Universidad de Málaga

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

Tema 4: Servicios básicos para el nivel de transporte en Internet

Profesores:

Mercedes Amor
Inmaculada Ayala
Lidia Fuentes
Gabriel Luque
Francisco Rus
Alberto Salguero
Daniel Muñoz
Francisco Servant

Contenido del tema

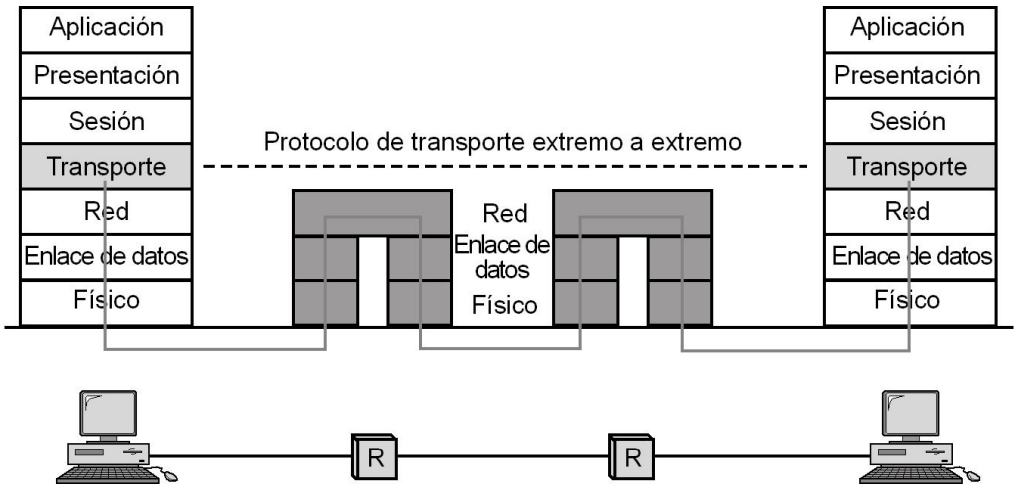
- Protocolos de Transporte
  - Funcionalidad del Nivel de Transporte
  - Protocolo UDP
  - Protocolo TCP
  - Otros Protocolos de Transporte
- Programación Distribuida
  - Programación básica de Aplicaciones Distribuidas (Sockets)
  - Esquemas de programación en TCP y UDP

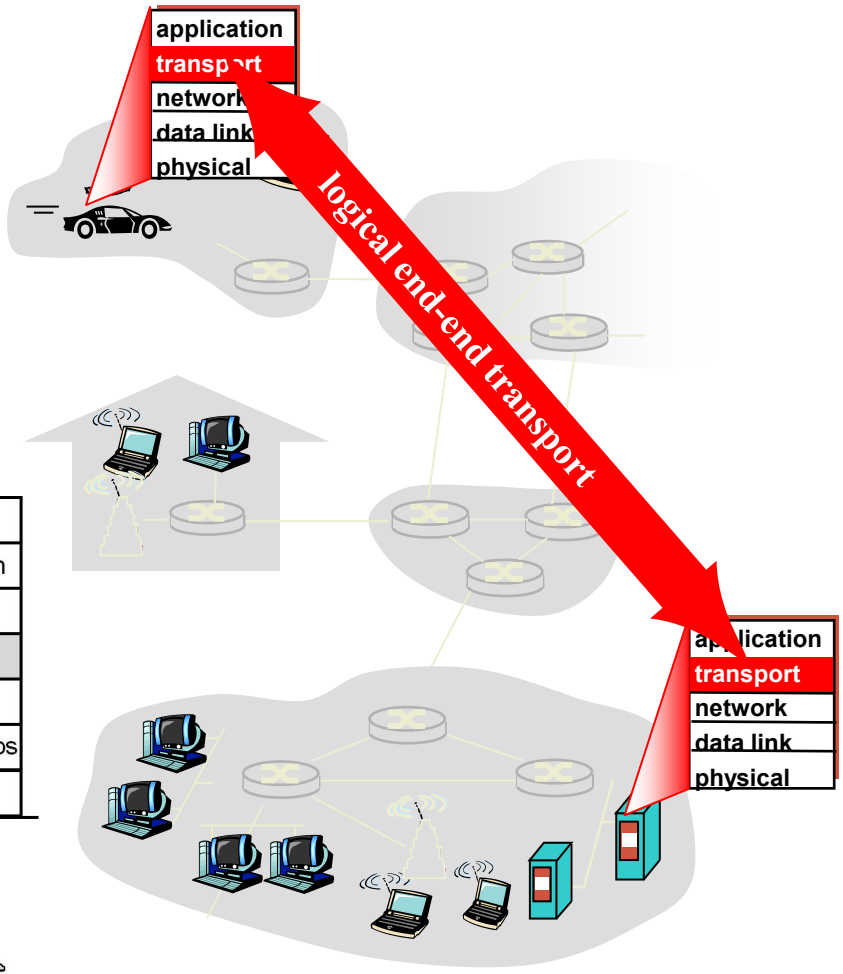
- Funcionalidades del nivel de transporte
- Protocolo UDP
- Protocolo TCP
- Otros Protocolos de Transporte

PROTOCOLOS DE TRANSPORTE

El nivel de transporte

- Objetivo básico
  - Proporcionar servicios para permitir la comunicación lógica entre procesos que se ejecutan en la capa de aplicación





Protocolos de la capa de transporte

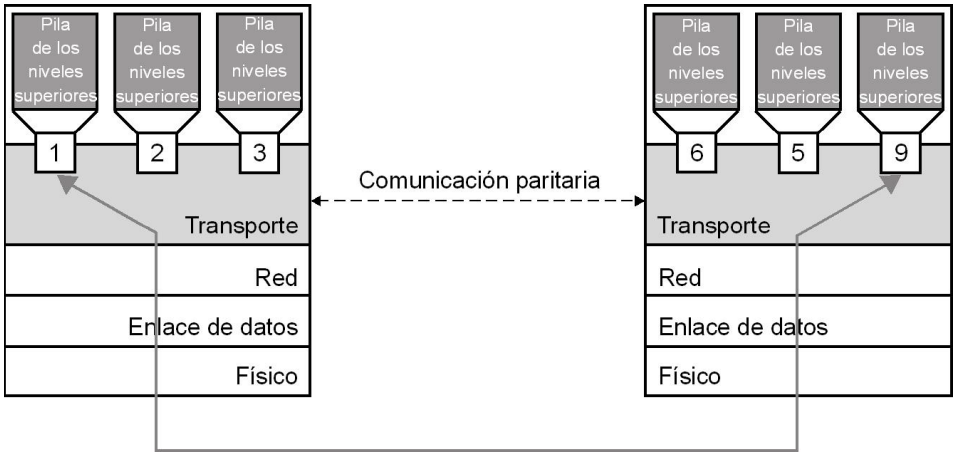
- Los protocolos de transporte se ejecutan en los nodos terminales que se están comunicando
- Objetivo:
  - Suplir todas las carencias que a nivel de control de errores presenta el protocolo de nivel de red utilizado
  - Ofrece al nivel superior (ej: sesión o aplicación según la arquitectura de red usada) una comunicación extremo-a-extremo fiable
- Protocolos de transporte disponibles en TCP/IP
  - TCP, UDP (clásicos), SCTP (reciente)

Servicios que se ofrecen

- Direccionamiento
  - Identificación de procesos
- Seguimiento de la comunicación individual entre procesos origen y destino
  - Protocolos orientados a la conexión (ej: TCP)
    - Transporte fiable de datos
    - Control de congestión y control de flujo
  - Protocolos no orientados a la conexión (ej: UDP)
- 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

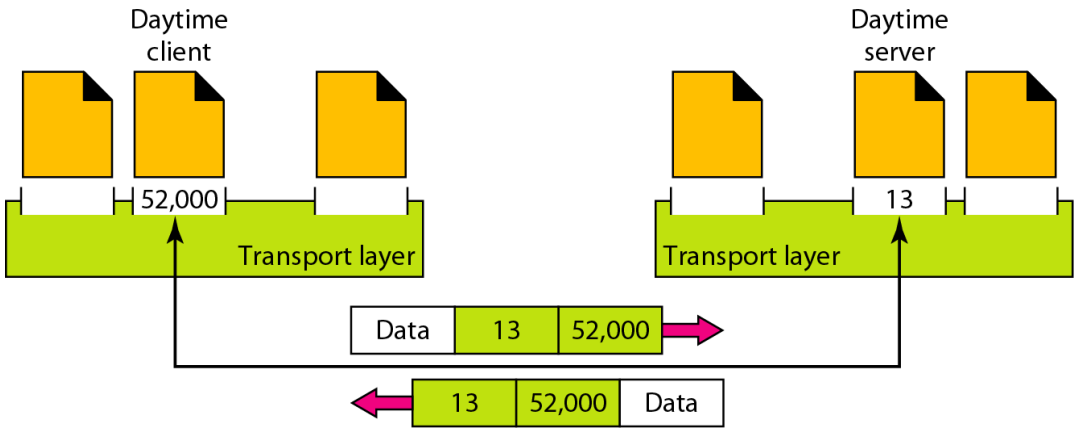
Direccionamiento

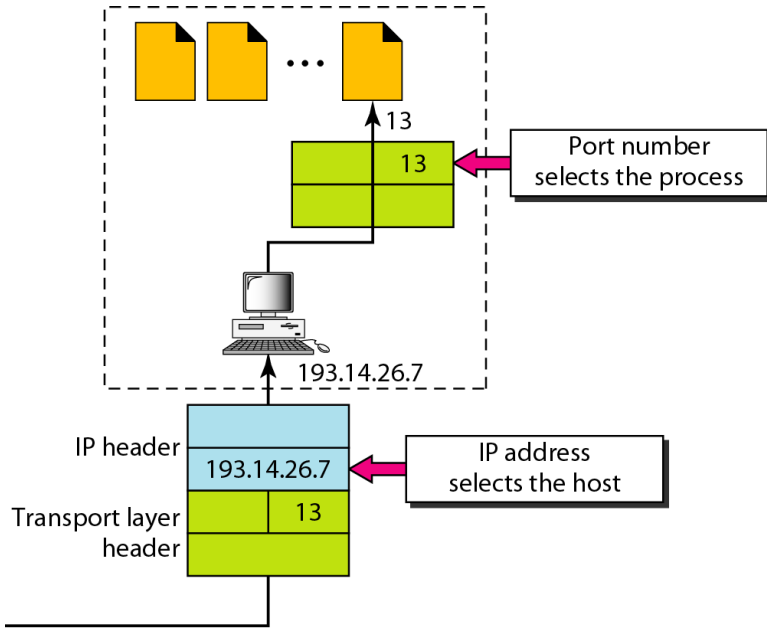
- Un proceso de la computadora local –cliente- necesita servicio desde un proceso remoto –servidor-
- En una misma máquina puede haber varios procesos al mismo tiempo interesados en comunicarse con otros
- Direccionamiento
  - Es necesario tener una dirección de nivel de transporte denominada TSAP (puerto en TCP y UDP)
  - Permite elegir entre los múltiples procesos que se ejecutan en una máquina destino



