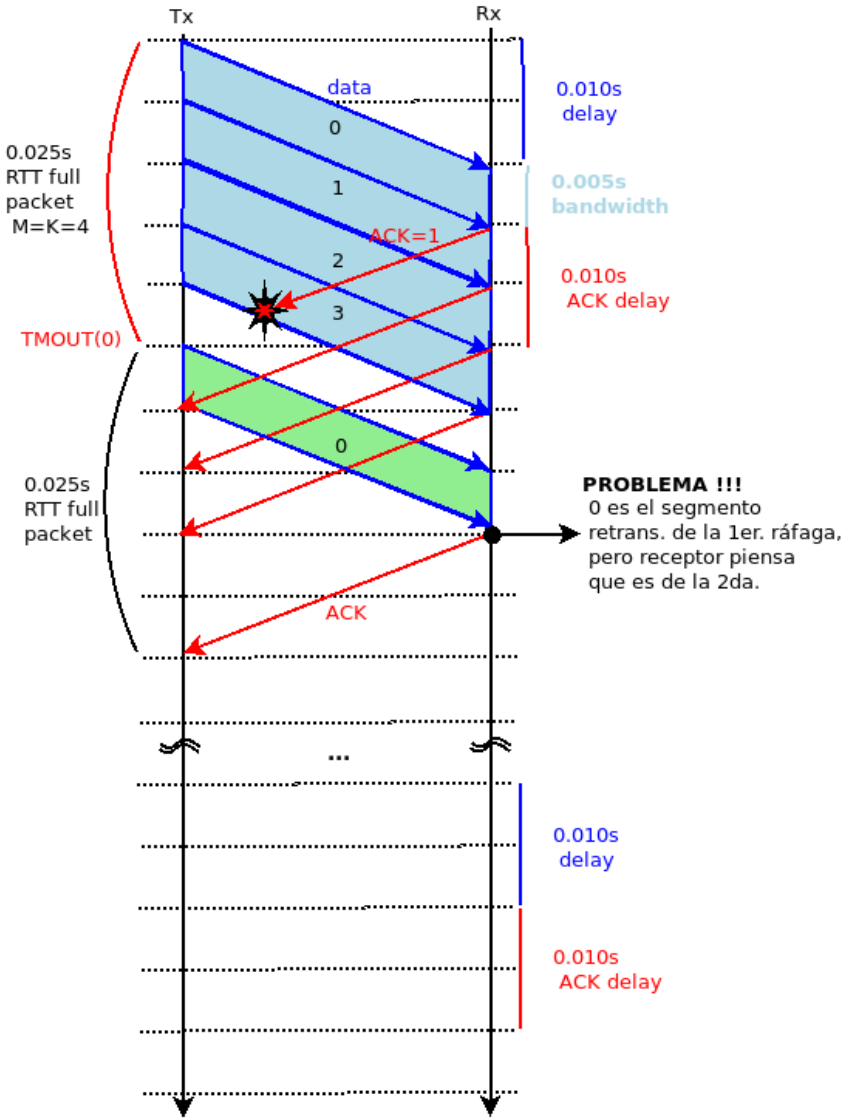
Go-back N

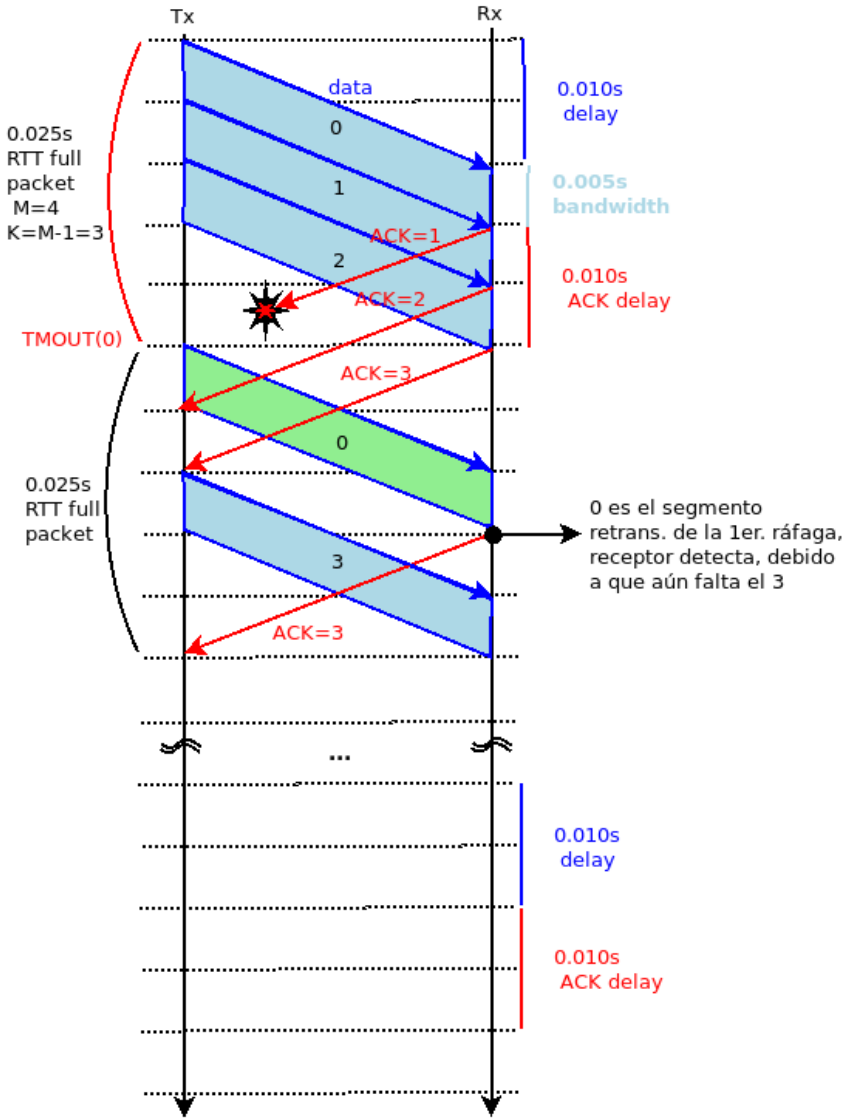
- Se tiene una ventana estática de tamaño $K = n, n > 1$. Numeración de segmentos, se realiza en módulo M, $K <= (M - 1)$.
- No admite segmentos fuera de orden, ni confirmaciones fuera de orden. Solo se confirman por la positiva los segmentos que se pudieron colocar en el buffer en orden.
- Se puede confirmar desde N hacia atrás (ACK acumulativos). No necesariamente se confirma cada segmento individualmente.
- Se puede re-enviar ante un timeout o un NAK. Requiere buffering extra en el emisor, no se pueden descartar del buffer de com. con la capa usuaria.
- Si se re-envía ante un timeout se hace desde N hacia adelante, los que ya se enviaron.
- Emisor puede mantener solo un timer para el segmento más viejo enviado y aún no confirmado, si este expira retransmite todos los no confirmados.
- Si llega una confirmación en orden arranca un nuevo timer.

