Control de Errores en TCP

Contenidos

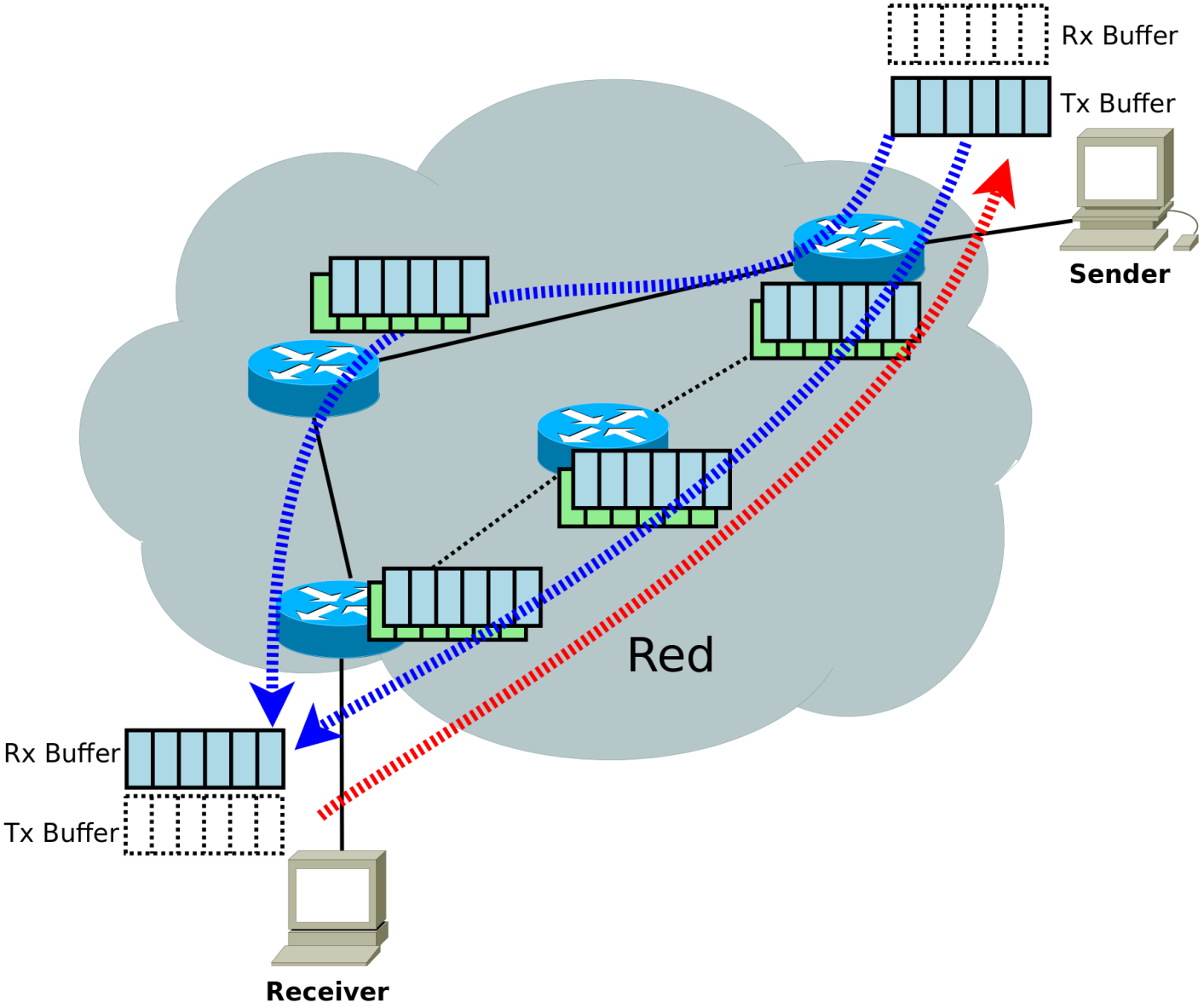
Control de Errores

- Se requiere un mecanismo de control sobre un canal no confiable, por ejemplo IP (best-effort).
- Se realiza con ARQ: Automatic Repeat reQuest/Automatic Repeat Query, End-to-End.
- Utiliza números de secuencia y confirmaciones para validar que los datos se recibieron OK (en orden y sin errores).
- Se confirman los datos que se reciben.
- Requiere mantener timer: *RTO* (Retransmission Timeout) o TMOUT.
- Requiere mantener buffer de segmentos a transmitir *TxBuf* y posiblemente para segmentos recibidos *RxBuf*.
- Puede corroborar mediante checksum/CRC errores de bits en los datos recibidos.

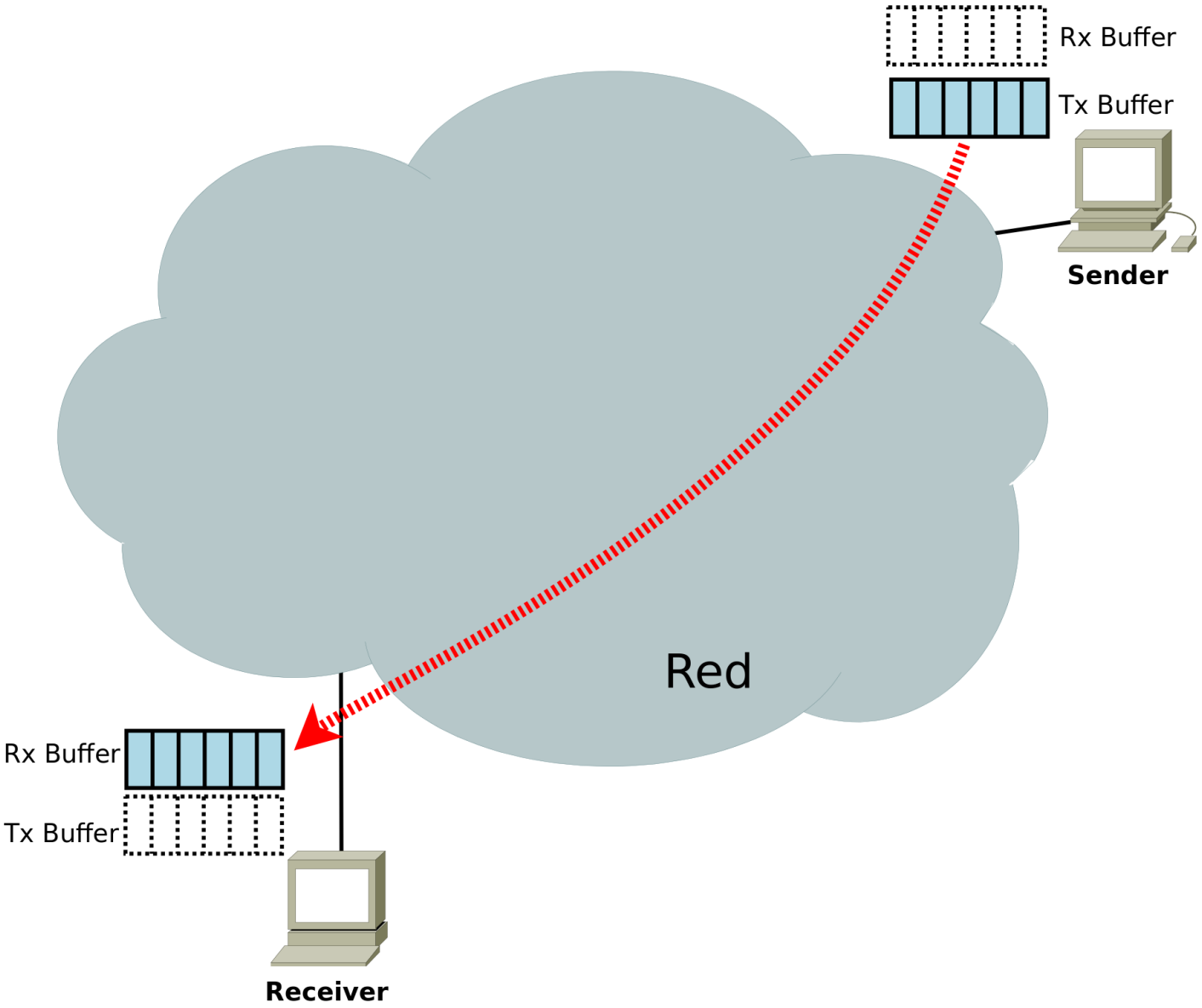
Control de Errores, Escenarios

- Para simplificar, en varios ejemplos se supone solo un emisor y un receptor, solo se transmite en un sentido datos y no se requiere elegir origen ni destino.
- Para simplificar se analiza primero un RTT (Round Trip Time) y BW (Bandwidth digital) fijos.
- Se comienza con un esquema S&W (Stop&Wait) hasta llegar al GBN (Go-Back-N) y SR (Selective-Repeat) como TCP.
- Esquema S&W solo a modo de introducción a los problemas y como se solucionan, aunque ineficiente.
- Control de errores End-to-End, abstracción de los nodos intermedios, la red no "colabora".

Capa de Red



Capa de Red para TCP



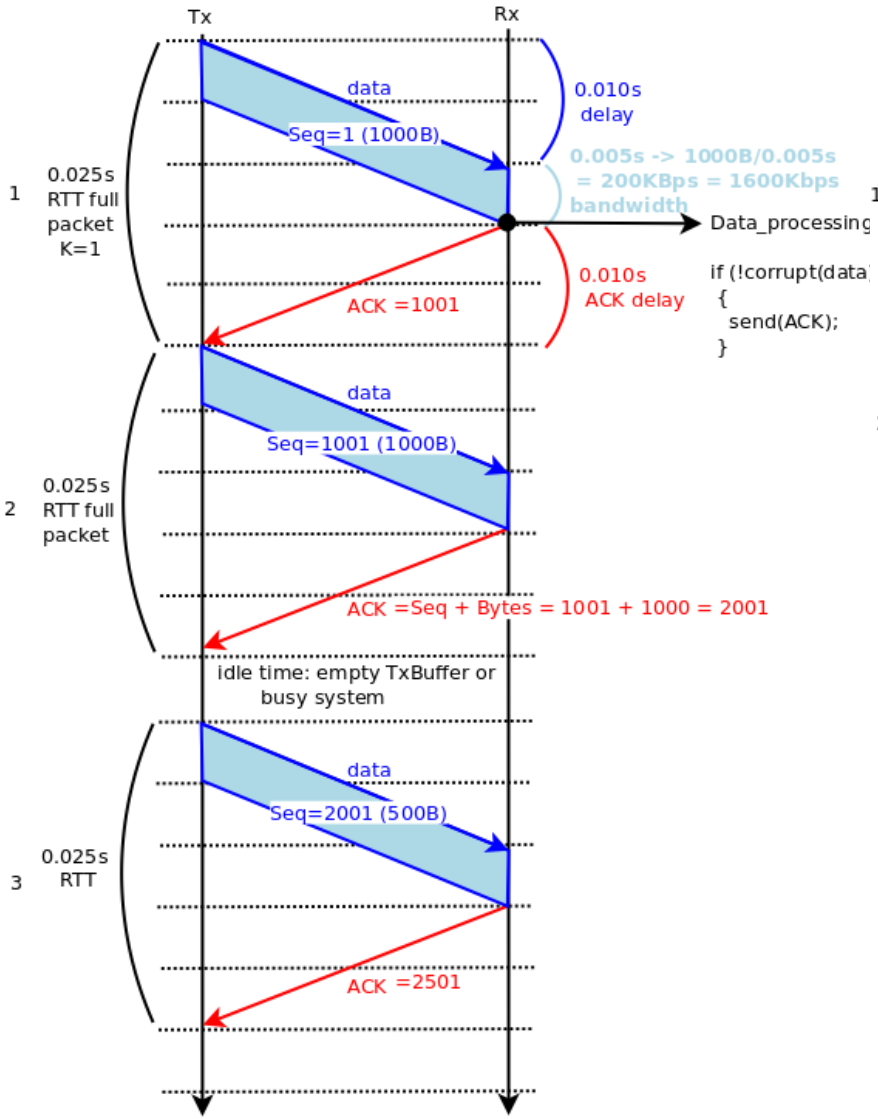
Errores Posibles en Capa de Red (o inferior)

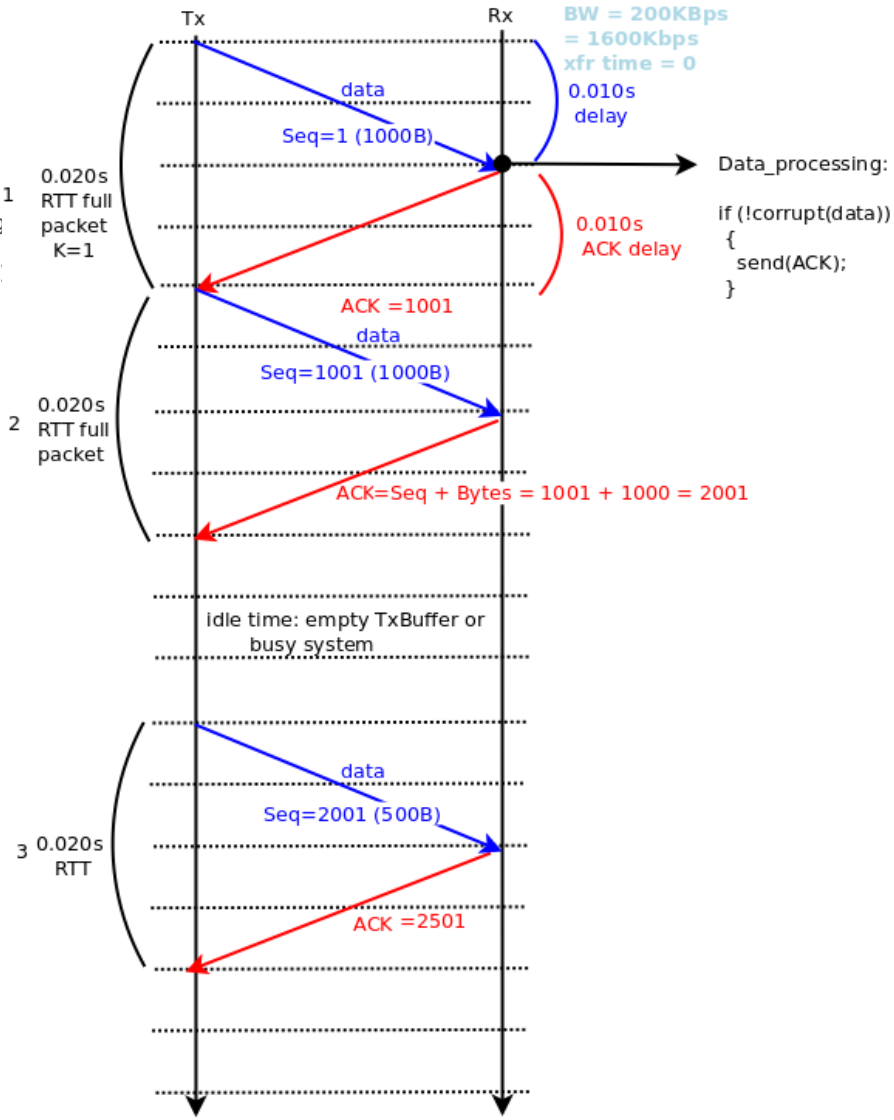
- Errores se pueden generar en los extremos o en la red.
- Se pierden datos y/o ACKs (por descartes o errores en la red).
- Se duplican datos y/o ACK, por retransmisiones o por errores en equipos (extremos o intermedios, red).
- Se desordenan por usar caminos múltiples o errores de procesamiento en equipos.
- Se corrompen datos y/o ACKs.
- Existe delay (retardos) mayores a 1 RTT y RTT podría ser variable.

