TEMA 4. SERVICIOS BÁSICOS PARA EL NIVEL DE TRANSPORTE EN INTERNET

PROTOCOLOS DE TRANSPORTE

- **EL NIVEL DE TRANSPORTE**

El **objetivo** básico es proporcionar servicios para permitir la comunicación lógica entre procesos que se ejecutan en la capa de aplicación.

Hay que tener clara la **diferencia** entre la **comunicación entre máquinas**, de la que se encarga el nivel de red, y la **comunicación entre procesos**, en la que toma parte el nivel de transporte.

- **PROTOCOLOS DE LA CAPA DE TRANSPORTE**

Los protocolos de transporte se ejecutan en los nodos terminales que se están comunicando.

- **En el emisor**: los mensajes se rompen en segmentos y se envían a la capa de red.
- **En el receptor**: Los mensajes se reensamblan a partir de los segmentos recibidos y se envían a la capa de aplicación.

Existen una serie de protocolos disponibles en TCP/IP para ser usados en aplicaciones:

- **TCP:** Protocolo que realiza la fragmentación anteriormente detallada. Es el más utilizado debido a su comodidad y a las garantías de su correcto funcionamiento, ya que realiza control de flujo, control de errores, etc.
- **UDP:** junto con TCP son los dos protocolos clásicos y su principal característica es que el programador es el responsable de realizar la fragmentación y si no somos precisos en esta práctica, el nivel de transporte es el responsable de hacer el corte. UDP es práctico para máquinas con poca capacidad de memoria. Esto se debe a que, a diferencia de TCP, no realiza los distintos controles ni comprobaciones y por ello, el protocolo puede ejecutarse sin problemas con baja capacidad.
- Otros protocolos más recientes son **SCTP** y **DCCP**.

- **SERVICIOS QUE SE OFRECEN**

Direccionamiento: identificación de procesos. Si quiero establecer una distinción, es indispensable dotar con una dirección (puerto) a los distintos procesos que quieren comunicarse (a los que no quieren comunicarse no se les asigna ningún puerto). De tal forma que cada máquina tiene disponible para la comunicación 65536 puertos TCP y 65536 puertos UDP.

Seguimiento de la comunicación individual entre procesos origen y destino:

- **Protocolos orientados a la conexión (ej: TCP):** son protocolos que presentan un transporte fiable de datos y un control de congestión y un control de flujo.
- **Protocolos no orientados a la conexión (ej: UDP):** no es necesario establecer una conexión previa para la comunicación.

Segmentación y reensamblado de datos: trocear los datos de usuario en varios mensajes o segmentos.

Multiplexación: multiplexar varias conexiones de transporte en una sola conexión de red para aumentar la eficiencia. Es decir, aprovecha una única conexión de red para enviar la información de todos los procesos.

- **DIRECCIONAMIENTO**

Un proceso de una computadora local -cliente- necesita servicio desde un proceso remoto -servidor- .

En una misma máquina puede haber varios procesos al mismo tiempo interesados en comunicarse con otros.

Para el direccionamiento es necesario tener una dirección de nivel de transporte denominada TSAP (puerto en TCP y UDP), que permite elegir entre los múltiples procesos que se ejecutan en una máquina destino.

A cada uno de los procesos que tienen en común dos máquinas les asignamos una TSAP (puerto si se trabaja con UDP o TCP) y de esa forma podrán comunicarse. Cuando un puerto mande un mensaje, en la cabecera del nivel de transporte aparecerá indicado que el origen de ese menaje es ese puerto.

- **DIRECCIONAMIENTO EN TCP/IP**

Los números de puerto son enteros de 16 bits [0-65535] que son elegidos de forma arbitraria y efímera por el proceso cliente. El proceso servidor también define su número de puerto, pero no de forma arbitraria -> **Número de puerto bien conocido**.

La comunicación proceso a proceso necesita al menos tres elementos:

- **Protocolo**
- **Dirección IP**
- **Número de puerto**

La combinación se denomina ***dirección de socket****:* **<protocolo, IP, puerto>** Un protocolo de transporte necesita un par de direcciones de sockets (para cliente y servidor).