Go-back N (Cont.)

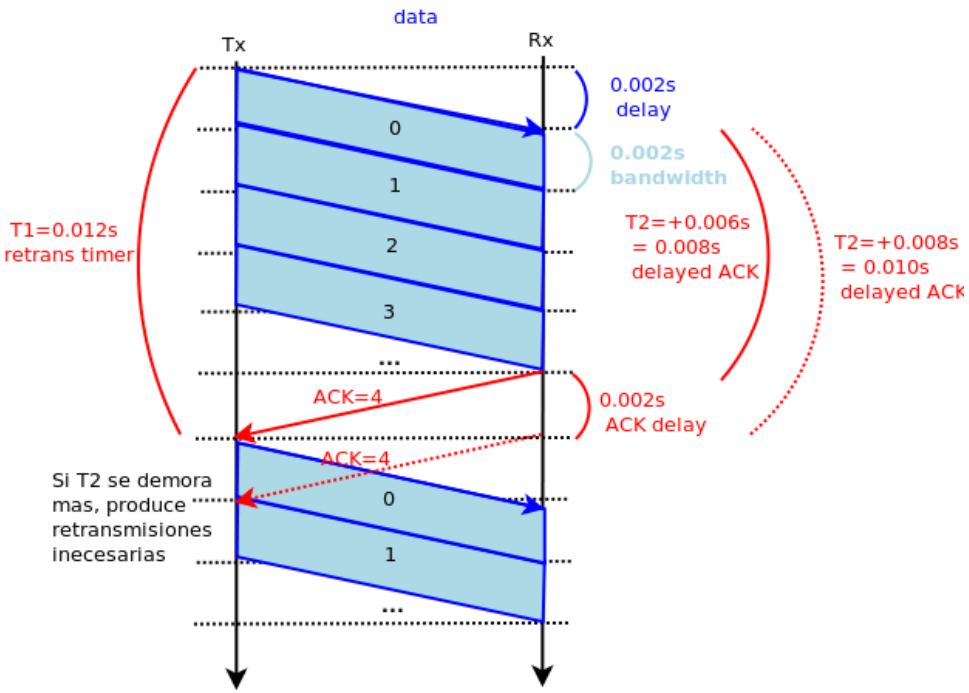
- Si llega un segmento de datos en orden se puede/debe confirmar por la positiva indicando el próximo.
- Si llega un segmento fuera de orden o esta corrupto se puede confirmar por la negativa indicando que se espera el próximo al último recibido de forma adecuada.
- Ante el error se puede esperar la retransmisión.
- Se pueden aprovechar tramas de datos para confirmar: Piggy-backing.
- El receptor puede usar: timer de ACK, $T2$, para confirmar. $T1 > T2$, $T1 > T2 + RTT$ (Delayed ACK).
- $T2$ debe aprovechar piggy-backing, y confirmaciones acumulativas pero sin demorar demasiado tiempo el flujo de datos.
- Si se pierde uno segmento es como "vaciar" el pipe y volver atrás: Go-Back.

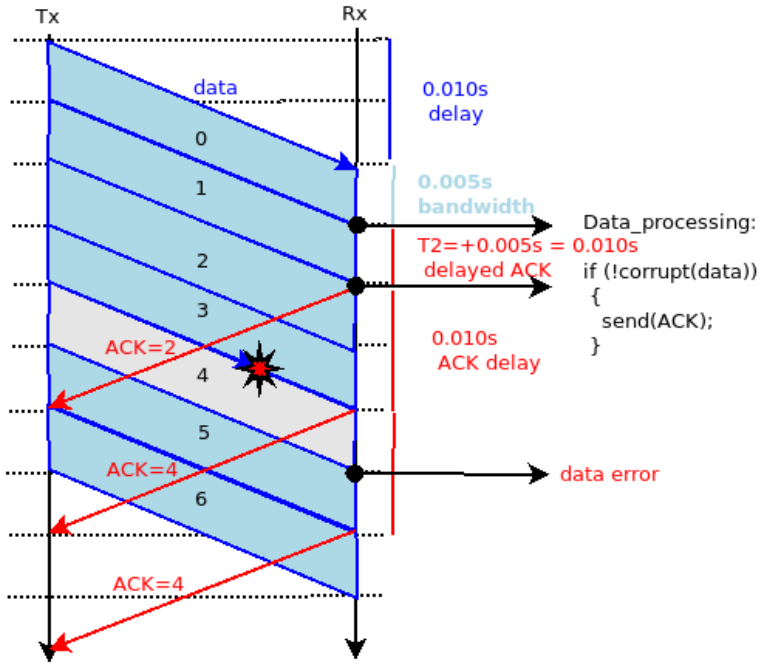
Relación K,M: Ej. M=K=4 ; M=4,K=(M-1)=(4-1)



