



MODULO III: Protocolos de Transporte

Objetivos



- Protocolo TCP
 - Servicios
 - Administración de la Conexión
 - Control de Error y Flujo
 - Administración de Tamaño de Ventana
 - Timers
 - Cálculo Timer de Retransmisión
 - Administración de Congestión
 - Multiplexado
 - Header TCP
 - Funcionamiento
- Protocolo UDP
 - Header UDP

TCP: “Transmission Control Protocol” – RFC 793 (año 1981)

- Las principales características son:
 - Protocolo Orientado a Conexión (“Connection-oriented”)
 - Se establece una conexión virtual antes del intercambio de datos
 - Transmisión Confiable (“Reliable”)
 - Utiliza Secuenciado de paquetes y confirmación de recepción (ACKs)
 - Control de flujo.
 - Comunicación del Tipo “Byte-Stream”
 - Protocolo orientado a “byte” no-estructurados (no a trama).
 - “Buffered transfer”
 - Prepara los segmentos de un tamaño tal que al ser encapsulados en datagramas IP la transferencia de datos sea óptima, independiente del tamaño de PDU de la Aplicación.
 - También puede segmentar (PDU de aplicación demasiado grandes)
 - Full-Duplex
 - Permite la comunicación simultánea entre transmisor y receptor a través de un mismo segmento TCP (“piggybacking”)

TCP – Temas a Resolver . . .



- Establecimiento de Conexión
- Terminación de Conexión
- Control de Flujo
- Entrega ordenada
- Estrategia de retransmisión
- Detección de segmentos duplicados
- Multiplexado
- Direcccionamiento
- Todo esto sobre un protocolo que es innatamente no-confiable y no orientado a conexión (IP)

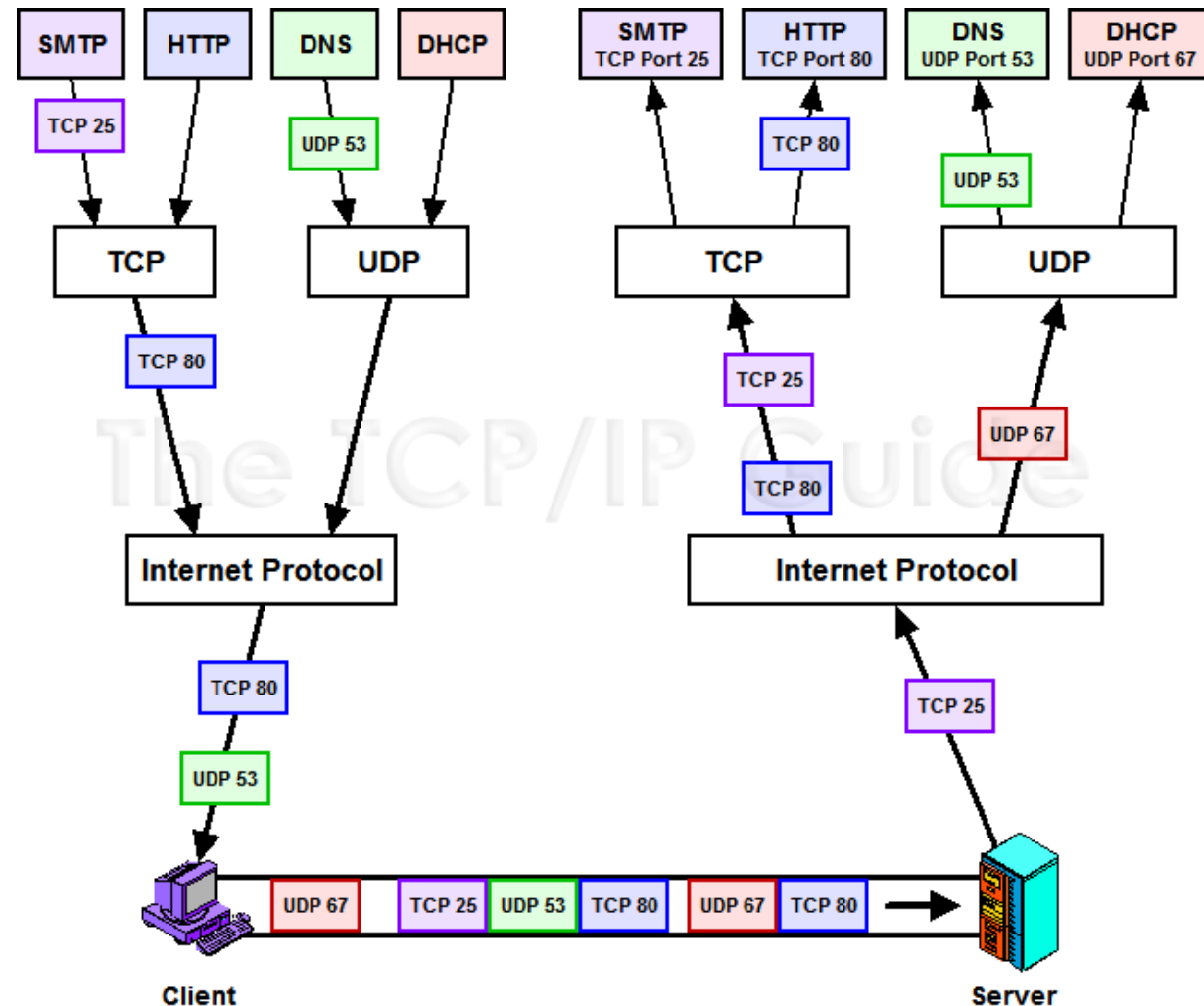
Direccionado



- Aplicación destino identificada por:
 - Generalmente Dirección IP del "host" y puerto
 - Denominado "socket" en TCP
 - Puerto representa un Servicio de Transporte de Usuario (o aplicación)
- Identificación de la Entidad de Transporte
 - Identificado por el "Protocol Number" en los datagramas IP.
 - En TCP/IP existen dos entidades de transporte:
 - TCP – "Protocol Number": 6
 - UDP – "Protocol Number": 17

Multiplexado

- Múltiples aplicaciones emplean el mismo protocolo de transporte
- Las aplicaciones se identifican por número de puerto.
- El multiplexado de protocolos permite que varias aplicaciones clientes (de un mismo host o de distintos hosts) se comuniquen con un única aplicación servidora.



Entrega Ordenada



- Segmentos pueden arribar fuera de orden (debido a IP)
- Se deberán numerar los segmentos transmitidos secuencialmente
- TCP numera cada **octeto** secuencialmente ("Byte oriented")
- Los segmentos son numerados por el primer número de octeto en el segmento.
 - Ejemplo: El primer segmento tiene número de inicio de secuencia a 100. Si este segmento tiene 200 Bytes de longitud, el segundo segmento tendría como número de secuencia a 300.

Estrategia de Retransmisión

- Dos eventos requieren la retransmisión de un segmento:
 - Daño de uno o más bits en tránsito.
 - Pérdida completa de uno o más segmentos.
- En ambos casos: Transmisor no conoce la falla.
- Receptor debe confirmar recepción **exitosa**
- Utiliza ACK **acumulativo** (delay ack...)
 - Envía un ACK cada cierto número de segmentos recibidos (en realidad cada un cierto tiempo).
 - Por ejemplo: Receptor recibe segmentos con SN = 1, 201, 401 y envía un AN (Ack Number) = 601, confirmando todos los segmentos.
- Si un segmento no llega o llega con error, el receptor no genera ACK
 - El transmisor mediante un "Time out" (**RTO**) que espera por ACK, dispara la retransmisión de segmento.

RTO: Timer de Retransmisión

- Crucial en la performance de TCP
 - Ejemplo: Timer en una LAN y en un enlace con alto retardo de propagación.
- El retardo para que lleguen los datos al destino y el ACK depende del tráfico en la red, de la distancia y del tipo de tecnología de comunicación.
- Conclusión: TCP debe manejar retardos que cambien con rapidez.
- Timer de retransmisión puede ser
 - Fijo: No funciona correctamente en interredes por los retardos variables que se producen de acuerdo a las diversas topologías (ver transparencia siguiente).
 - Variable o adaptable: Utilizado por TCP
 - TCP supervisa retardo actual de cada transmisión y modifica el cronómetro de retransmisión.
 - Al transmitir un mensaje, TCP registra el tiempo de ida y vuelta ("round-trip delay"), y va estimando estadísticamente el retardo promedio.
- Mas adelante se ve el RTO con más detalles

Detección de Duplicación



- Si ACK es perdido, los segmentos son retransmitidos. Puede aparecer duplicidad de segmentos.
- Receptor debe reconocer duplicados (número de secuencia ayuda)
- Existen dos casos:
 - Se recibe un segmento duplicado antes de cerrar la conexión TCP.
 - Se recibe un segmento duplicado luego de cerrar la conexión TCP.
- Más adelante se estudiarán estos casos.

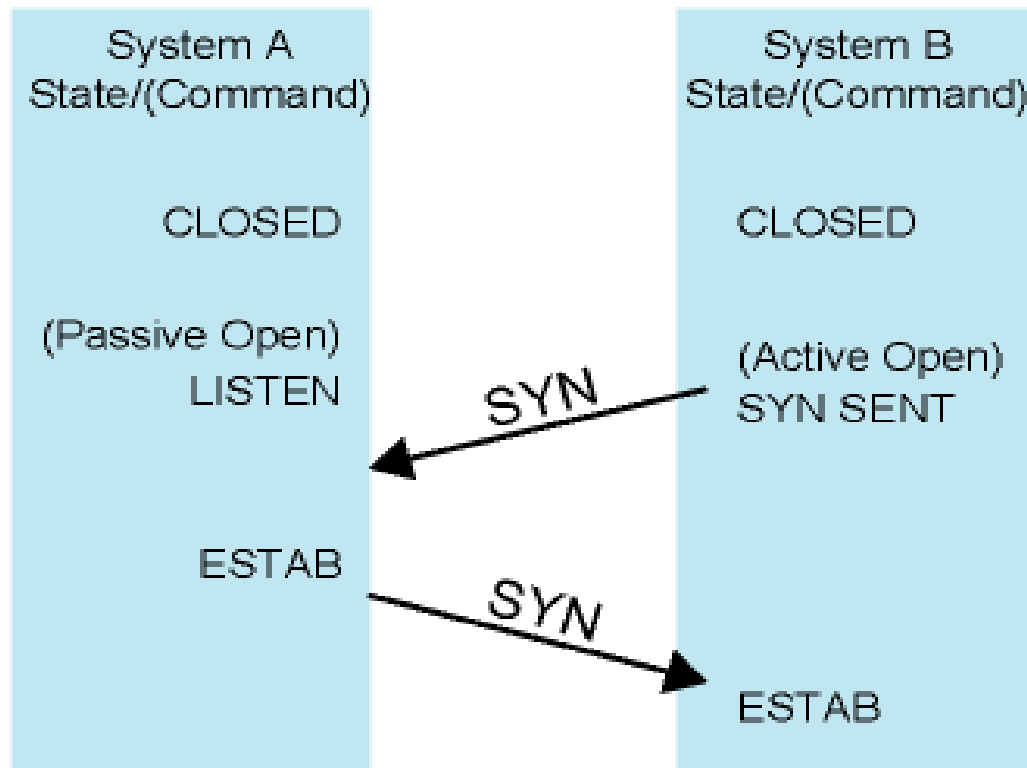
Control de Flujo

- Implementación compleja a nivel de capa de transporte (comparado con capa de enlace), debido a que el retardo de propagación puede ser mayor que el tiempo de transmisión real.
 - Debido a los retardos de procesamiento de routers y retardos en cola (lo que conlleva a un retardo en comunicación del **control de flujo**).
 - El retardo de transmisión puede ser variable (dificulta el uso "timeouts").
- Se necesita control de flujo a nivel de capa de transporte debido a:
 - La **aplicación** receptora no puede mantener la tasa de recepción de datos.
 - La entidad de **transporte** receptora no puede mantener dicha tasa de recepción de datos.
- Resulta en buffers que rebalsan en la capa de transporte.

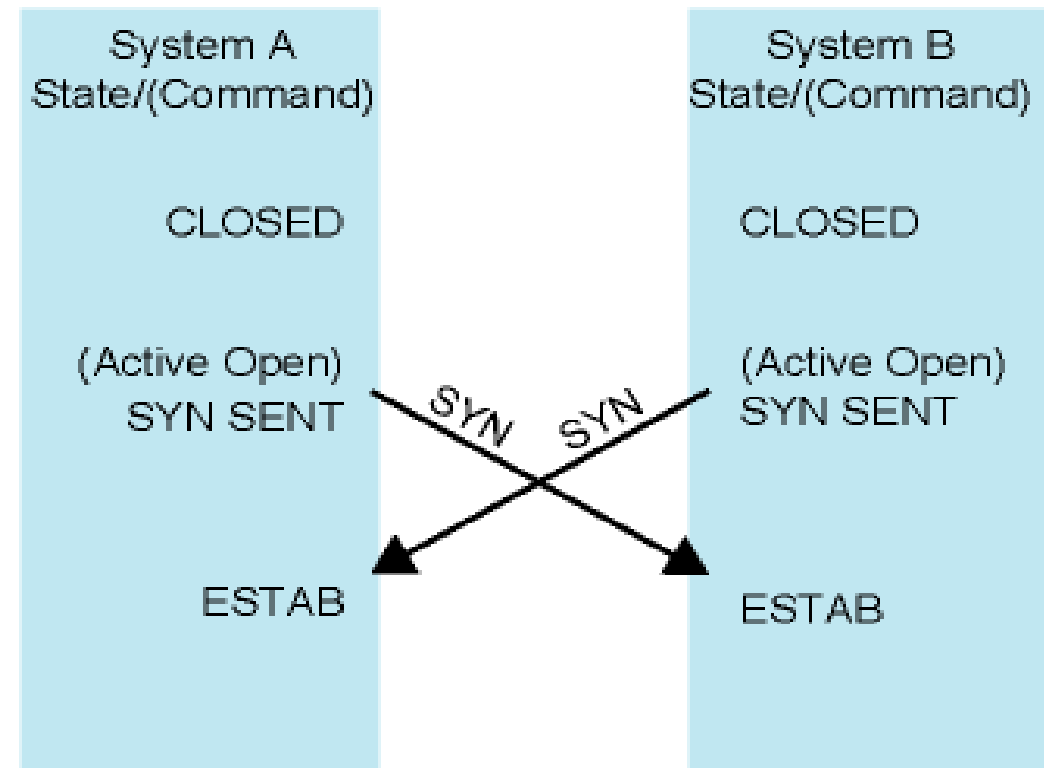
Establecimiento y Terminación de la Conexión

- TCP es un protocolo orientado a Conexión
- Una conexión TCP cumple con tres objetivos:
 - Permite a cada extremo saber que el otro existe.
 - Permite el establecimiento de parámetros iniciales (ISN, Tamaño de ventana, ...) y negociación de ciertos parámetros opcionales (tamaño máximo del segmento, factor de escala de ventana, etc.)
 - Inicializa la asignación de recursos de la entidad de transporte (buffers, entradas en tabla de conexión, etc.)
- El establecimiento de la conexión es por mutuo acuerdo entre los sistemas finales.
- Para **estudiar** el problema, se asume que el servicio de red es **confiable (lo cual obviamente NO ocurre en la realidad)**.
 - Se tiene el diagrama de estado mostrado en la próxima transparencia.
- Luego se estudiará el **caso real** en una red IP no-confiable

Escenarios de Establecimientos de Conexión

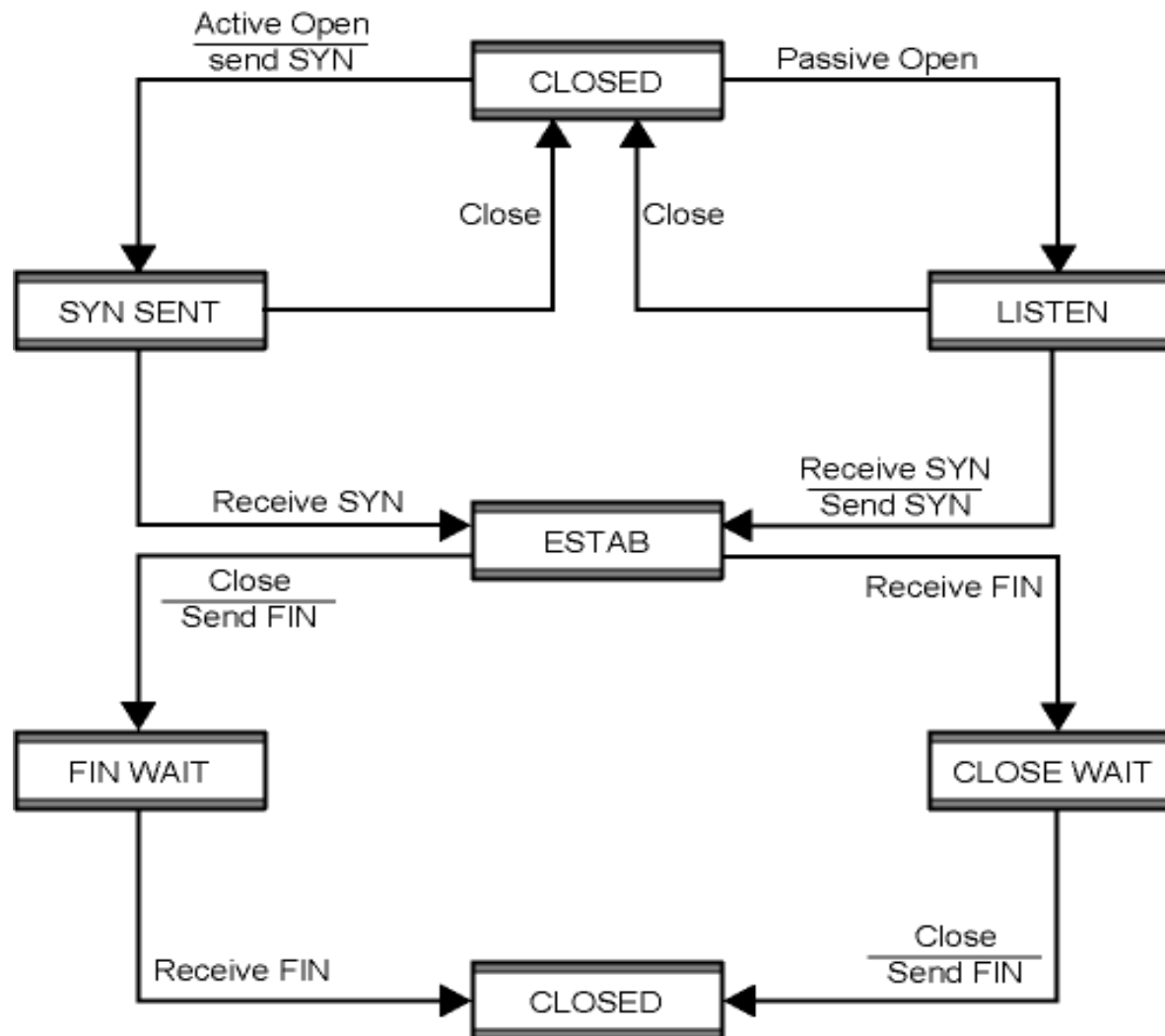


(a) Active/Passive Open



(b) Active/Active Open

Diagrama de Estado de Establecimiento y Terminación de Conexión (IP Confiable)



Terminación de la Conexión (IP Confiable)



- Puede ser iniciada por uno o ambos extremos
- La conexión se cierra por mutuo acuerdo
- La terminación puede ser:
 - Abrupta
 - Se pueden perder datos en tránsito
 - Agradable ("graceful")
 - Un extremo que pasa al estado **FIN WAIT**, debe continuar aceptando datos hasta que llegue segmento **FIN**.