Stop & Wait (No lo usa TCP)

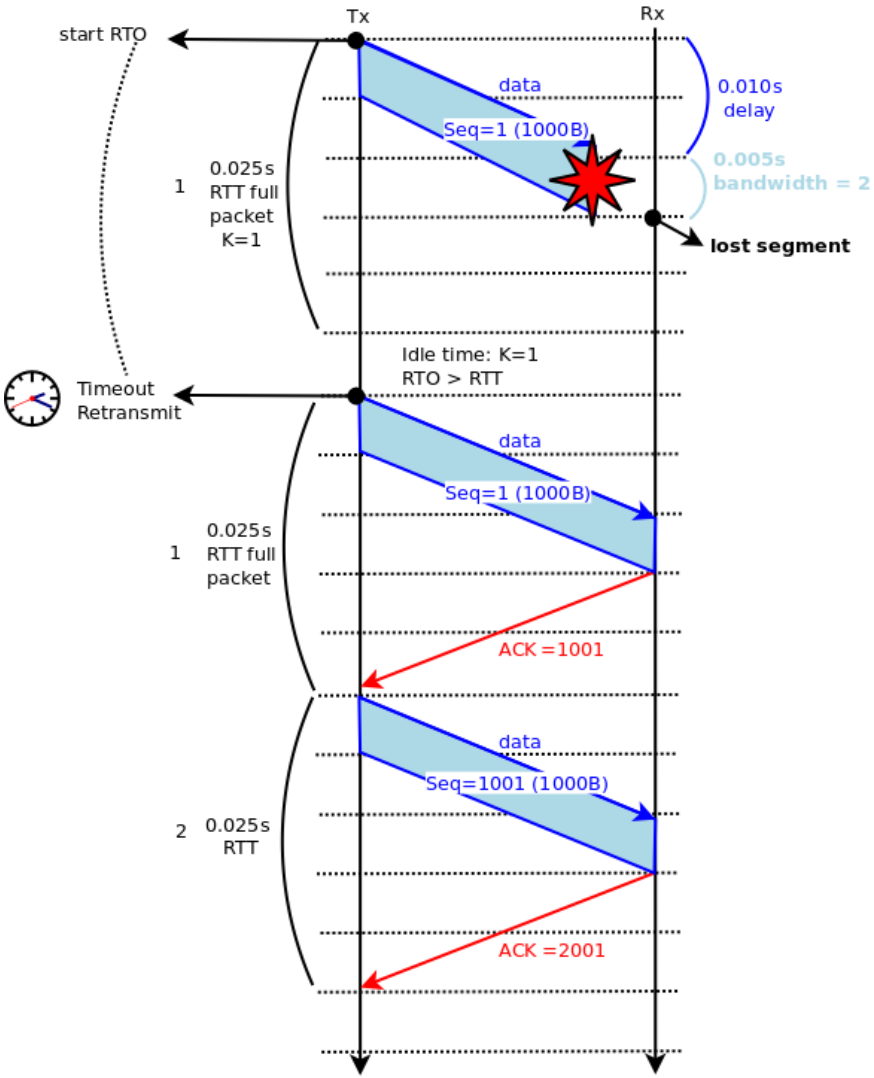
- Emisor manda un segmento numerado y espera confirmación.
- Emisor arranca un *RTO* al enviar el segmento, si no recibe ACK (confirmación) re-envía.
- Receptor, cada vez que recibe segmento, confirma indicando (Num. por segmentos o bytes, en el ejemplo por bytes). Se puede confirmar el actual o el que se espera (en el ejemplo el que se espera).
- Receptor, si tiene errores, descarta o podría confirmar indicando qué Num.(#) espera. Funcionan como NAK (No Acknowledge).
- Emisor si recibe confirmación envía nuevo segmento, si tiene en el *TxBuf* , e inicia nuevo *RTO* (en caso que la capa superior dejo datos para enviar en el buffer).
- Emisor descarta confirmaciones fuera de secuencia (out-of-order) , e.g. atrasadas.
- Si recibe confirmación del anterior (similar NAK) podría re-enviar, en caso de $ACKs > N, N >= 2$.

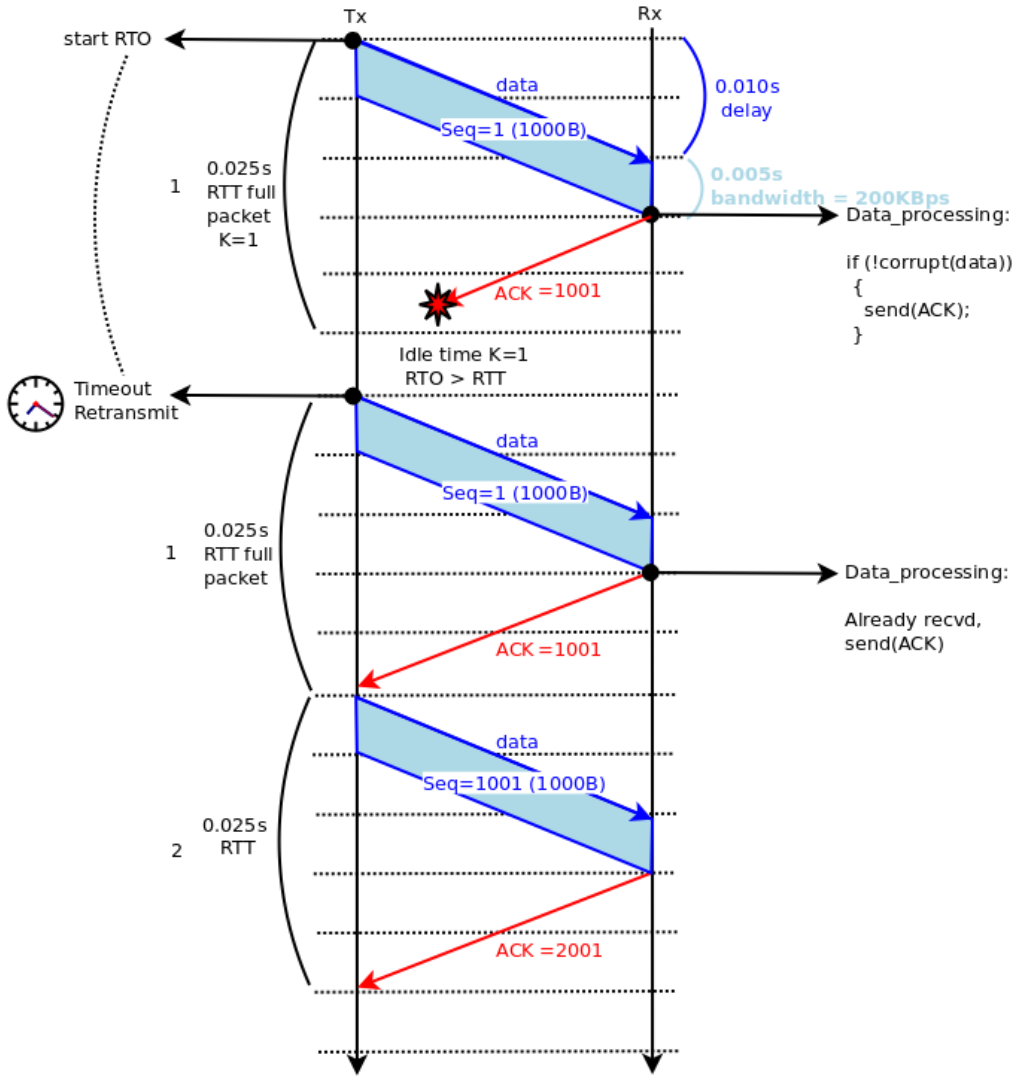
Stop & Wait (gráfico)



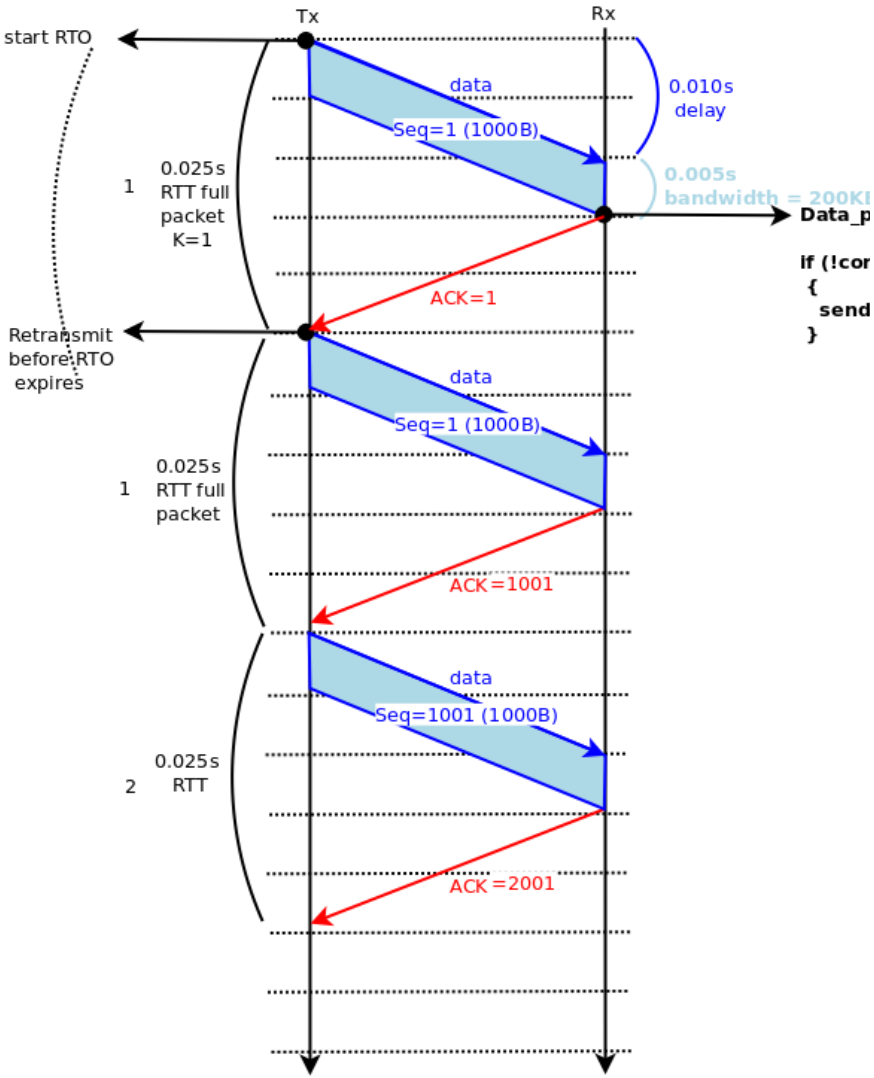


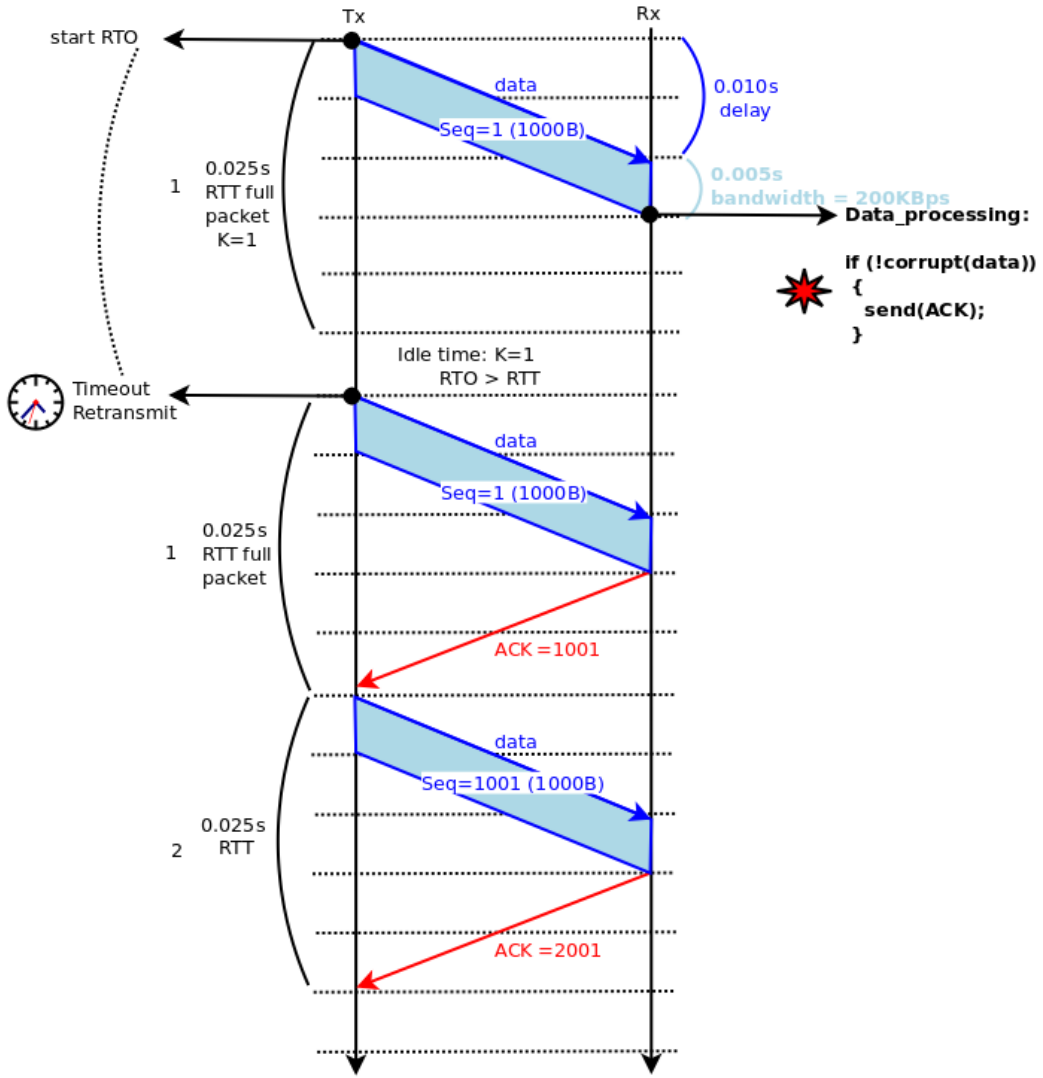
Stop & Wait Errores (Segmentos Perdidos)