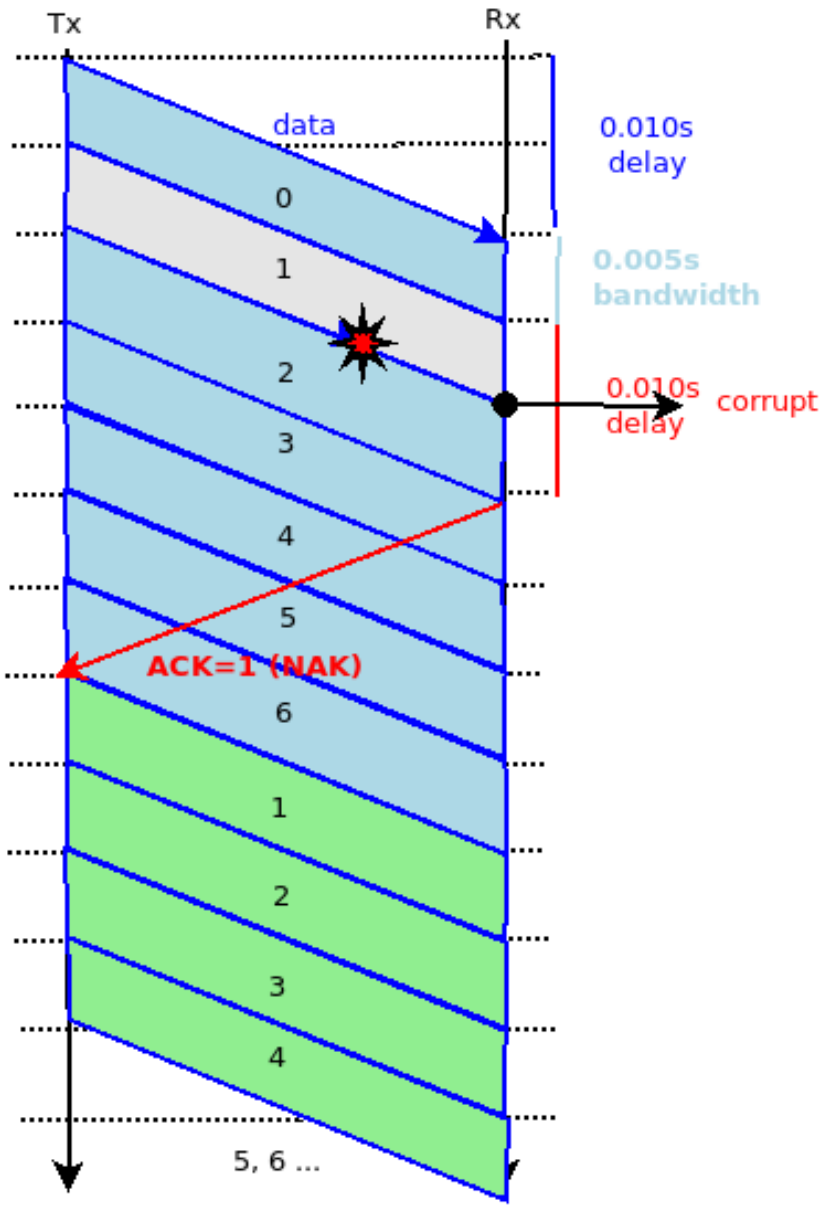


Timers Go-Back N





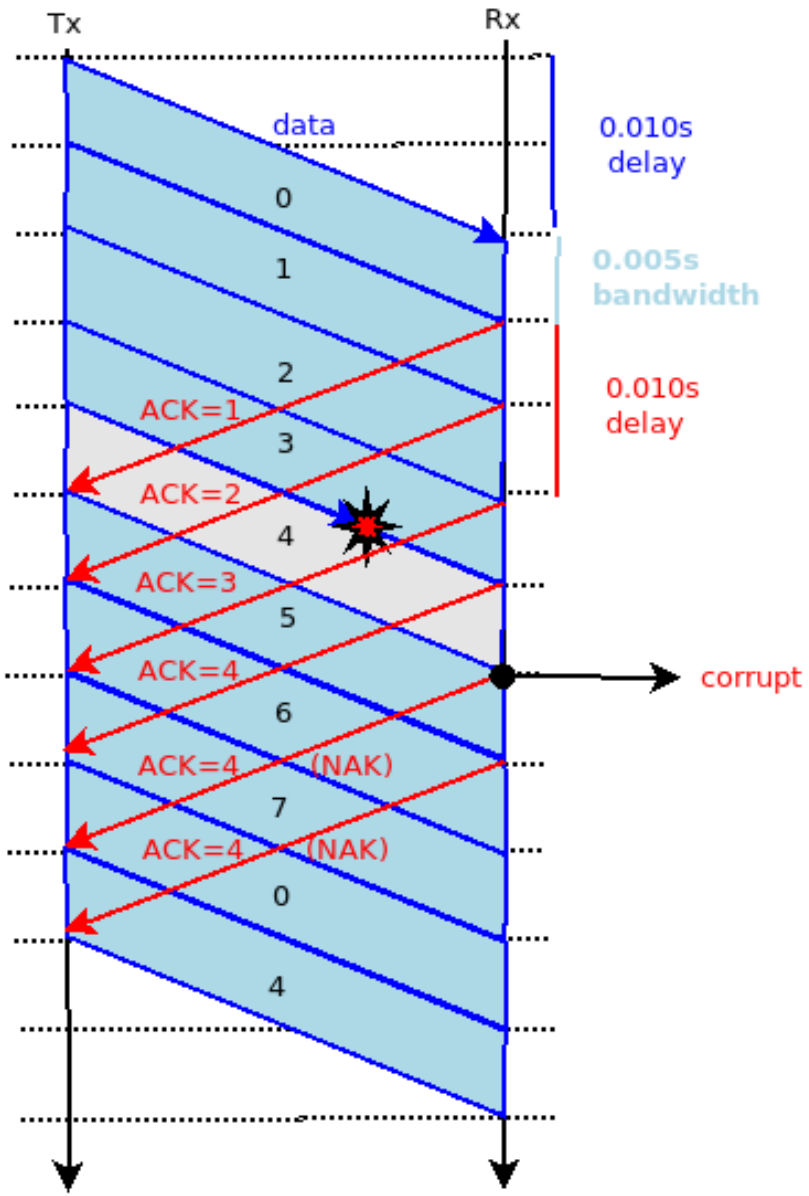
Go-Back N, Segmento Corrupto



Go-back N (Cont.)

- Las confirmaciones por la negativa pueden generar ACK duplicados (NAK).
- Mensajes fuera de orden:
  - Buffering hasta recibir los que llenan los huecos. Hasta cuando se mantienen? (requiere buffering de Rx antes de pasarlos a la capa usuaria).
  - Descartarlos y esperar retransmisiones (no requiere buffering del receptor, solo recordar la secuencia que se espera).