Direccionamiento en TCP/IP

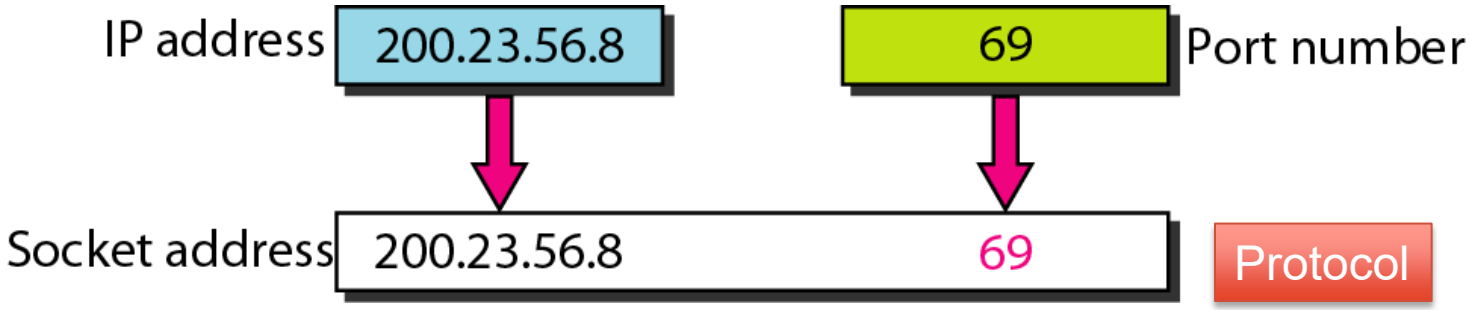
- Los números de puerto son enteros sin signo de 16 bits [0-65535]
- Tres tipos: Bien conocidos (0-1023), Registrados (1024-49151), dinámicos (49152-65535)
- El proceso cliente elige (define) un número de puerto arbitrario
  - Puerto efímero (normalmente elegido de entro los dinámicos)
- El proceso servidor también define su número de puerto, pero no de forma arbitraria
  - Número de puerto bien conocido
    - Puerto de un servicio estándar (ej: 80)
    - Puerto de un servicio no estándar





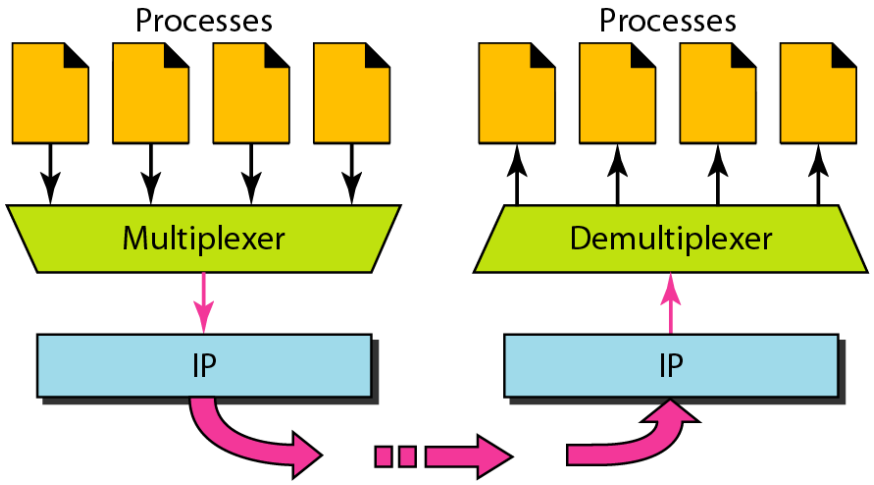
Direccionamiento en TCP/IP

- La comunicación proceso a proceso necesita tres identificadores
  - Dirección IP
  - Protocolo
  - Número de puerto
- La combinación se denomina *dirección de socket*
  - *<protocolo,IP,puerto>*
    - *Ej: <tcp,150.215.12.37,80>, <udp,180.220.21.33,9>*
- Un protocolo de transporte necesita un par de direcciones de sockets (para cliente y servidor)



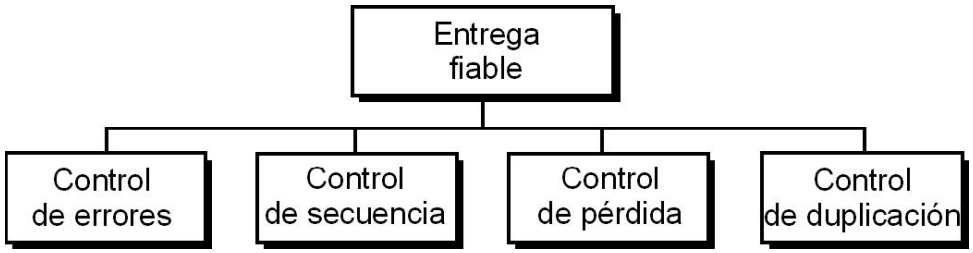
Multiplexación en TCP/IP

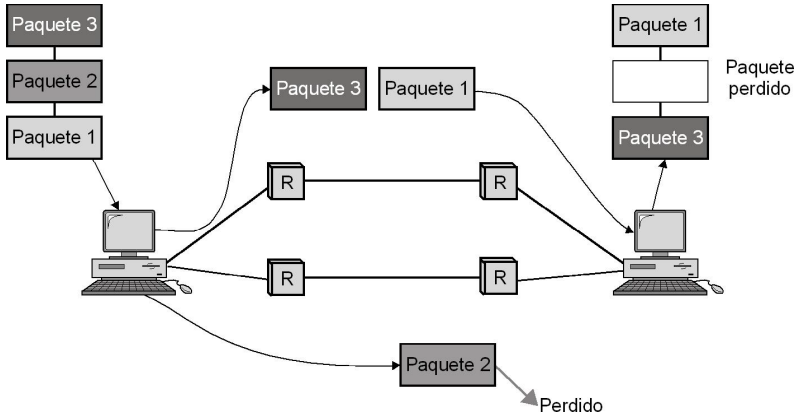
- El mecanismo de direccionamiento permite multiplexar y demultiplexar las direcciones del nivel de transporte
  - Multiplexación (relación muchos-uno)
    - El protocolo acepta mensajes de distintos procesos diferenciados por el nº de puerto asignado
  - Demultiplexación (relación uno-muchos)
    - El protocolo de transporte recibe datagramas del nivel de red y entrega cada mensaje al proceso adecuado basándose en el nº de puerto



Servicio orientado a la conexión: control de errores

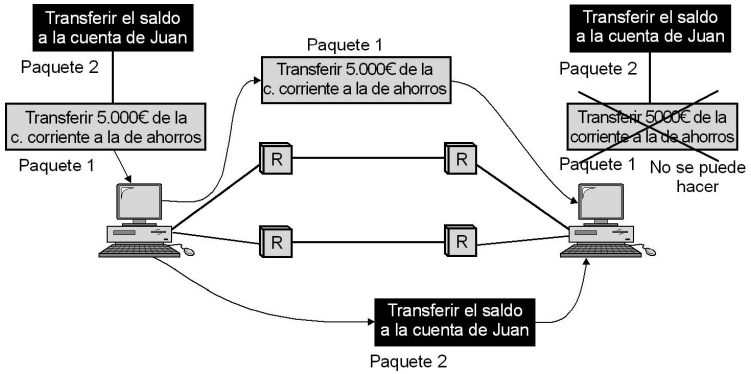
- Es posible controlar los errores extremo a extremo - Control de errores
  - Asegura que todo el mensaje llega al nivel de transporte del receptor sin errores
    - Control de mensajes corruptos, desordenados, pérdida de mensajes o mensajes duplicados
  - Habitualmente los errores se corrigen con **retransmisiones**