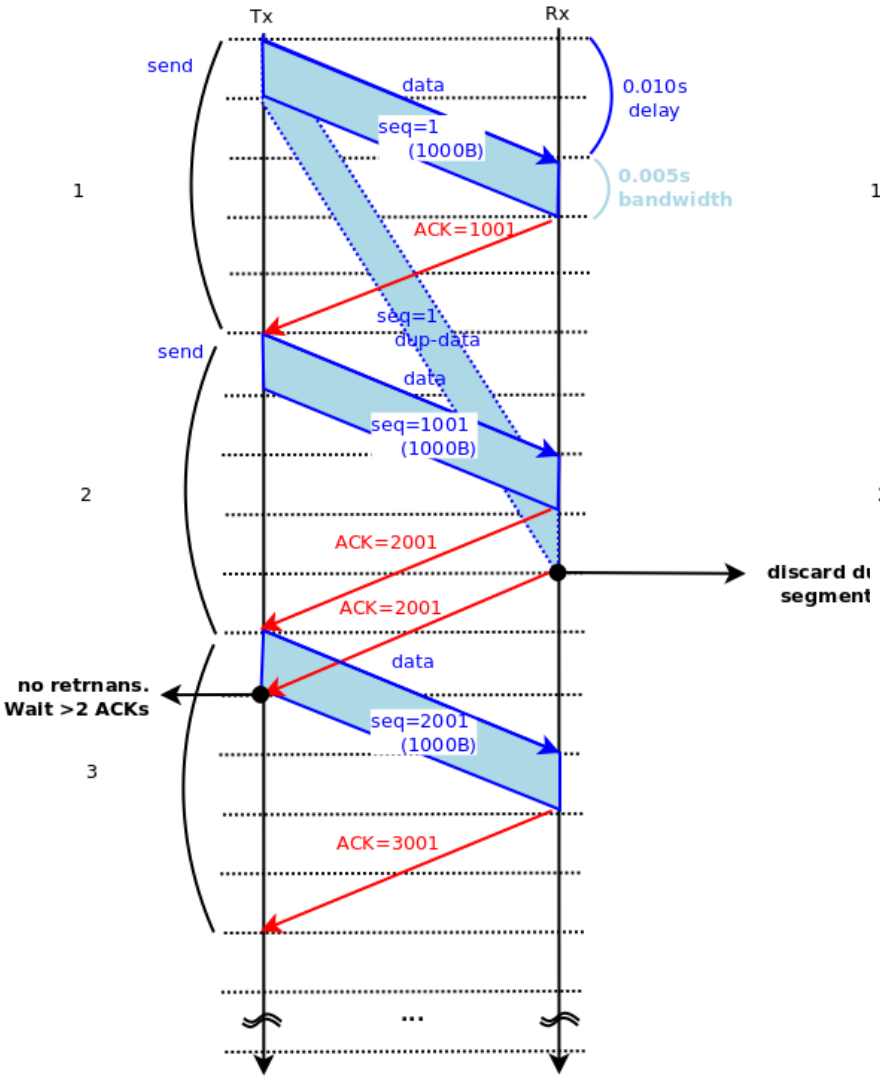


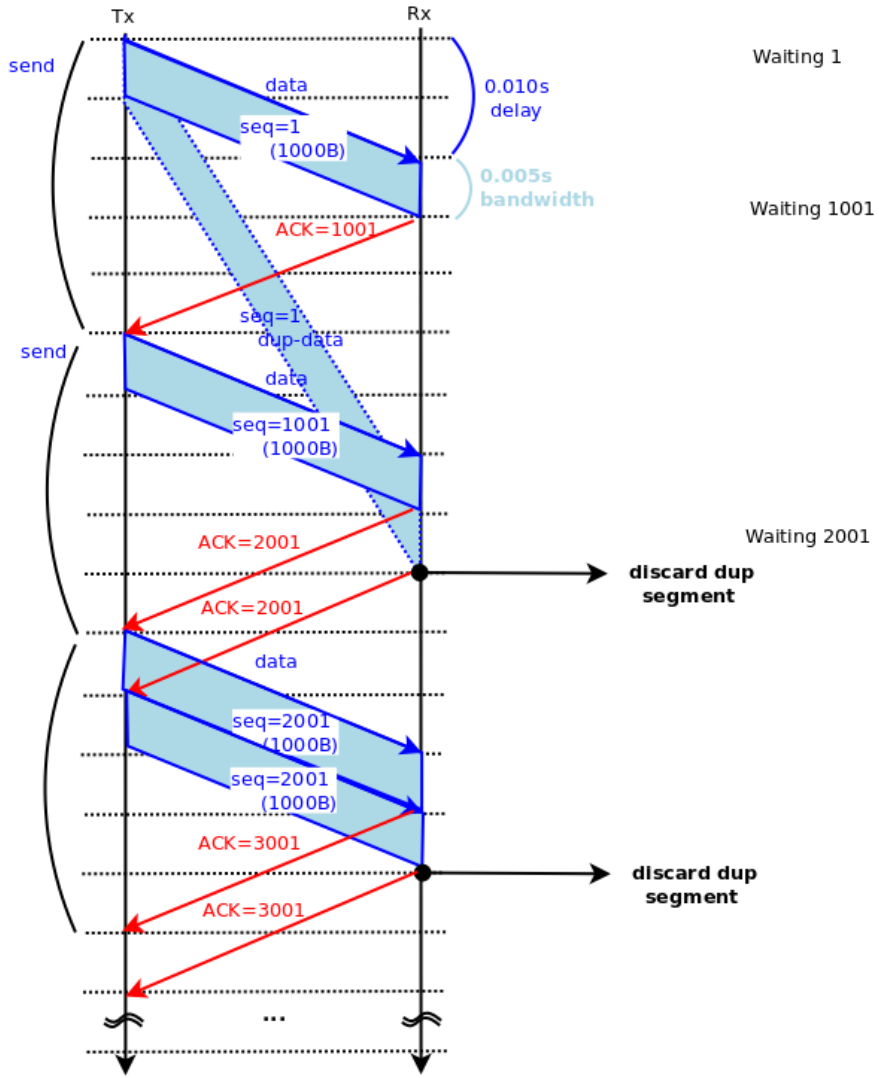
Stop & Wait Errores (Errores Checksum/CRC)



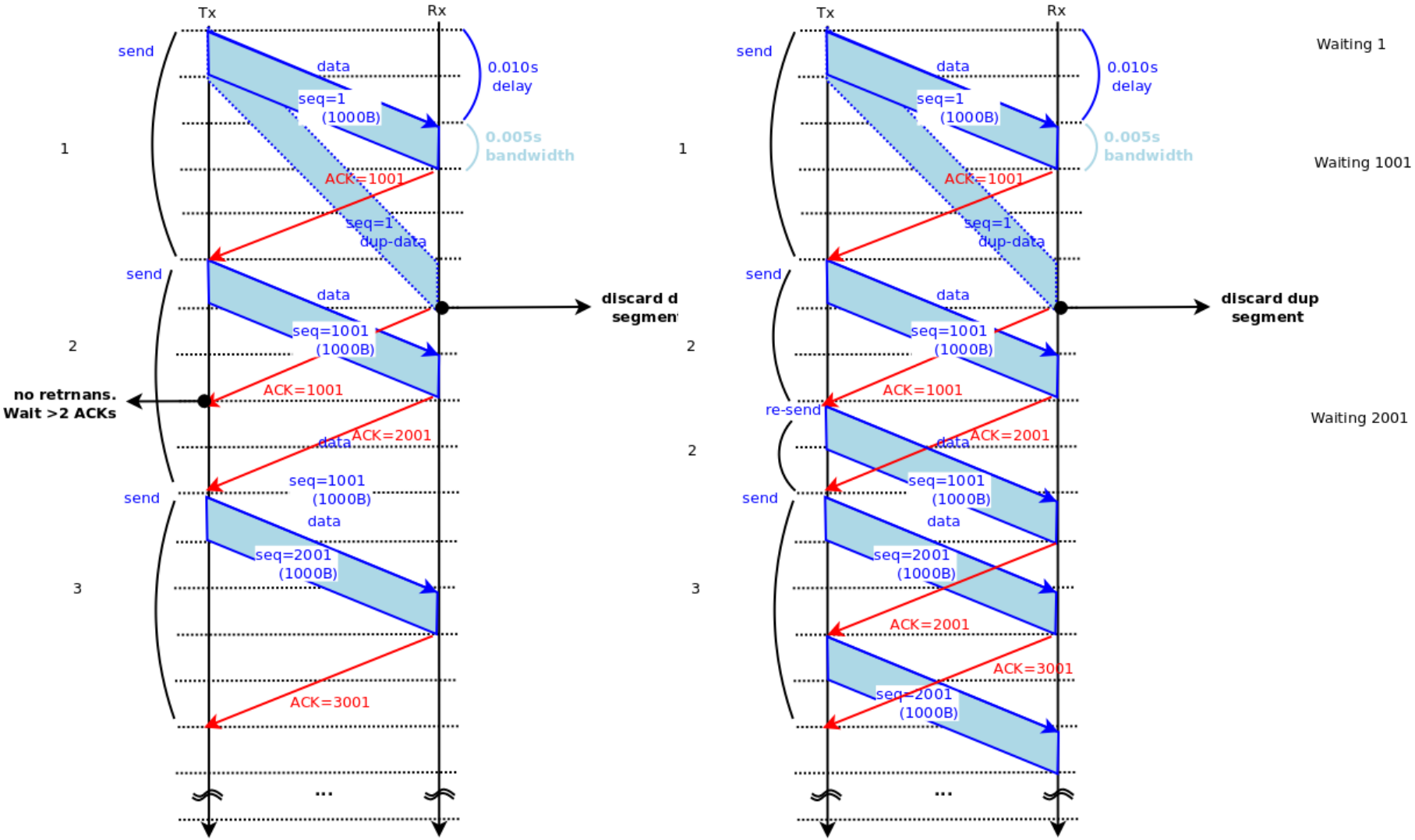


Stop & Wait Errores (Segmentos Duplicados)