Terminación Agradable



- Extremo que **inicia** la terminación:
 - Aplicación solicita un **Close**
 - TCP envía segmento con **FIN**
 - Conexión pasa al estado **FIN WAIT**
 - Continúa aceptando datos del otro extremo y entregando datos a la aplicación.
 - No envía más datos al otro extremo
- Cuando se recibe el segmento **FIN** (del otro extremo), se informa a la aplicación y se cierra la conexión

Terminación Agradable



- Extremo que **no** inicia la terminación
 - Segmento **FIN** recibido
 - Informa a Aplicación y pone la conexión en estado **CLOSE WAIT**
 - Continúa aceptando datos de la aplicación (local) y transmitiéndola
 - Cuando la aplicación ejecuta la primitiva **CLOSE**, **TCP** envía segmento **FIN**
 - Se cierra la Conexión
- Este esquema asegura que ambos extremos recibieron todos los datos pendientes y que acuerdan terminar la conexión.

Conexión TCP



- En redes **no-confiables**, un handshake de dos vías puede llevar a pérdidas o retardos de segmentos **SYN**
 - A envía **SYN**, B responde con **SYN**
 - **SYN** perdido (A o B) manejado por timer de retransmisión –SYN (**Retransmit-SYN** Timer: A o B reenvía en SYN si este timer expira)
 - Puede llevar a **SYNs** duplicados, los cuales deben ser ignorados una vez que se estableció la conexión
- La pérdida o retardo de segmentos de **datos** puede causar problemas en la transferencia de datos.
 - Ver escenario en próxima transparencia.
 - Solución: comenzar los números de segmento con valores **lejanos** al del último valor SN de la conexión previa y además:
 - Se deberá usar **SYN i**
 - El ACK debe incluir a **i**
- Existen otros problemas (no cubiertos por el momento)
- Todos estos problemas se resuelven con un handshake de 3 vías.

Handshake 2

Vías con Segmento Datos Obsoleto

- Asuma que con cada inicio de conexión se establece $SN = 1$.
- Se observa como el segmento $SN = 401$ de la comunicación antigua se puede confundir con la nueva conexión.
- Si el número de secuencia de la nueva conexión se elige lejos del rango de los SN de la última conexión, este problema se elimina.
- Es necesario crear conexiones con distintos SN y además no utilizar $SN = 1$ para cada nueva conexión.
- Además se debería **confirmar** dicho número de secuencia

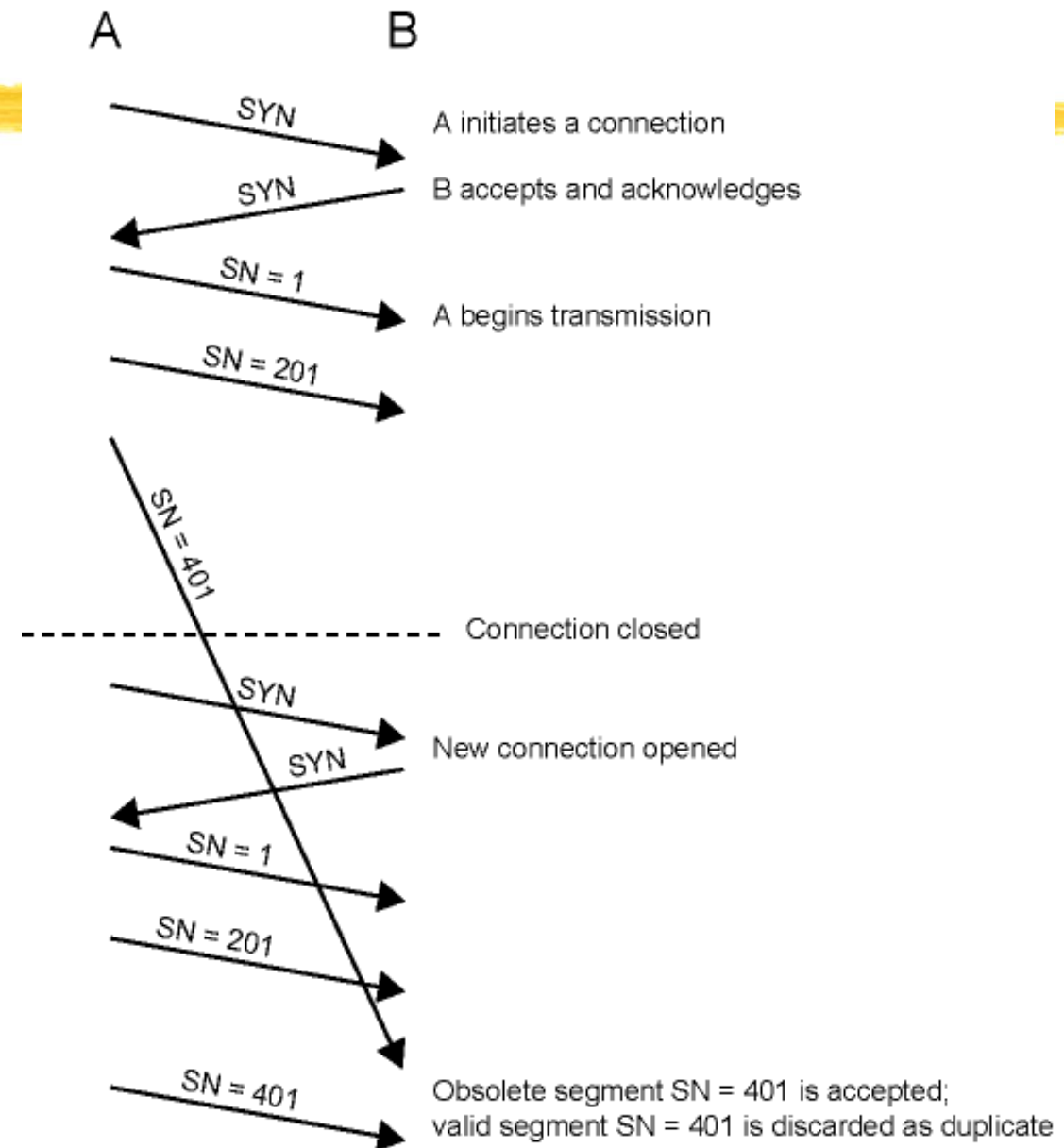
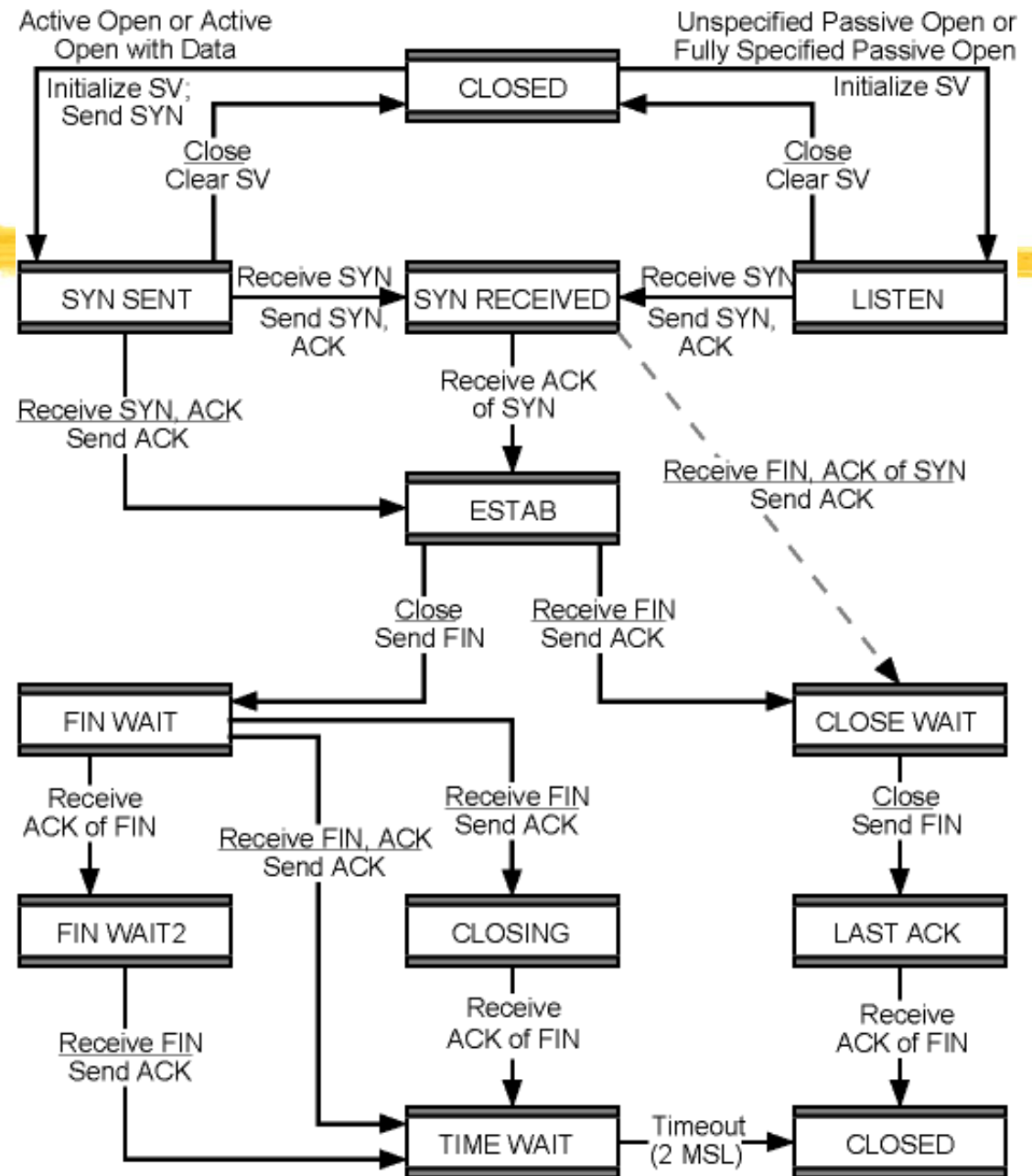


Diagrama de Estado TCP - Handshake 3 vías.

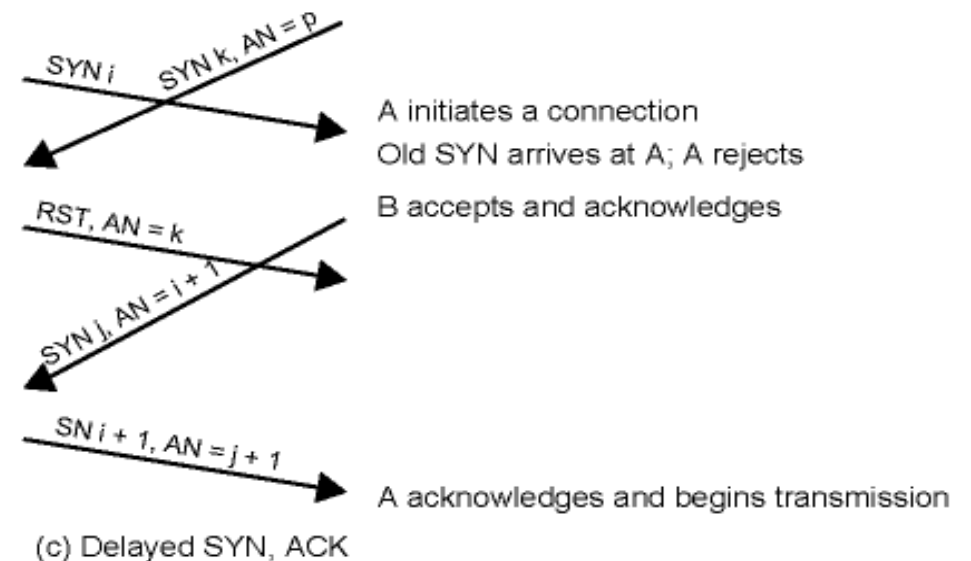
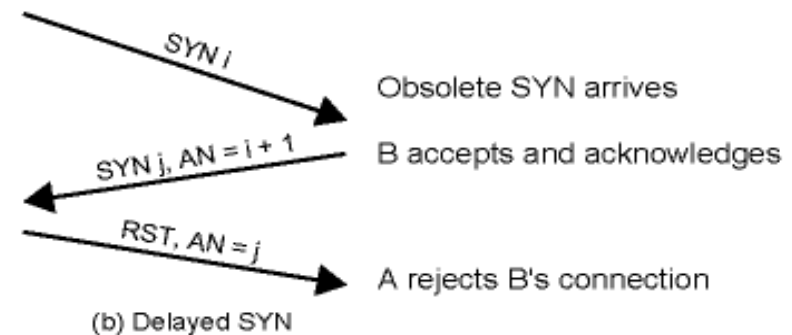
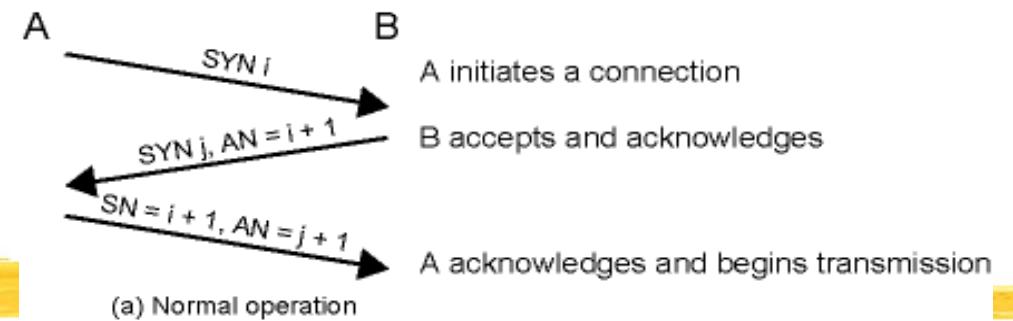
- Establecimiento de Conexión: Parte superior del diagrama
- Se introduce un nuevo estado: **SYN RECEIVED**.
- Estando en este estado y al recibir ACK de ambos SYN se establece la conexión.
- El esquema permite la conexión simultánea entre ambas entidades (open activos enviados por ambas entidades).



SV = state vector
MSL = maximum segment lifetime

Ejemplos de Handshake de 3 vías

- (a) A inicia la conexión enviando **SYN** y **SN=i**. B responde con **AN=i+1** y envía su propio **SYN=j**. A confirma y termina la etapa de establecimiento de la conexión.
- (b) y (c) muestra como se comporta en protocolo de conexión en dos casos particular de segmentos retardados.
 - (b) SYN obsoleto (de una conexión ya cerrada)
 - (c) SYN/ACK antiguo llega a A en el medio de una nueva conexión.



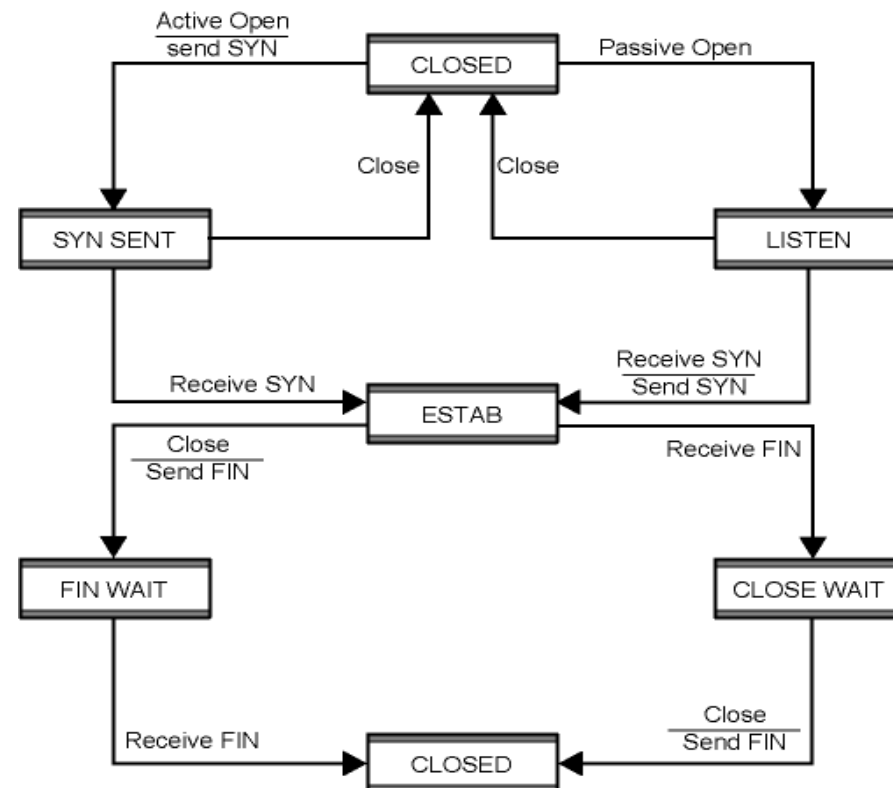
Intercambio de Parámetros en Inicio de Conexión



- Los siguientes parámetros se establecen al inicio de una conexión:
 - Número de Secuencia (ambos sentidos)
 - MSS ("Maximum Segment Size") (536 Bytes Default)
 - Corresponde solo al PAYLOAD TCP
 - Tamaño de la Ventana (ambos sentidos)
 - Factor de Escala de Ventana
 - SACK
 - Método de cómputo de Checksum alternativo
- Los últimos tres parámetros fueron definidos en RFC posteriores al original (y son **opcionales**)