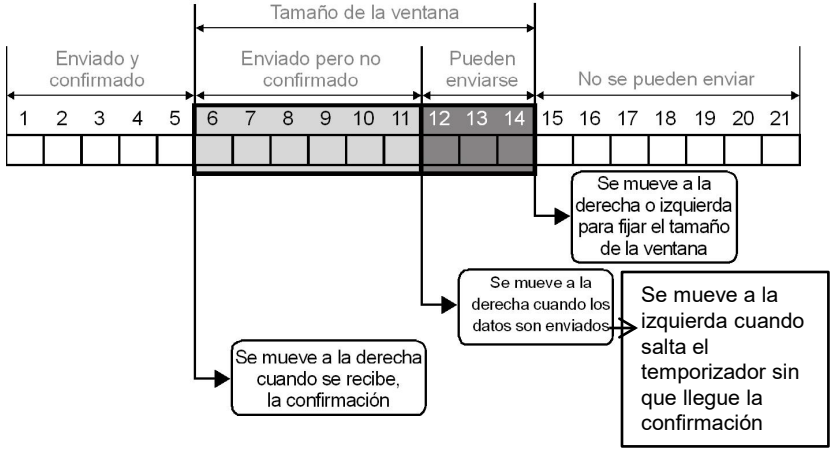




Servicio orientado a la conexión: Control de flujo

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



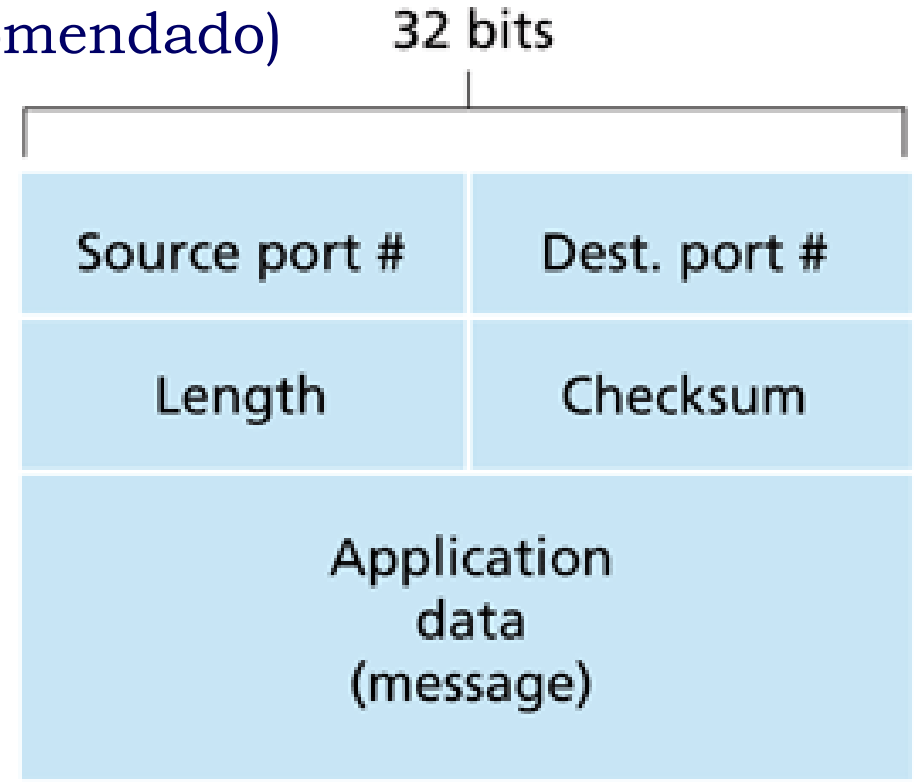


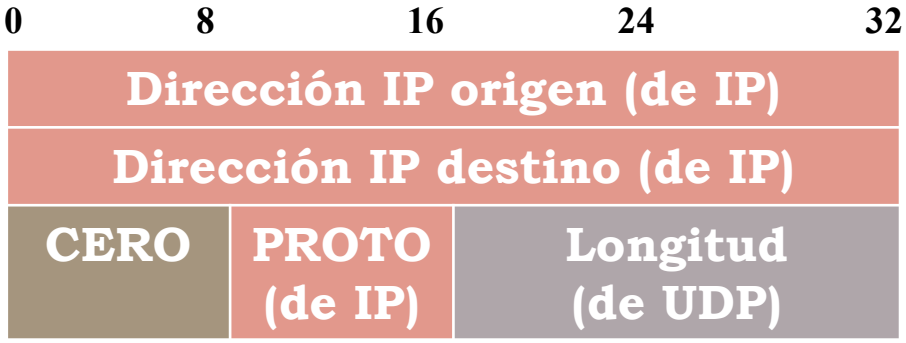
El protocolo UDP

- Características
  - Orientado al mensaje: envía datagramas
  - Protocolo no orientado a la conexión
  - No fiable
  - No incorpora control de congestión
- Servicio
  - **sendto**(puerto origen, puerto destino, datos)
  - **recvfrom**(puerto origen, puerto destino, datos)
- Extremos
  - Extremo 1: <udp,IPlocal,Plocal>
  - Extremo 2: <udp,IPremoto,Premoto>

El protocolo UDP

- Formato datagrama UDP (8 bytes)
  - Puerto origen y destino (local –opcional- y remoto)
  - Longitud total
  - Checksum aplicado todo el datagrama UDP + pseudo-cabecera IP (opcional - recomendado)
  - Datos del nivel superior



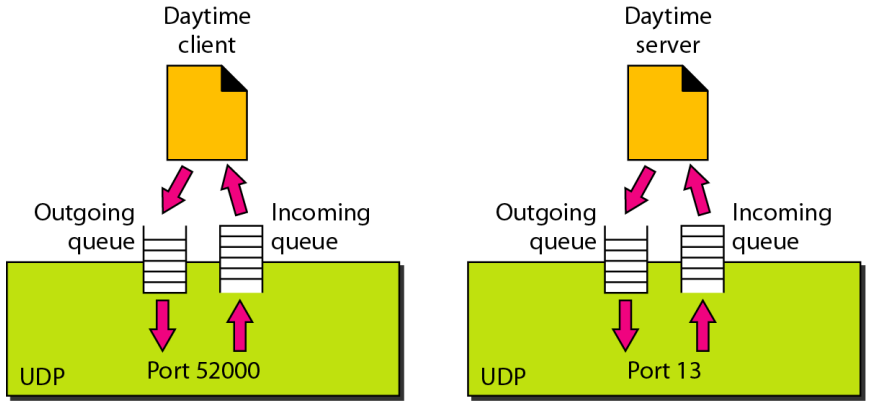


Pseudo-cabecera IP para el cálculo del checksum

(no se transmite, se crean con los datos existentes en el origen y destino del datagrama)

Funcionamiento de UDP

- No hay control de flujo, ni de errores, ni fase de conexión ni desconexión
- El mensaje es encapsulado en un datagrama IP
- Colas
  - Cada puerto lleva asociada **dos colas**: entrada y salida
  - UDP saca mensajes de las colas de salida, añade la cabecera y entrega a IP
  - Al llegar un mensaje, UDP comprueba si hay cola de entrada para el 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

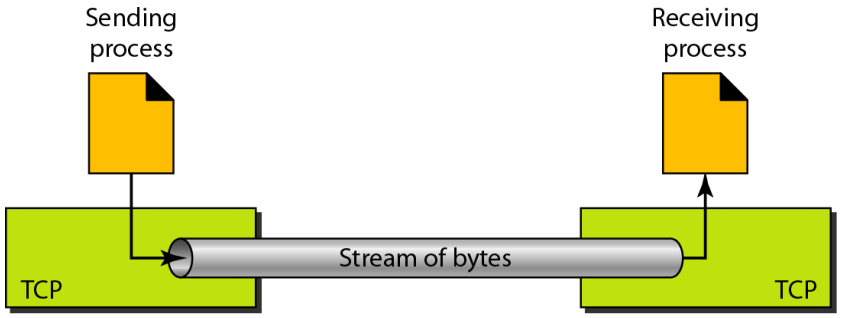


Protocolo UDP

- Ventajas frente a TCP
  - Más rápido (no requiere conexión, sin control de errores)
  - Menor penalización en la transmisión al tener una cabecera más corta
- Uso:
  - Envío de mensajes pequeños
  - Protocolos de nivel de aplicación tipo mensaje/respuesta (ej: daytime)
  - Protocolos de nivel de aplicación tolerantes a fallos
    - No preocupa la fiabilidad (ej: envío de datos multimedia)
  - Necesidad de envíos Multicast/broadcast

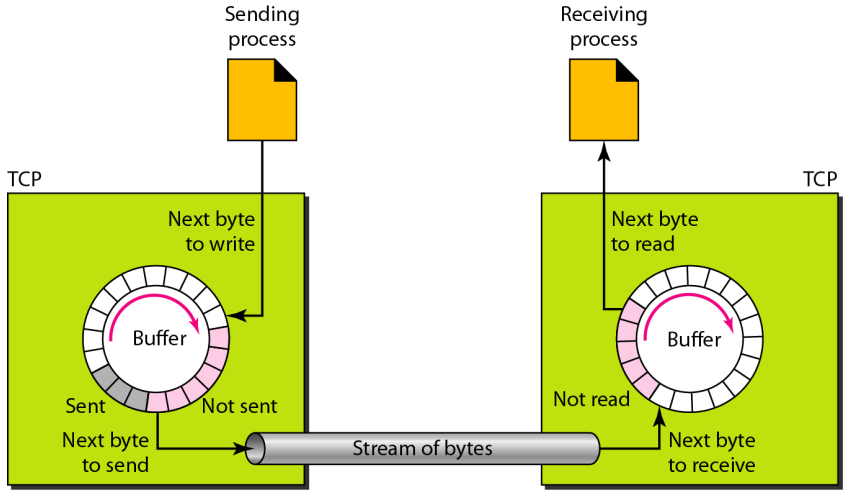
El protocolo TCP

- Envía segmentos TCP
- Características del servicio
  - 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
    - No orientado a datos estructurados (*stream* de bytes sin tipo)