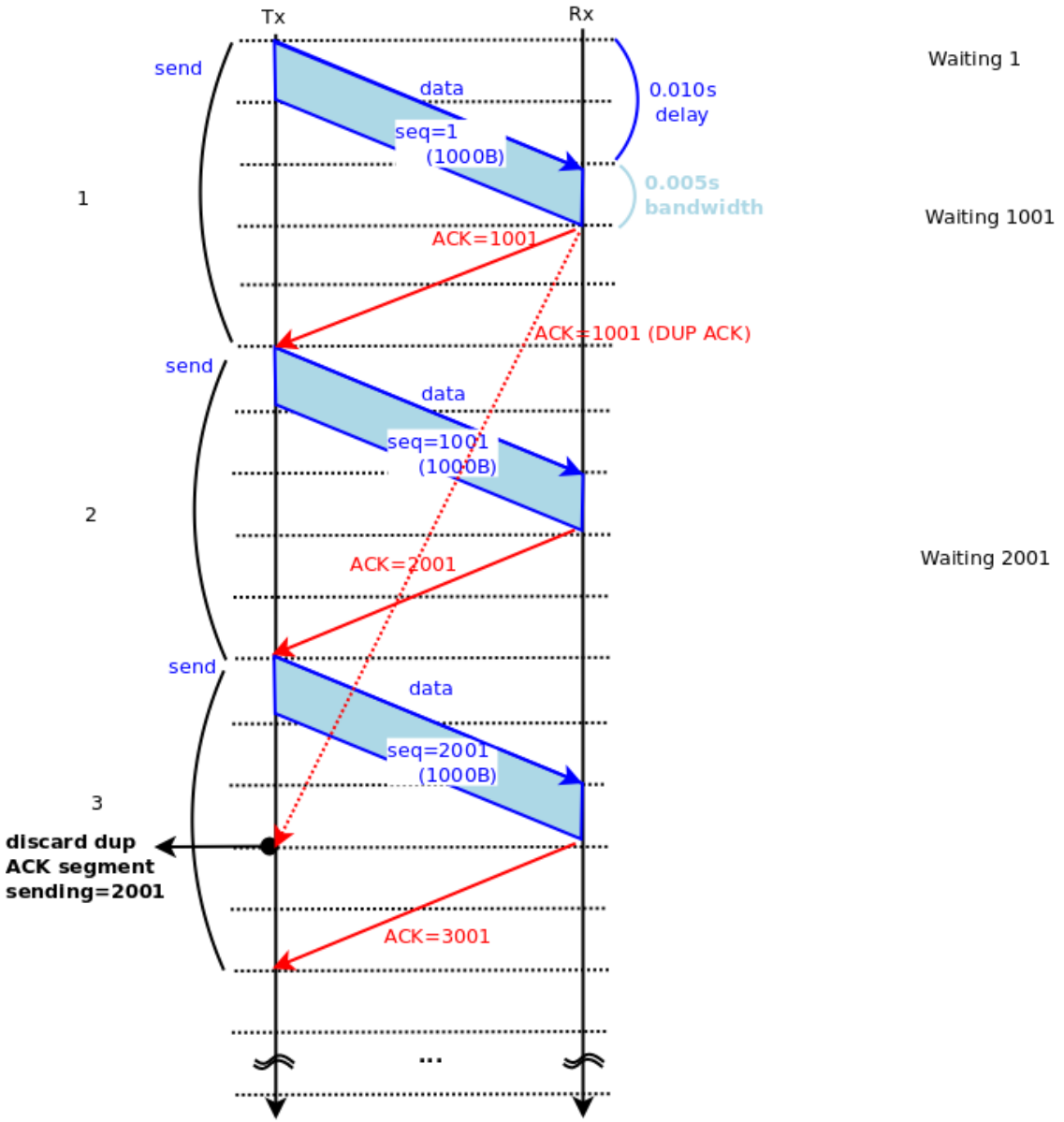




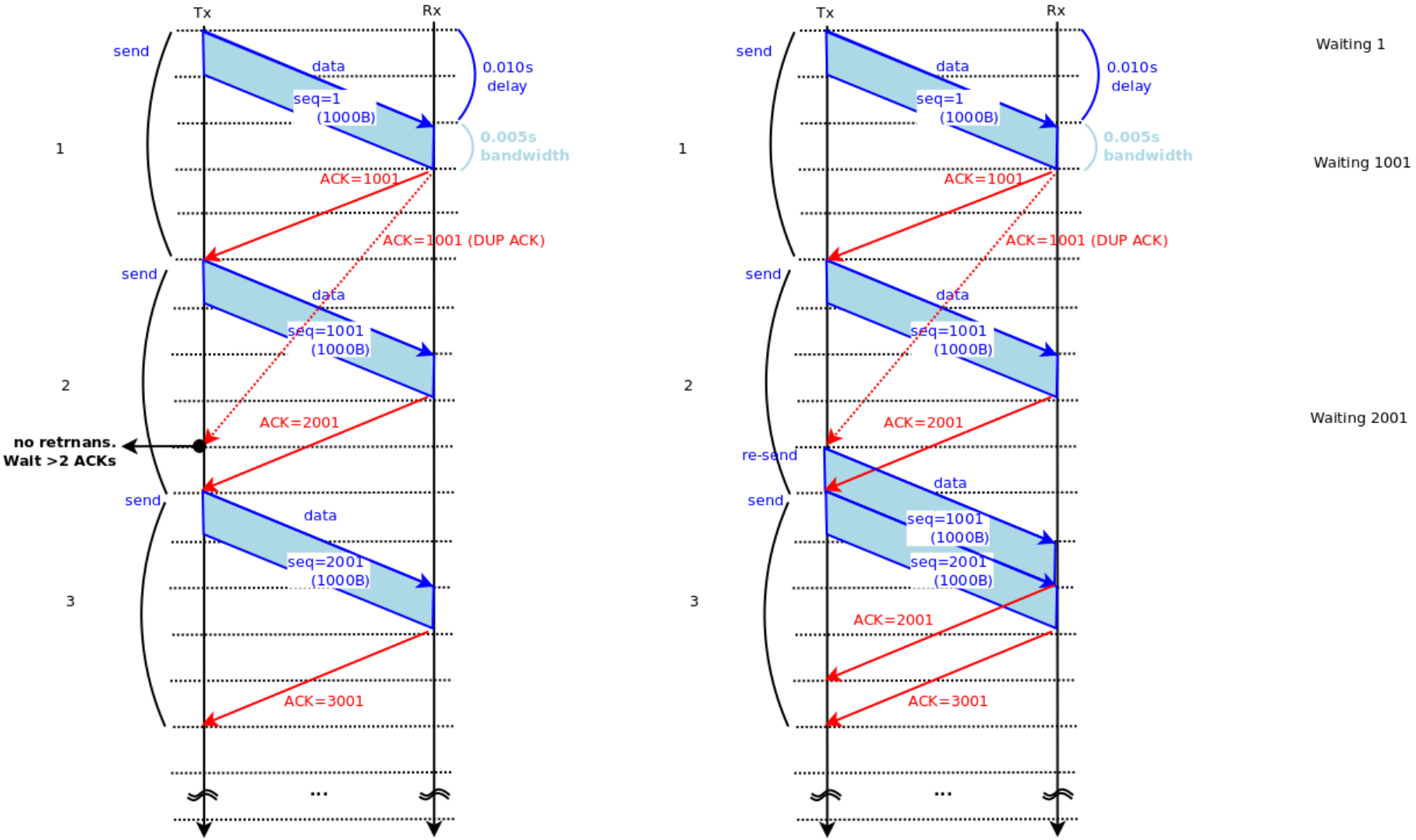
Stop & Wait Errores (Segmentos Duplicados 2)



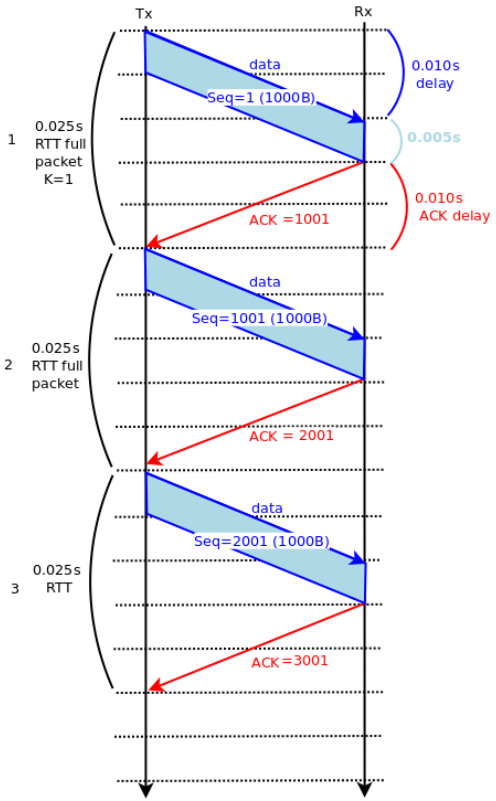
Stop & Wait Errores (ACK Duplicados)



Stop & Wait Errores (ACK Duplicados 2)



Análisis de Rendimiento S&W



$$S = MaxSgmt_{bytes} = 1000B$$

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

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

$$R = BW_{bps} = \frac{S \times 8bits}{L} =$$

$$\frac{1000 \times 8}{0.005} = 1600Kbps = 1.6Mbps = 200KBps$$

$$U = Utiliz = \frac{\frac{L}{R}}{RTT + \frac{L}{R}} =$$

$$0.005/(0.020 + 0.005) = 0.2(20\%)$$

Se obtiene: $1.6Mbps \times 0.2 = 0.32Mbps = 320Kbps$

Análisis de Rendimiento S&W (Cont.)

- Si RTT aumenta y BW aumenta se hace peor.
- Por ejemplo enlace de 1 Gbps y 50ms de latencia ida y vuelta $BDP = 1Gbps \times 50ms$.

$$L = 1Gbps$$
$$RTT = 0.050s$$
$$S = MaxSgmt_{bytes} = 1500B$$
$$\frac{L}{R} = \frac{1500 \times 8}{(1 \times 1000^3)} = 0.0000120s$$
$$U = \frac{0.0000120}{(0.050 + 0.0000120)} = 0.00024(0.0024\%)$$

Se obtiene: $1Gbps \times 0.00024 = 240Kbps$

Conclusiones S&W

- S&W+SeqNumbers (#) $0..M-1$ en datos/ACK recupera los problemas en la red.
- Problema: Sequence Number Wrap Arround, PAWS (Protection Against Wrapping Sequence) (aumentar el espacio de números de secuencia, puede usarse time-stamping).
- El sistema es ineficiente, envía un dato por vez, ventana de transmisión/recepción: $K = 1$, $W = 1$. No se envía el próximo mensaje hasta que no se confirma el que se envió.
- Sistema simple: 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.
- Si no recibe confirmación se vence el timer y se retransmite.

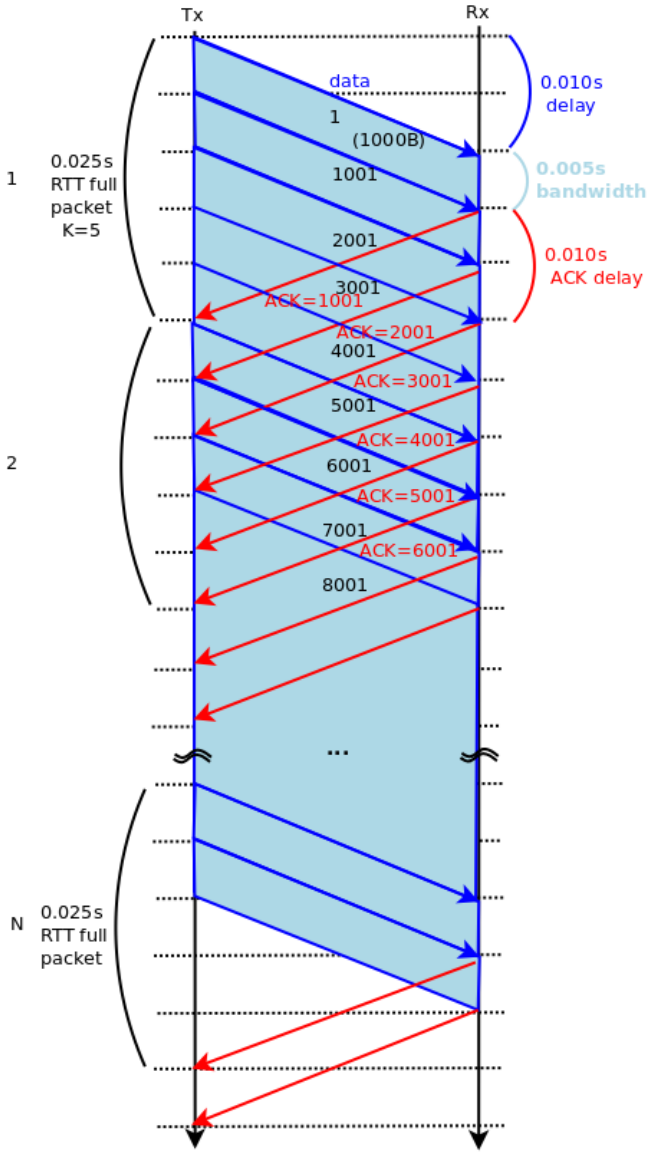
Pipelining/Sliding Window

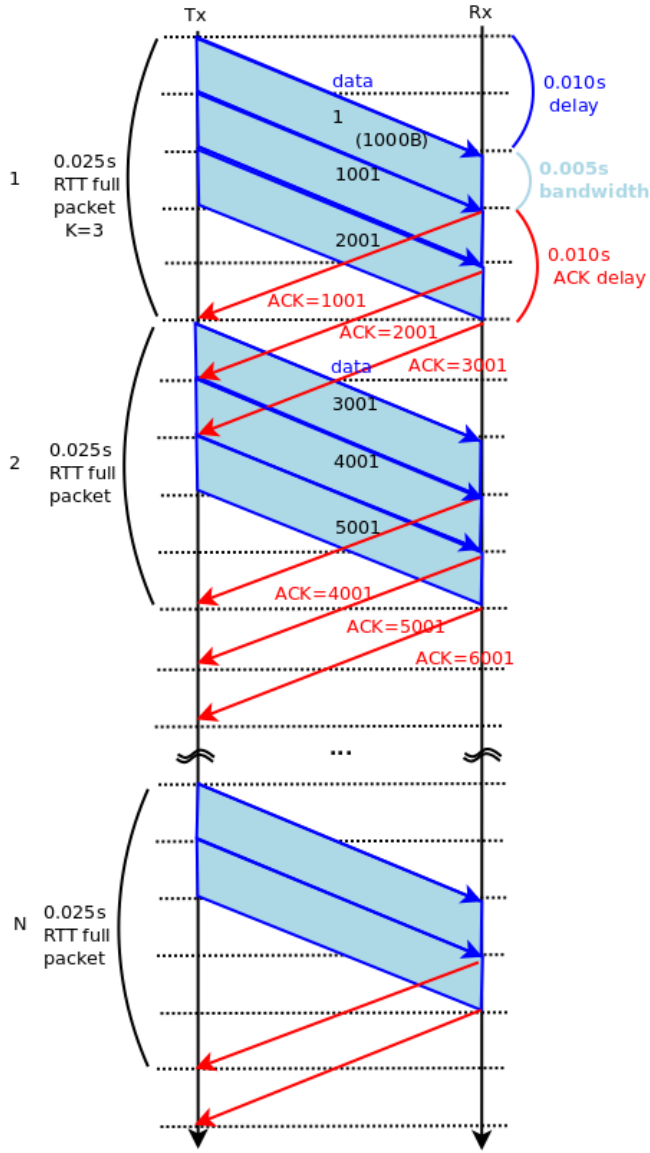
- Pipelining: permitir enviar múltiples segmentos o paquetes por RTT (ráfaga) sin aún haber recibido confirmaciones, paquetes "in-flight".
- Emisor debe saber la cantidad de segmentos o bytes que se puede enviar sin aún recibir confirmación se llama **Ventana**, notado como K o W, $W = n$, donde $n > 1$.
- Requiere buffering de Tx, *TxBuf* (lo que deja la capa superior y aún no se confirmó) y de Rx, *RxBuf* (lo que se va recibiendo hasta entregar a la capa superior).
- Por cada mensaje enviado se podría iniciar un timer de retransmisión, *RTO* (no escala).
- Se mantiene un *RTO* por ráfaga, para el segmento más viejo no confirmado.
- Receptor debe confirmar los segmentos recibidos (Se confirma indicando lo que se espera).

Pipelining/Sliding Window (Cont.)

- Confirmaciones deslizan la ventana sobre *TxBuf* habilitando nuevos segmentos a transmitir.
- Podría generarse Confirmaciones Negativas: NAK (NO-Acknowledge), implícitas o explícitas. (TCP usa implícitas con varios ACK dups).
- Algoritmo más eficiente, óptimo, si llena el pipe.
- Los números de secuencia, $0..M-1$, si se numeran en módulo, no necesariamente son $M = K$, Sucede que $K < M$. Puede ser por bytes o por PDU (segmento). (TCP lo hace por bytes).
- Aumentar M (números de secuencia) permite ser tolerante a fallas de ACK delayed.
- Alternativas:
  - Go-back-N (confirma en orden).
  - Selective-Repeat (confirma fuera de orden).