- **MULTIPLEXACIÓN**

En este contexto podemos considerar dos tipos de multiplexación:

- **Hacia arriba:** ayuda a reducir los costes de transporte da datos (más lento)
- **Hacia abajo:** Disminuir el tiempo de transmisión (menos eficiente)

- **MULTIPLEXACIÓN EN TCP/IP**

El mecanismo de direccionamiento permite multiplexar y demultiplexar las direcciones del nivel de transporte.

- **Multiplexación (relación muchos-uno):** el protocolo acepta mensajes de distintos procesos diferenciados por el número de puerto asignado
- **Demultiplexación (relación uno-muchos):** El protocolo de transporte recibe datagramas de nivel de red y entrega cada mensaje al proceso adecuado basándose en el número de puerto.

- **ENTREGA FIABLE**

El nivel de transporte es responsable de controlar errores extremo a extremo. Asegura que todo el mensaje llega al nivel de transporte del receptor sin errores.

- Control de mensajes corruptos, desordenados, pérdida de mensajes o mensajes suplicados.

Habitualmente los errores se corrigen con **retrasmisiones.**

- **CONTROL DE FLUJO**

El nivel de transporte es responsable de controlar el flujo extremo a extremo. La idea principal es controlar que el emisor no envíe más de lo que el receptor puede albergar.

- Permite al receptor controlar la velocidad (cantidad) de datos enviados por el emisor.
- Habitualmente el flujo se controla mediante técnicas basadas en ventana deslizante.

Congestión: cuantos más mensajes se encuentren atascados en una serie de routers, más información se vuelca a la red porque los orígenes empiezan a retransmitirse. Es decir, se pierden mensajes porque los routers no son capaces de procesar toda la información, debido a que se les agota la memoria y tienen que descansar.

En el nivel de transporte, en los protocolos TCP/IP existe un **control de congestión.**

- **PROTOCOLO UDP**

Características:

- Envía **datagramas**
- Protocolo no orientado a la conexión
- No fiable
- No incorpora control de congestión

Ventajas frente a TCP:

- Más rápido (no requiere conexión, sin control de errores)
- Usa conexiones sin **estados** (*stateless*)
- Menor penalización en la transmisión al tener una cabecera más corta

Uso:

- Mensajes pequeños, poco intercambio, no preocupa la fiabilidad, envíos Multicast

Nombre de TPDU (*Transport Protocol Data Unit*) **: segmento UDP**

Formato del segmento UDP (cabecera 8 bytes):

- Puerto origen y destino (local y remoto)
- Longitud total
- Checksum aplicado a una pseudocabecera que incluye la cabecera UDP, IPs origen y destino y los datos
- Datos de orden superior

Número del puerto de origen	Número del puerto de destino
Longitud	Checksum
Datos de aplicación (mensaje)	

----------32 bits----------

- **FUNCIONAMIENTO DE UDP**

No hay control de flujo, ni de errores. No hay fase de conexión ni desconexión.

El mensaje es encapsulado en un datagrama IP.

Colas:

- Cada puerto lleva asociada dos colas: entrada y salida
- UDP extrae mensajes de las colas de salida, les añade la cabecera y los entrega a IP
- Cuando llega un mensaje, UDP comprueba si hay una cola de entrada asociada a ese número de puerto
  - Si es así, deposita el mensaje al final de la cola
  - Si no existe, descarta el datagrama y pide a ICMP que envíe un mensaje de puerto no alcanzable
- Todos los mensajes enviados por un mismo proceso son encolados en la misma cola de salida, independientemente del destino
- Todos los mensajes recibidos por un mismo proceso son encolados en la misma cola de entrada, independientemente del origen

- **PROTOCOLO TCP**

Características:

- Protocolo orientado a la conexión
- Las conexiones son full-duplex
- Canales punto a punto (no broadcasting ni multicasting)
- Ofrece un servicio fiable
- Conexión orientada a flujos de bytes, no a flujo de mensajes
- TPDU: segmentos TCP

Buffering:

- Los **segmentos TCP** se envían a un buffer de envío
- TCP enviará los datos en el buffer de envío cuando lo estime oportuno
- Existe también un buffer de recepción

MSS (*Maximum Segment Size):*

- Máxima cantidad de datos que puede llevar un segmento TCP
- Depende de la implementación

Estructura del segmento TCP:

- Cabecera de 20 bytes + opciones

<table>
<tr><td colspan="8">Número de puerto origen</td><td colspan="2">Número de puerto destino</td></tr>
<tr><td colspan="10">Número de secuencia</td></tr>
<tr><td colspan="10">Número de confirmación</td></tr>
<tr><td>Longitud cabecera</td><td>Inutilizado</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td colspan="2">Tamaño de la ventana</td></tr>
<tr><td colspan="8">Internet Checksum</td><td colspan="2">Puntero de datos urgentes</td></tr>
<tr><td colspan="10">Opciones</td></tr>
<tr><td colspan="10">Datos</td></tr>
</table>

----------32 bits----------

Los **números de secuencia** están asociados a **bytes,** no a segmentos de TCP.