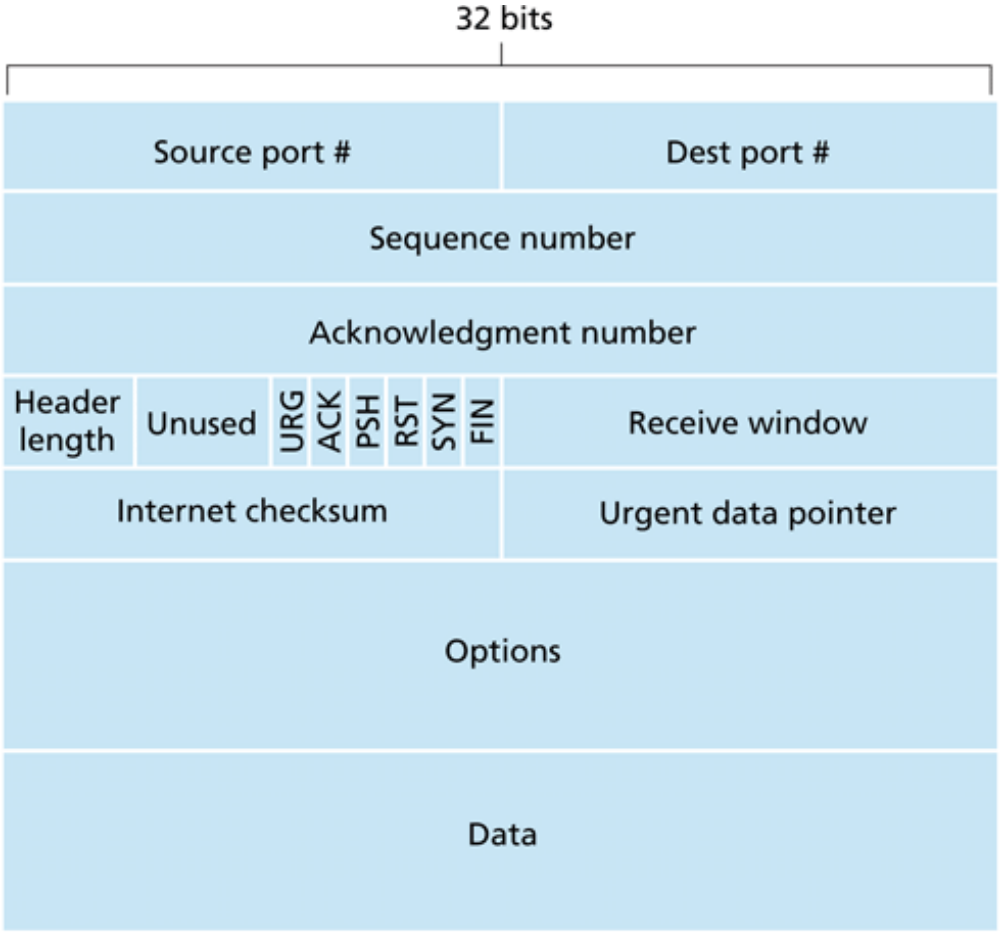
El protocolo TCP

- Buffering
  - Los segmentos TCP se envían a un buffer de envío
  - TCP enviará los datos en el buffer de envío cuando lo estime oportuno (distintos algoritmos – configurable)
  - Existe también un buffer de recepción (los datos se leen a petición del protocolo de la capa superior)
- MSS (*Maximum Segment Size*)
  - Máxima cantidad de datos que puede llevar un segmento TCP
  - Se negocia durante el establecimiento de conexión
  - Depende de la implementación
  - Ej.: 1460 bytes en Ethernet



El protocolo TCP

- Estructura del segmento TCP
  - Cabecera de 20 bytes + opciones

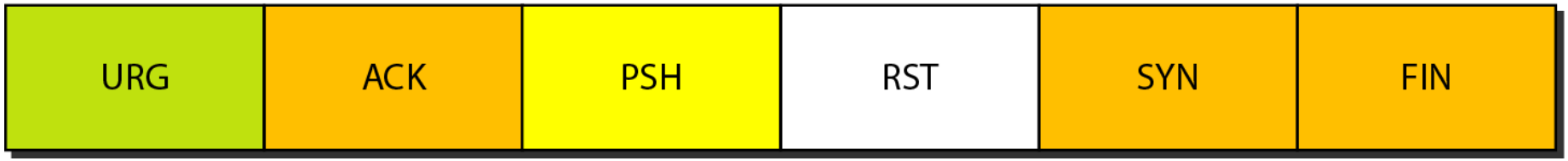


Campos

- 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 TCP (4)
    - En palabras de 32 bits
  - Tamaño de la ventana (16)
    - Para control de flujo
  - Bit ACK: el valor del campo ACK es válido
  - Bits RST, 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 connection



Números de secuencia y ACKs

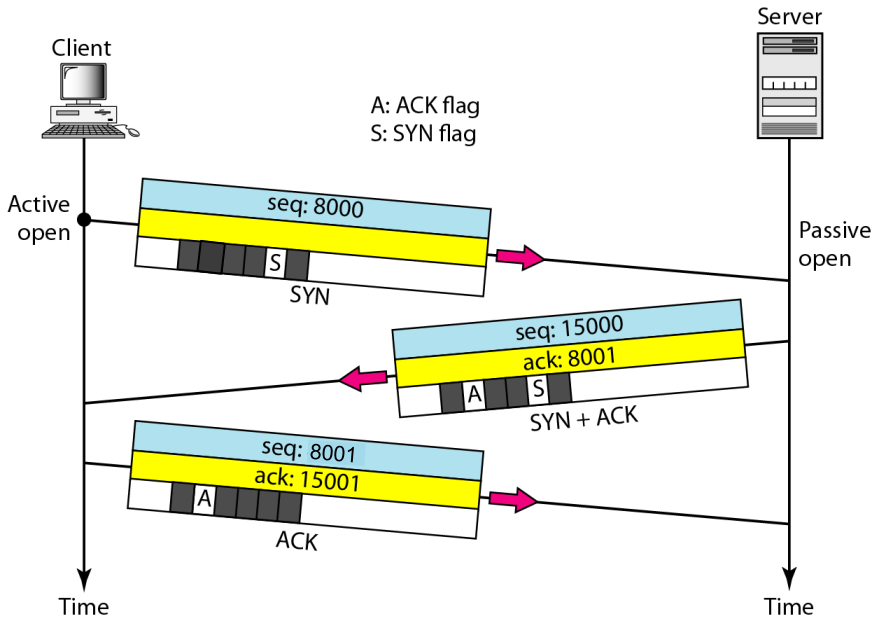
- Números de secuencia:
  - TCP utiliza el **número de byte** para numerar los datos
    - TCP numera todos los bytes de datos que se transmiten
    - La numeración no comienza en 0 → nº aleatorio [0 — 2^{32}-1]

Ejemplo:
Nº aleatorio = 1057
y el total de datos = 6000 bytes
→ Los bytes se enumeran de 1057 a 7056

    - El número de secuencia es el del primer byte que transporta el segmento (contando desde el valor de inicio)
    - El número de ACK (cuando el flag está activo) indica el siguiente byte que se espera recibir
  - Si un segmento no lleva datos de usuario no consume número de secuencia
    - Excepción: segmentos de control (SYN y FIN de conexiones)

Establecimiento de conexiones

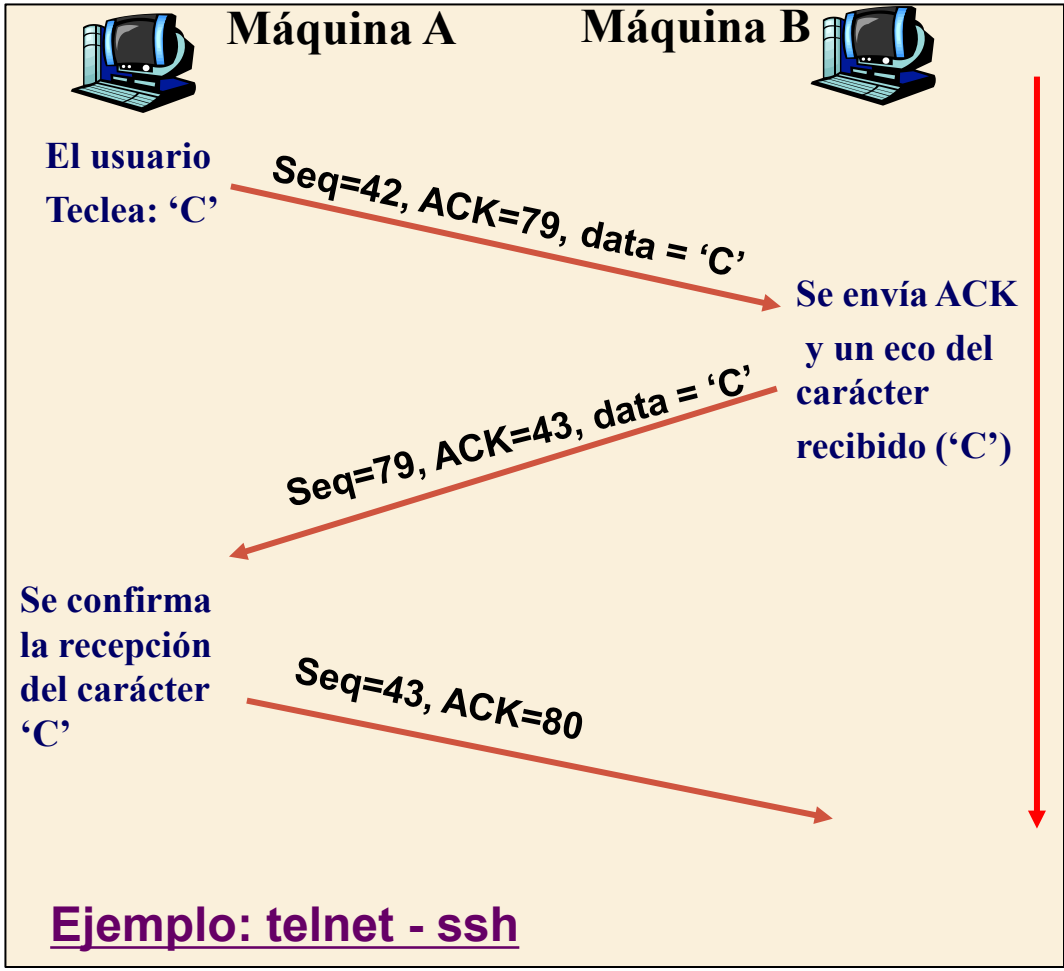
- Se usa un protocolo de negociación a tres bandas (*three-way handshake*)