Pipelining


Tx
Rx
data
1
(1000B)
1001
2001
3001
4001
5001
6001
7001
8001
ACK=1001
ACK=2001
ACK=3001
ACK=4001
ACK=5001
ACK=6001
0.010s
delay
0.005s
bandwidth
0.010s
ACK delay
1
0.025s
RTT full
packet
K=5
2
N
0.025s
RTT full
packet


Tx
Rx
data
1
(1000B)
1001
2001
3001
4001
5001
ACK=1001
ACK=2001
ACK=3001
ACK=4001
ACK=5001
ACK=6001
0.010s
delay
0.005s
bandwidth
0.010s
ACK delay
1
0.025s
RTT full
packet
K=3
2
N
0.025s
RTT full
packet

Análisis de Rendimiento

$$W = Window = 3, S = MaxSgmt_{bytes} = 1000B$$
$$RTT = Latencia_{seg} = 0.020s = 0.010s + 0.010s$$
$$L = DelayTransf_{seg} = 0.005s$$
$$R = \frac{1000 \times 8}{0.005} = 1600000bps = 1.6Mbps$$
$$U = \frac{W \times \frac{L}{R}}{RTT + \frac{L}{R}} =$$
$$(3 \times 0.005)/(0.020 + 0.005) = 0.6(60\%)$$

- Se obtiene: $1.6Mbps \times 0.6 = 0.96Mbps$
- Si $W = 5$ se obtiene el 100%, si el tráfico es constante, $1.6Mbps$.
- Optimo:
  $BDP = RTT \times BW = 0.025s \times 200KBps = 5K = 5000B = 5MSS$.

Go-back N

- Se tiene una ventana en bytes o en MSS (Max. Segment Size) de tamaño $W = n, n > 1MSS$. Numeración de segmentos, se realiza en módulo M, $W <= (M - 1)$.
- Go-back N 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.
- Confirmar cada segmento tiene ventajas ante la pérdida de los ACK.
- Emisor transmite tantos segmentos o bytes que admite W de acuerdo a lo que tiene en el buffer dejados por la capa superior.
- Inicia RTO al enviar segmento.

Go-back N, Emisor

- Eventos en el lado del Emisor (Tx):
  - Si tiene datos en el buffer *TxBuf* y la ventana lo permite, $W > in\_flight\_segments$, los transmite.
  - Si no tiene *RTO* activo arranca un nuevo *RTO* para el primero que manda.
  - Si recibe un ACK en orden, desliza/actualiza la ventana, si hay segmentos "in-flight" re-arranca nuevo *RTO*, sino lo descarta.
  - Si vence *RTO*, re-envía todo a partir del segmento más viejo aún no confirmado.
- A diferencia de S&W, más de un segmento "in-flight", más de un ACK "in-flight".

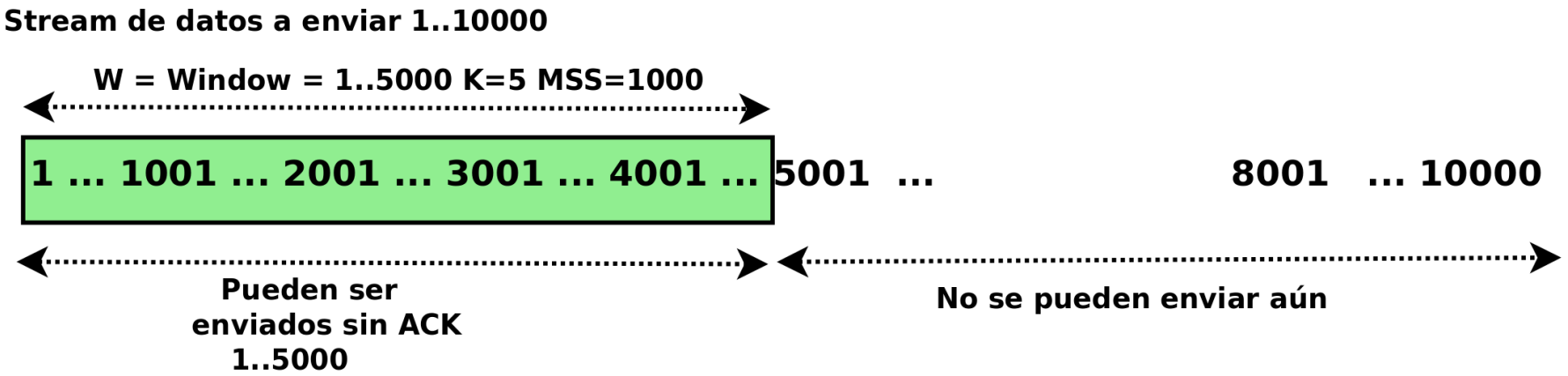
Go-back N, Receptor

- Eventos en el lado del Receptor (Rx):
  - Si recibe segmento en orden confirma por la positiva indicando el próximo que espera.
  - Si segmento recibido corrupto lo descarta como si se hubiese perdido, espera retransmisión.
  - Si recibe segmento fuera de orden confirma indicando que espera uno anterior (NAK).
    - Puede descartar el segmento fuera de orden, ya que será retransmitido.
    - Puede bufferear el fuera de orden, pero no entregar a capa superior, esperando que llegue el/los segmento/s que llena/n el hueco y luego confirmarlo.

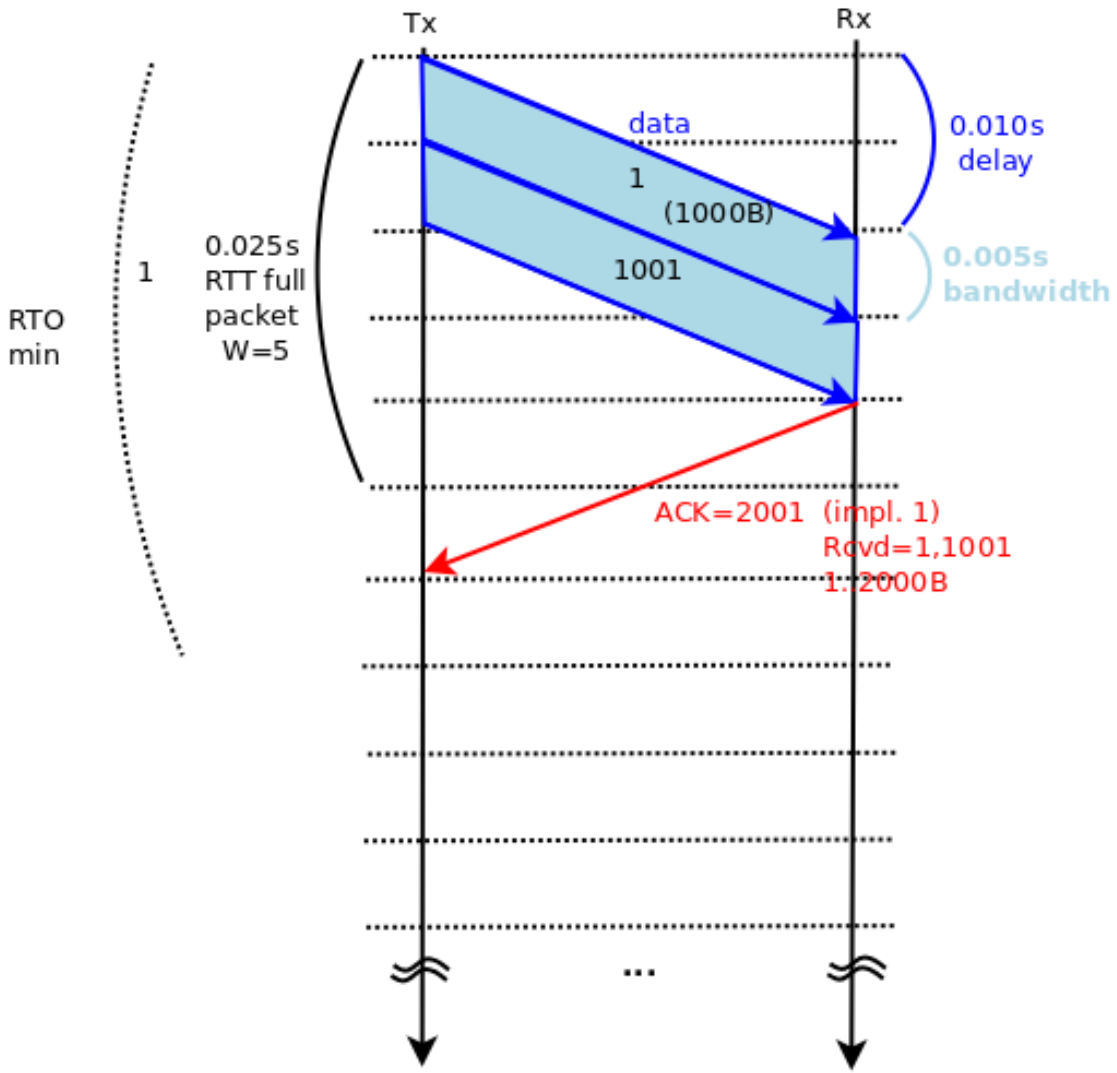
Go-back N, Receptor Extras

- Se aprovechan segmentos de datos para confirmar: Piggy-backing.
- El receptor puede usar: timer de ACK, *DelACK*, para confirmar. $RTO > DelACK + RTT$ (Delayed ACK) y aprovechar datos en el otro sentido.
- *DelACK* debe aprovechar piggy-backing, y confirmaciones acumulativas pero sin demorar demasiado tiempo el flujo de datos, sino podría generar retransmisiones innecesarias.