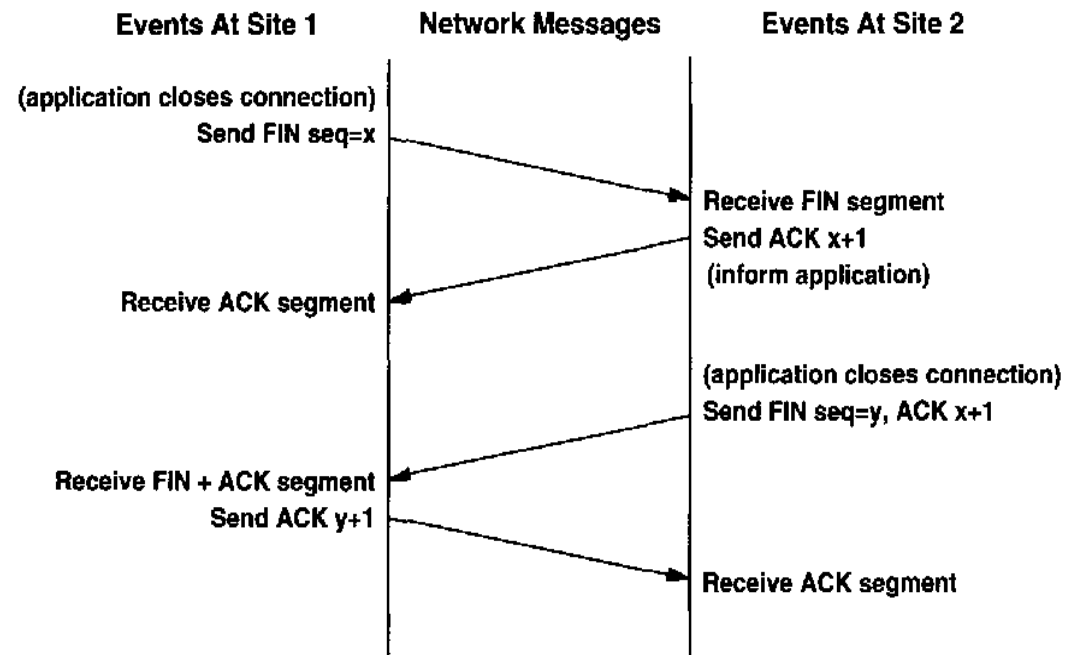
Terminación de la Conexión TCP

- Es necesario un doble handshake de dos vías (para cerrar la conexión full duplex)
- Ejemplo:
 - TCP en estado **CLOSE WAIT** envía el último segmento de datos, seguido por segmento **FIN**
 - Segmento **FIN** arriba **antes** que el último segmento de datos
 - Receptor acepta **FIN**
 - Cierra la conexión
 - Pierde el segmento de datos!
- Solución: Se debe asociar el número de secuencia con **FIN**
- Receptor espera por todos los segmentos antes del número de secuencia del segmento **FIN**
- Problema más serio con pérdidas de segmentos o presencia de segmentos obsoletos



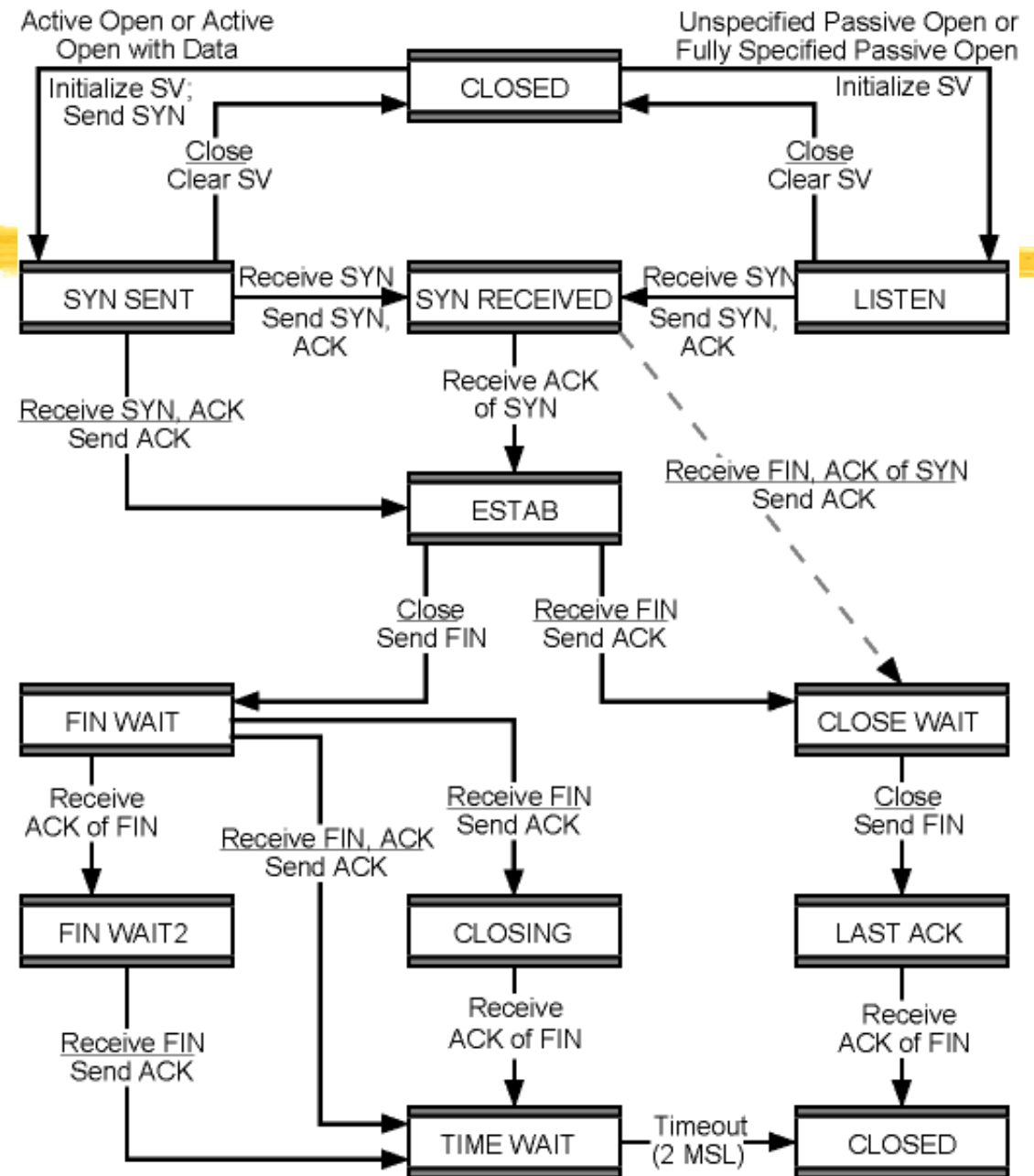
Terminación de Comunicación

- Para una terminación amistosa:
 - Cada extremo debe confirmar el segmento FIN del otro lado, usando un ACK con el SN del FIN recibido.
 - Aparecen 4 segmentos TCP en el cierre de conexión.
 - El extremo que inicia el cierre de la conexión debe enviar **FIN i**
 - El otro extremo envía un **AN = i + 1**
 - Luego (cuando la aplicación informa a TCP que se cierre la conexión), se envía un **FIN j** (con **AN = i + 1 nuevamente**)
 - Se recibe el **FIN j** y se envía un **ACK = j + 1** terminando la conexión.



Cierre de Conexión TCP

- Incluye el cierre simultáneo y también incluye el cierre con solo 3 mensajes TCP (no 4 como en la transparencia anterior)



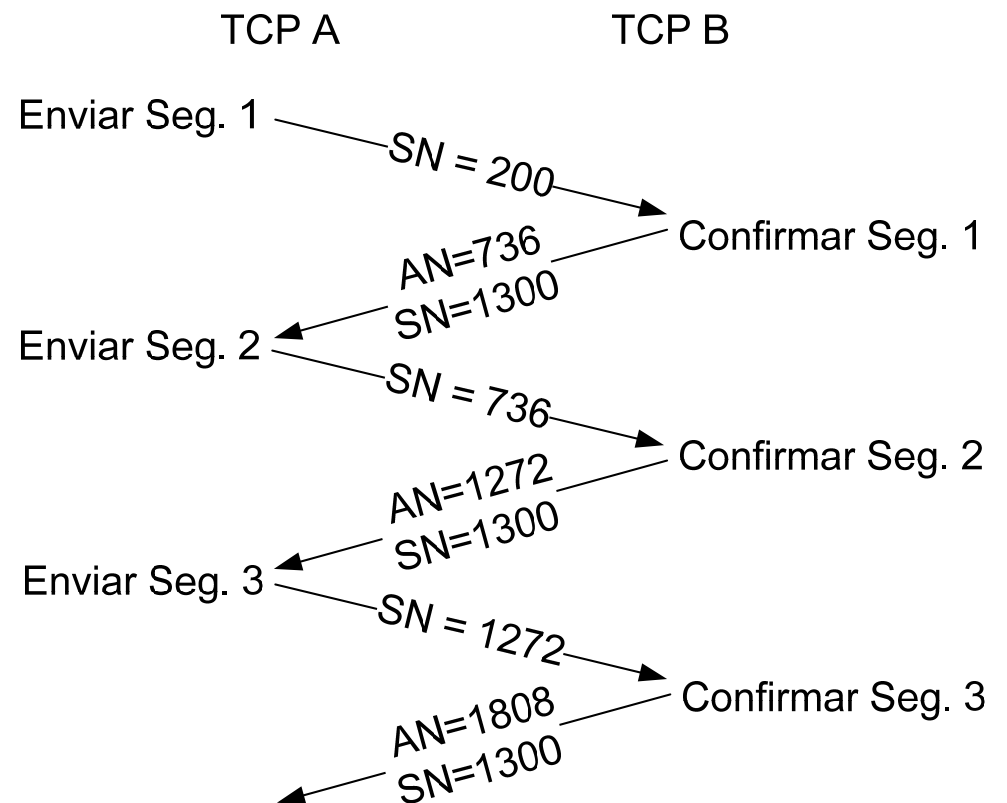
SV = state vector
MSL = maximum segment lifetime

TCP Keepalive Timer

- TCP provee un mecanismo (**no** estándar) para ver si una conexión trafica datos o se encuentra en estado *idle*.
- Cada cierto tiempo ("keepalive timer") se envía un segmento de datos vacío.
 - Si se responde con un ACK, se mantiene la conexión
 - Caso contrario se envía un mensaje de RST y se cierra la conexión.
- Su uso es controvertido
 - Algunos sostienen que estos mensajes ocupan ancho de banda y son innecesarios.
 - Otros que pueden liberarse recursos consumidos por sesiones TCP abiertas de un solo extremo.
- Al no estar especificado como parte del estándar, algunos dispositivos lo implementan y otros no.

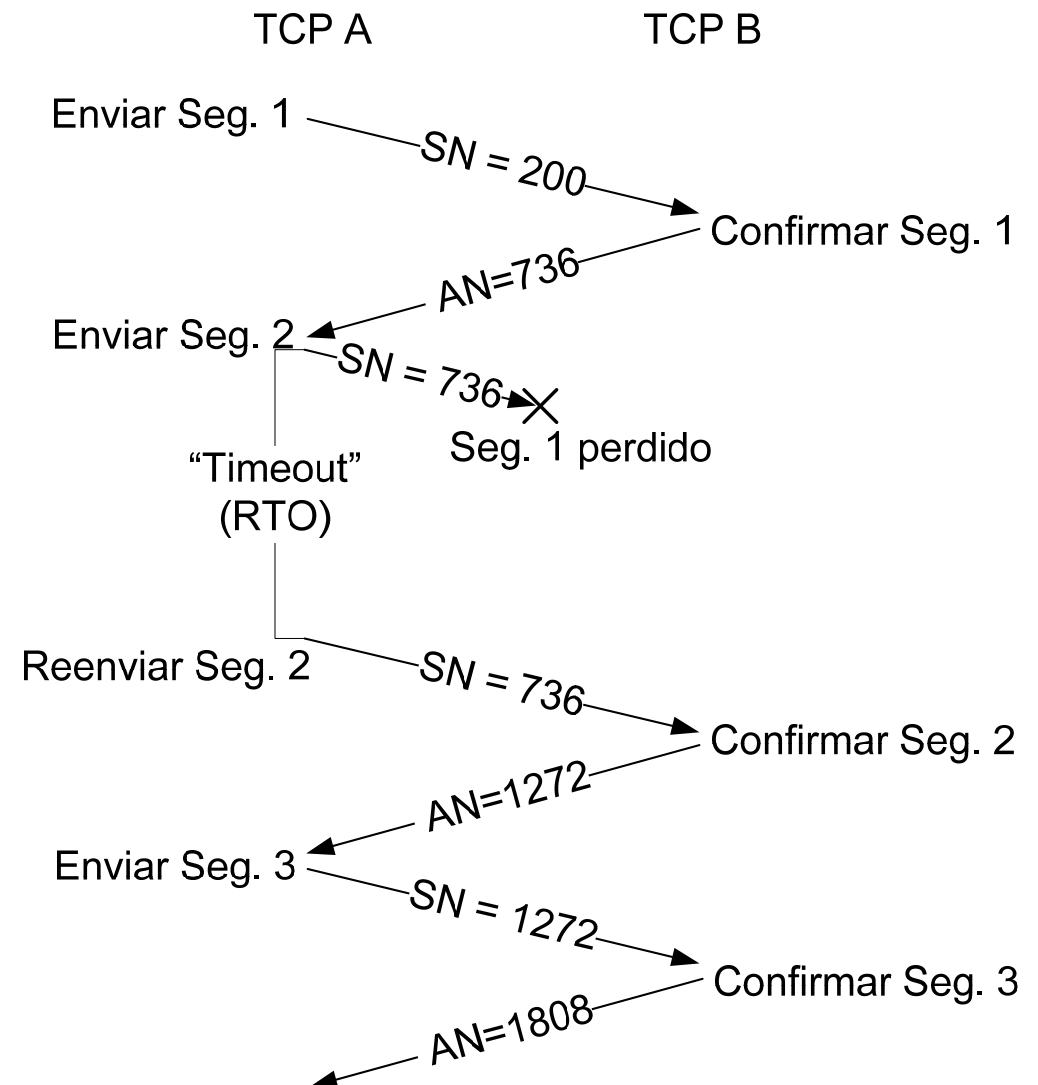
Transmisión de Datos en TCP

- Una vez establecida la conexión se conocen los ISN y el MSS.
- Sea:
 - ISN (A->B): 200
 - ISN (B->A): 1300
 - MSS (A->B): 536 Bytes
 - Flujo de A a B solamente
- RTT: Round Trip Time

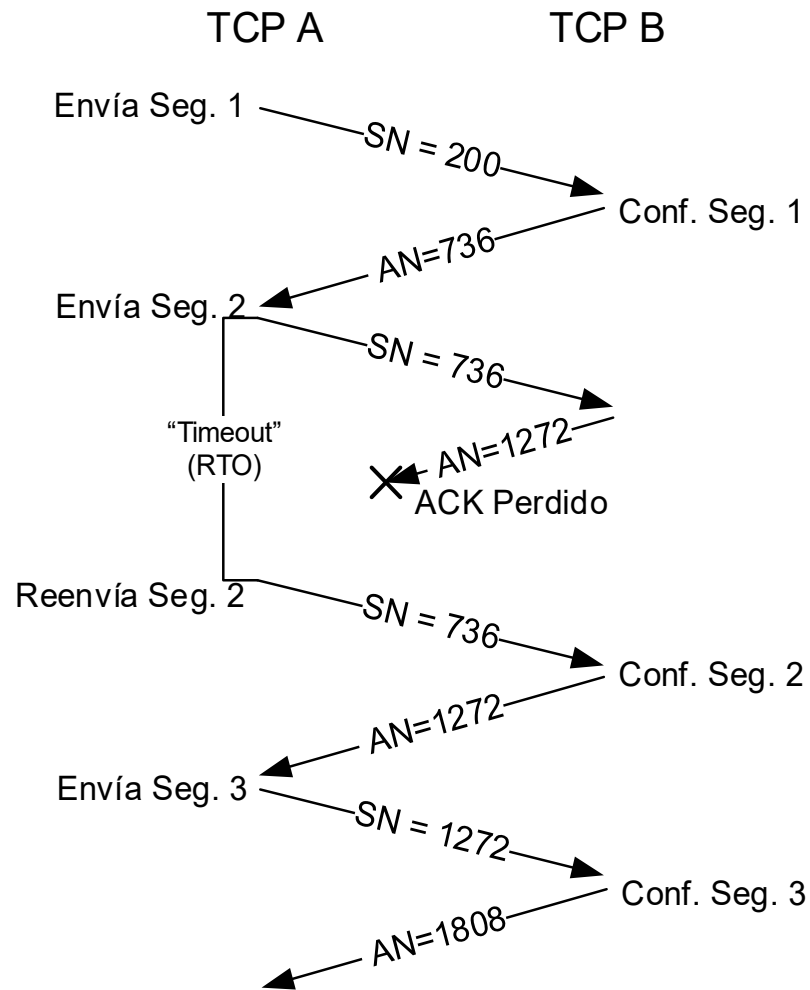


Transmisión de Datos con Error

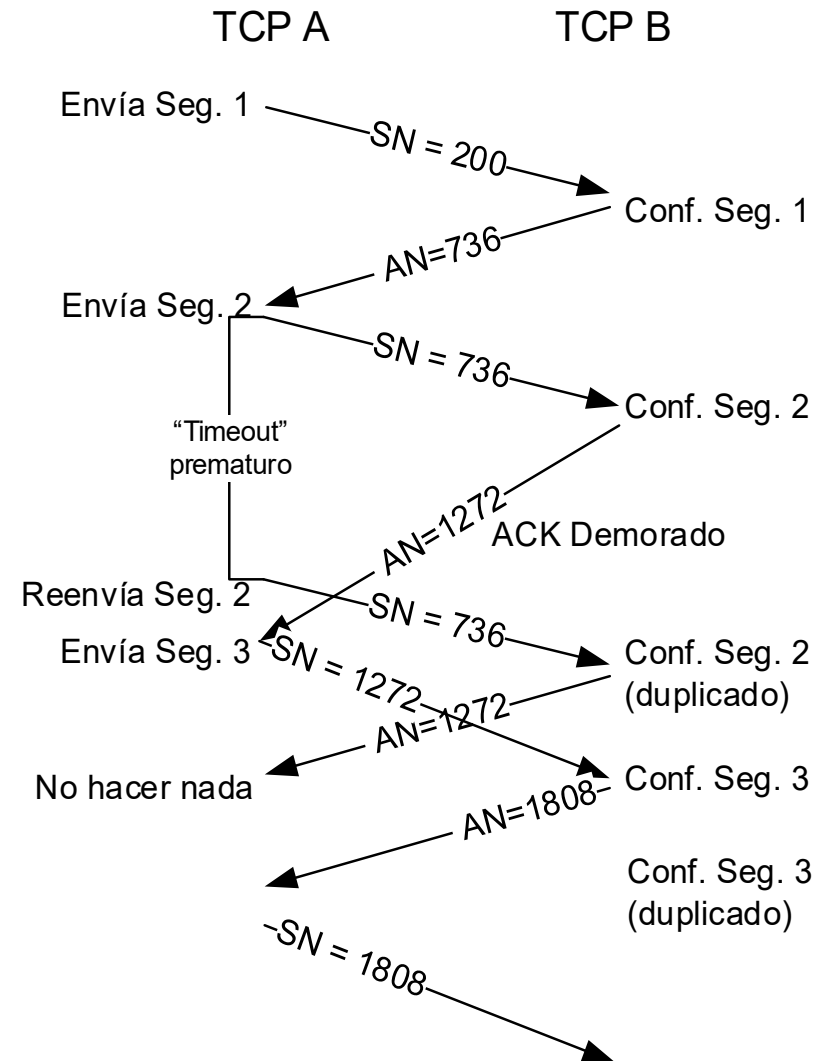
- Igual escenario pero el segmento 2 no llega a destino (o llega con errores).
- RTO ("Retransmission time out") > RTT.
- Estrategia de Retransmisión: PAR (Positive Ack with Retransmission)



Otros escenarios posibles.