La longitud de cabecera se muestra en múltiplos de 4 bytes.

Campos:

Campos de la cabecera TCP:

- **Puertos origen y destino** (16)
  - TSAPs que identifican los extremos de la conexión
- **Número de secuencia** (32)
- **Número de confirmación** (32)
- **Longitud de la cabecera** (4)
  - En palabras de 32 bits
- **Tamaño de la ventana** (16)
  - Para el control de flujo
- Bit **ACK**: el valor del campo ACK es válido
- Bits **RTS**, **SYN**, **FIN**: establecimiento/fin de conexiones

  URG: *Urgent Pointer is valid*
  ACK: *Acknowledgment is valid*
  PSH: *Request for push*
  RST: *Reset the connection*
  SYN: *Synchronize sequence numbers*
  FIN: *Terminate the conection*

Números de secuencia y ACKs:

TCP utiliza el **número de byte** para numerar el segmento

- TCP numera todos los bytes de datos que se transmiten
- La numeración no comienza desde 0 -> número aleatorio [0 – 2^{32}-1]
- El número de secuencia de un segmento es el del primer byte que transporta el segmento

Si un segmento no lleva datos de usuario no consume número de secuencia

- **Excepción**: segmentos de control (inicio y fin de conexiones)

ACKs: número de secuencia del siguiente byte que se espera recibir

Hay algunos segmentos de control que consumen número de secuencia porque requieren confirmación.

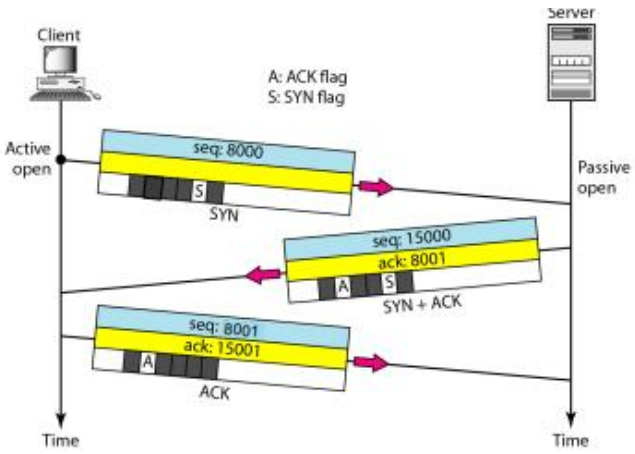
- Consumen 1 byte, aunque no lleven datos

ESTABLECIMIENTO DE CONEXIONES

Se usa un protocolo de negociación a tres bandas (*three-way hansshake)*

El segmento inicial de secuencia se elige aleatoriamente

Los segmentos SYN y SYN+ACK no llevan datos, pero consumen



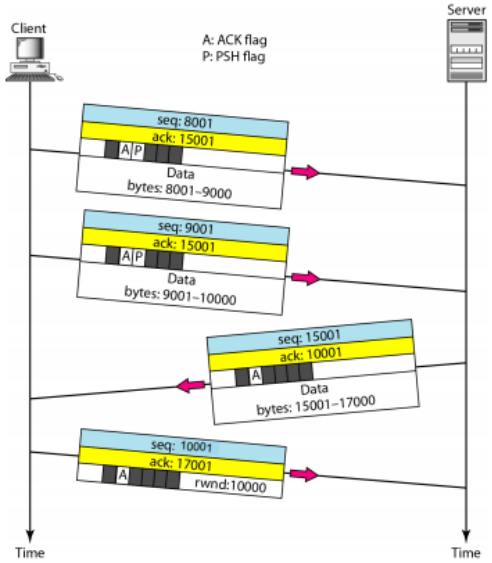
TRANSFERENCIA DE DATOS TCP

Transmisión de datos bidireccional:

- Ambos extremos pueden enviar datos y confirmaciones
- La confirmación se incluye con los datos

Entrega inmediata de datos (pushing):

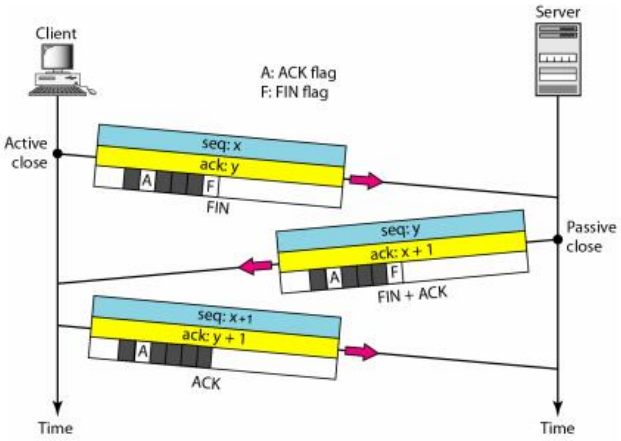
- El emisor no debe esperar a completar una ventana
- El receptor entrega los datos lo antes posible



- **FINALIZACIÓN DE LAS CONEXIONES**

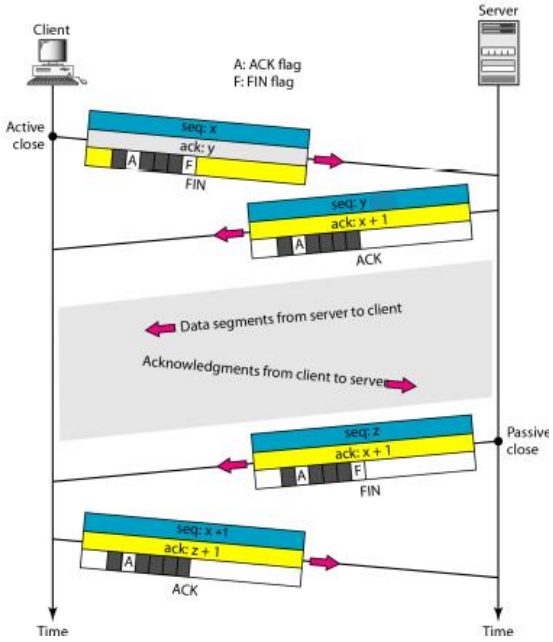
Negociación en tres pasos – *three-way handshaking*:

- Cualquiera de los dos extremos inicia la desconexión

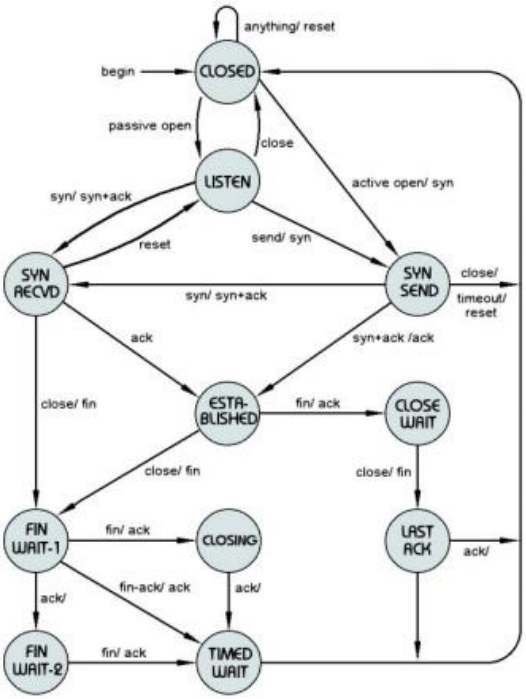


Semicierre:

- Uno de los extremos deja de enviar datos mientras sigue recibiendo



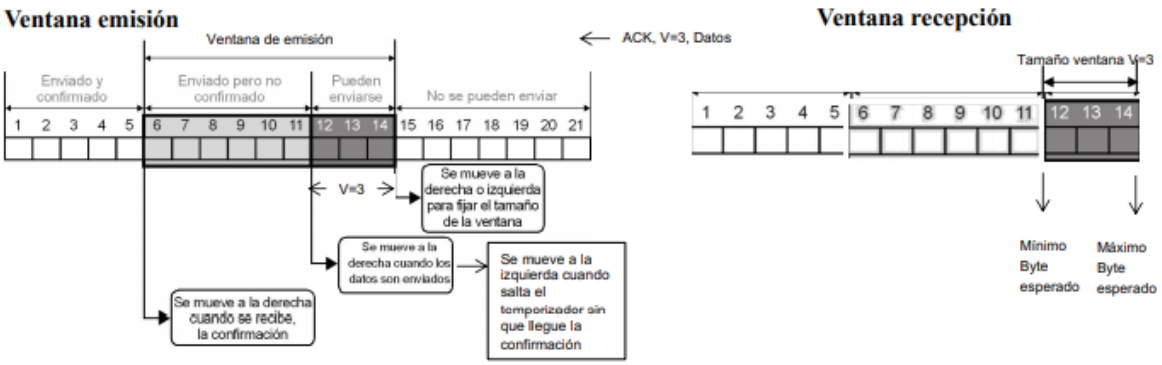
- **MÁQUINA DE ESTADOS TCP**



- **CONTROL DE FLUJO EN TCP: VENTANA DESLIZANTE**

Se usa una ventana deslizante para hacer la transmisión más eficiente

- Las ventanas son orientadas a bytes
- Son de tamaño **variable**
  - Los bytes dentro de la ventana se pueden enviar sin preocuparse de la confirmación
- La ventana de recepción es el valor notificado por el segmento opuesto en un segmento que contiene una confirmación -> representa un número de bytes que puede aceptar antes de que su almacén se desborde y tenga que descartar datos
- Hay cuatro ventanas para una conexión full-dúplex



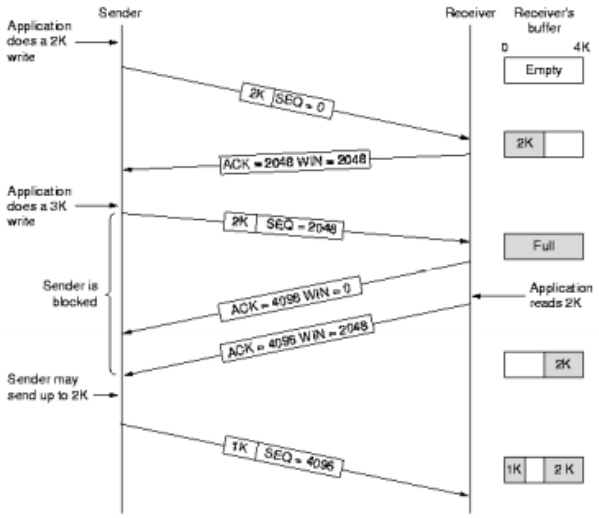
- **CONTROL DE FLUJO EN TCP: AJUSTE DE LA VENTANA**

Síndrome de Silly window:

- Una aplicación receptora/emisora lenta provoca la apertura de la ventana con tamaños pequeños => se generan segmentos de datos pequeños (muy ineficiente)

Solución (lado del receptor):

- Tras cerrar una ventana, el receptor no debe anunciar una ventana distinta de 0 hasta que no pueda anunciar una ventana de tamaño suficiente.
- Habitualmente, el tamaño "suficiente" se fija a ***T = min(MSS, buffer_recepción/2)***



- **ALGORITMO DE NAGLE**

Solución al síndrome de Silly Window del lado del emisor:

- Algoritmo de Nagle es un método eficiente para evitar la transmisión de segmentos demasiado pequeños.

Algoritmo de Nagle:

- Si no existen datos esperando ser confirmados => ***write(datos)***
- Si hay datos pendientes de ser confirmados => no se envían nuevos segmentos de datos hasta que suceda:
    - Que se reciba la confirmación de todos los datos pendientes de confirmación, o bien, la longitud de los datos a transmitir sea mayor que MSS

Todo esto respetando que long(datos) <= tamaño ventana de recepción

- **CONTROL DE ERRORES EN TCP**

TCP incluye mecanismos para detectar y corregir errores por segmentos:

- Corruptos
- Perdidos
- Fuera de orden
- Duplicados

Estos mecanismos son:

- Suma de comprobación
- Confirmación (positiva)
- Retransmisión (con temporizadores)
- Números de secuencia

Suma de comprobación de 16 bits:

- Detecta segmentos corruptos, que son descartados considerados perdidos

Confirmación:

- Indican la recepción de segmentos de datos
- Los segmentos de control que no llevan datos, pero si número de secuencia se confirman (SYN, FIN)
- Los segmentos ACK no se confirman

Retransmisión:

- Se retransmite en dos ocasiones:
  - Cuando expira un temporizador de retransmisión
  - Cuando se reciben 3 ACKs duplicados para el mismo número de secuencia
  - Los segmentos ACK no se retransmiten

Retransmisión después de un plazo de retransmisión (*RTO – retransmission time out*):

- El emisor mantiene un temporizador RTO para todos los segmentos enviados pero no confirmados
- Cuando vence el temporizador, el segmento pendiente anterior se retransmite
  - Aunque la falta de una ACK se deba a un segmento retrasado, un ACK retrasado o la pérdida de una confirmación
- No se activa un temporizador RTO para los segmentos que sólo confirman
- El valor del temporizador RTO es dinámico y se actualiza en base al **tiempo de ida y vuelta** (RTT)

Retransmisión después de tres segmentos ACK:

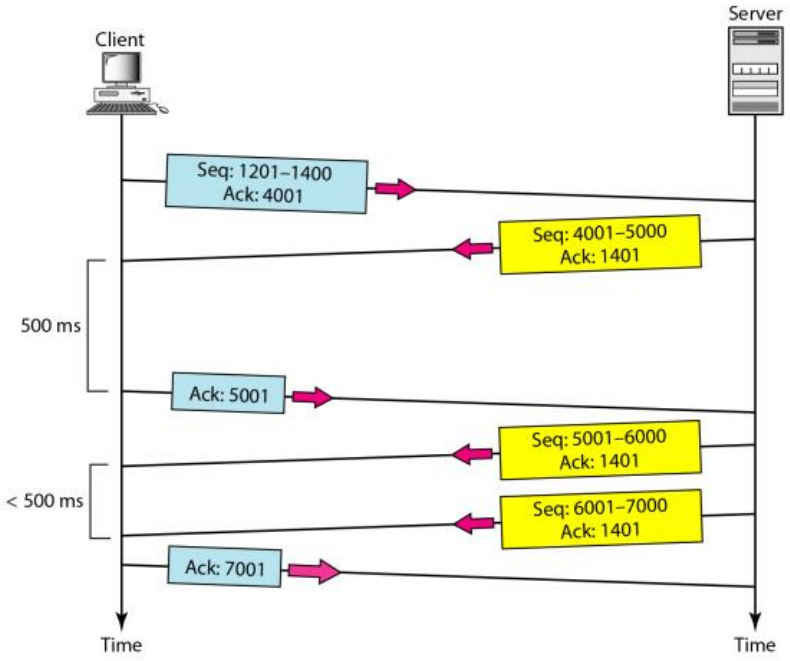
- Escenario: se pierde un segmento y se reciben algunos fuera de orden
  - Un problema si el almacén del receptor es limitado
- Se reciben 3 ACKs duplicados para el mismo número de secuencia -> se reenvía el segmento perdido inmediatamente (**retransmisión rápida**)

Segmentos fuera de orden:

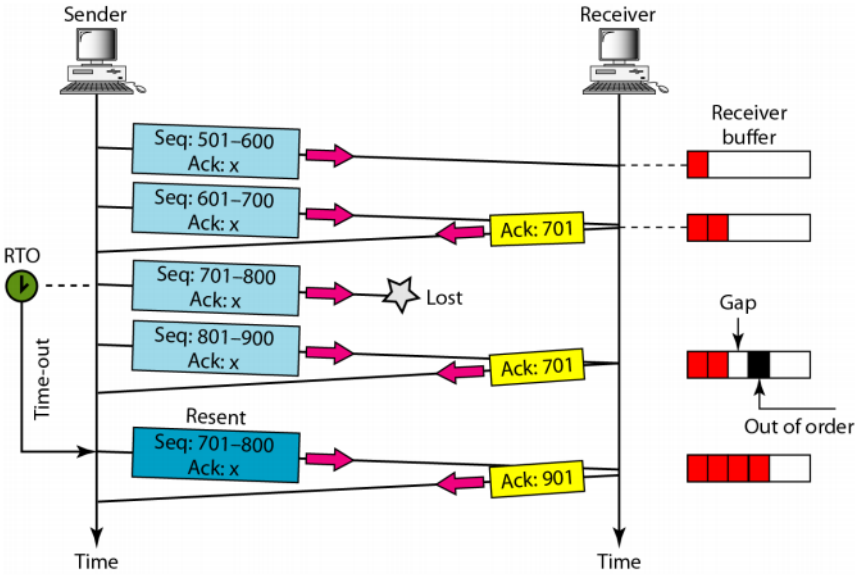
- Antes se descartaban los segmentos fuera de orden, que eran retransmitidos