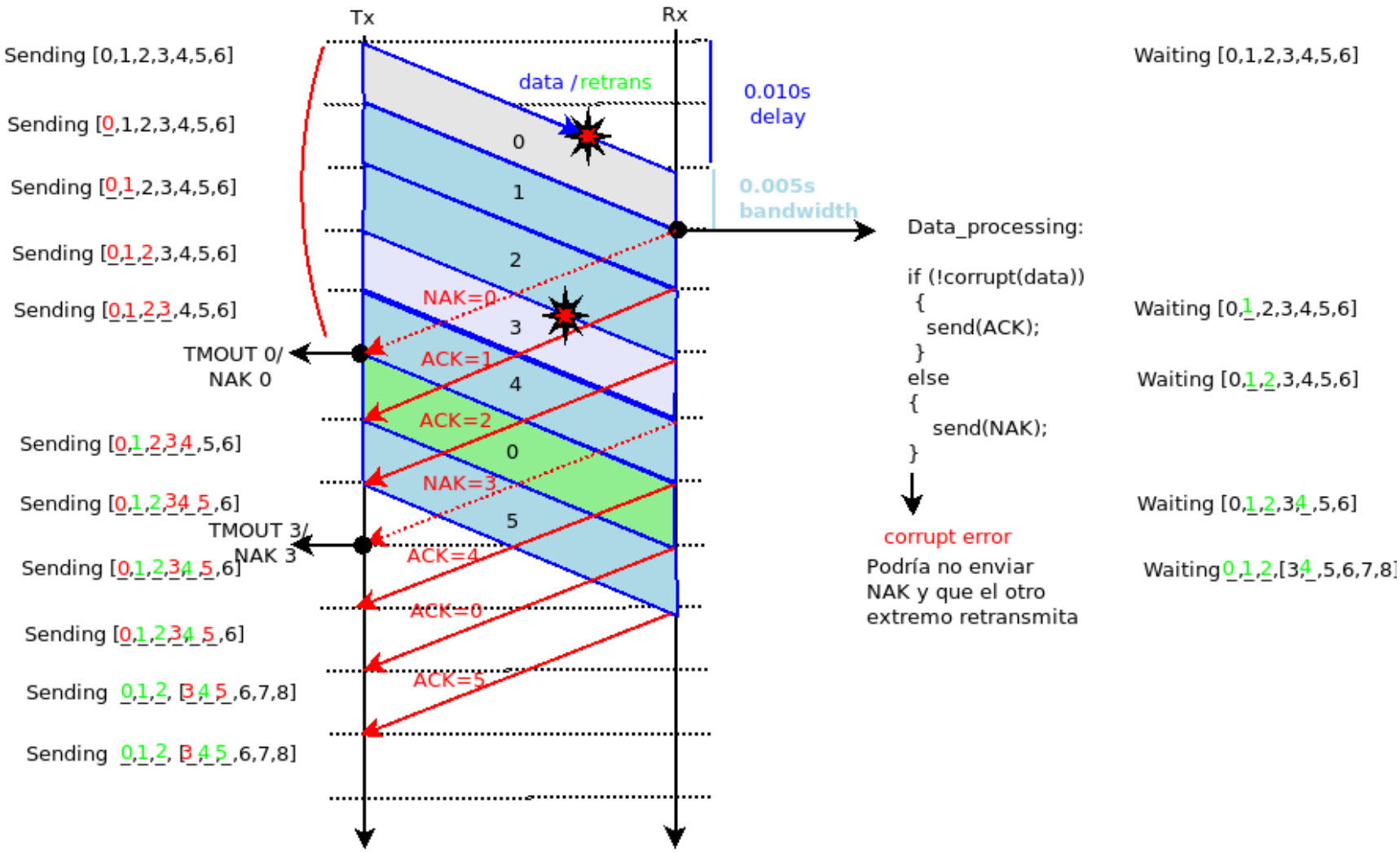
Go-Back N, ACK duplicados



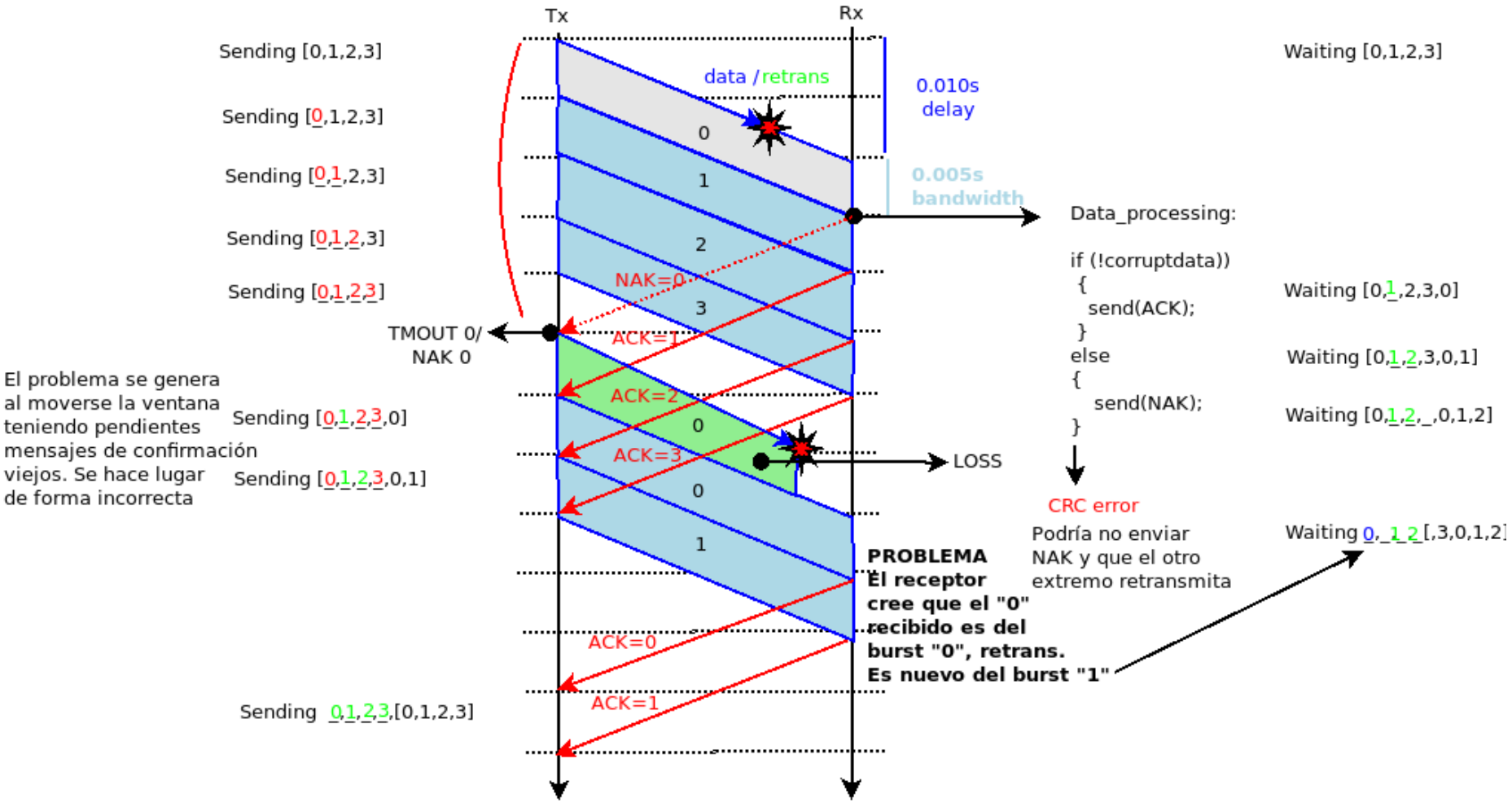
Selective Repeat (SR)

- Go-Back-N ante pérdidas retransmite segmentos innecesarios si se perdió uno del medio del stream de datos solamente y hubo timeout.
- Selective Sliding Window/Selective Repeat solo retransmite los que no se confirmaron.
- El receptor puede confirmar de a uno o usar bit vectors/intervalos de confirmaciones.
- No se puede usar confirmaciones acumulativas.
- No se deben confundir los segmentos de diferentes ráfagas. No se deben reusar #ID/SEQ hasta asegurarse que tiene todos los mensajes previos o estos no están en la red.
- Se realiza en módulo M, $K <= \frac{(M-1)}{2}$, para evitar confundir los ACK de segmentos.
- La ventana se desliza sin dejar huecos, desde los confirmados más viejos.