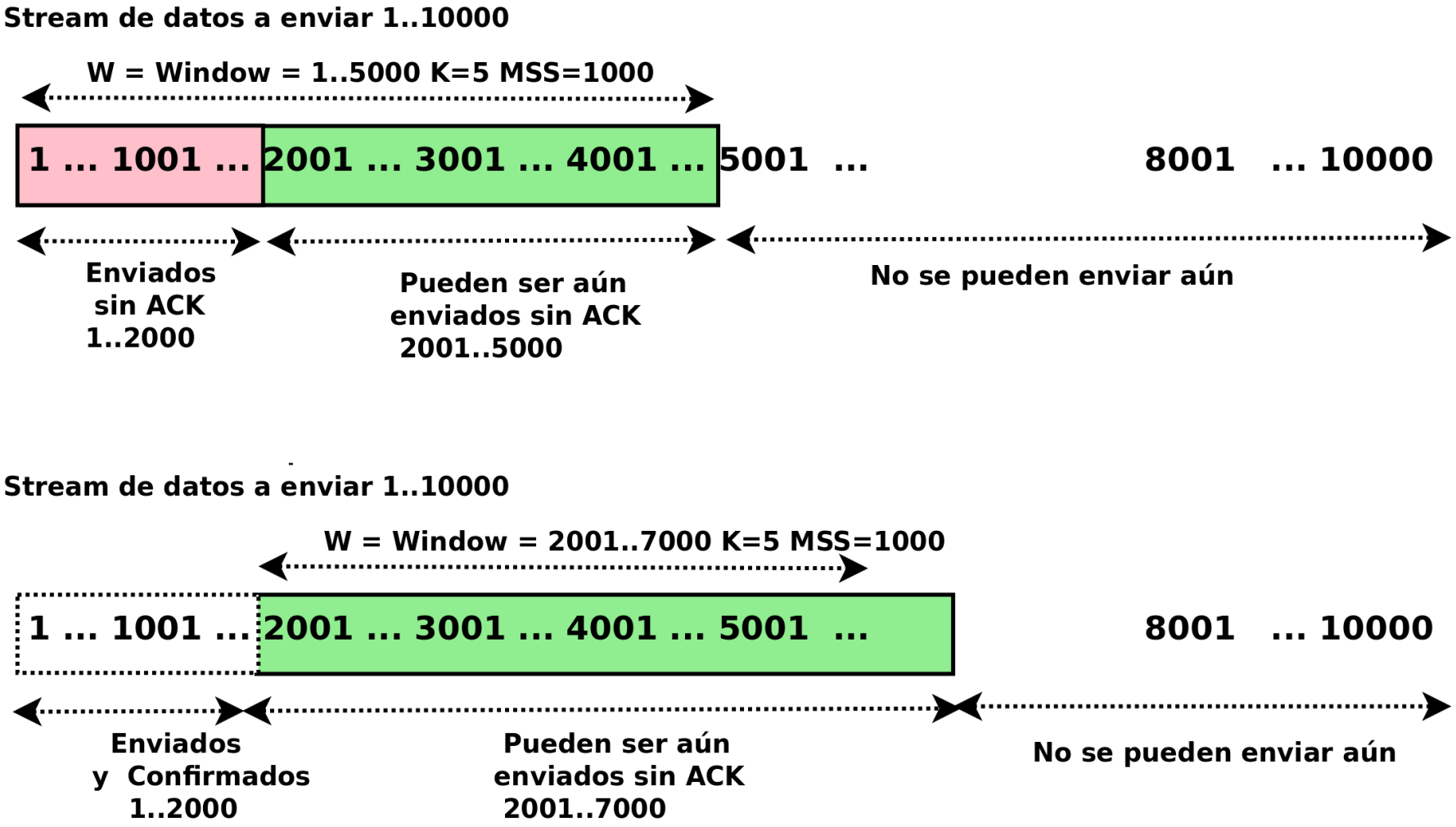
Go-back N, Ventana Inicial



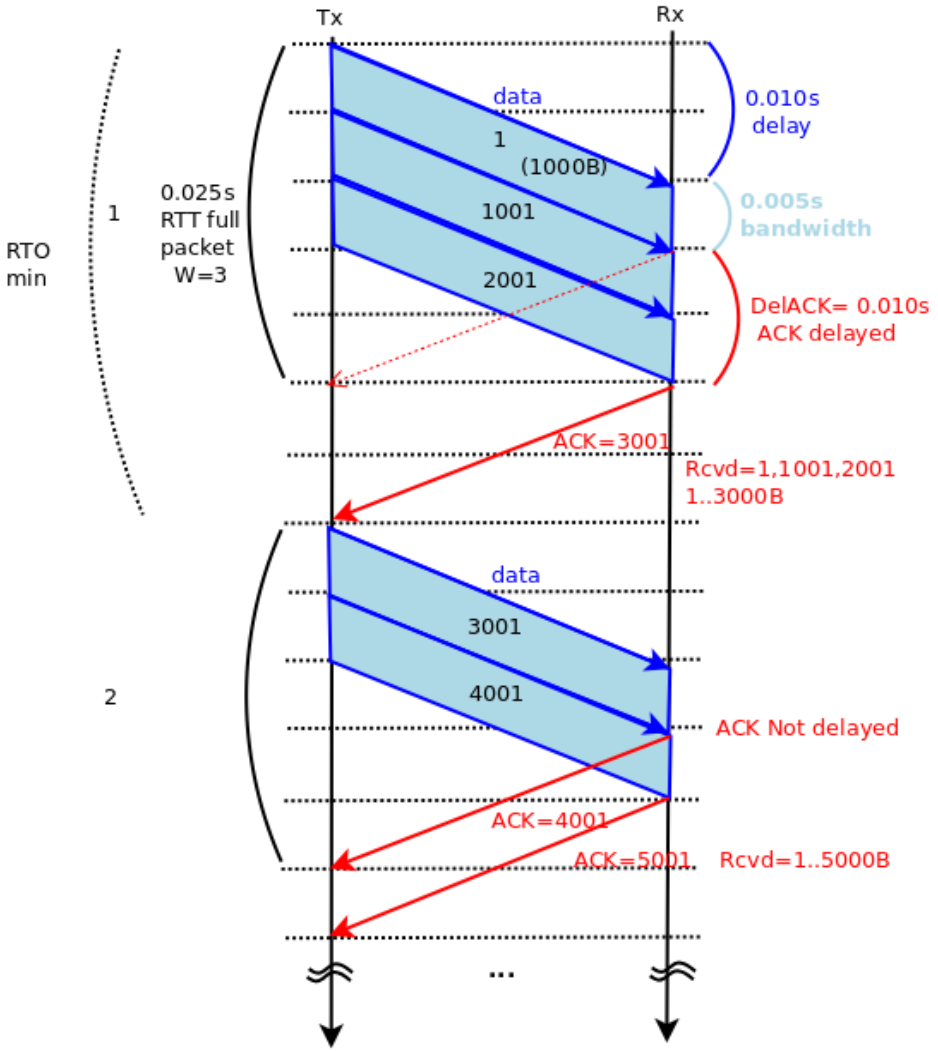
Go-back N, Ventana Envíos

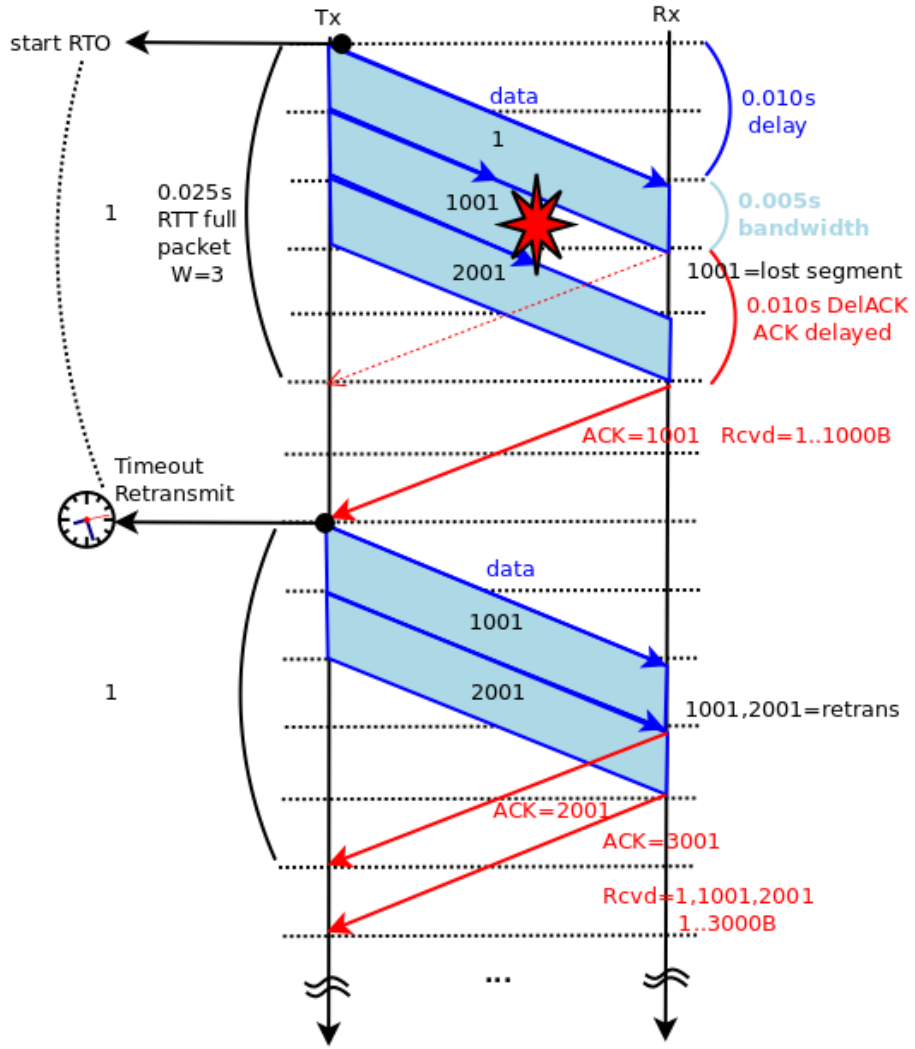


Go-back N, Ventana Actualizada

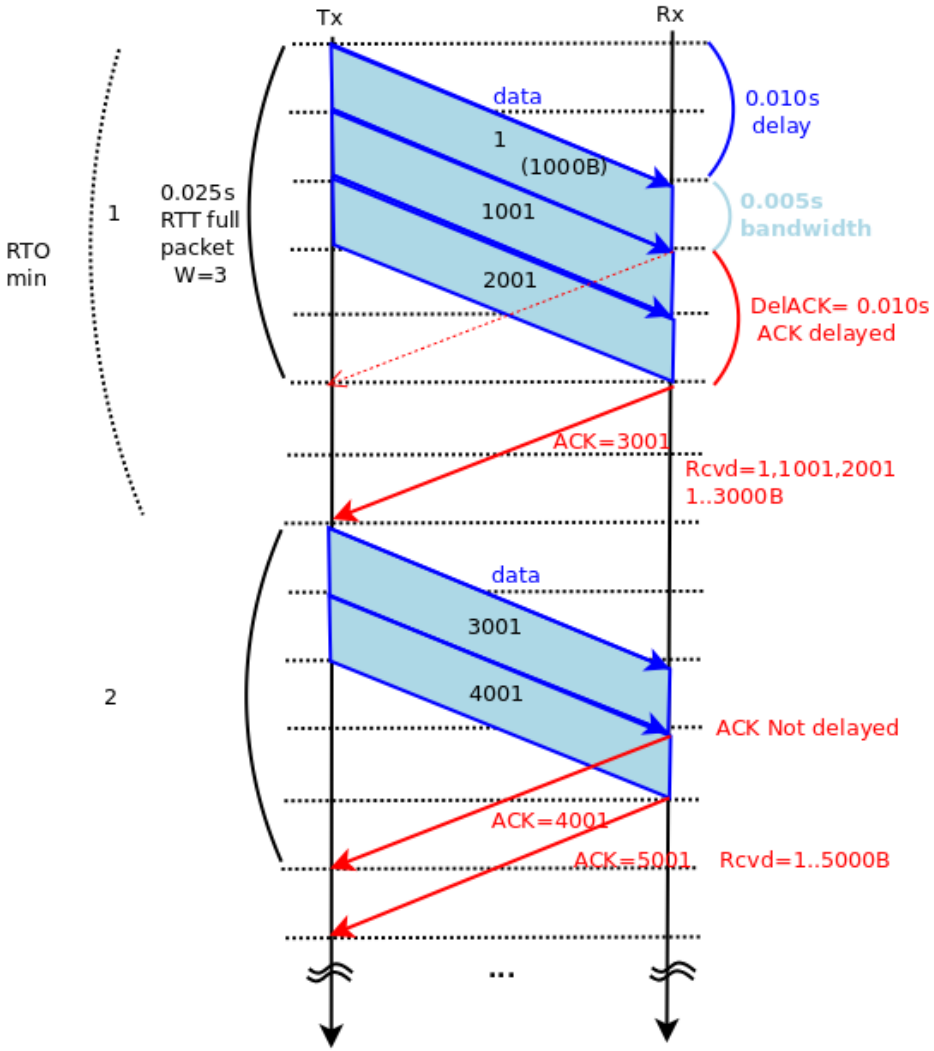


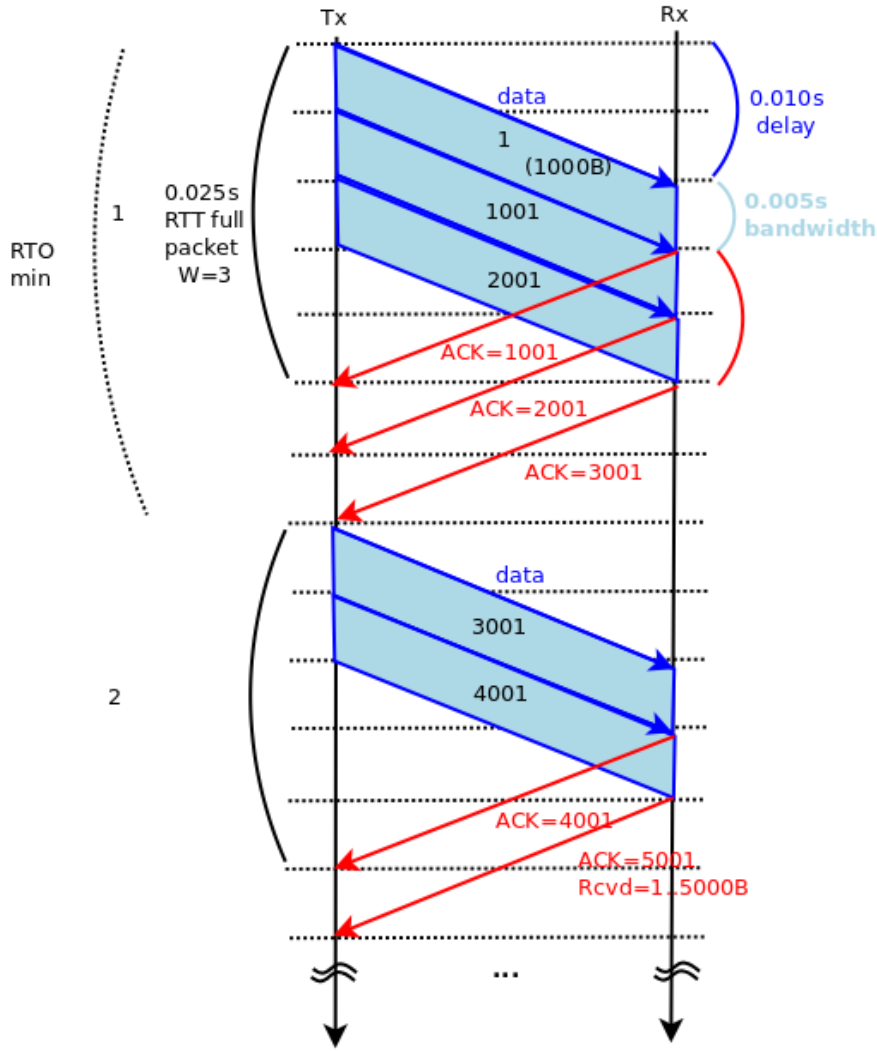
Go-back N (lost segment)



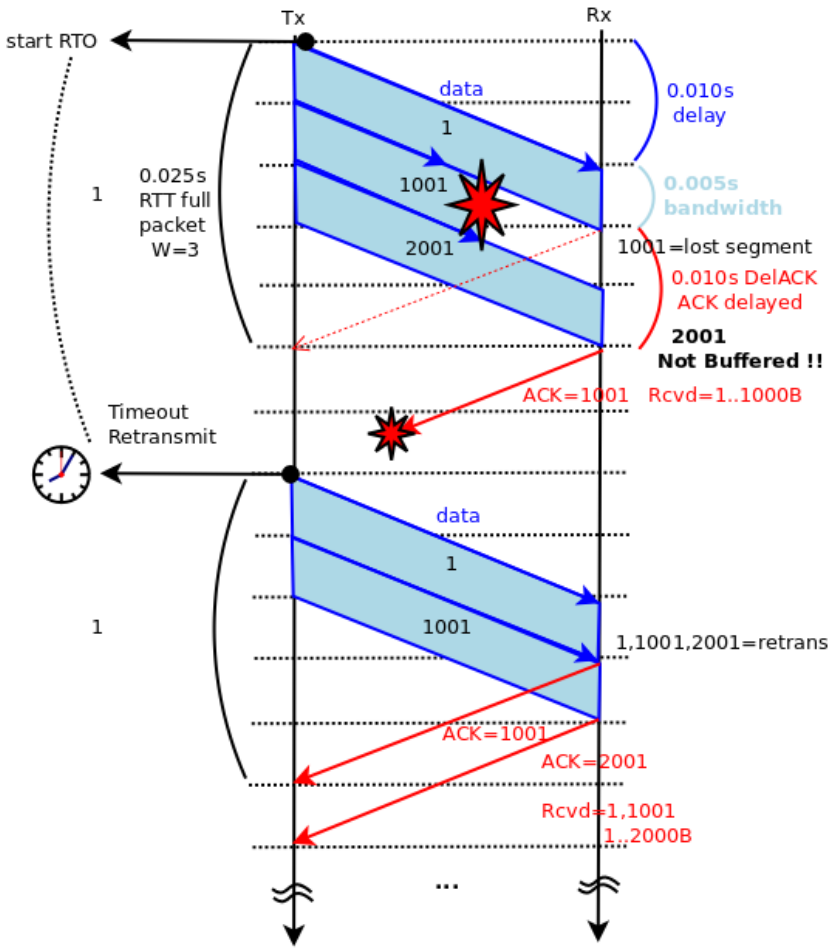


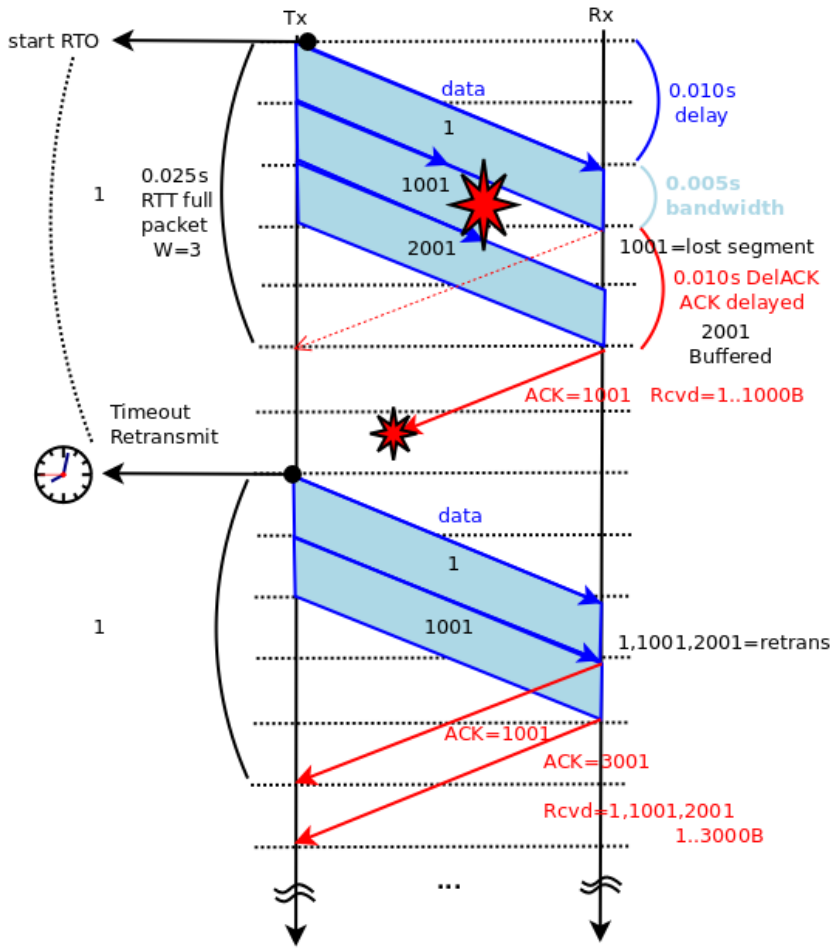
Go-back N (ACK acum.)