(a)



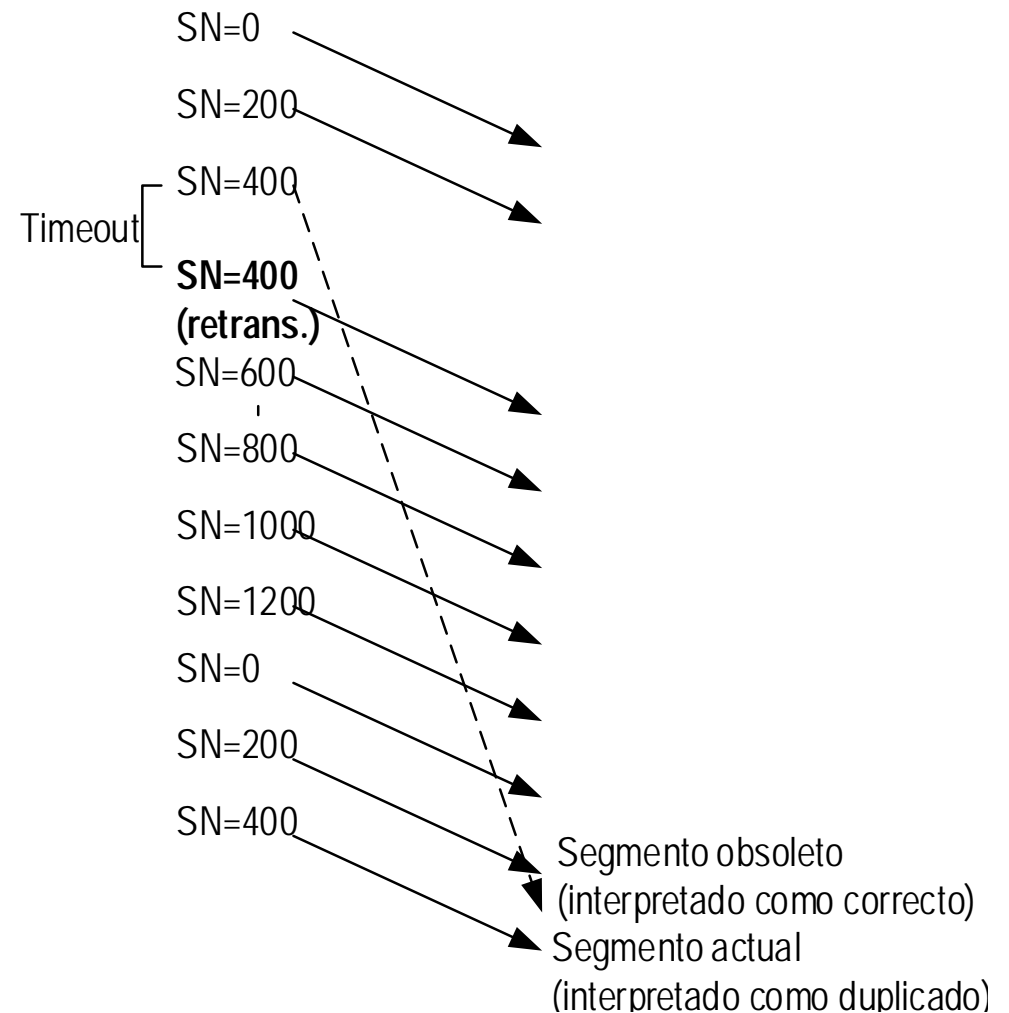
(b)

Observaciones PAR o Stop & Wait

- Es necesario usar número de secuencia.
 - Debido a la naturaleza orientada a datagrama de IP puede haber ambigüedades
- Detección de errores en segmento transmitido y tb. en ACK
- Al ser TCP Full-Duplex, el ACK puede transportar datos.
- Desperdicio del ancho de banda del canal!
 - Retardo de Propagación >> Tiempo de Transmisión
- ¿¿¿Rango de Número de Secuencia???

Espacio de SN

- Suficientemente largo para no presentar un ciclo en un tiempo menor que el MSL (Max. Segm. Lifetime)
- Sino pueden aparecer errores de datos no detectables por TCP.
- Ejemplo:
 - SN = 1400 (0 ... 1399)
 - MSS = 200
- Rango de Números de Secuencia:
 - 2^{32} (0 ... $2^{32} - 1$)
 - 4 Giga



Estrategias de Control de Flujo



- PAR no permite más de un segmento en tránsito.
- En necesario una estrategia de envío de varios segmentos simultáneamente -> aumenta la utilización del canal y además la recepción de mayor cantidad de bytes por parte de la aplicación.
- ¿Qué hacer con el control de flujo? Se puede reaccionar de tres modos:
 - No realizar nada
 - Segmentos que provocan sobreflujo, son descartados
 - TCP origen no obtendrá ACK y retransmitirá
 - Este tráfico de retransmisión se agrega al tráfico de transmisión.
 - “Backpressure” a nivel de Red
 - En situaciones de overflow de buffers de transporte, rechazar datos adicionales de capa de **red**.
 - Activa mecanismos de control de congestión a nivel de capa de red (en trasmisor)
 - Este a su vez rechaza nuevo segmentos TCP en trasmisor.
 - Utilizar esquema de crédito.
 - Emplea número de secuencias en segmentos, una ventana de segmentos en tránsito y el uso de ACKs para ajustar el tamaño de la ventana.

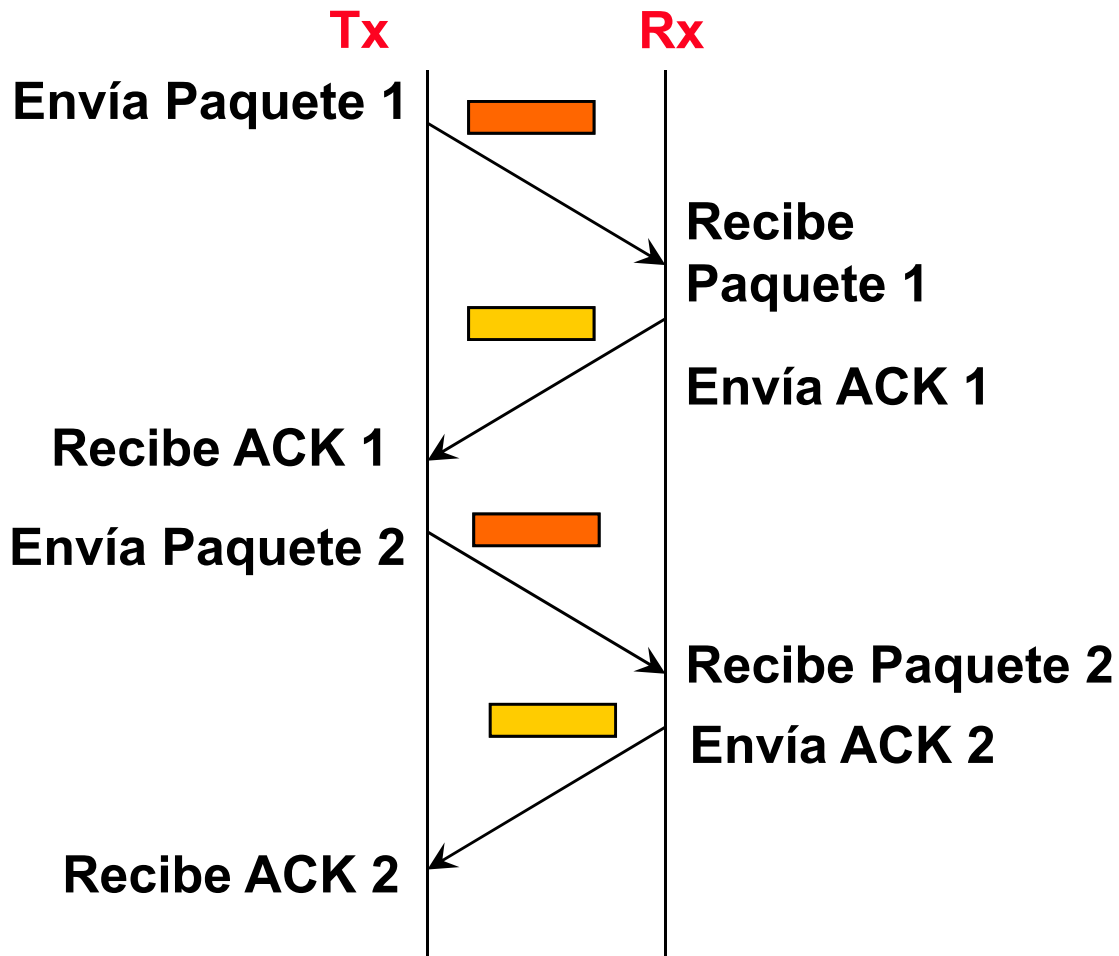
Esquema de Crédito



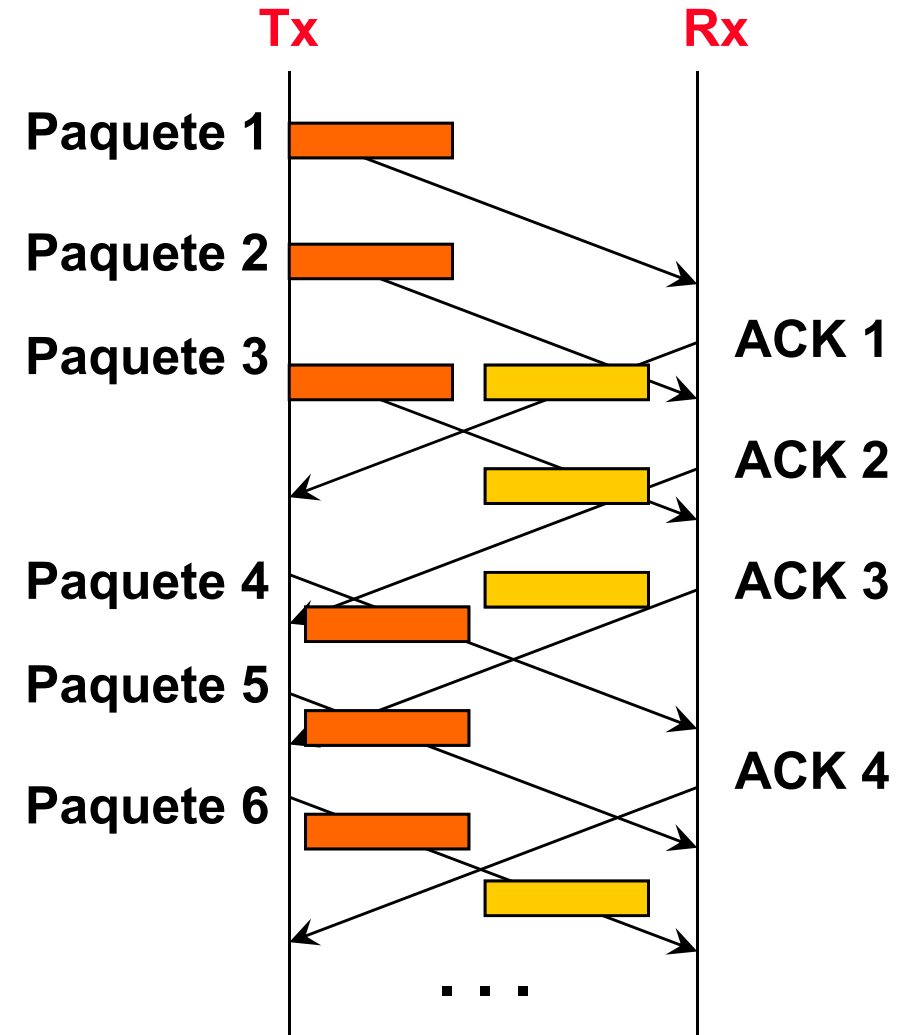
- Utilizado por TCP
- Mejora las ineficiencias provocadas por el mecanismo de "Stop and Wait" (S&W)
 - Con S&W solo un PDU en tránsito entre Tx y Rx.
 - Si el tiempo de propagación (inherente al vínculo físico de comunicación) es mayor que el tiempo de transmisión (inherente al "data rate" o ancho de banda del canal), S&W es ineficiente.
 - Lo ideal sería tener varios segmentos en tránsito e **inundar** el canal de datos.
- Esto es lo que hace "Sliding Window"
- Para ello, cada segmento de transporte, tiene un número de segmento, un número de ACK y un tamaño de ventana.

Protocolo de Ventana Deslizante ("Sliding Windows")

Stop & Wait (PAR)



Sliding Windows (W=3)



Campos de Header TCP

- Cada segmento transmitido incluye tres campos relacionados al Control de Flujo:
 - SN ("Sequence Number"): Cuando se envía un segmento, el número de secuencia corresponde al primer Byte del segmento.
 - AN ("Ack Number"): Número de bytes recibidos correctamente.
 - **W ("Window"): Tamaño de Ventana**
- Interpretación: Si ACK incluye $AN=i$, $W=j$
 - Todos los octetos hasta $SN=i-1$ son confirmados.
 - Próximo octeto esperado es i
 - Permiso para enviar ventana adicional de $W=j$ octetos
 - Es decir el transmisor puede enviar octetos correspondientes a i hasta $i+j-1$

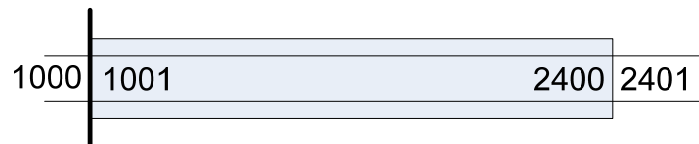
Ejemplo



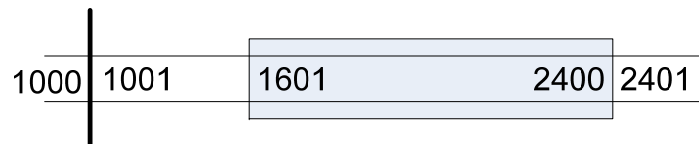
- Suponga el siguiente escenario:
 - TCP de host A envía segmentos a B (solo en esa dirección existe la transferencia de datos)
 - A obtiene como número inicial de secuencia a 1001 (a través del establecimiento de la conexión).
 - El tamaño inicial de la ventana es de 1400 octetos
 - Es decir A puede enviar 1400 octetos, sin recibir ACK de B.
 - MSS = 200 Bytes
- La secuencia que se muestra en la próxima transparencia muestra el control de flujo ejercido por el protocolo de ventanas deslizantes.

Ejemplo...

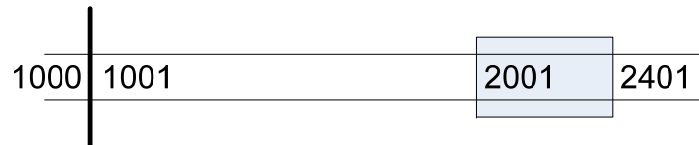
TCP Transmisor A



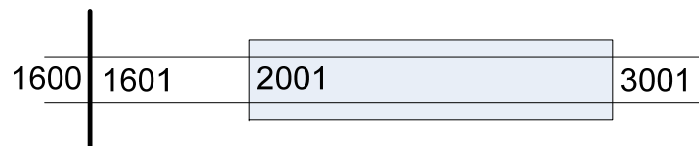
A puede enviar 1400B



A encoge su ventana transmisora con cada transmisión

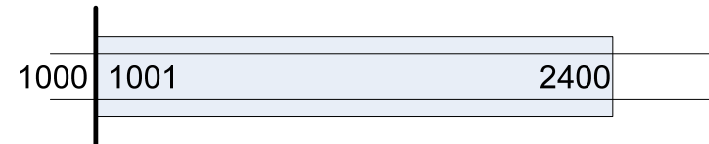


A encoge nuevamente su ventana transmisora. Crédito actual: 400B

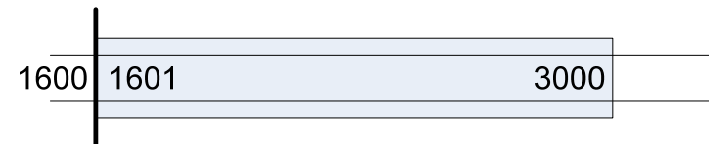


A aumenta su ventana al llegar un nuevo crédito (ACK de B). Crédito actual: 1000B

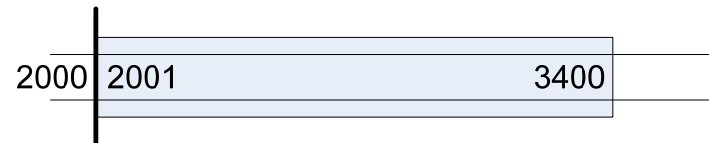
TCP Receptor B



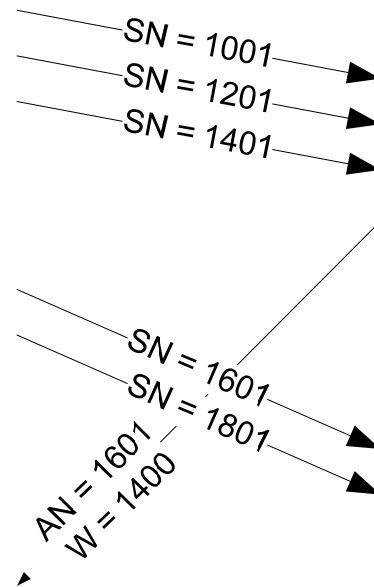
B se prepara para recibir 1400B



B confirma 3 seg. (600B). Se prepara para recibir 1400B adicionales.



B confirma 2 seg. y se prepara para recibir 1400B adicionales.