  - El número inicial de secuencia se elige aleatoriamente
  - Los segmentos SYN y SYN+ACK no llevan datos pero consumen números de secuencia

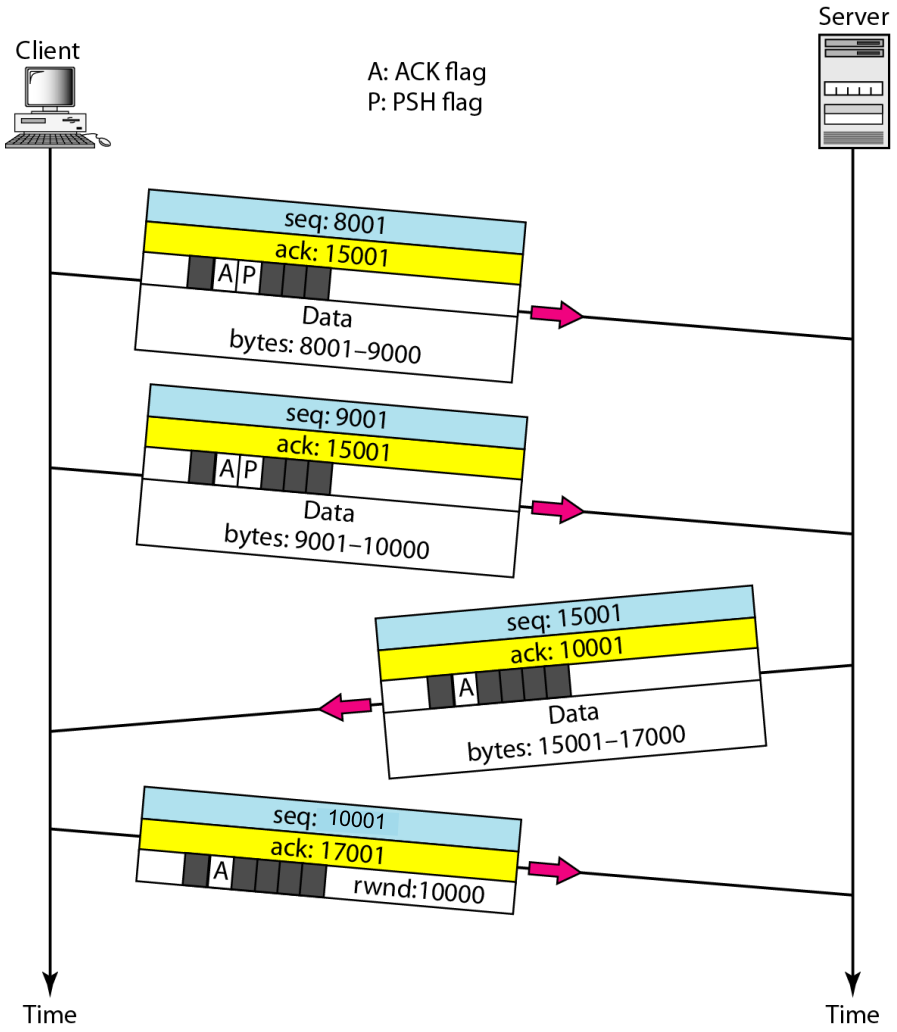
Transferencia de datos TCP

- ACKs:
  - Número de secuencia del siguiente byte que se espera recibir
  - ACK = SEQ + long(datos)



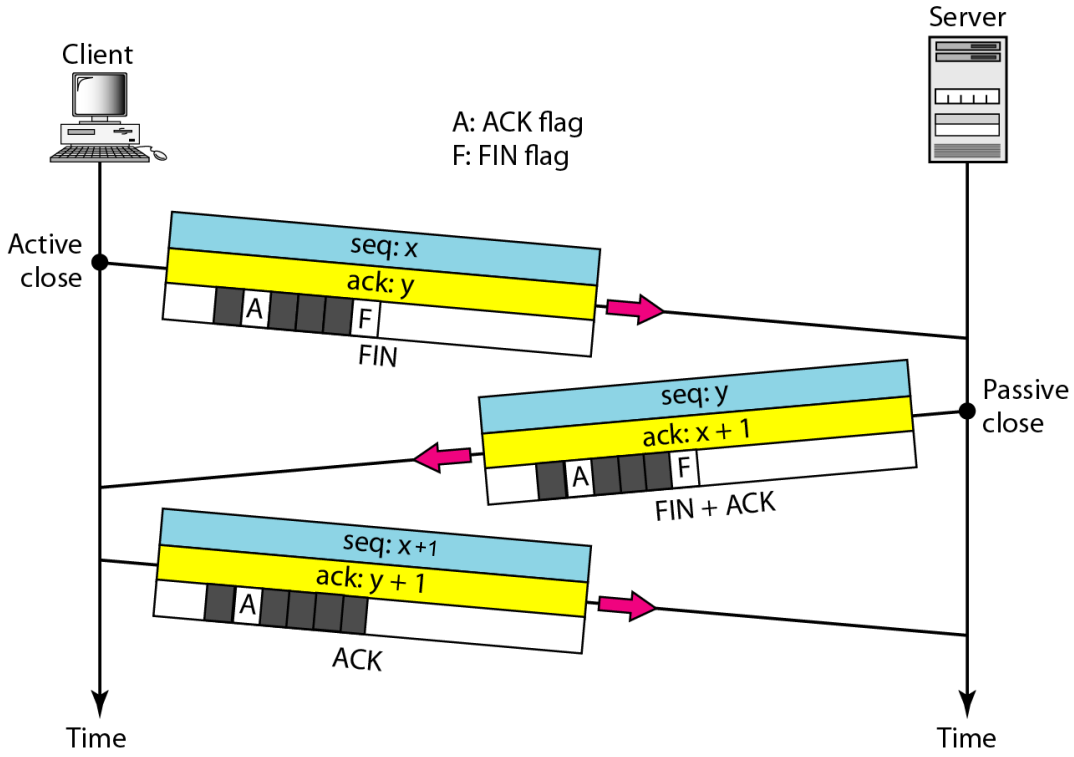
Transferencia de datos TCP

- Transmisión de datos bidireccional
  - Ambos extremos pueden enviar datos y confirmaciones
  - La confirmación se incluye con los datos (piggybacking)
- Envío inmediato de datos (pushing, PSH)
  - El emisor no debe esperar a completar el buffer
  - El receptor entrega los datos lo antes posible



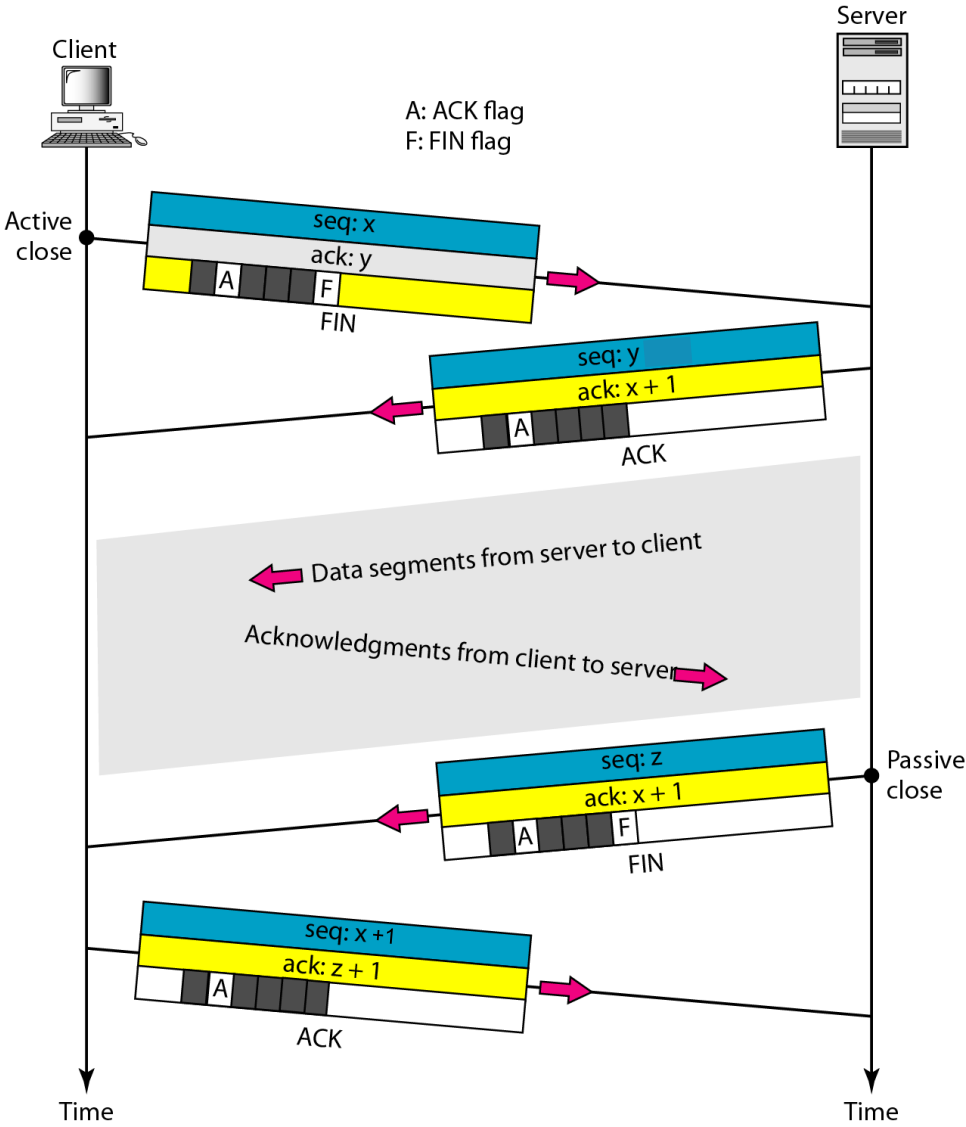
Finalización de las conexiones

- Cierre en tres pasos – *three-way handshaking*
  - Cualquiera de los dos extremos inicia la desconexión