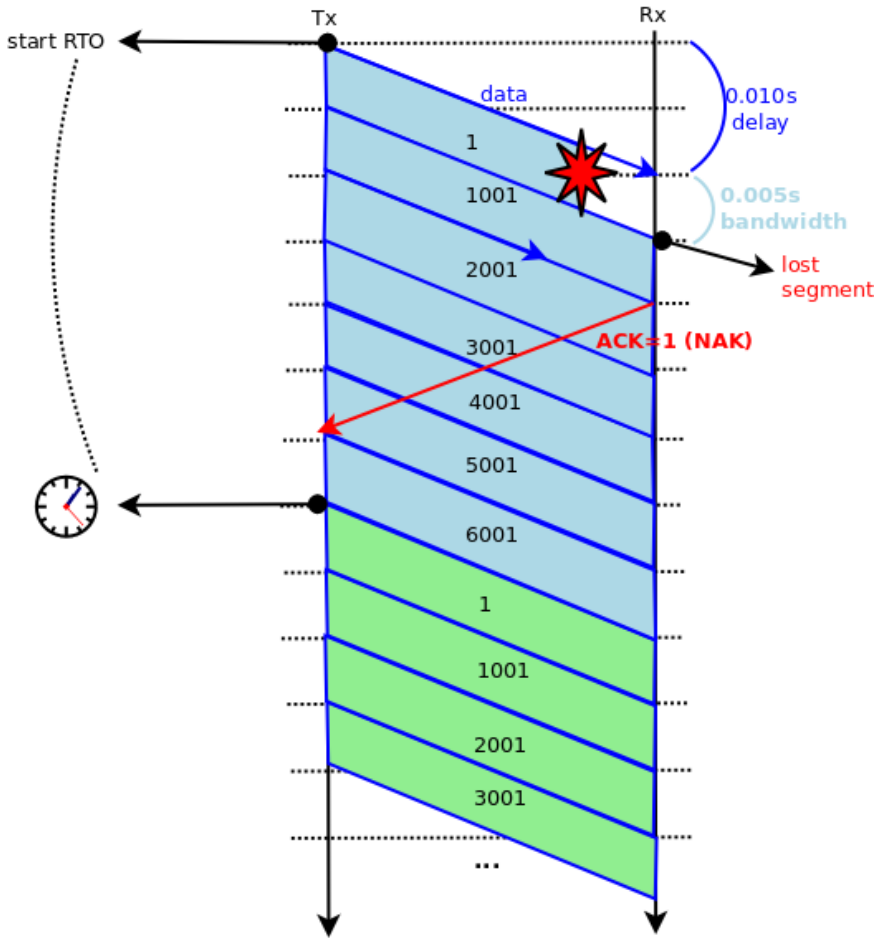


Go-back N (Buffer out-of-order)





Go-Back N, *RTO*

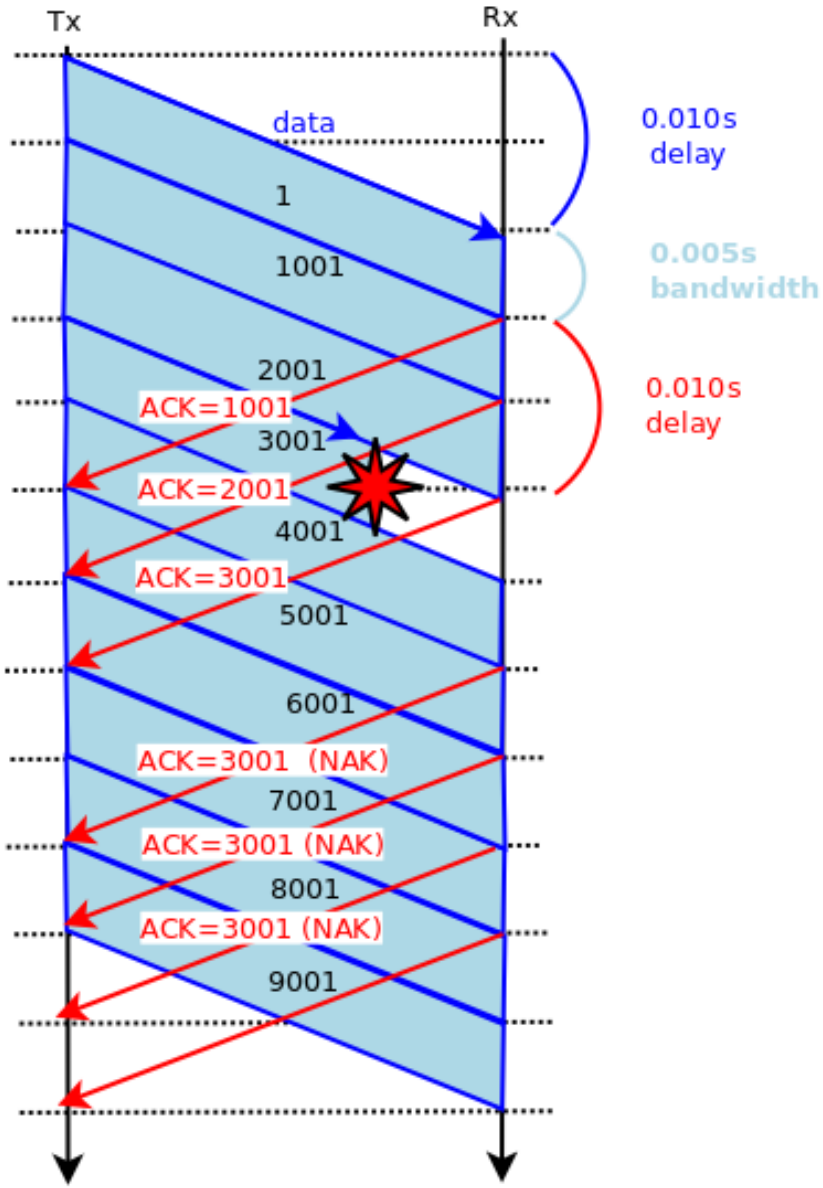


- Si pierde uno segmento y se enviaron varios siguientes es como "vaciar" el pipe y volver atrás: Go-Back-N.
- Si se mantienen segmentos en el receptor más fácil la recuperación.

Go-back N, Segmento fuera de Orden

- Las confirmaciones por la negativa pueden generar ACK duplicados (NAK).
- Mensajes fuera de orden:
  - Buffering hasta recibir los que llenan los huecos. (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).
- Los NAK por si solos no generan retransmisiones, requiere varios duplicados para retransmitir.
- UN NAK solo no necesariamente indica pérdida, puede ser re-ordering en la red.

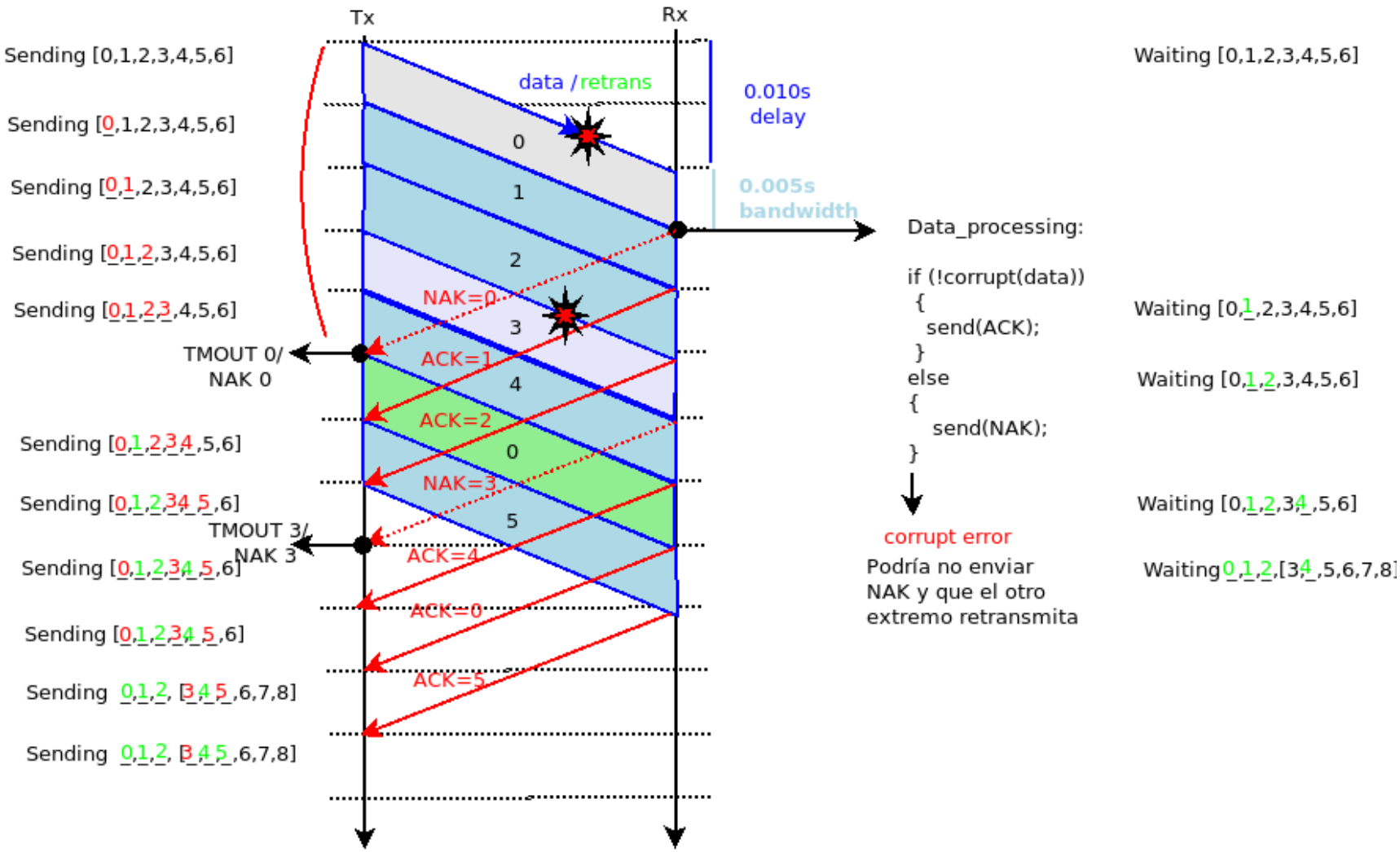
Go-Back N, ACK duplicados, NAK



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, $W <= \frac{(M-1)}{2}$, para evitar confundir los ACK de segmentos.
- La ventana se desliza sin dejar huecos, desde los confirmados más viejos.

Selective Repeat Ejemplo



Selective Repeat, Emisor

- Eventos en el lado del Emisor (Tx):
  - Si tiene datos en el buffer y la ventana lo permite los transmite igual que GBN.
  - A diferencia que GBN cada segmento requiere su *RTO*, se puede simular con un solo timer.
  - Si recibe un ACK en orden, desliza/actualiza la ventana, detiene el *RTO* para el segmento confirmado.
  - Si recibe un ACK fuera de orden, no desliza/actualiza la ventana, detiene el *RTO* para el segmento confirmado y lo marca como ya recibido.
  - Si vence *RTO*, re-envía el segmento asociado.

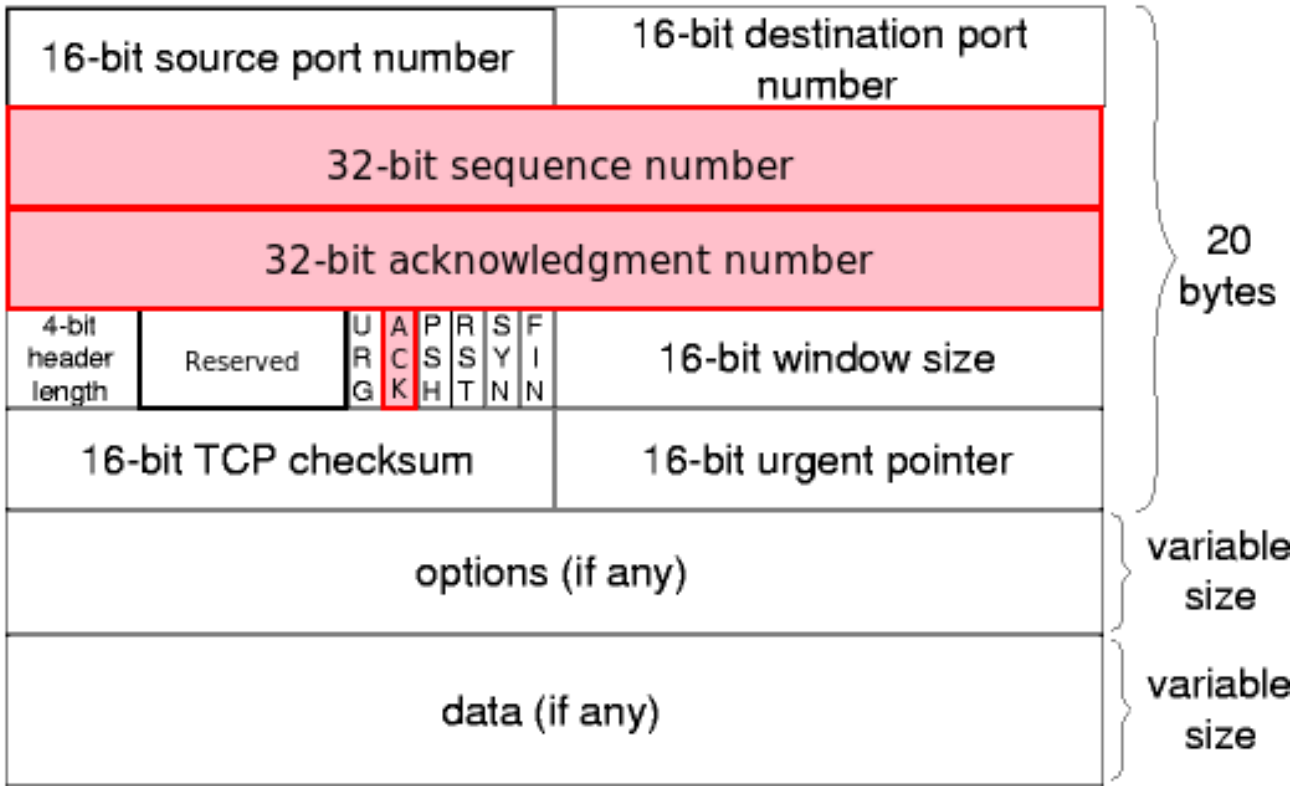
Selective Repeat, Receptor

- Eventos en el lado del Receptor (Rx):
  - Si recibe segmento dentro de la ventana esperada confirma el segmento particular (grupo de bytes).
  - Si recibe segmento dentro de la ventana esperada y no lo tiene lo almacena, si ya lo tiene lo descarta. Siempre confirma. Requiere bufferear los segmentos *RxBuf* .
  - El receptor actualiza su ventana conforme recibe los segmentos en orden.