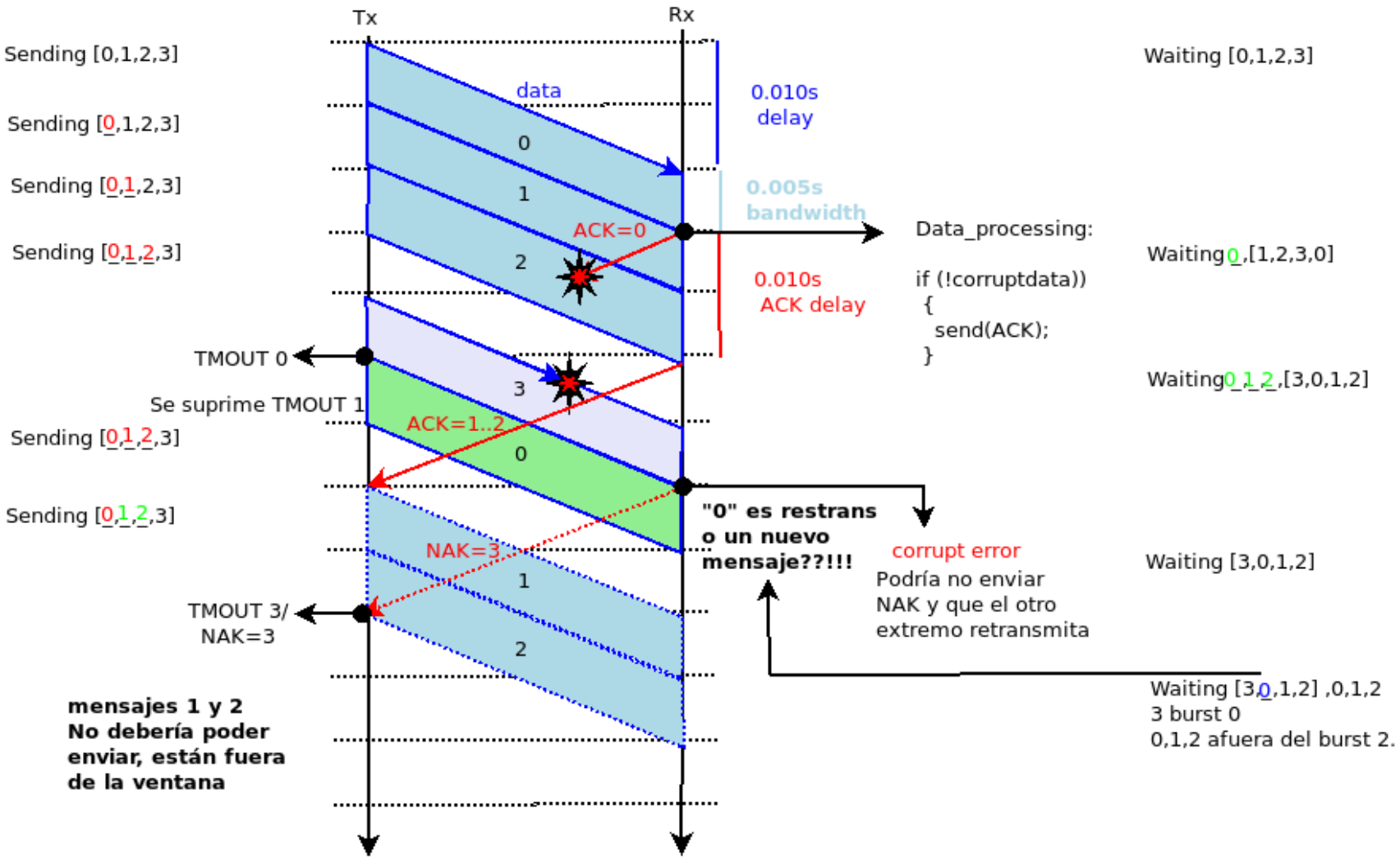
Selective Repeat Ejemplo



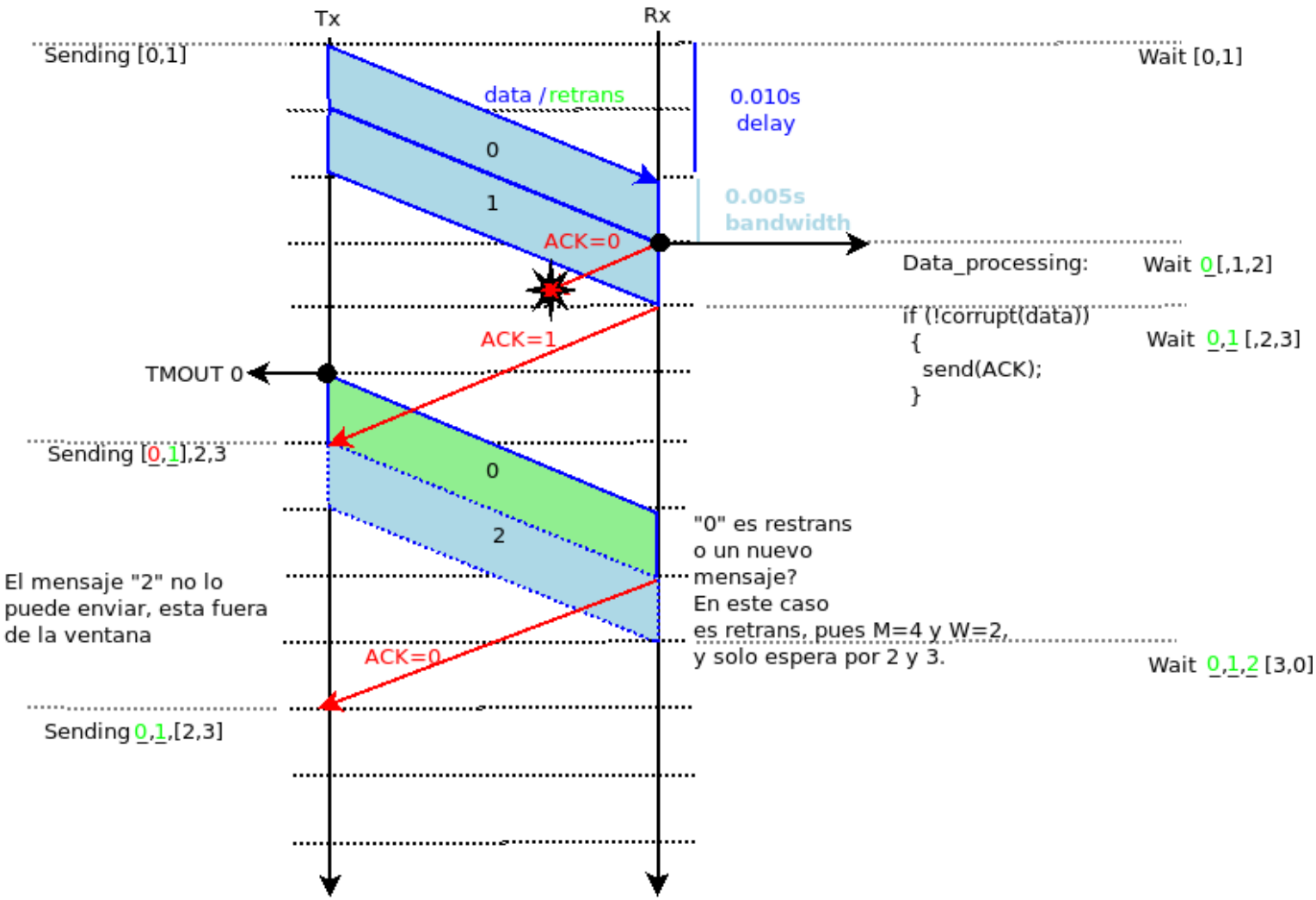
Selective Repeat, Problemas



Selective Repeat, Problemas



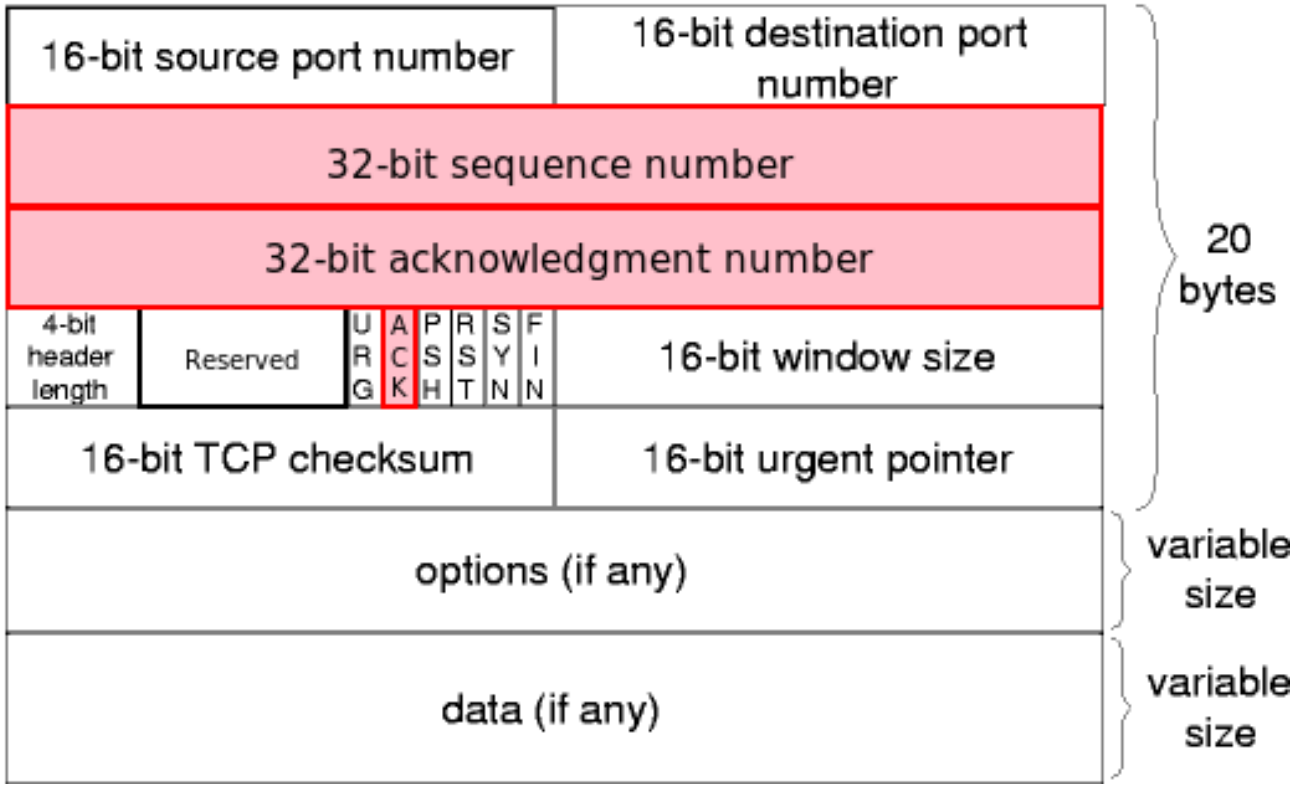
Selective Repeat, K adecuado



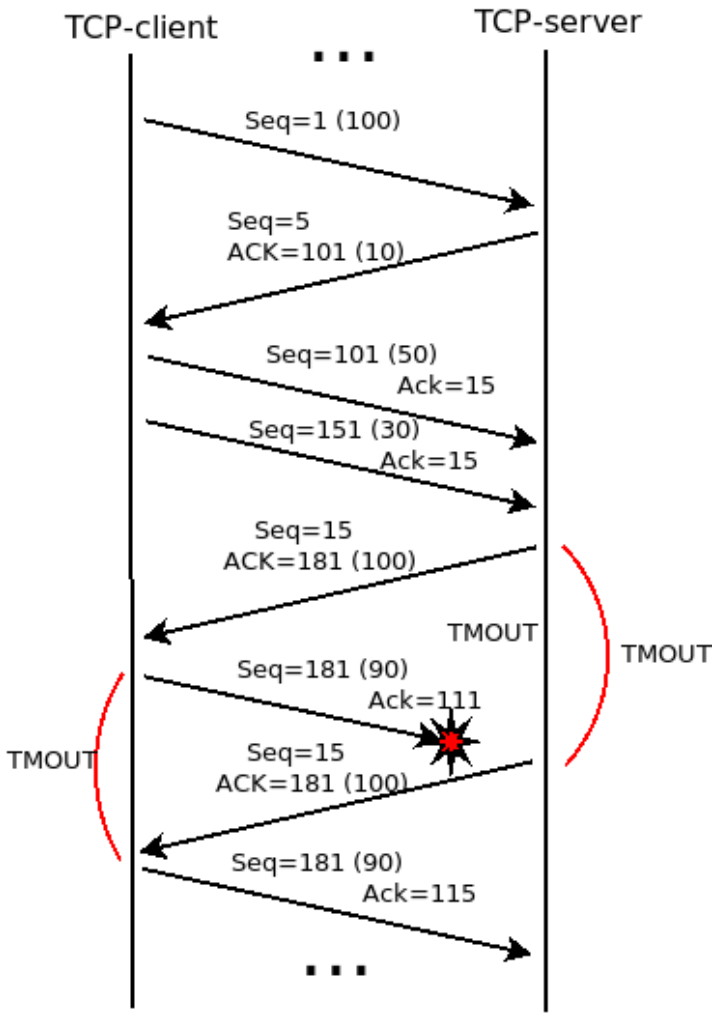
Control de Errores en TCP

- TCP hace el control de errores por bytes (byte oriented), no por segmentos.
- Los segmentos se numeran de acuerdo a bytes enviados (nro. del primer byte).
- Los números se negocian al establecer la sesión y cada implementación los elige libremente (ISN).
- Las confirmaciones son "anticipativas", indican el nro. de byte que esperan.
- Utiliza Go-back-N con ventana dinámica (flow-control), utiliza piggy-backing y permite negociar Ventana Selectiva con Opciones.
- Para control de errores TCP utiliza los campos: #SEQ, #ACK, flag ACK más timer y algunas opciones.