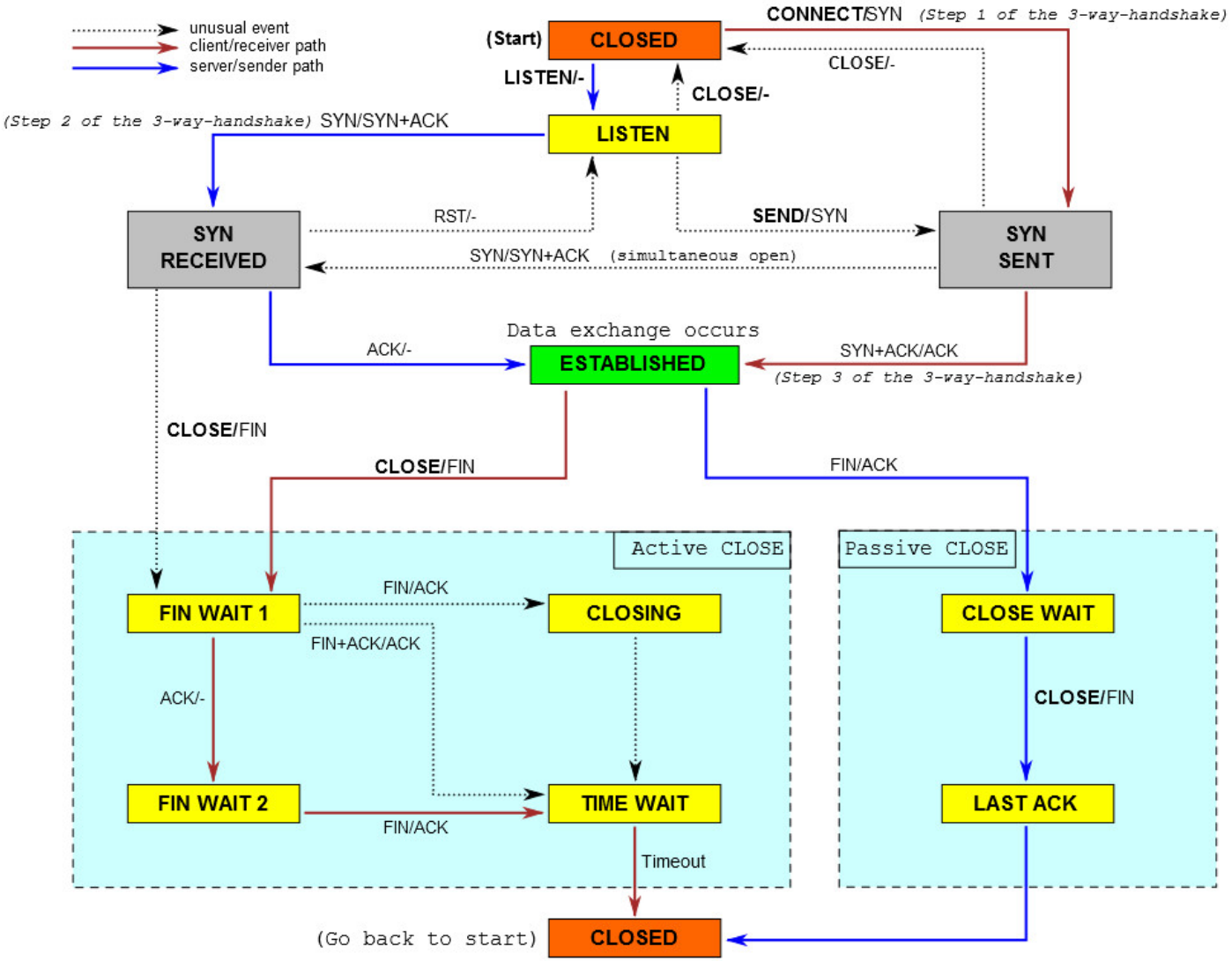


Finalización de las conexiones

- Semicierre
  - Uno de los extremos deja de enviar datos mientras sigue recibiendo
  - Actualmente incluso sin tener datos pendientes que enviar, se suele utilizar este esquema (difícil coordinar *close*)