- Actualmente se almacenan temporalmente y se marcan como “fuera de orden” hasta que llega el segmento perdido
    - Al ser marcados, no se entregan al proceso destino
- Ningún segmento fuera de orden es entregado al proceso destino

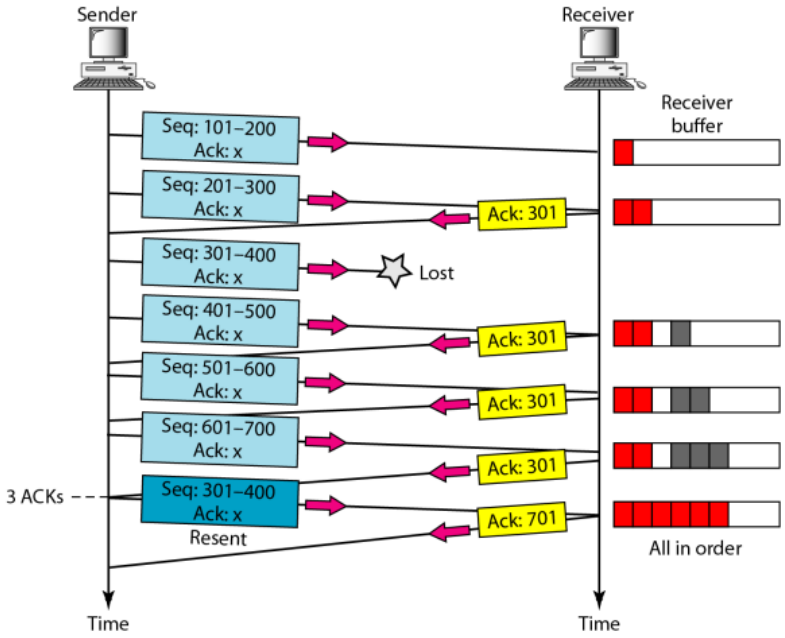
- **ESQUEMA SIN ERRORES**



- **SEGMENTO PERDIDO**



- **RETRANSMISIÓN RÁPIDA**



- **CONTROL DE CONGESTIÓN:**

Conceptos diferentes:

- **Control de flujo:**
  - No enviar más paquetes de los que puede recibir el receptor

Control de congestión en TCP:

- La detección de congestión se basa en observar los siguientes hechos:
  - El aumento de los RTT de los segmentos confirmados, respecto a anteriores segmentos.
  - La finalización de temporizadores de retransmisión

Mecanismos de control de congestión: slow start, congestion avoidance (obligatorios).

Slow start :

- Inicialmente longitud de ventana de congestión (del emisor)
  - ***Long(ventana_congestión) = MSS***
- Cada vez que se recibe un ACK
  - ***Long(ventana_congestion) = long(ventana_congestion) + MSS***
- **El crecimiento de la ventana es exponencial**

Congestion avoidance:

- Intenta suavizar el crecimiento de la ventana del slow start
- Comienza cuando el tamaño de la ventana de congestión es mayor o igual al umbral (Threshold)
- Si salta el temporizador (signo de congestión)
  - ***Long(ventana_congestion) = MSS*** (se vuelve al slow start)
- Si hay ACKs duplicados (signo de congestión)

    - Se reduce la subida del tamaño de la ventana de congestión de exponencial a lineal
    - ***Long(ventana_congestion) = long(ventana_congestion) + MSS/long(Ventana_congestion)***

- **PROTOCOLO SCTP**

Características:

- Orientado al mensaje (como UDP)
- Confiable, con control de flujo y secuenciación (como TCP)
- Permite opcionalmente el envío de mensajes fuera de orden

Ventajas:

- *Multihoming* (posibilidad de varias IP en cada extremo de la comunicación)
- Selección y monitorización de caminos
- Mecanismos de validación y protección contra ataques