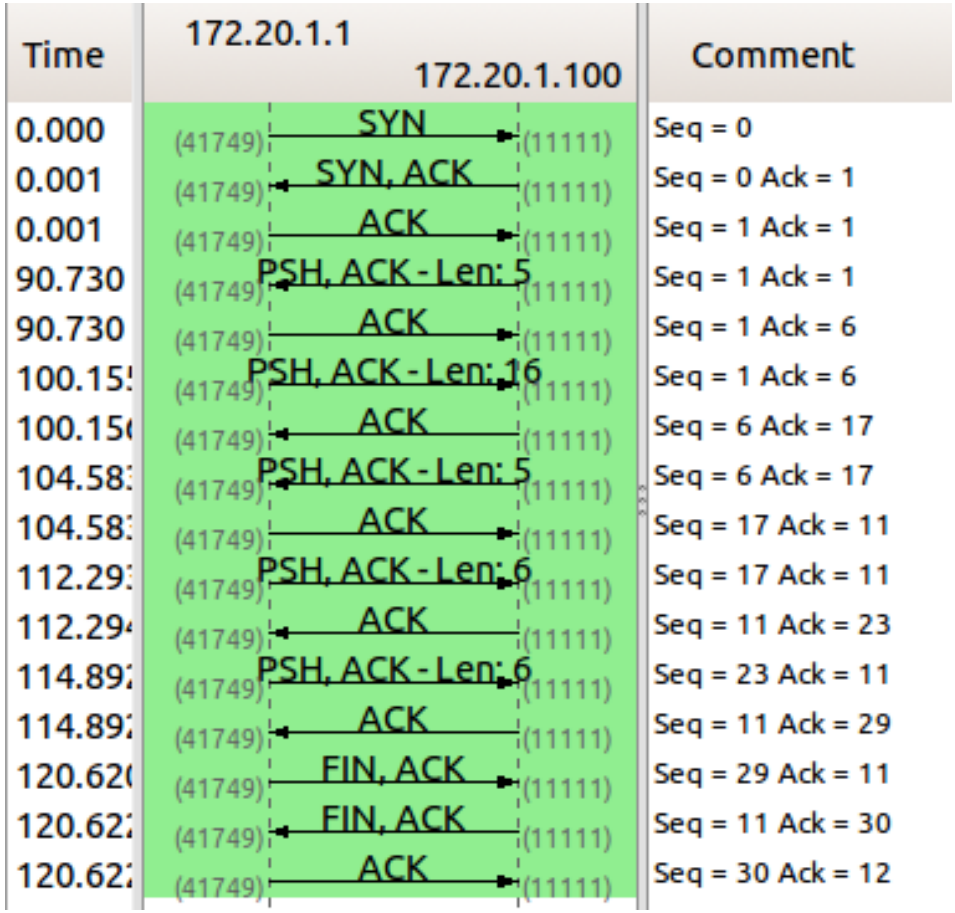
Segmento TCP, Control de Errores



Ejemplo de Control de Errores de TCP



Otro Ejemplo de Control de Errores de TCP



Control de Errores en TCP (Cont.)

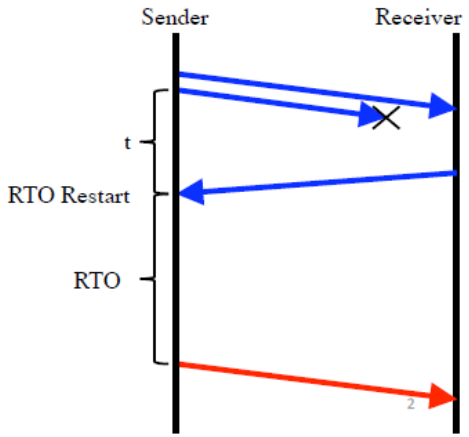
- Por cada segmento (con datos) que envía TCP es como si iniciara un Timer local, *RTO* y pone copia del segmento en cola local (RFC-793) *TxBuf*.
- Por cada segmento ACKed descarta el timer asociado y descarta la copia del segmento (RFC-793) del *TxBuf*. Hace lugar para nuevos segmentos a Tx.
- Si *RTO* expira antes que se confirme el segmento TCP lo copia del *TxBuf* y retransmite (RFC-793).
- Segmentos ACKed no indica leído por aplicación, sí recibido por TCP (RFC-793) (ubicado en el *RxBuf* del receptor).
- Si el receptor detecta error en el segmento simplemente descarta y espera que expire *RTO* en el emisor (podría envía un NAK, re-enviar ACK para el último recibido en orden, forma de solicitar lo que falta).

Control de Errores en TCP (Cont.)

- Receptor con segmentos fuera de orden descarta directamente y podrá re-enviar ACK (podría dejar en *RxBuf* pero no entregar a la aplicación, tiene huecos).
- Se puede confirmar con ACK acumulativos.
- TCP NO arrancar un *RTO* por cada segmento, solo mantiene un por el más viejo enviado y no ACked y arranca uno nuevo solo si no hay *RTO* activo.
- Si se confirman (ACked) datos, se inicia un nuevo *RTO* (RFC-6298) recomendado. Si todo confirmado se detiene RTO.

Control de Errores en TCP (Cont.)

- El nuevo *RTO* le esta dando más tiempo al segmento más viejo aún no confirmado. Mejor: RFC-7765: $RTO_{new} = RTO - T_{earliest}$ (menos el tiempo que pasó del pendiente más viejo).