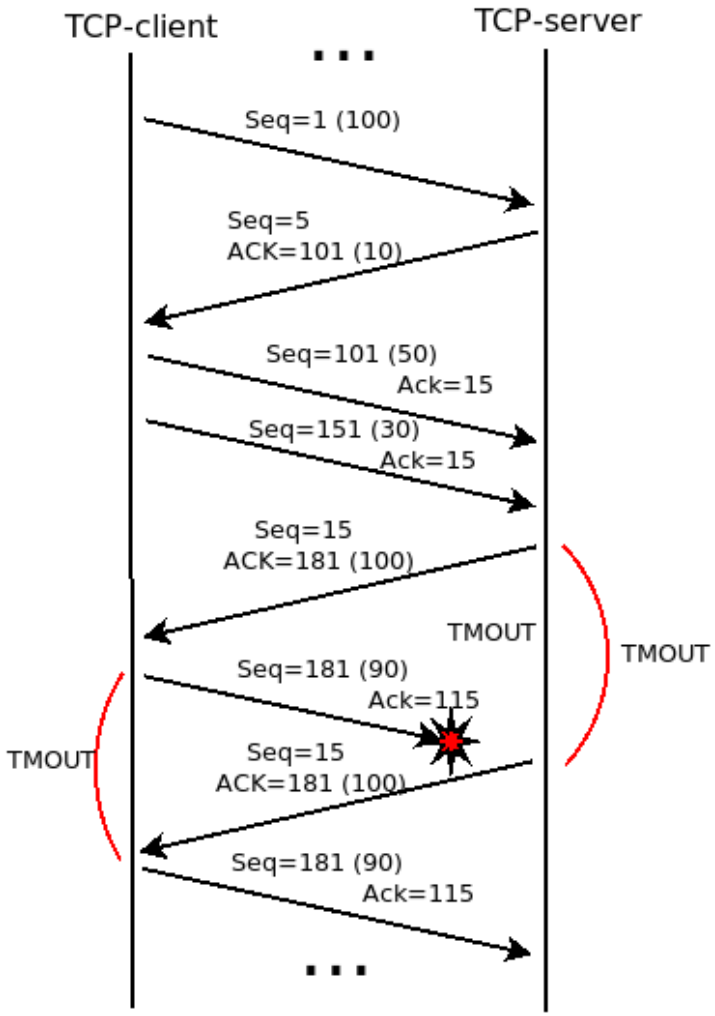
Control de Errores en TCP

- Utiliza Go-back-N (GBN) con ventana dinámica (flow-control), utiliza piggy-backing y permite negociar Selective Repeat con opciones SACK (Selective ACK).
- TCP mantiene el control de errores por bytes (byte oriented), no por segmentos.
- Los segmentos se numeran de acuerdo a bytes enviados (nro. del primer byte) de datos dentro del segmento.
- 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.
- 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.)

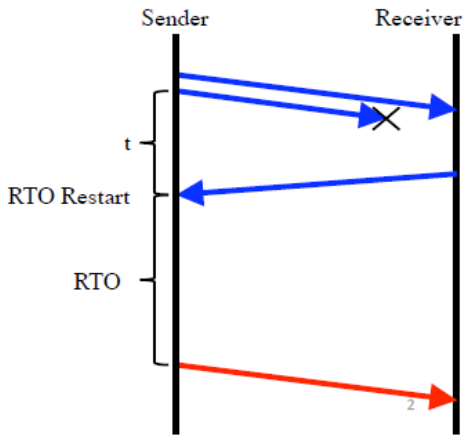
- Por cada segmento (con datos) que envía TCP es como si iniciara un Timer local, *RTO* y pone copia del segmento en cola local *TxBuf* (RFC-793).
- Por cada segmento ACKed en orden descarta el timer asociado y descarta la copia del segmento (RFC-793) del *TxBuf*. Hace lugar para nuevos segmentos a Tx.
- 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*.

Control de Errores en TCP (Cont.)

- 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 y mensajes sin confirmar.
- Receptor con segmentos fuera de orden descarta directamente, mejor podrá re-enviar último ACK solicitando lo que espera (podría dejar en *RxBuf* pero no entregar a la aplicación, tiene huecos).
- Se puede confirmar con ACK acumulativos.
- Si se confirman (ACked) datos, se inicia un nuevo *RTO* (RFC-6298) recomendado. Si todo confirmado se detiene RTO.

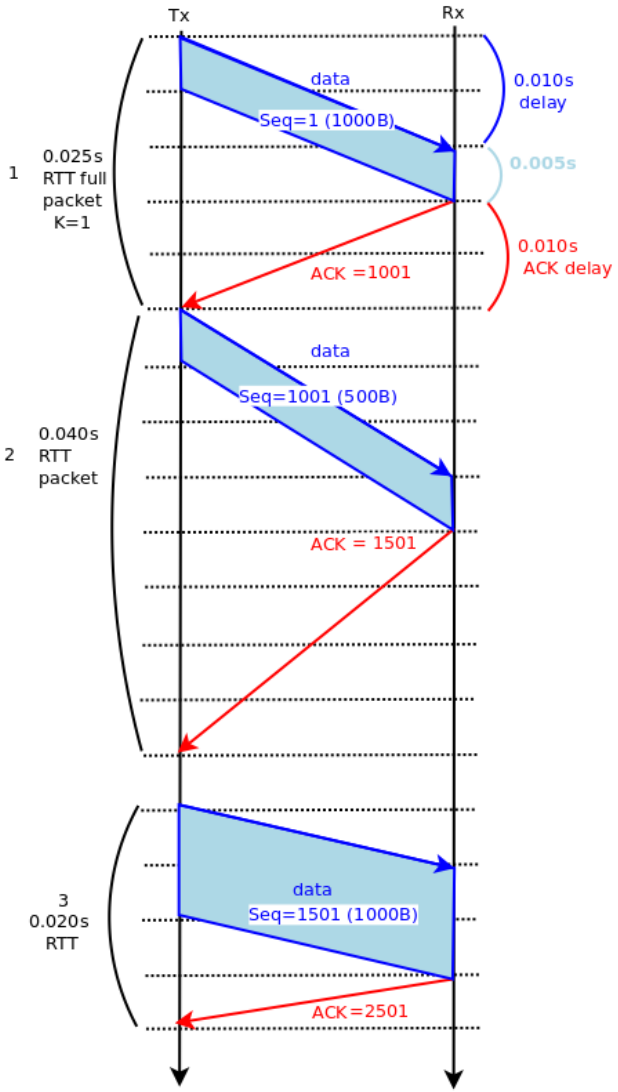
Control de Errores en TCP (Cont.)

- 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.

RTT Dinámico

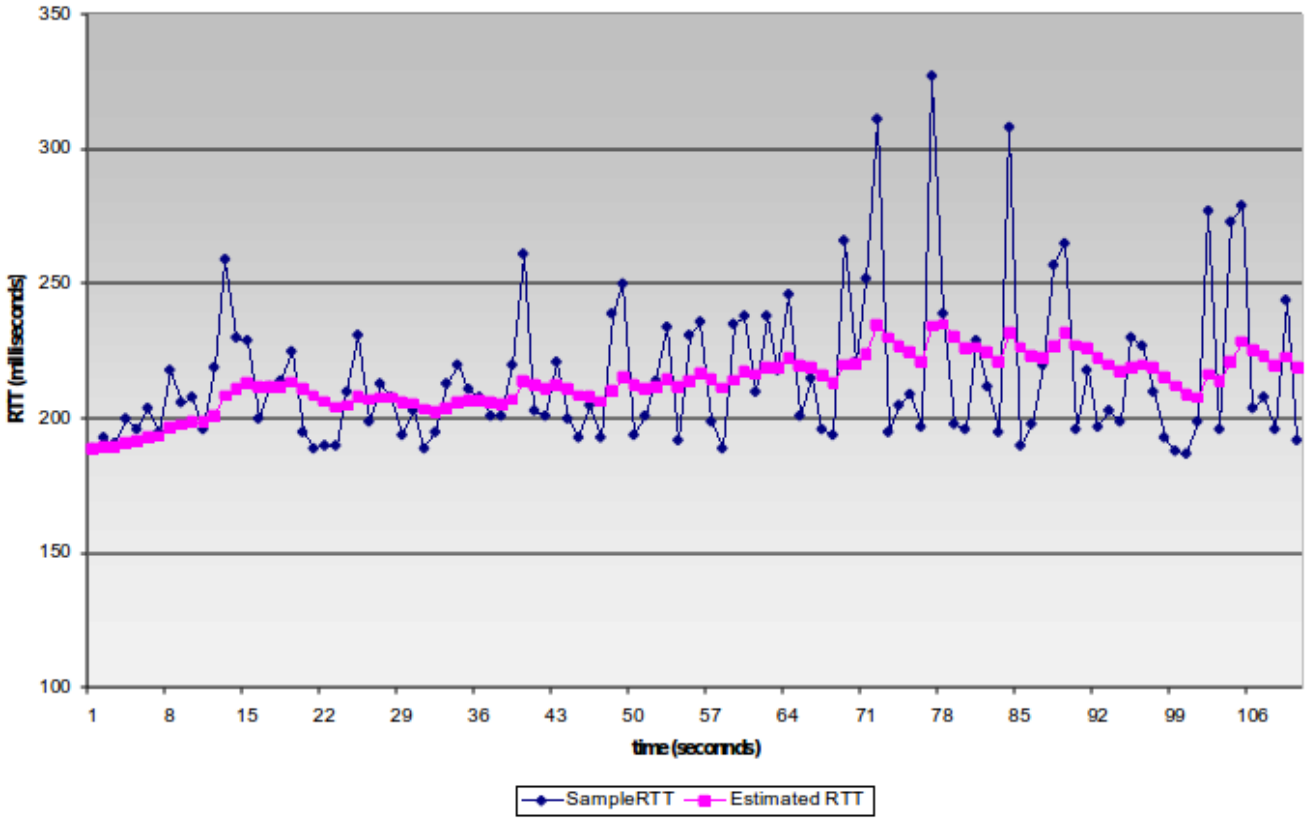


- *RTO* calculado en base el RTT.
- *RTO* debe ser dinámico, debe contemplar estado de la red.
- *RTO* estático solo serviría para L2 (directamente conectados).

Cálculo de RTO

- 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

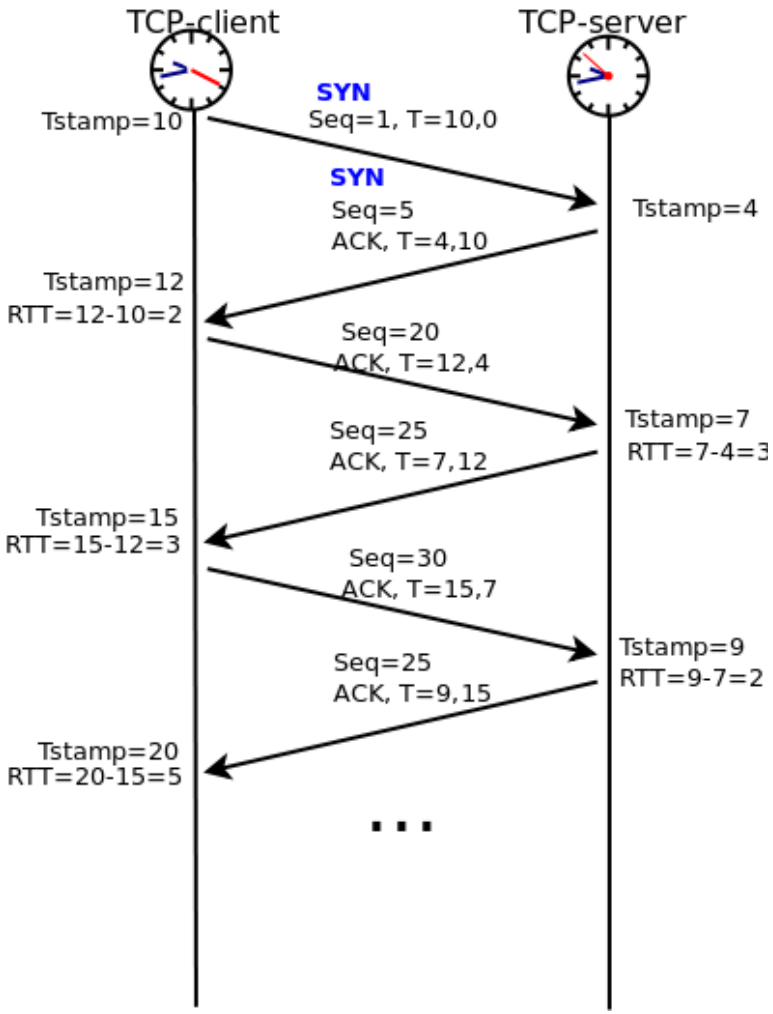
Mínimo RTO

- $RTO < 1seg$: 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, 8e. James F. Kurose & Keith W. Ross.