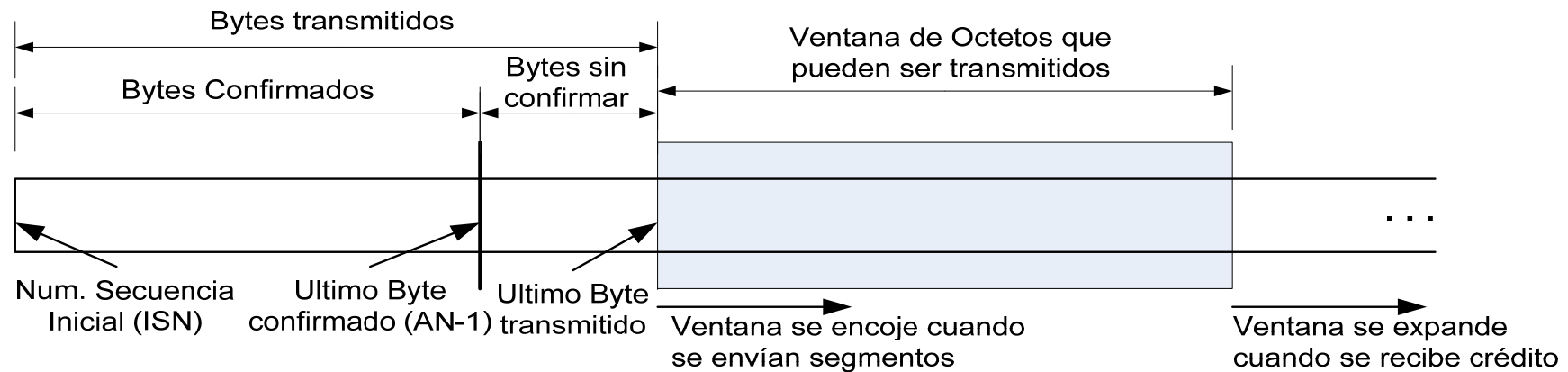


Observaciones

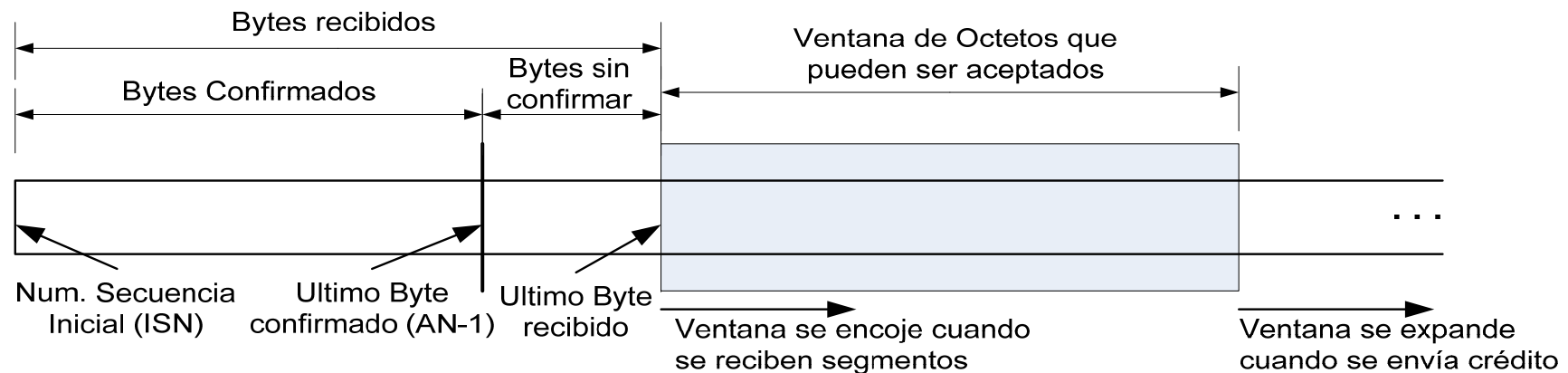


- El tamaño de la ventana se mide en bytes, no en segmentos.
 - En el ejemplo los segmentos son de 200 Bytes y el tamaño inicial de ventana es de 1400 Bytes
- El Número Inicial de Secuencia y de Tamaño de ventana se establece en el inicio de conexión como ya se vio.
- Observar como se van deslizando las ventanas de A (Transmisor) y de B (Receptor).
- Cuando la comunicación es en ambos sentidos (A -> B y B -> A), se utilizan los SN y AN en ambos extremos con su correspondiente significado.
 - Es decir en una transmisión full duplex, se mantienen ventanas en ambos extremos.
- En la transparencia siguiente se describe el significado de cada elemento de estas ventanas (TX y RX)

Ventanas Deslizantes TX y RX



TX



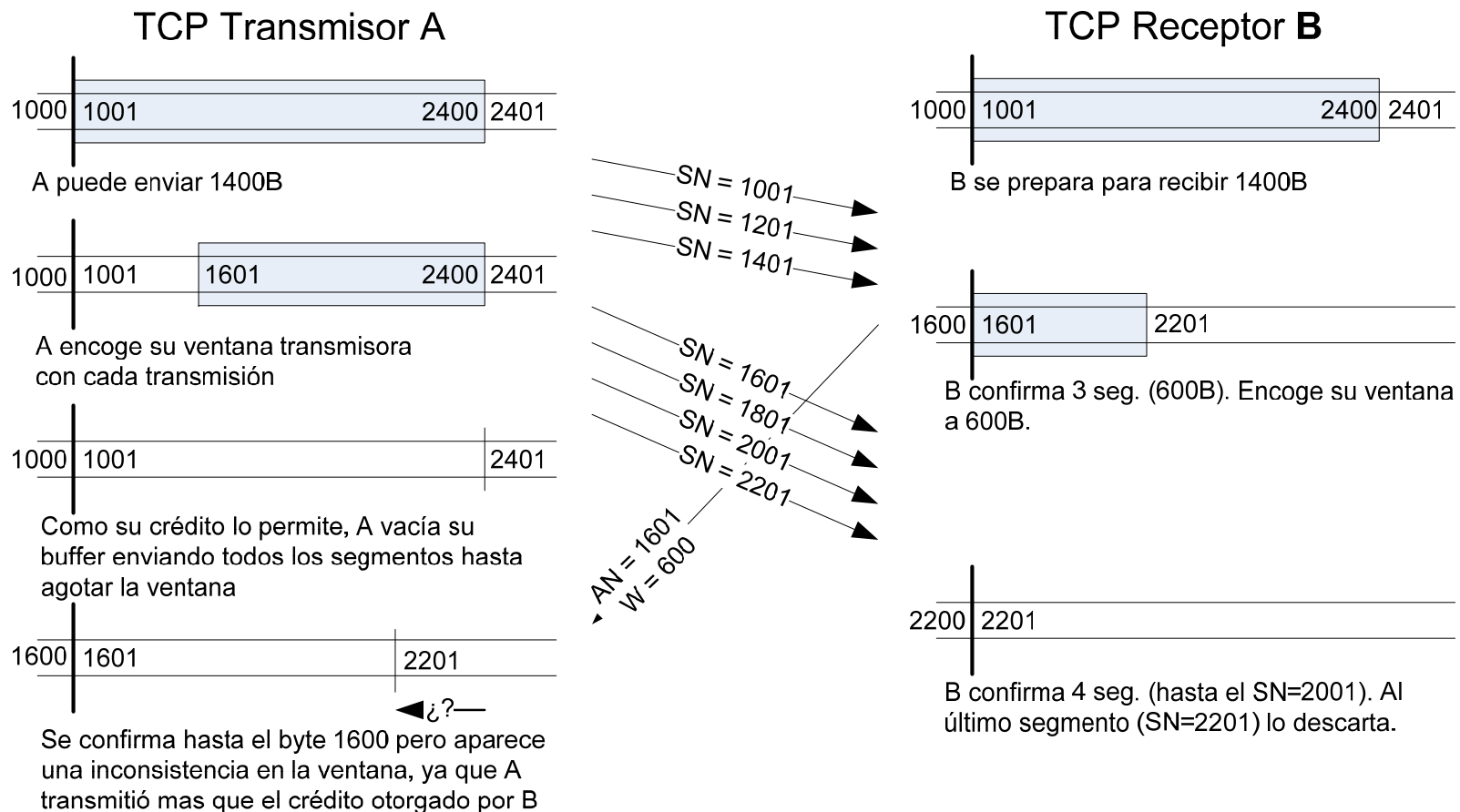
RX

Tamaño de la ventana (W)

- W se establece en ambos extremos en el momento de la conexión TCP
- W rige el control de flujo:
 - W Disminuye: Restricción de bytes en el receptor.
 - W Incrementa: Aumenta la cantidad de bytes que pueden ser recibidos en el receptor.
- Importante
 - W depende de la capacidad de consumo de bytes de los buffers de TCP por parte de la aplicación.
 - W no controla la congestión en la red de datos.
- Nota: Existe una opción que permite ampliar en 2^F el tamaño de la ventana ("Windows Scale")
 - F: Se pasa como argumento de la opción
 - Solo se envía con los segmentos con SYN = ON
 - El factor de escala permanece invariante durante una conexión

Administración de W

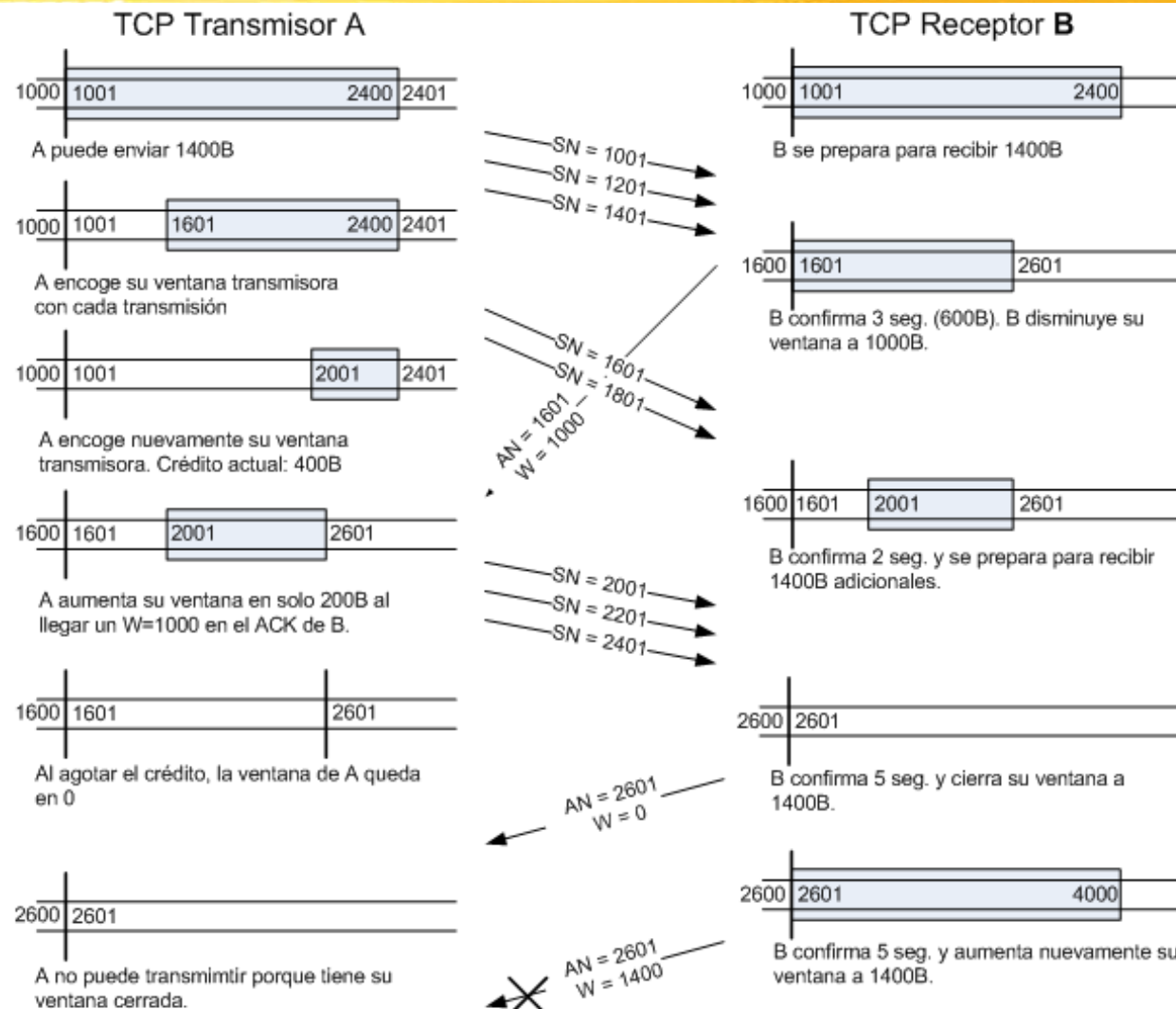
- W nunca puede disminuir de tal manera que el borde derecho de la ventana se desplace a la izquierda
- Puede llevar a segmentos rechazados por el Receptor (sin errores).



W = 0 en Transmisor

- Si $W=0$ en Transmisor, se detiene el envío de datos.
- Si se pierde el ACK de ampliación de ventana, el Tx no enviará mas datos
 - En TCP no se envían ACK de ACK por lo que el Rx asume que el Tx no manda mas datos porque no tiene para enviar.
- El transmisor puede inferir problemas en el receptor si la ventana permanece en $W=0$ y cerrar la conexión.
- Solución:
 - Transmisión de mensajes de prueba con 1 Byte de datos.
 - Se permite esta transmisión pese a que $W=0$.
 - El byte que se envía es siempre el mismo.
 - Frecuencia de envío: "Persist Timer" (5 ~ 60 seg)

W = 0 en Transmisor...



Segmento perdido. A permanece con la ventana cerrada y no puede transmitir nunca más

Silly Windows Syndrome



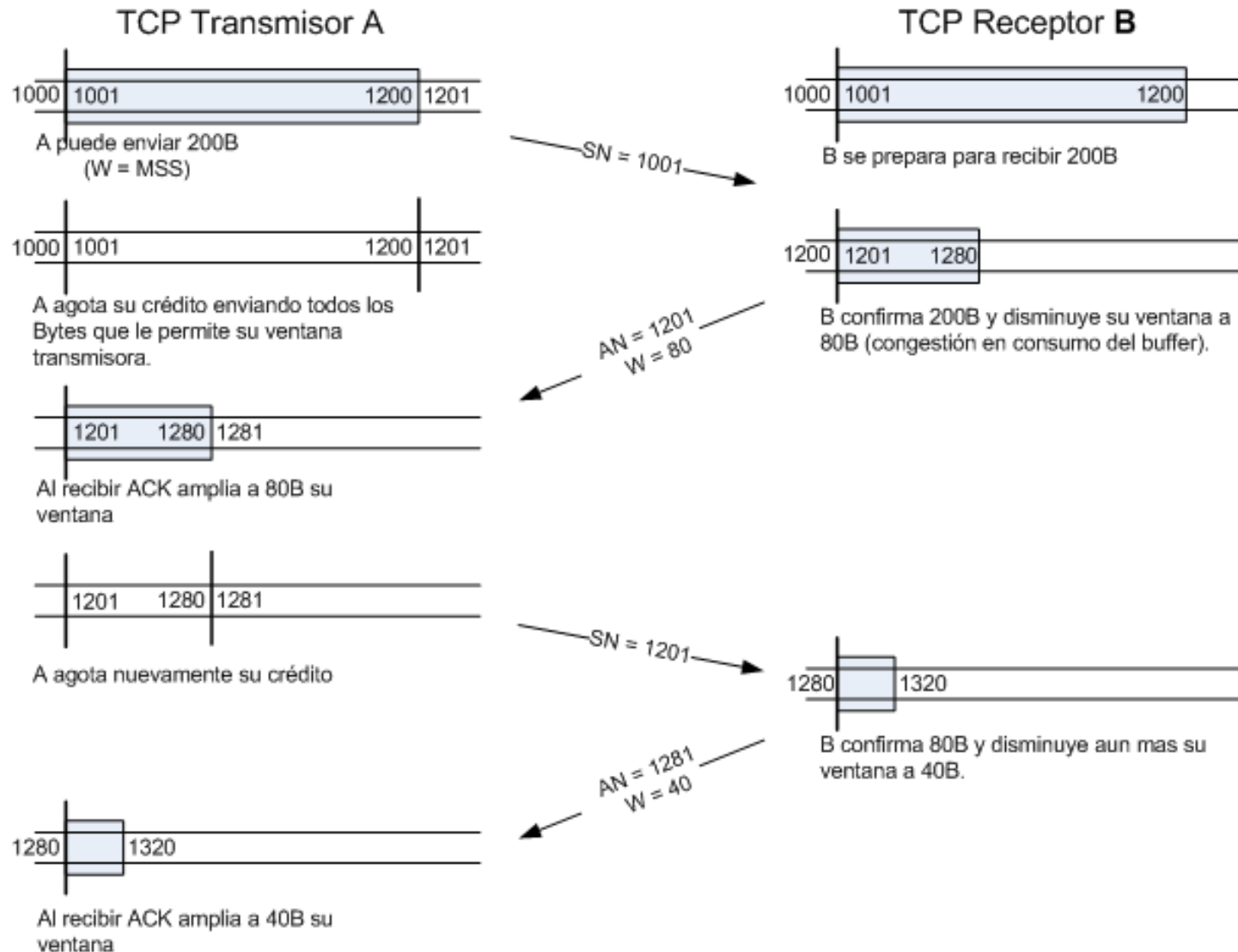
- Puede llevar a intercambio de segmentos pequeños ($<< \text{MSS}$).
- Mayor "overhead" -> Menor Rendimiento global de la red.
- Puede ser causado por:
 - **Receptor**: Avisa W de tamaño pequeño (en lugar de esperar por W mayor)
 - **Transmisor**: Puede enviar segmentos pequeños (en lugar de esperar por datos adicionales y completar el MSS).
 - Algoritmo de Nagle
- En ambos casos "Sliding Windows" trabaja correctamente!
- Para evitar el síndrome, se deben implementar políticas en ambos extremos de la conexión.

Silly Windows: Extremo Receptor



- Estación cliente (Tx) puede enviar datos en forma ininterrumpida a la servidora (Rx)
- Servidor tiene restricción de espacio de buffers
 - Cada vez que envía ACK al cliente, disminuye W.
- Cliente disminuye cada vez más la cantidad de segmentos o incluso el tamaño del segmento.
 - En el límite envía segmentos de 1 Byte.
- Ver escenario siguiente donde $W = MSS$ y el RX va disminuyendo la ventana en cada ACK.