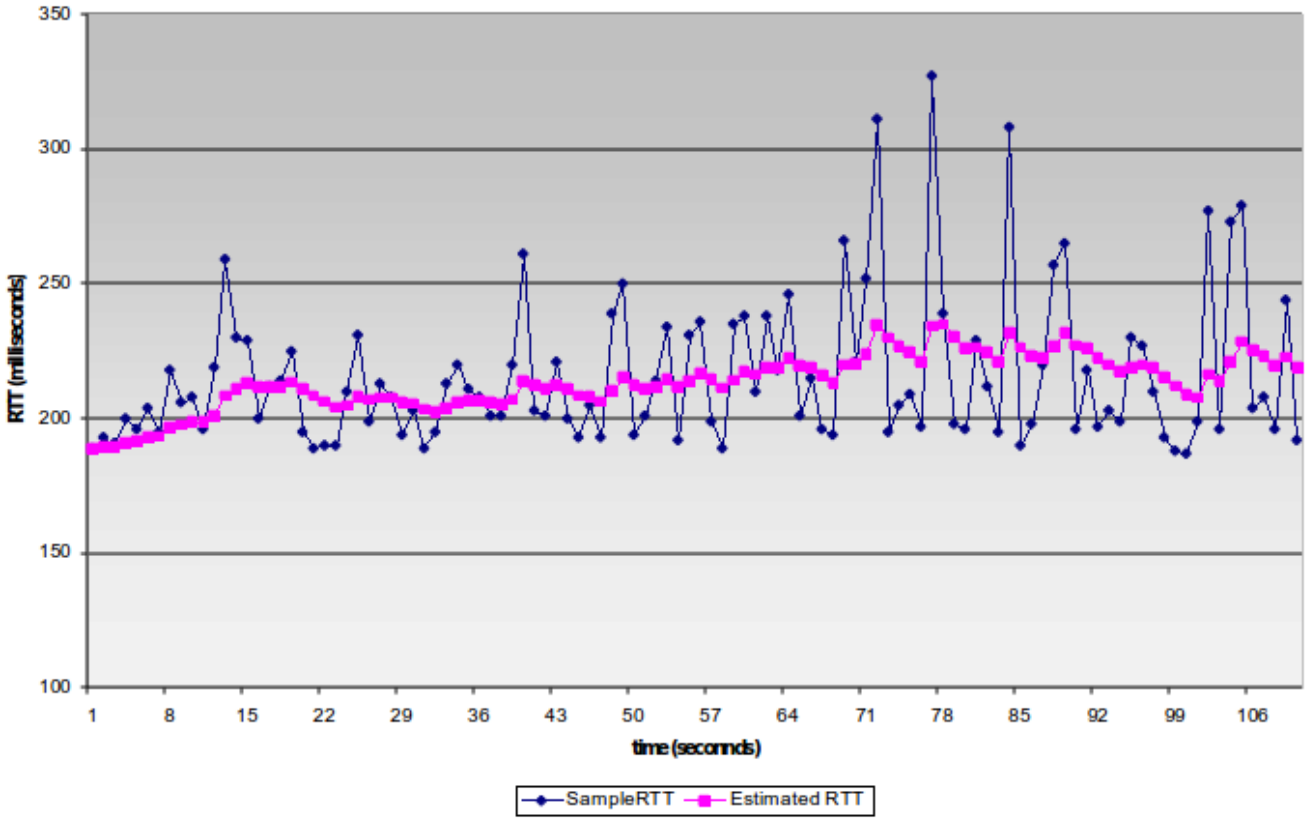


- Si vence un RTO se debe retransmitir el segmento más viejo no ACKed y se debe duplicar: Back-off timer $RTO_{new} = RTO * 2$ $RTO_{MAX} = 60s$ (RFC-6298) recomendado.

Cálculo de RTO

- *RTO* debe ser dinámico, debe contemplar estado de la red.
- *RTO* estático solo sirve para L2 (directamente conectados).
- Para calcular *RTO* se estima RTT (Round Trip Time). RTT inicial RFC-2988(2000), 3seg - RFC-6298(2011), 1 seg. Cambio en las redes.
- $RTO = SRTT + (4 * DevRTT)$ (RFC-6298).
- $SRTT_i = (1 - \alpha) * SRTT_{i-1} + \alpha * RTT, \alpha = 1/8$
- Influencia de las muestras pasadas decrece exponencialmente.
- $DevRTT_i = (1 - \beta) * DevRTT_{i-1} + \beta * |RTT - SRTT_i|, \beta = 1/4$
- Si hay gran variación en $SRTT_i$ se usa un mayor margen.
- $RTO < 1seg$: redondeado a 1 seg (RFC-6298).
- Se mide por cada RTT. Se puede utilizar la opción TimeStamp.

SRTT


350
300
250
200
150
100
RTT (milliseconds)
1
8
15
22
29
36
43
50
57
64
71
78
85
92
99
106
time (seconds)
SampleRTT
Estimated RTT

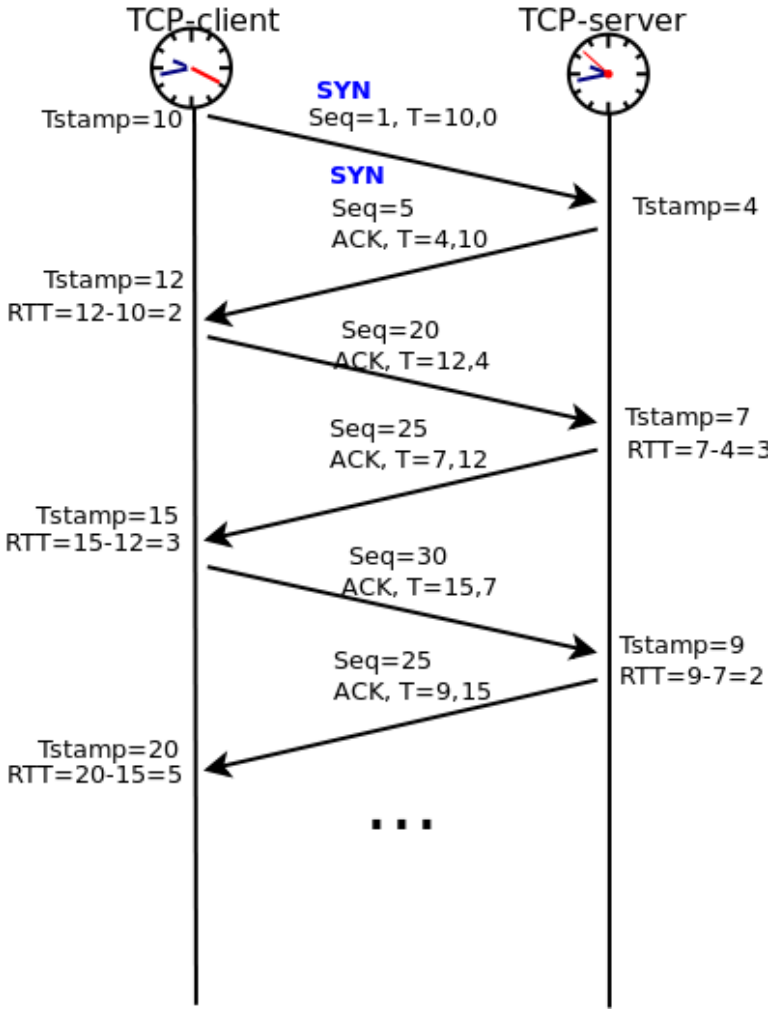
Mínimo RTO

- $RTO < 1seg$: redondeado a 1 seg (RFC-6298).
- Por qué tan conservador?
  (The TCP Minimum RTO Revisited, Ioannis Psaras and Vassilis Tsaoussidis)
  - Considerar sistemas con granularidad de 500ms.
  - Delayed ACK ($T2 = DelACK = 200ms$).
- Sistemas reales ignoran esta recomendación:
  - Linux $RTO >= 200ms$, $DelACK = dyn$.
  - Windows $RTO >= 300ms$, $DelACK = 200$.
  - BSD $RTO >= 30ms$.
  - Solaris $DelACK = 50..100ms$.

TimeStamp

- RFC-1323.
- Se envía en el primer SYN el timeStamp local, opcional.
- En cada mensaje TCP con esta opción, se copia el timeStamp local y se hace echo del último timeStamp recibido desde otro extremo.
- Con el valor recibido como echo y el valor del reloj local se calcula el RTT.
- Si el mensaje no es un ACK válido no se actualiza la estimación del RTT *SRTT*.
- Relaja la necesidad de usar timer por cada segmento para estimar RTT.
- Protección contra Wraparounds de num. secuencia (PAWS).

Ejemplo de TimeStamping



Referencias

[K-R] Computer Networking International Edition, 6e. James F. Kurose & Keith W. Ross. ISBN: 9780273768968.