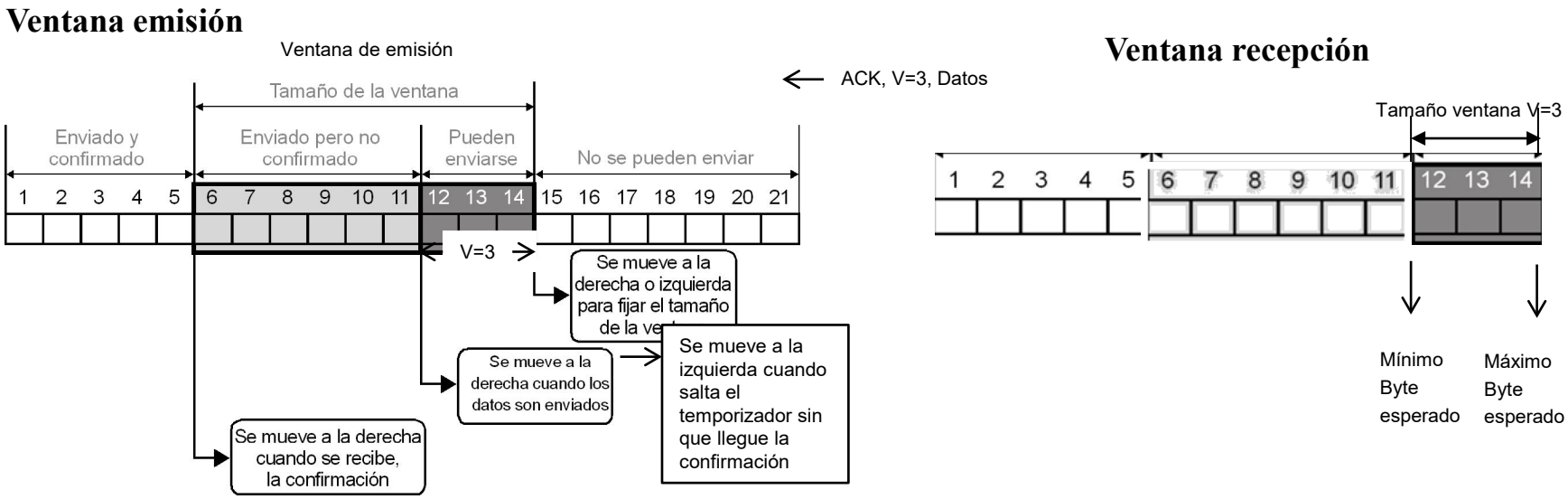


Máquina de estados TCP

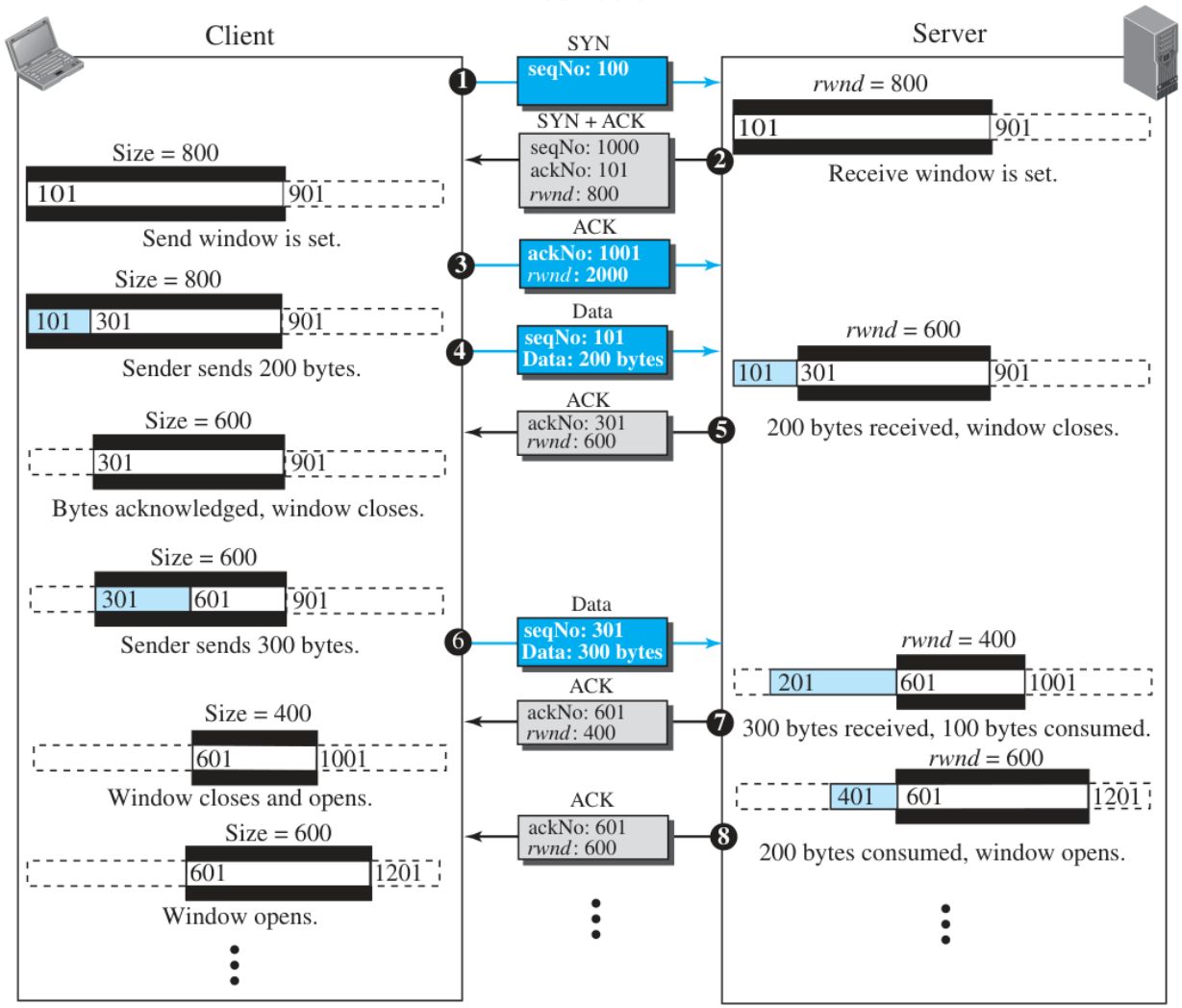


Control de flujo TCP: ventana deslizante (sliding window)

- Se usa una ventana deslizante para hacer la transmisión más eficiente
  - Las ventanas son orientadas a **byte**
  - 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 el número de bytes que puede aceptar antes de que su almacén se desborde y tenga que descartar datos
  - Hay 4 ventanas para una conexión full-dúplex



Control de flujo TCP: ventana deslizante (sliding window)

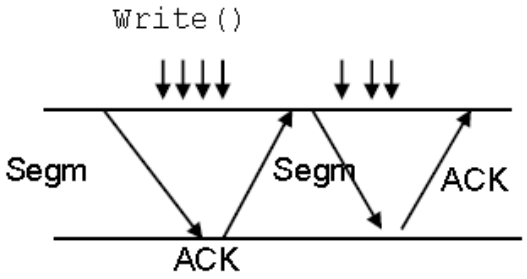


Control de Flujo: ajuste de la ventana

- Silly Window Syndrome
  - Se produce cuando el emisor o el receptor envían o procesan datos de manera muy lenta lo que provoca ajustes excesivos en las ventanas de transmisión y recepción.
  - Cuando la cantidad de datos de usuario enviados son menores que el tamaño de la cabecera de TCP (20 Bytes) decimos que la transacción está sufriendo el Silly Window Syndrome
  - Está provocado por implementaciones pobres de TCP

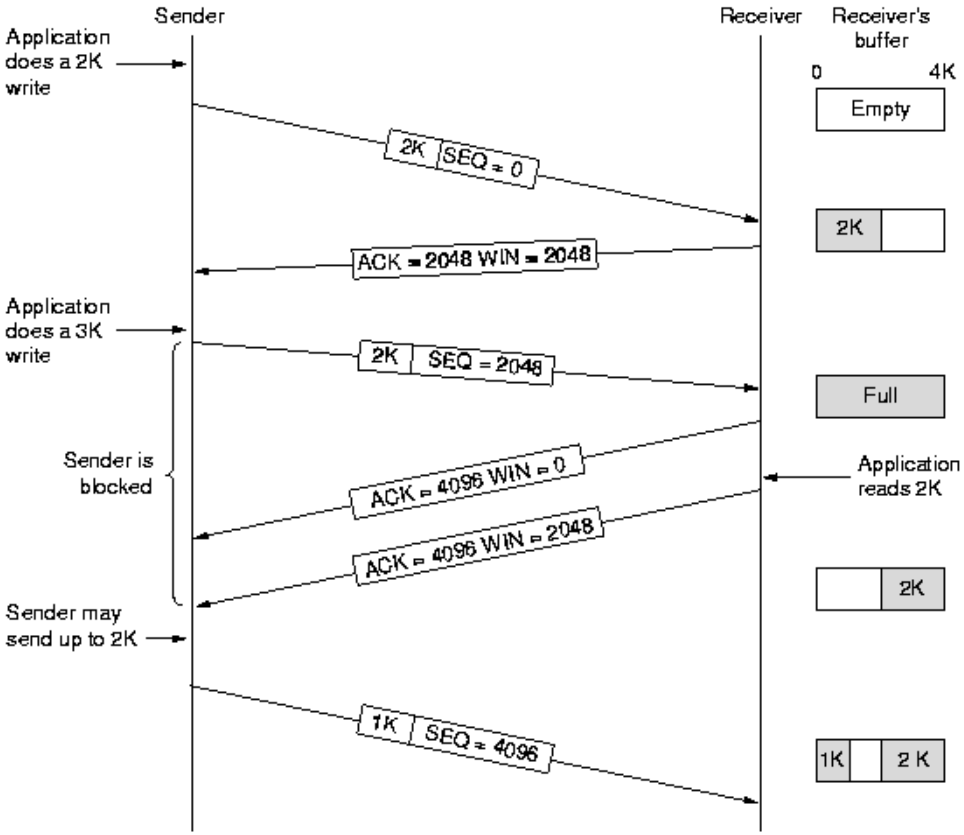
Control de flujo TCP: algoritmo de Nagle

- 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
- Algortimo de Nagle
  - Si no existen datos esperando ser confirmados => `write(datos)` independientemente del tamaño de los 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
  - Siempre respetando que long(datos)<=tamaño ventana de recepción (a.k.a. rwnd)



Control de flujo TCP: Solución de Clark

- Solución al Silly Window Syndrome en 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)`



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 acumulativa (positiva)
  - Retransmisión (con temporizadores)
  - Números de secuencia

Control de Errores en TCP

- Suma de comprobación de 16 bits
  - Detecta segmentos corruptos, que son descartados y considerados perdidos
- Confirmación
  - Indican la recepción de segmentos de datos
  - Los segmentos de control que no llevan datos pero si nº 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

Control de Errores en TCP

- Retransmisión después de un plazo de retransmisión (RTO – retransmission time out) **RFC 6298**
  - El emisor mantiene un temporizador RTO por conexión
  - Cuando vence el temporizador envía el primer segmento sin confirmar
  - No se activa un 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): usa su media (SRTT) y varianza (RTTVAR)

```
SRTT = (1-α).SRTT + α.R'  ; α=1/8 y R' es una nueva muestra
RTTVAR = β.|SRTT-R'| + (1-β).RTTVAR ; β = 1/4
RTO = SRTT+ MAX(G, 4*RTTVAR) ; Se sugiere G <= 100ms
Mínimo valor >= 1s y Máximo valor <= 60s
```

- 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)

Control de Errores en TCP

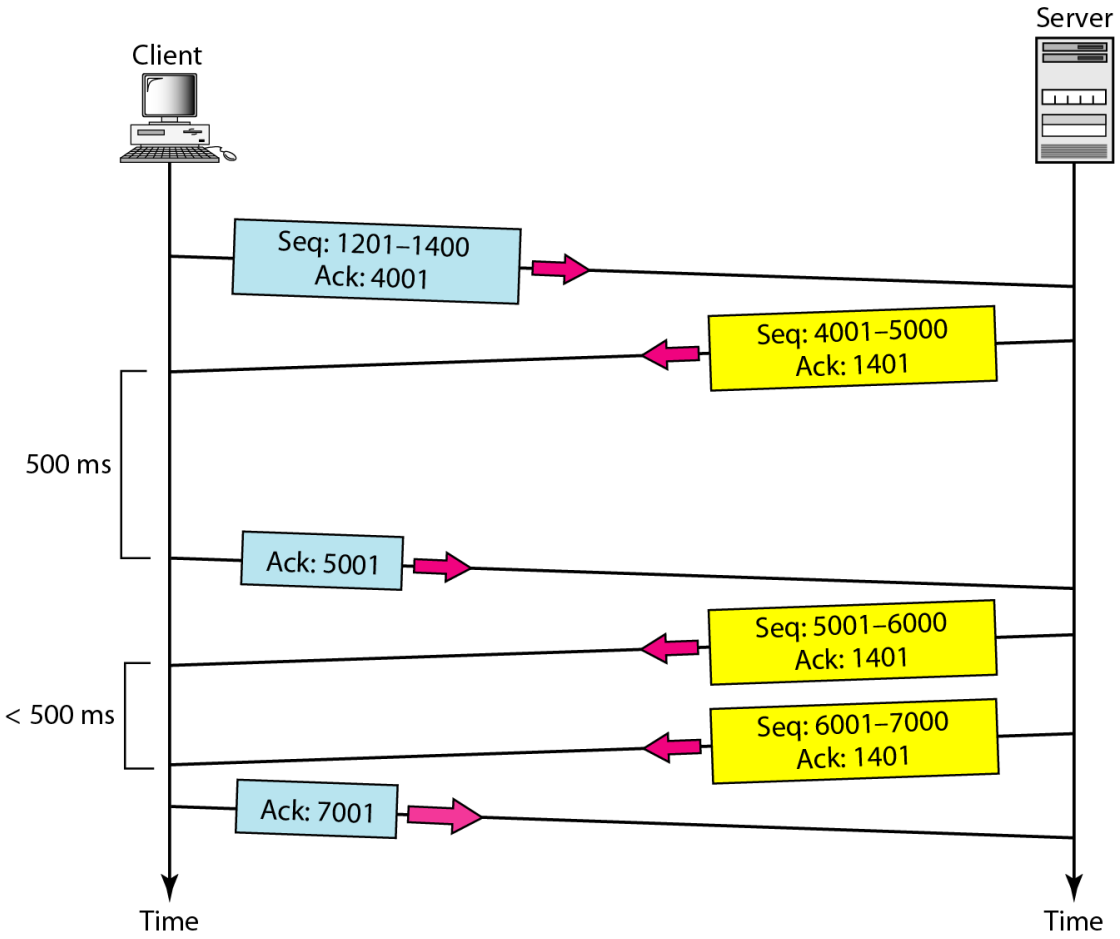
- Emisor TCP simplificado

```
1  sigNumSec=NumeroSecuenciaInicial;
2  baseEmision=NumeroSecuenciaInicial;
3
4  loop(true){
5      switch(suceso){
6          suceso: datos recibidos de aplicación superior
7              Crear segmento TCP con número de secuencia sigNumSec
8              if(el temporizador no está funcionando){
9                  iniciar temporizador
10             }
11             pasar segmento a IP
12             sigNumSec=sigNumSec+longitud(datos);
13             break;
14         suceso: fin de temporización del temporizador
15             retransmitir el segmento no reconocido con el número de secuencia más pequeño
16             iniciar temporizador
17             break;
18         suceso: ACK recibido, con valor de campo ACK igual a y
19             if(y>baseEmision){
20                 baseEmision=y;
21                 if(existen actualmente segmentos aún no reconocidos){
22                     iniciar temporizador
23                 }
24             }
25             break;
26     }
27 }
```