Silly Windows: Extremo Receptor

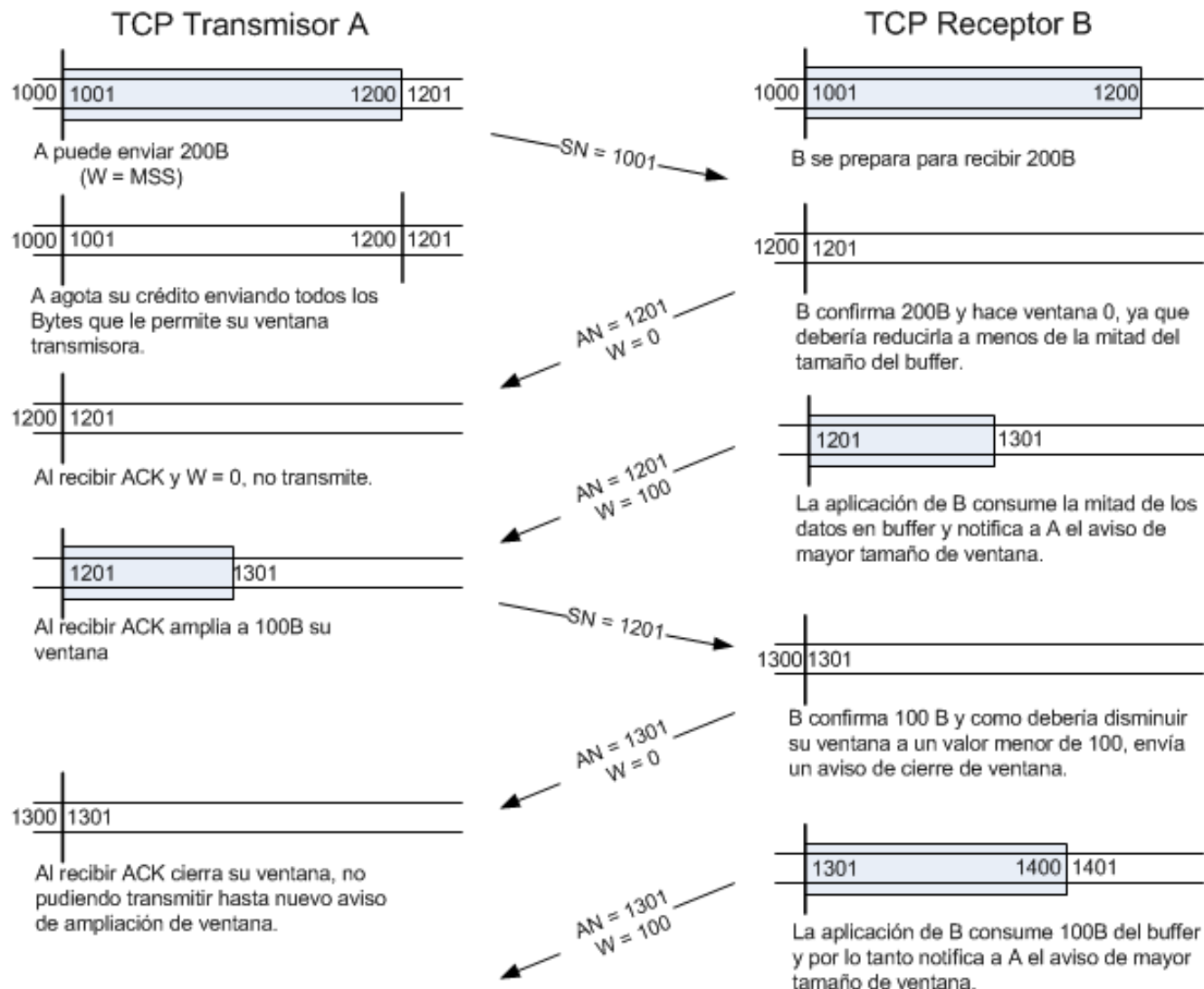


Silly Windows: Extremo Receptor

➤ Posible solución:

- Impedir al receptor que envíe avisos de ventana que permitan enviar al transmisor segmentos pequeños.
- El receptor **NO** debe enviar avisos de ventanas pequeños
- Un criterio podría ser:
 - Receptor no envía W más chico de la que está avisando (puede ser $W = 0$) hasta que W pueda aumentar a un valor igual al MSS o a la mitad del tamaño del buffer receptor (el que sea menor entre esos dos valores).
- Ver escenario siguiente en donde $MSS = \text{Tamaño del Buffer Receptor} = 200 \text{ Bytes}$ (por simplicidad)
- Se observa en el diagrama que los segmentos transmitidos no pueden nunca ser de un tamaño pequeño.

Silly Windows: Extremo Receptor



Timers en TCP



- TCP para su funcionamiento utiliza varios timers:
 - Timer de Establecimiento de la Conexión ("Retransmit SYN timer")
 - Espera respuesta a un SYN durante este timer (75 seg).
 - Para este tipo de segmentos (SYN) no se utiliza el RTO (no transporta datos el segmento).
 - Timer de Retransmisión (RTO o "Retransmission Time Out")
 - Permite la retransmisión de un segmento no confirmado.
 - Acotado al valor 1 a 64 seg
 - Timer de ACK retardado ("Delayed ACK Timer")
 - Utilizado por los ACK Acumulativos.
 - Tiempo que espera para confirmar segmentos recibidos correctamente.
 - 200 mseg
 - Timer de Inactividad ("Keepalive" timer)
 - Expirado este timer, si no se recibió ningún segmento, se envía uno (sin datos) para saber si la conexión sigue activa.
 - 2 horas

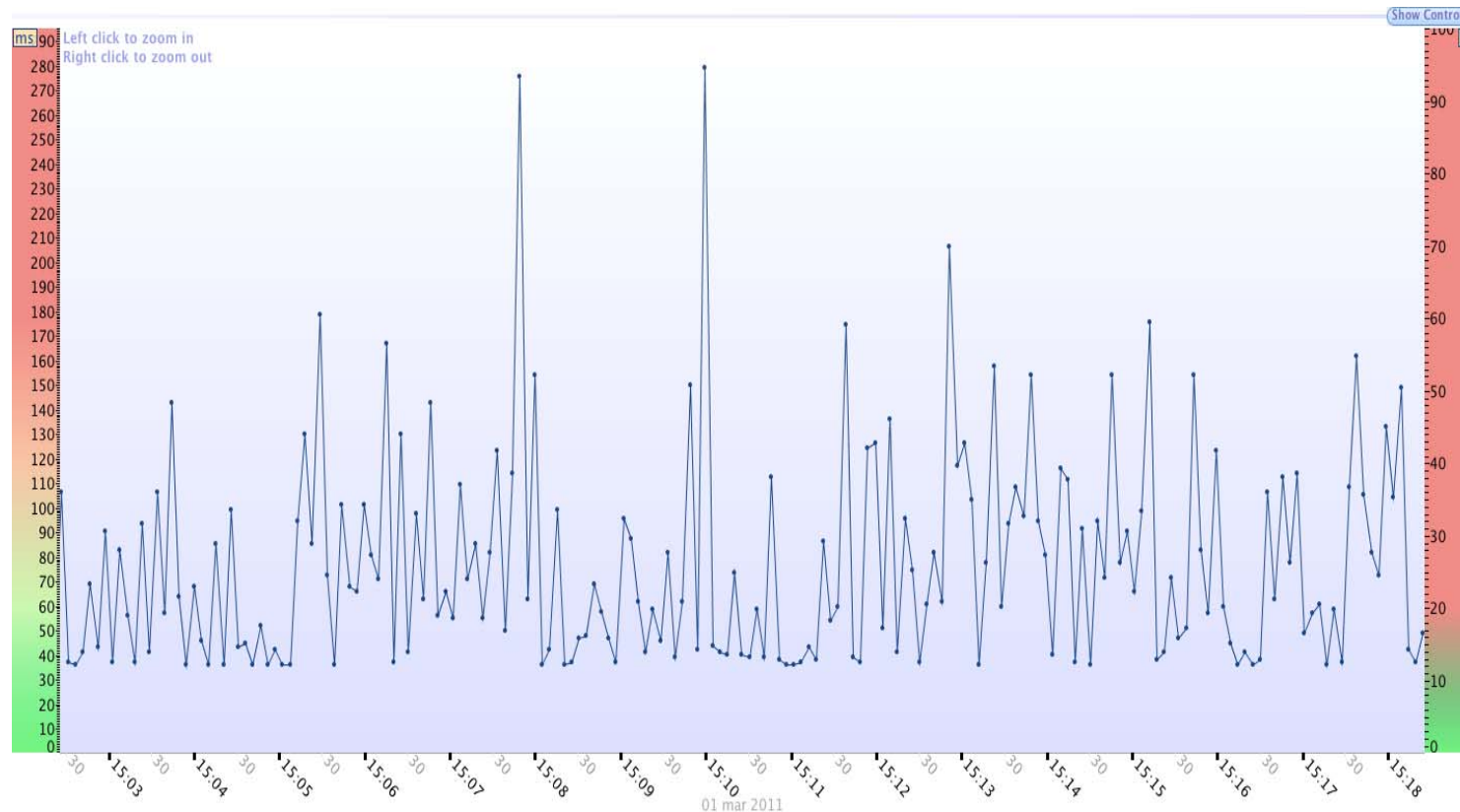
Timers en TCP



- Timer de Persistencia ("Persist timer")
 - Utilizado para caso de $W=0$
 - Extremo con ventana 0, envía cada "Persist Timer" un paquete de prueba para saber si la ventana aumentó su valor.
 - 5 a 60 seg.
- Timer FIN WAIT
 - Iniciado cuando TCP transiciona de FIN_WAIT_1 a FIN_WAIT_2
 - 10 minutos
 - Si expira el timer y no se recibe FIN, se espera 75 seg. adicionales y si no se recibe FIN se cierra la conexión.
- Timer TIME WAIT
 - Inicializado cuando se TCP entra al estado TIME_WAIT.
 - Al expirar se cierra la conexión.
 - 2 MSL
- ...

Timer de Retransmisión (RTO)

- RTT (Aplicación) medido sobre www.google.com en un intervalo de 15 minutos.
- Varianza de 280 a 32 mseg -> RTO debe ser variable!



RTO



➤ Cálculo RTO según RFC 793

➤ $RTT \leftarrow (\alpha \times RTT_{\text{anterior}}) + (1 - \alpha) \text{Nuevo_RTT}_{\text{medido}}; 0 \leq \alpha < 1$

➤ Donde:

➤ RTT: Round Trip Time

➤ $\alpha \approx 0 \rightarrow$ RTT refleja cambios bruscos en RTT

➤ $\alpha \approx 1 \rightarrow$ RTT inmune a cambios en RTT

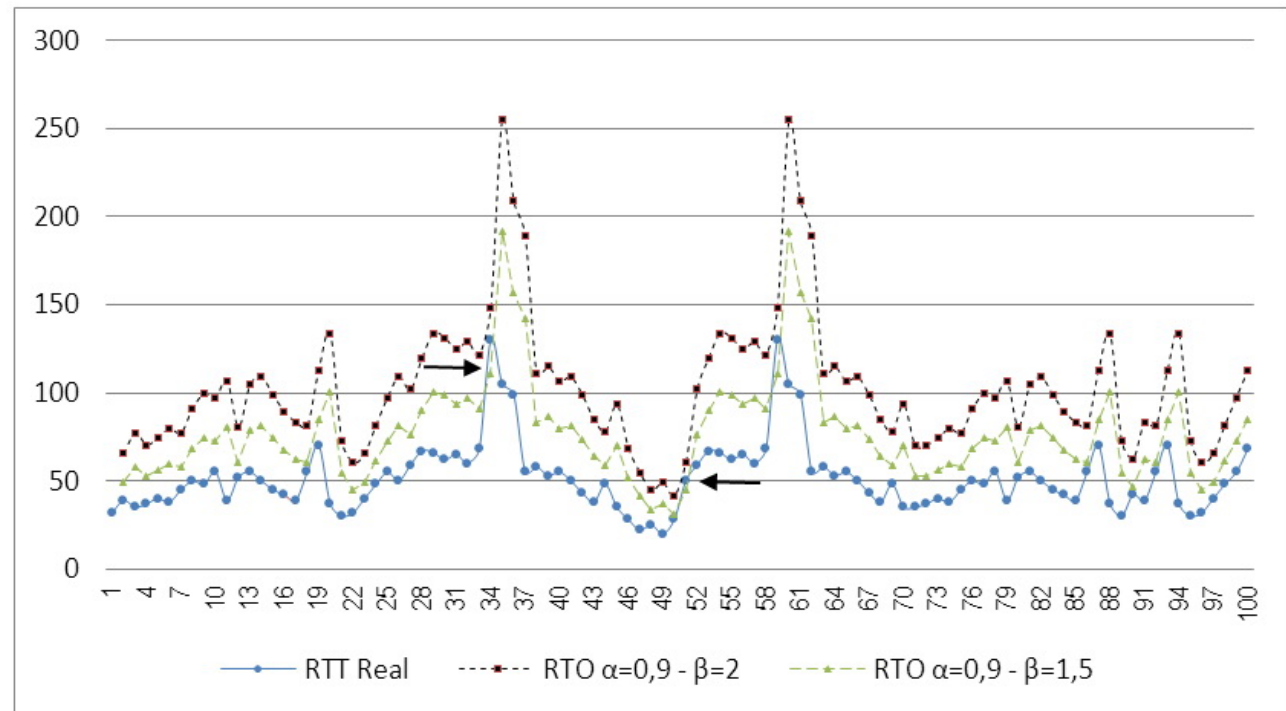
➤ $\alpha \approx 0,8$ a $0,9$ **recomendado** en RFC 793

➤ El RTO se calcula como:

➤ $RTO = \beta \times RTT; \beta > 1$

Valor de β en el RTO

- Si $\beta \approx 1$ -> Rápídamamente se detectan paquetes no confirmados y se procede a la retransmisión
- Si $\beta \gg 1$ -> TCP es más lento en las retransmisiones.
- Valor recomendado en RFC 793: β entre 1,3 y 2.
- Observaciones Gráfico:
 - $\beta = 1,5$ RTO < RTT en dos casos.
 - $\beta = 2$ RTO >> RTT



Aporte de Jacobson & Karels ('88)

- Problema del cálculo del RTO aparece cuando hay grandes fluctuaciones en el RTT
- Estas fluctuaciones pueden depender de:
 - Fluctuaciones en el tamaño del **datagrama IP** que transporta el segmento TCP (especialmente cuando el tiempo de transmisión >> retardo de propagación)
 - Congestiones transitorias en la red.
 - TCP receptora puede no reconfirmar segmentos rápidamente debido a congestión propia del host receptor.
- En el RFC 793 esas variaciones del RTT se atenúan con β ...
- Pero si la red es estable -> RTO estimado es alto!

Cálculo del RTO según Jacobson

- Jacobson introdujo el valor medio y la varianza en el cálculo del RTO:

$$Dif = RTT_{Muestra} - RTT_{anterior}$$

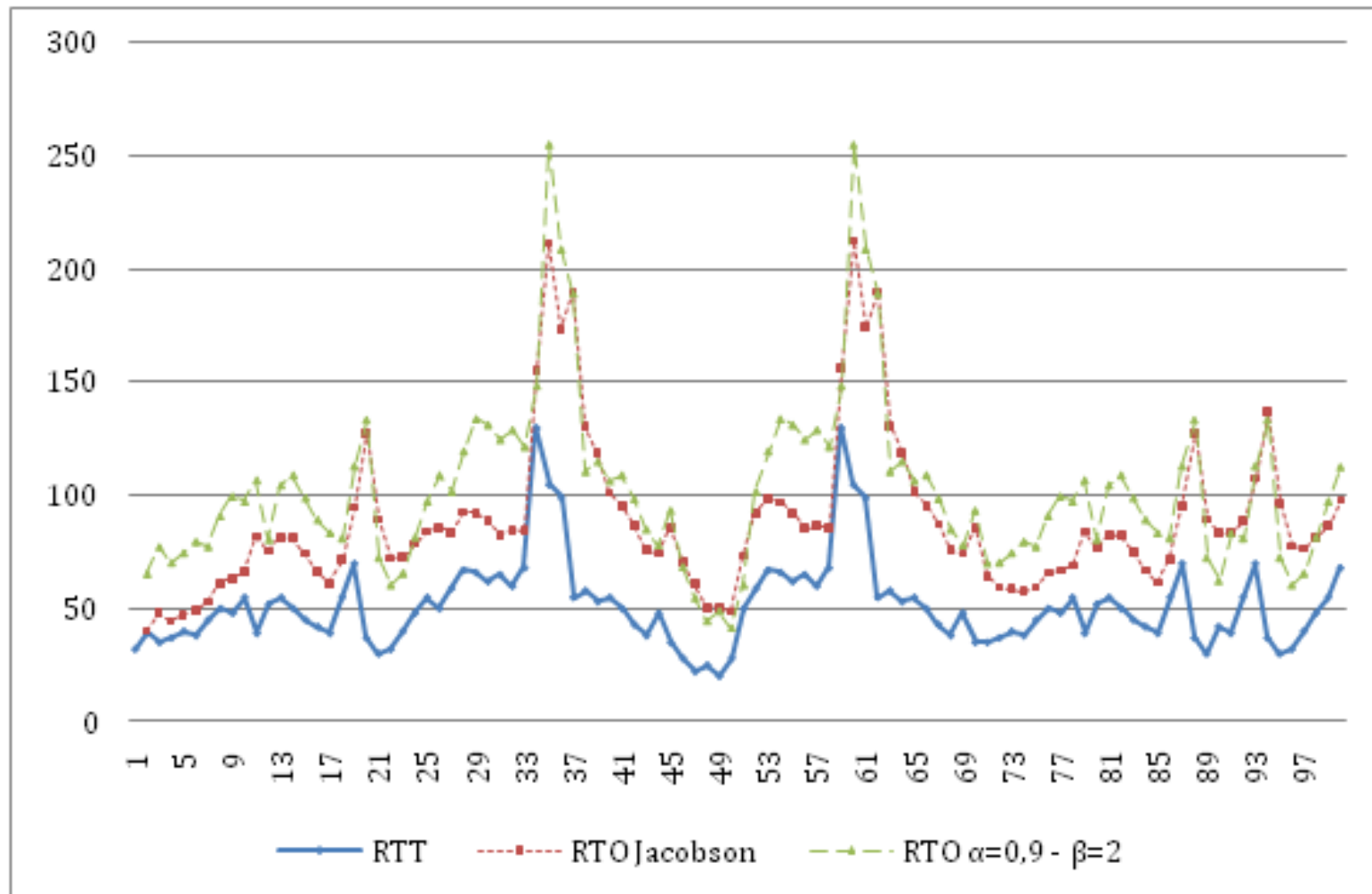
$$RTT_{Suavizado} = RTT_{anterior} + \delta \times Dif$$

$$Dev = Dev_{anterior} + \rho \times (|Dif| - Dev_{anterior})$$

$$RTO = RTT_{Suavizado} + \eta \times Dev$$

- En las implementaciones actuales de TCP los parámetros que se usan son:
 - $\delta = 1/8 = 0,125$
 - $\rho = 1/4 = 0,25$
 - $\eta = 4$

Ejemplo RTO según Jacobson



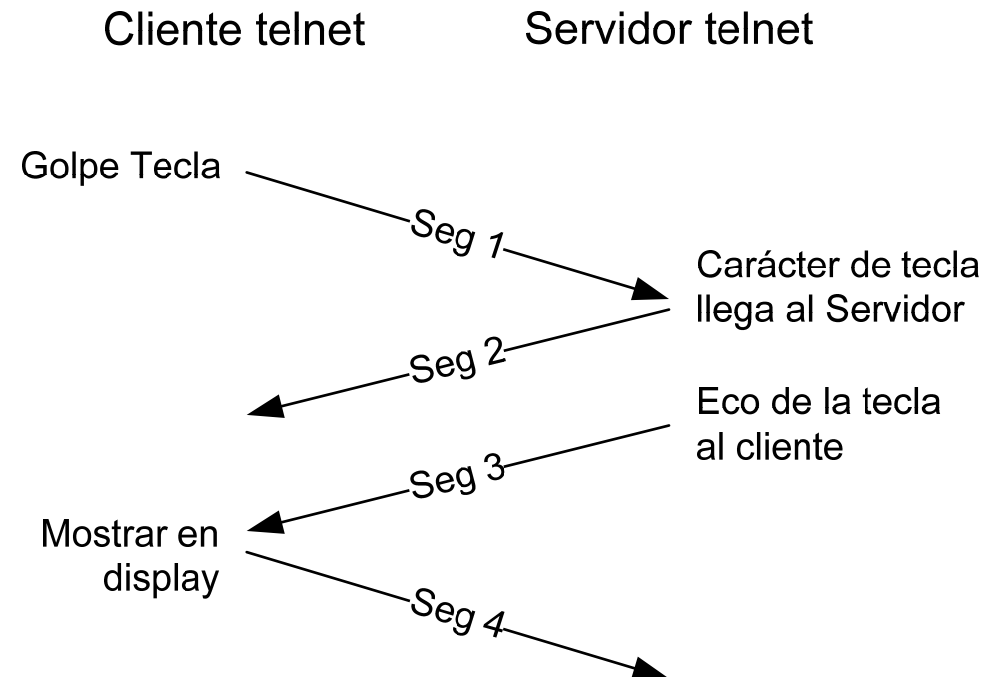
Retardo Exponencial del RTO



- Un segmento transmitido que no recibió ACK se retransmite según el RTO
- Ante el caso de una segundo timeout, el valor del RTO se duplica para la retransmisión y así sucesivamente por cada retransmisión sucesiva.
- De esta manera TCP tiende a disminuir la potencial congestión en la red.
- Técnica parecida a la usada por Ethernet para disminuir la contención en el medio.

Flujo de Datos Interactivos en TCP

- TCP orientado a tráfico de volúmenes de datos.
- Si se trafican pocos bytes -> mucho "overhead" y congestión en red.
- Ejemplo: Telnet
 - Seg1: ASCII Tecla "a" (41 Bytes)
 - Seg2: ACK Tecla "a" (41 Bytes)
 - Seg3: Echo Tecla "a" (41 Bytes)
 - Seg4: ACK Echo Tecla "a" (41 Bytes)
 - Nota: Seg2 y Seg3 combinados gralmente. en 1 solo segmento



Algoritmo de Nagle (RFC 896 – ‘86)



- Impone que en una conexión TCP solo puede haber 1 segmento pequeño ($< \text{MSS}$) pendiente de confirmación.
- El transmisor no puede enviar segmentos adicionales hasta no recibir ACK.
- Particularmente útil en redes WAN que pueden sufrir de congestión por segmentos chicos.
- Los paquetes pequeños de datos son almacenados en el buffer de salida de TCP hasta que arribe el ACK.
- Ventaja: El algoritmo tiene un reloj autoincorporado:
 - En redes WAN lentas, llegan mas lentos los ACK's y por ende se inunda la red con menos segmentos -> mejora la performance global
- Desventaja: aumenta retardo percibido por usuario de la aplicación.
- Nota: El algoritmo de Nagle puede ser deshabilitado por administrador del S.O.

Manejo de Congestión en TCP



- Ventanas Deslizantes no controla la congestión (ni el flujo) en la red sino en los **Buffers** que maneja TCP.
- Por cada segmento generado puede aparecer una retransmisión.
- Si aumenta la congestión en la red, los segmentos llegan con retardo al receptor y tb. los ACK -> RTO expira -> Retransmisión de Segmentos >> Congestión!
- Es un efecto retroalimentado positivamente
- La red puede colapsar: "Colapso de Congestión"
- TCP debería tratar de colaborar tratando de evitar que se produzca congestión o si se produce, de disminuirla.
- RFC 2001 "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", 1991

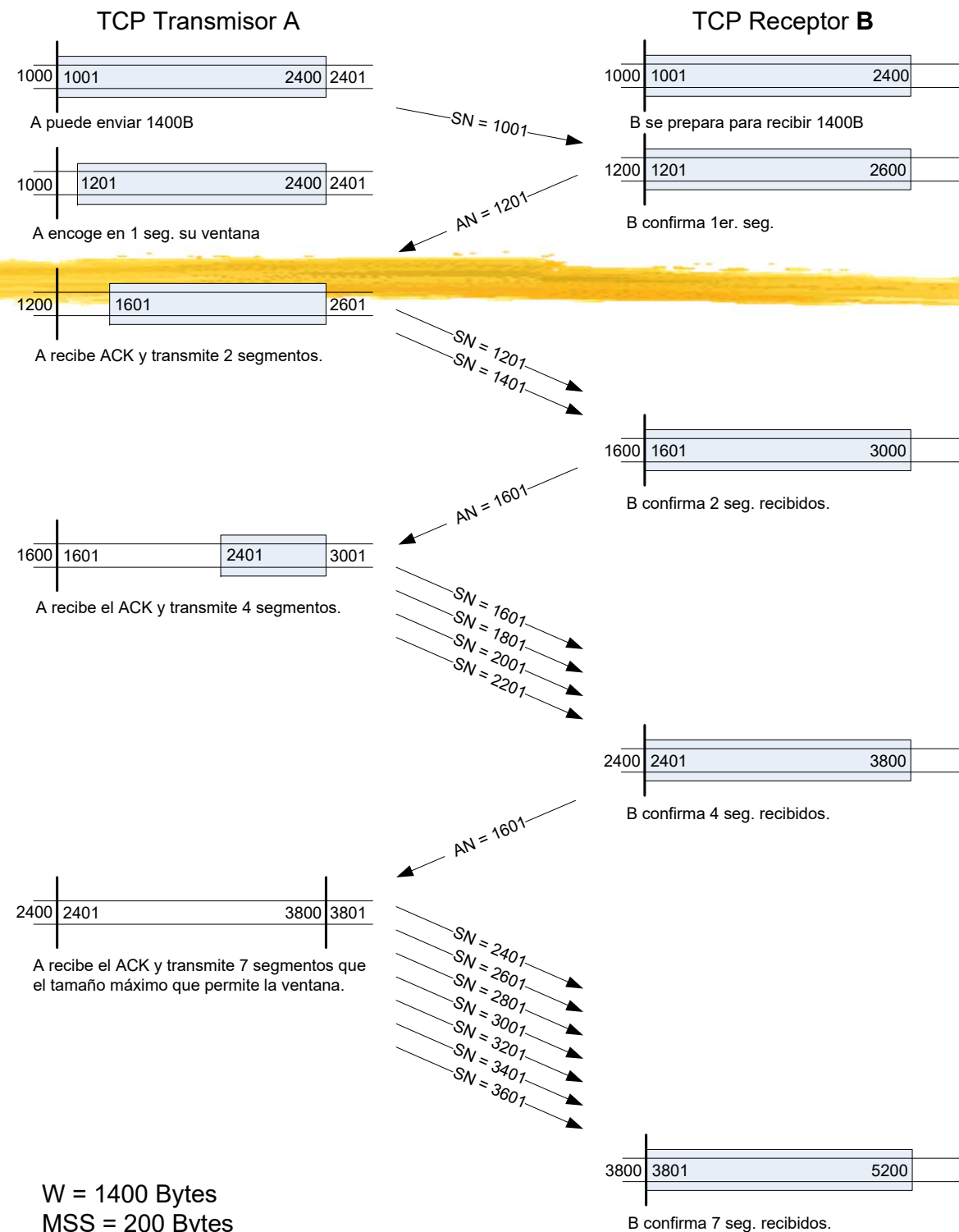
“Slow Start”



- Inmediatamente luego de establecida la conexión TCP, se puede enviar tantos segmentos como lo indique la ventana.
- Si la red se encuentra con tráfico importante -> Se aumentará considerablemente el tráfico en forma abrupta.
- Definiciones (Jacobson)
 - cwnd: “Congestion Window” [Segmento]
 - Windows usado por TCP en el momento del Start Up y reducción de flujo en congestión.
 - awnd: “Allowed Window” [Segmento]
 - Similar a W pero medido en segmentos
- Slow Start:
 - Establecida la conexión: $cwnd = 1$
 - Luego $cwnd = 2 \dots 4 \dots 8 \dots$
 - Hasta llegar a awnd o presencia de congestión (RTO)

“Slow Start”

- Ejemplo:
 - $W = 1400$
 - $MSS = 200$
- TCP (A) envía:
 - 1 segmento.
 - 2 segmentos.
 - 4 segmentos.
 - 7 segmentos (alcanza el tamaño de W)
- Como no se produce un timeout, no se activa el mecanismo para **evitar congestión** que posee TCP



Evitando la congestión ("Congestion Avoidance")

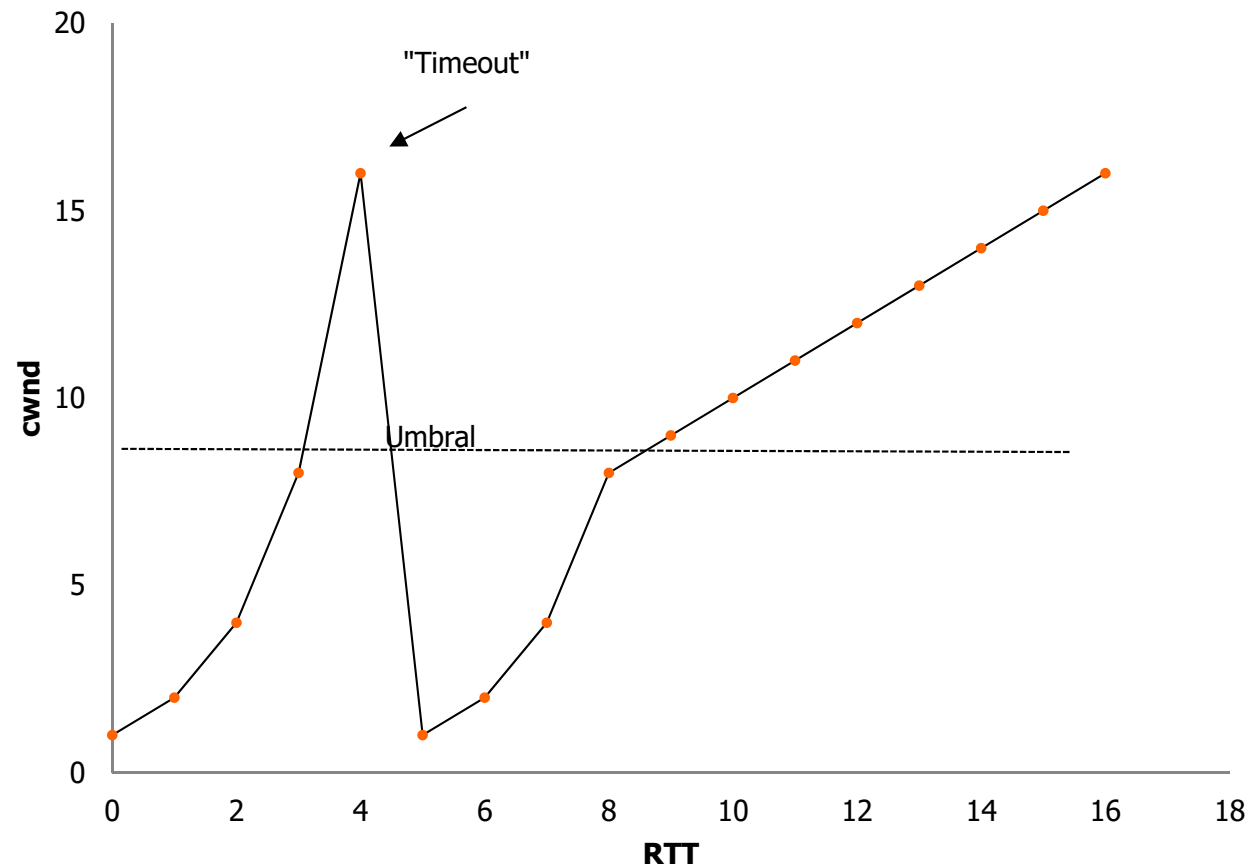


- TCP infiere congestión cuando expira el RTO
- Se podría en ese caso usar "Slow Start" una vez inferida la congestión
- Jacobson propone inyectar segmentos en forma más paulatina que en forma exponencial (salir de saturación lleva tiempo!)
 - Empezar como "slow start" ($cwnd = 1$)
 - Cuando aparece la retransmisión de segmento se registra el valor de ventana ($cwnd (Max)$)
 - Una vez alcanzado dicho valor, TCP activa "slow start" pero con crecimiento exponencial solo hasta $cwnd(max)/2$.
 - A partir de $cwnd(max)/2$ el crecimiento de segmentos transmitidos es lineal.
 - De esta manera se trata de evitar que aparezca nuevamente congestión.

“Congestion Avoidance”

➤ Ejemplo:

- Hasta $RTT = 4$ se utiliza “Slow Start”
 - $cwnd(max) = 16$
- Al ocurrir timeout se usa slow start hasta $cwnd(max) / 2 = 8$
- Luego se inyectan segmentos linealmente.
- Se necesitan 11 segmentos para llegar al valor de $cwnd(max)$.
- Posiblemente ya haya desaparecido la congestión en ese tiempo



Retransmisión rápida & Recuperación Rápida



- RTO calculado (por cualquier método) \gg que el RTT.
- Si un segmento se pierde \rightarrow TCP es lento en la retransmisión.
 - Puede llevar al descarte de varios segmentos independientemente de si TCP utiliza una estrategia de aceptación "in-order" o "in-window"
 - "in-order": Segmento que llega fuera de secuencia es descartado.
 - "in-window": Segmento que llega fuera de secuencia pero dentro de la ventana receptora, es almacenado temporalmente hasta que llega el segmento correcto.
- Jacobson propuso dos procedimientos para mejorar la performance debido a RTO (lento):
 - "Fast Retransmit"
 - "Fast Recovery"
- Estos algoritmos no se verán en el curso.

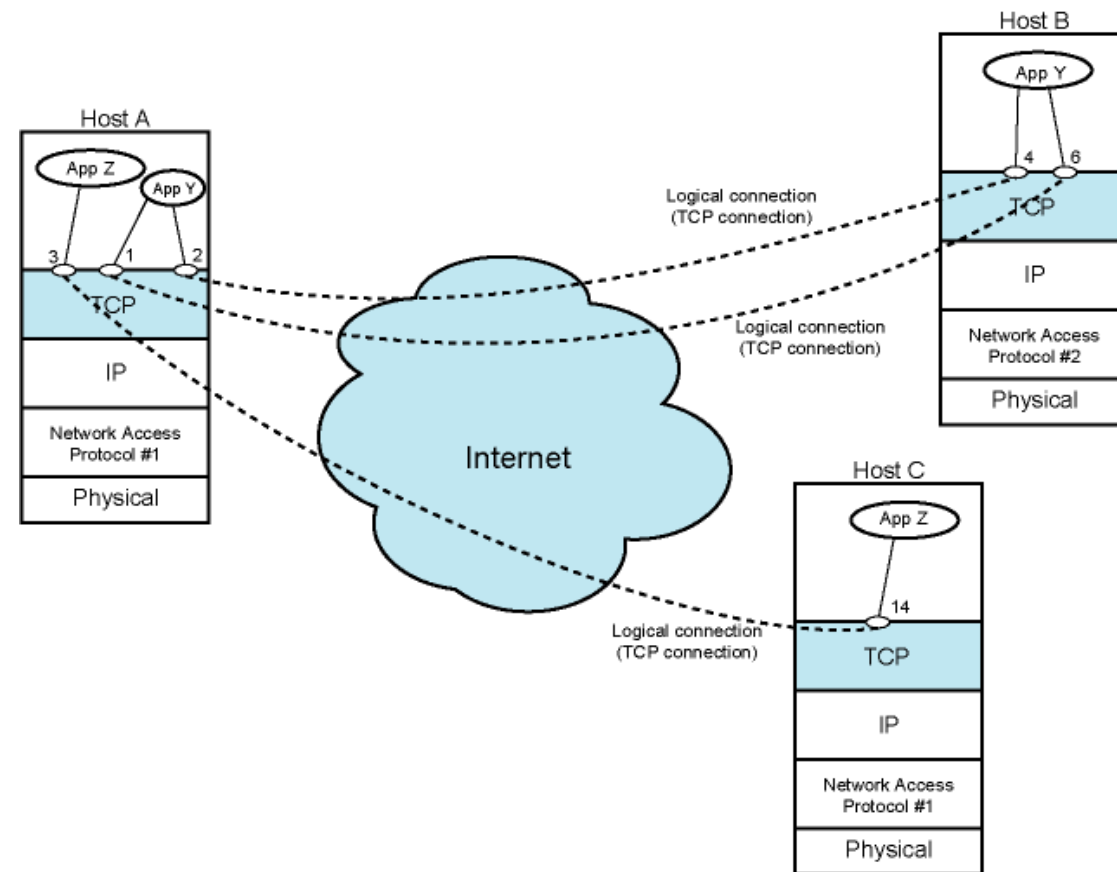
TCP: Servicios TCP



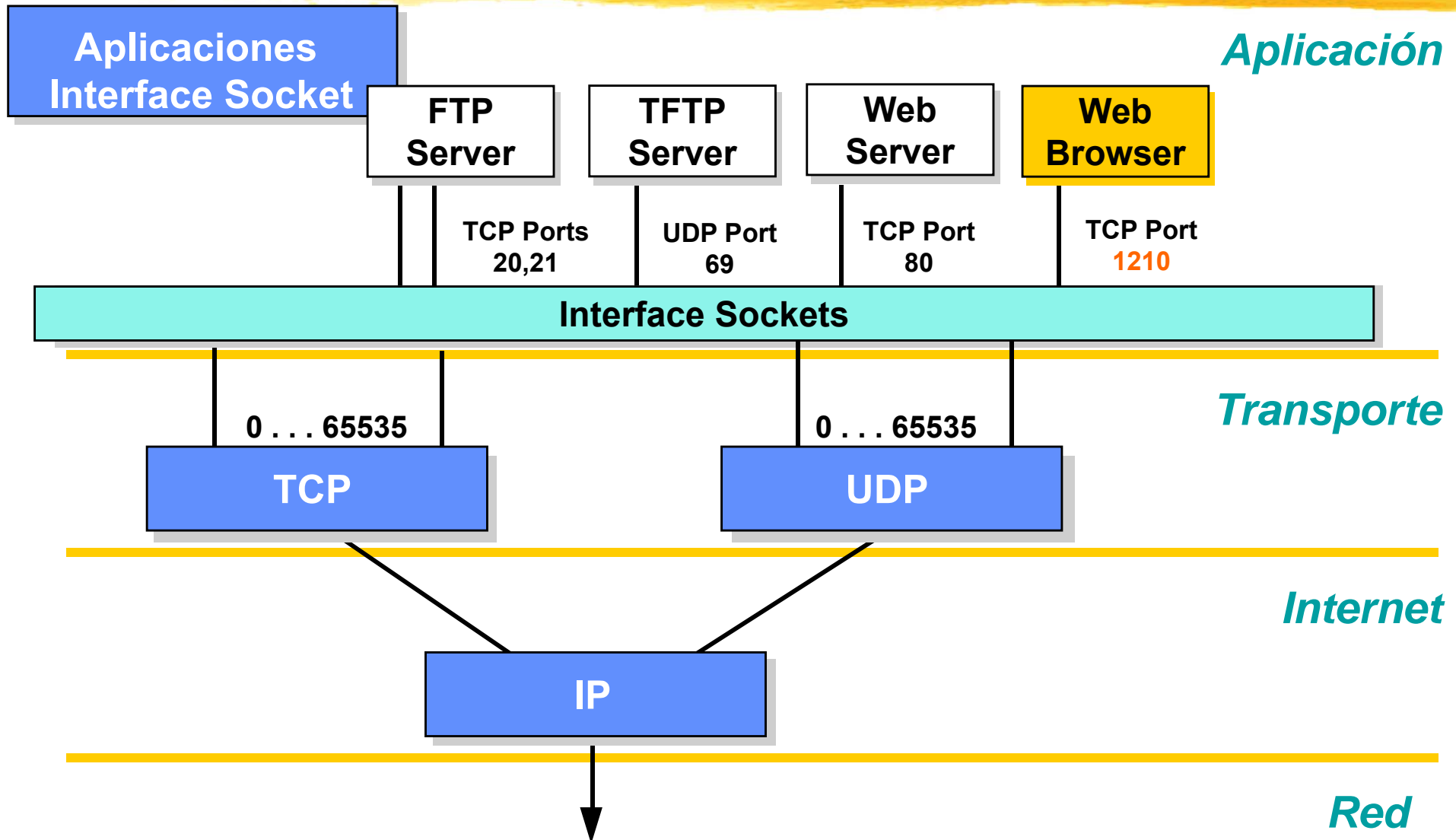
- TCP ofrece un servicio de transporte confiable extremo a extremo
- Ofrece según vimos los siguientes servicios:
 - Multiplexado
 - Soporte de múltiples aplicaciones por medio de puertos
 - Administración de Conexión
 - Establece, mantiene y termina conexiones
 - Transporte de Datos
 - Transporte de datos Full Duplex (flujo de datos bidireccional entre dos puertos de la conexión), en forma ordenada, con control de error y flujo.
 - Reporte de errores
 - Reporta fallas de servicio.
 - Otros
 - Señalización de datos urgentes
 - Mecanismo de Push

Multiplexado: Puertos

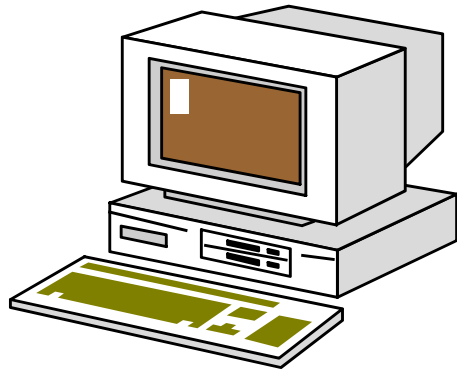
- Multiplexado
 - TCP puede proveer servicios a múltiples procesos en forma simultánea.
 - Procesos se identifican por puertos.
 - Un puerto más la dirección IP constituye un “**socket**” que es único en la Internet.
 - En TCP una conexión identifica a dos sockets.
 - En la figura se observa dos conexiones TCP activas entre Host A y Host B (aplicación Y) y una conexión entre A y C (aplicación Z)
 - La entidad TCP de A, multiplexa tres conexiones en forma simultánea.



Puertos (“Port”) y Sockets



Puertos Bien Conocidos y Asignados Dinámicamente

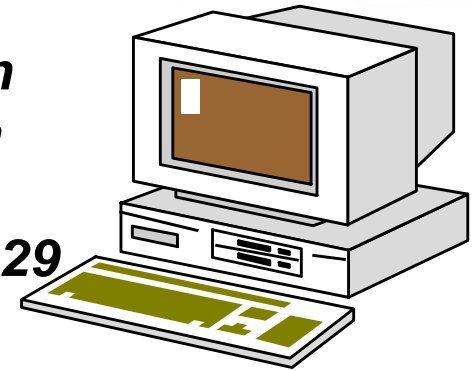


*Aplicación
Cliente*

IP: 134.14.14.8

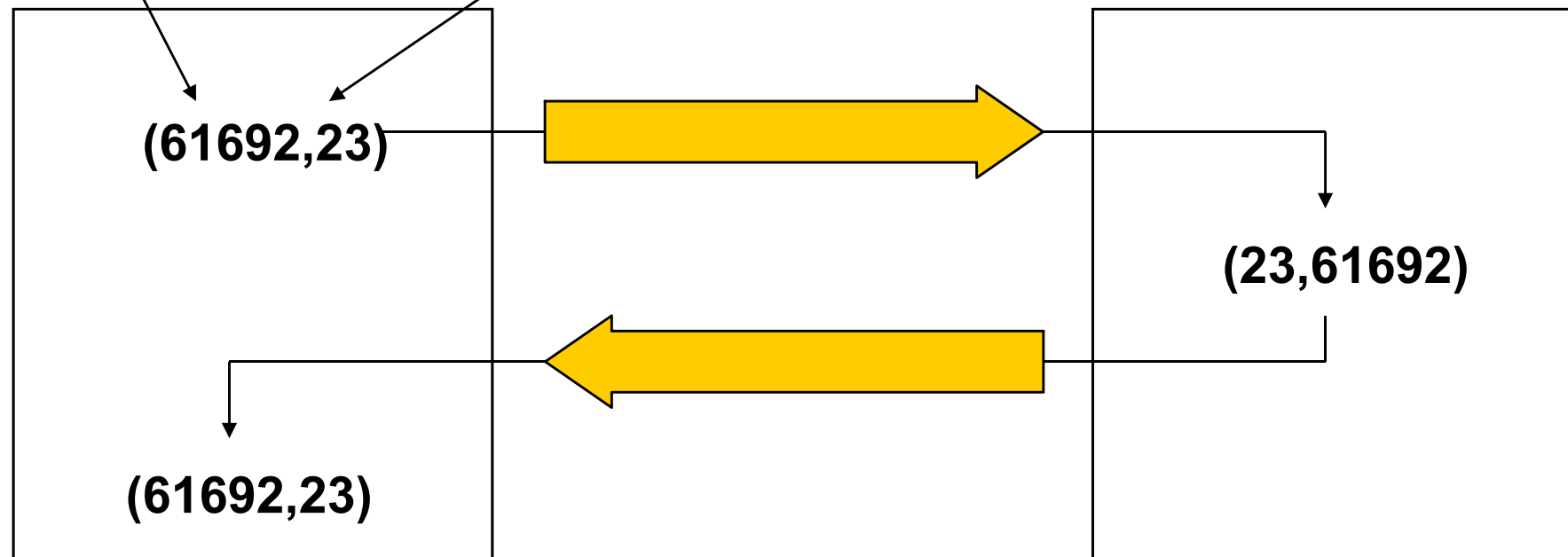
*Aplicación
Servidora*

IP: 195.3.58.29



Puerto Origen

Puerto Destino



Puertos Bien Conocidos (“Well-Known Ports”)

- Utilizados en aplicaciones servidoras
- Asignados por IANA (<http://www.iana.org/assignments/port-numbers>)
- Universalmente adoptados y conocidos
- Pueden tomar valores desde 0-1023
- Ejemplos:

Puerto (TCP/UDP)	Protocolo
TCP/20	FTP - Data
TCP/21	FTP - Control
TCP/23	Telnet
UDP/53	DNS
UDP/69	TFTP
UDP/513	rwhod

Otros tipos de puertos

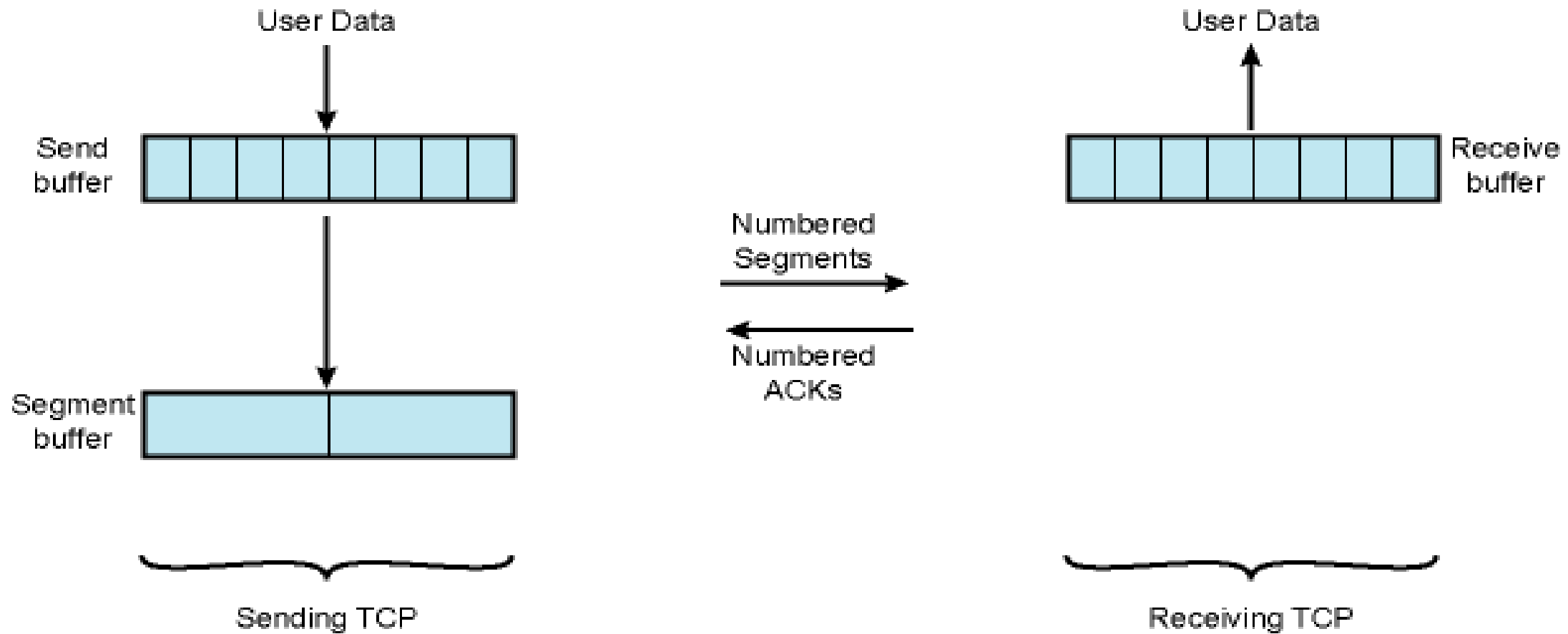
- Puertos Registrados
 - Utilizados por programas (registrados) de usuarios
 - Van desde el 1024 al 49151
- Dinámicamente asignados ("private port number")
 - Utilizados por aplicaciones privadas
 - Ocupan el rango de 49152 hasta 65.535
- Puertos Efímeros
 - Son utilizados por aplicaciones clientes.
 - El rango depende del sistema operativo
 - Se puede configurar para que usen el rango de los puertos asignados dinámicamente
- El uso de puertos bien conocidos y efímeros, permite el multiplexado de TCP:
 - Comunicaciones simultáneas desde un mismo cliente al mismo servidor
 - Comunicaciones simultáneas desde un mismo cliente al distintos servidores
 - Un mismo host ser cliente o servidor
 - Otros tipos de configuraciones

Operación Básica de TCP

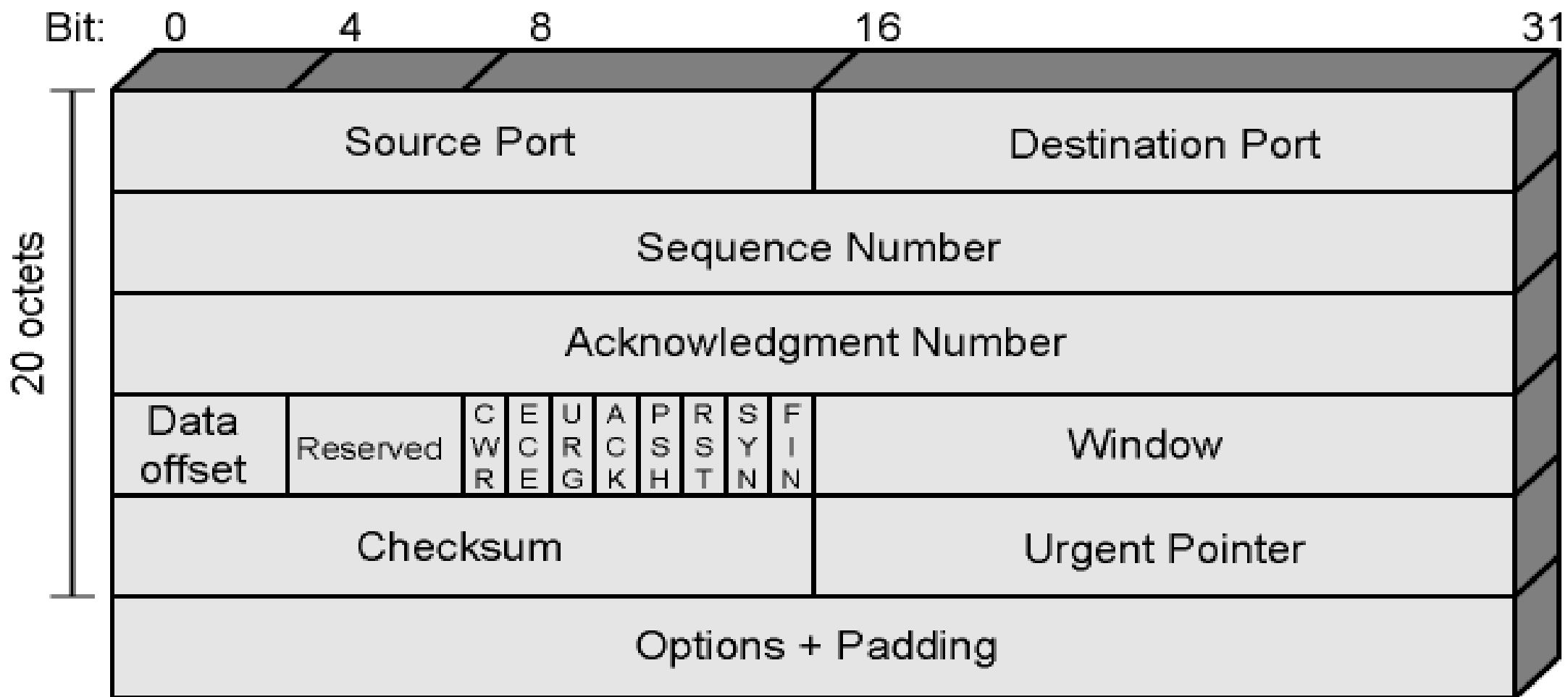


- Datos transmitidos en segmentos
 - Encabezamiento TCP y datos de aplicación
 - Algunos segmentos no transportan datos
 - Manejo de Conexión
- Datos pasados por aplicación a TCP en secuencia de primitivas **Send**.
- Almacenadas en un "send buffer"
- TCP ensambla datos del buffer y crea segmentos que luego se transmiten.
 - Con la opción **PSH** se obliga a enviar los datos del buffer al destino
- Segmentos son transmitidos por servicio de IP
- Entregados a la entidad TCP destino
- Elimina el "header" y ubica los datos en el buffer de recepción
 - Igualmente con **PSH** se pasan los datos a la aplicación, sin espera de buffers
- TCP notifica a la aplicación receptor a través de la primitiva "Deliver"

Operación Básica TCP



Header TCP



Header TCP



- Puerto origen y destino – 16 bits (c/u)
 - Interface con la capa de aplicación.
 - Identifica de que aplicación (origen) viene y a que aplicación (destino) se dirige el segmento TCP
- Número de secuencia – 32 bits
 - Posición en el “stream” de bytes enviados por el transmisor.
 - Con esto el protocolo se convierte en **orientado a byte** (“byte stream”)
 - Si el flag SYN es 1, este campo contiene el Número de Secuencia Inicial
- Número de ACK – 32 bits
 - El número de secuencia que espera recibir a continuación el **receptor**.
 - Este campo tiene sentido si el flag ACK está en 1.

Header TCP



- Observaciones sobre Número de secuencia y Número de ACK:
 - El número de secuencia se refiere al “stream” que fluye en la misma dirección del segmento
 - El número de ACK se refiere al número de secuencia que fluye en oposición al segmento.
 - El esquema de enviar ACK con datos se denomina “piggybacking”

Header TCP



- Data Offset: Longitud del Header – 4 bits
 - Medido en palabras de 32 bits
 - Se utiliza porque luego del “urgent pointer” puede ir un campo de opciones
- Reservado – 4 bits
 - Uso futuro.
- Flags – 8 bits
 - Indica significado especiales del segmento.
 - CWR: “Congestion Windows Reduced”
 - ECE: ECN-Echo (“Explicit Congestion Notification”)
 - CWR y ECE son utilizados para la función de Notificación de Congestión Explícita
 - Ver RFC 3168
 - Uso opcional de estos bits (se debe establecer en el inicio de la conexión su uso)

Header TCP



- Flags (continúa)
 - URG: Campo de puntero “urgent” es válido
 - ACK: Campo de ACK válido
 - PSH: Función de PUSH
 - RST: Reset de Conexión
 - SYN: Sincronización del Número de Secuencia
 - FIN: No hay más datos del Trasmisor.
- Tamaño de Ventana – 16 bits
 - Indica el tamaño de la ventana (en octetos)
 - El tamaño de la ventana es variable en TCP
 - De esta forma se implementa control de flujo

Header TCP



- Checksum – 16 bits
 - Utilizado para “header” y **datos**
 - El checksum se aplica al segmento entero y a un **“pseudo header”** que incluye a ciertos campos del header IP (Direcciones, Protocol number, ...)
 - De esta manera TCP se asegura de posibles errores de entrega de IP.
- Puntero “Urgent” – 16 bits
 - Medio de notificar a la aplicación destino que han llegado datos urgentes.
 - El flag **“urgent”** debe estar en estado **set**
 - El valor del Puntero “Urgente”, cuando se adiciona al número de secuencia del segmento, contiene el número de secuencia del **último** octeto que contiene datos urgentes.
 - El RFC de TCP no especifica donde comienza el dato de urgente (sino donde termina). Esto se deja a la aplicación.
 - Una aplicación que usa esta característica es FTP (para abortar una transferencia de archivos)
 - **Normalmente el uso de opción URG esta asociado al uso de la opción PSH**

Opciones TCP



- Tamaño máximo del segmento
 - Incluido en el segmento SYN
- Escala de la Ventana
 - Permite usar la escala 2^F (F indicado en esta opción) como tamaño de la ventana, en lugar de octetos (default)
- SACK Permitido
 - Se permite "Selective ACK"
- SACK
 - Permite al receptor indicar cuales son los segmentos que se recibieron correctamente.
 - El trasmisor solo retransmitirá segmentos que no fueron recibidos correctamente.
 - RFC 2018
- Checksum alternativo

UDP (“User Datagram Protocol”)




- RFC 768
- Protocolo **NO** Orientado a Conexión (“Connectionless”)
 - No se Establece una Sesión antes de la transferencia de datos
- No se garantiza la entrega
 - No existen los Números de Secuencia
 - No existen los ACK's
 - No provee Control de Flujo
- Confiabilidad es responsabilidad de la aplicación
- Utilizado principalmente por aplicaciones que típicamente transmiten poca cantidad de datos
 - Ejemplo: DNS, SNMP, DHCP

Estructura Header UDP



Puerto UDP Origen
Puerto UDP Destino
Longitud del Mensaje
Checksum

 = 1 Byte

Header UDP



- Puerto Origen y Destino –16 bits cada uno
 - Igual significado que en TCP
- Longitud del Mensaje – 16 bits
 - Número de **octetos** en el mensaje UDP
 - Incluye el Header
 - Valor mínimo: 8 (solo el "header")
- Checksum – 16 bits
 - Opcional
 - Si todo 0's -> no fue computado
 - Si es computado, tiene la misma forma que el checksum de TCP (esto no convierte a UDP en protocolo confiable).