Control de Errores en TCP

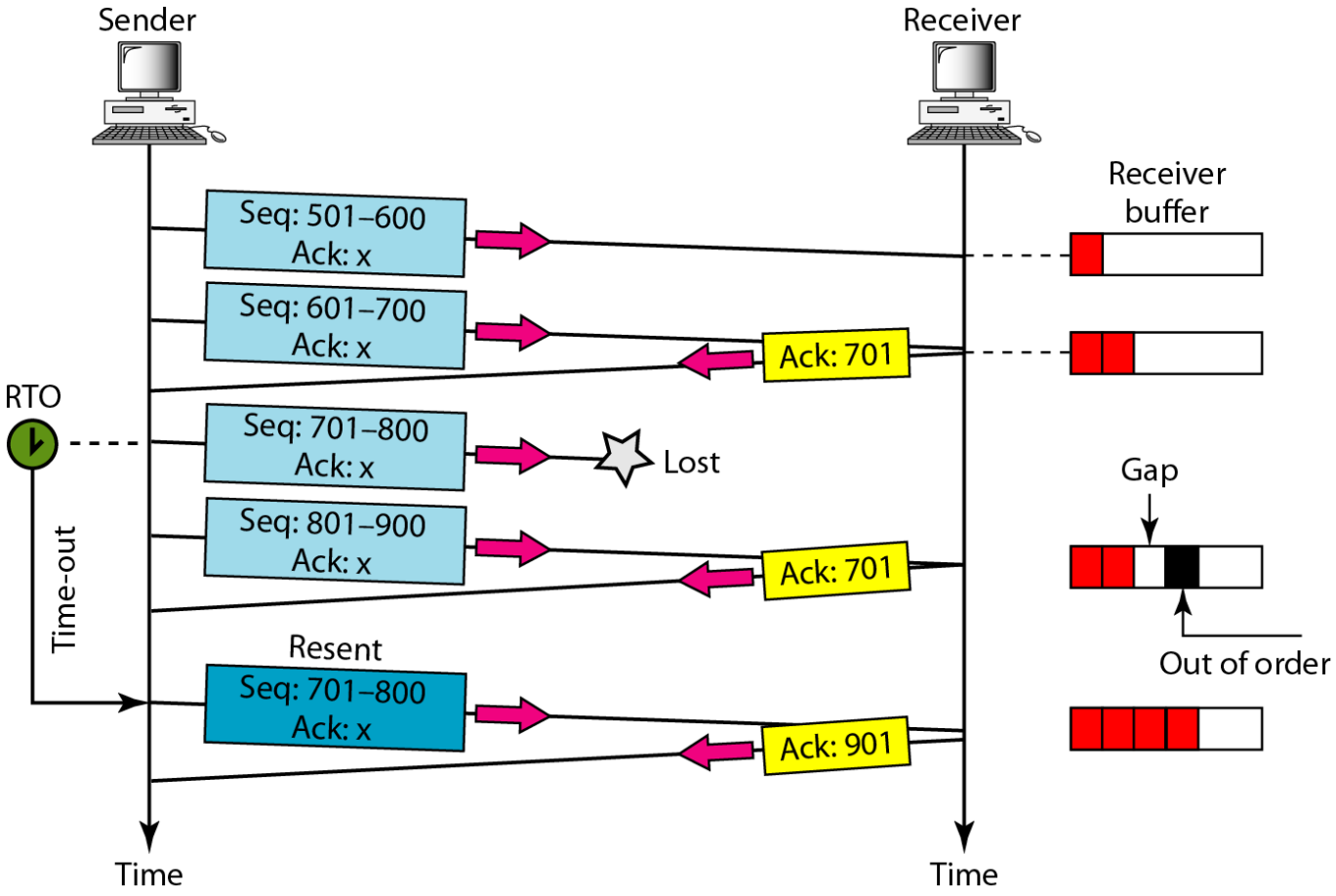
- Segmentos fuera de orden
  - Antes se descartaban los segmentos fuera de orden, que eran retransmitidos
  - Actualmente se almacenan temporalmente y se marcan como “fuera de orden” hasta que llega el segmento perdido
    - Al ser marcados, no se entregan al proceso destino
  - Ningún segmento fuera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

Funcionamiento de TCP: esquema sin errores



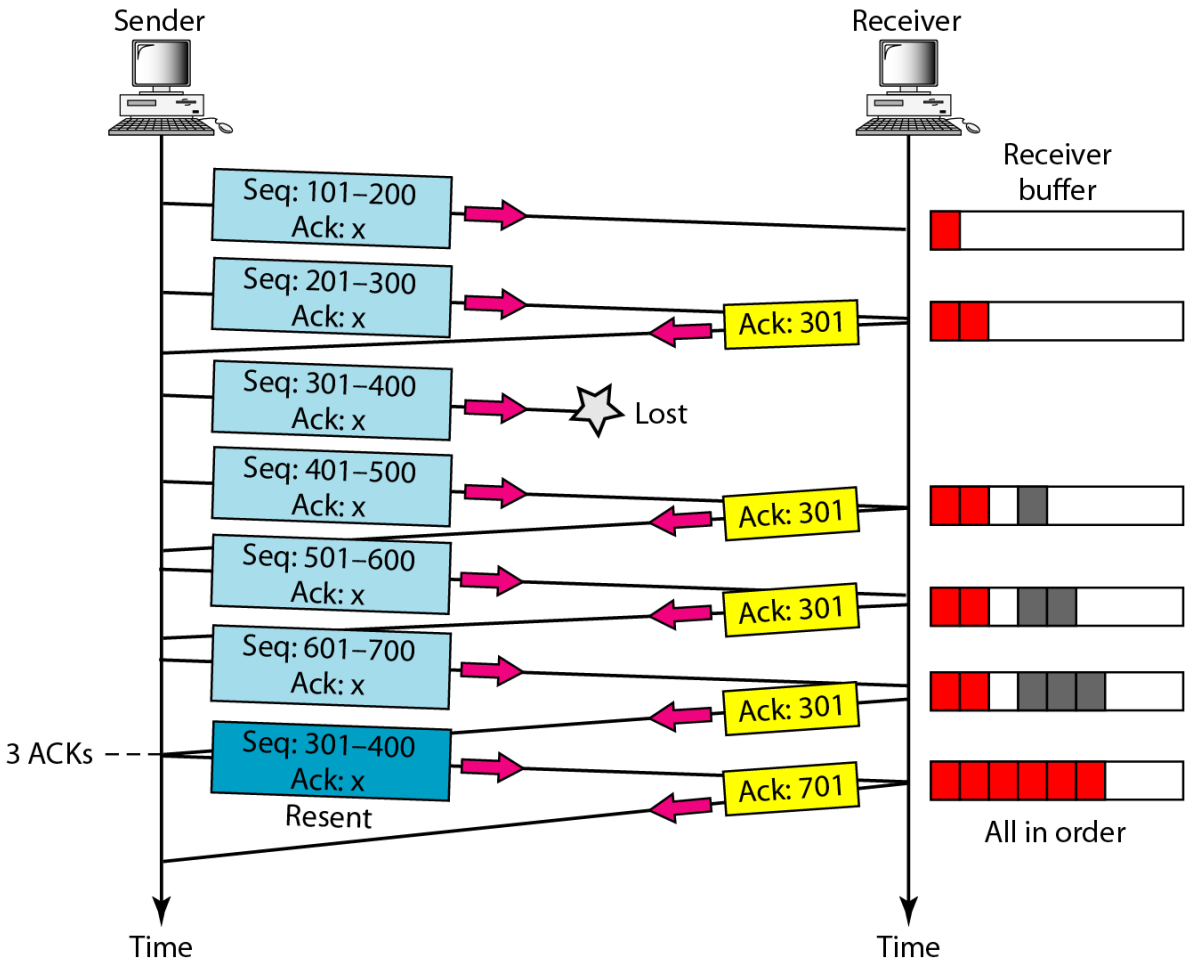
Funcionamiento de TCP: segmento perdido

- Retransmisión por temporizador



Funcionamiento de TCP: retransmisión rápida

- Retransmisión por triple ACK

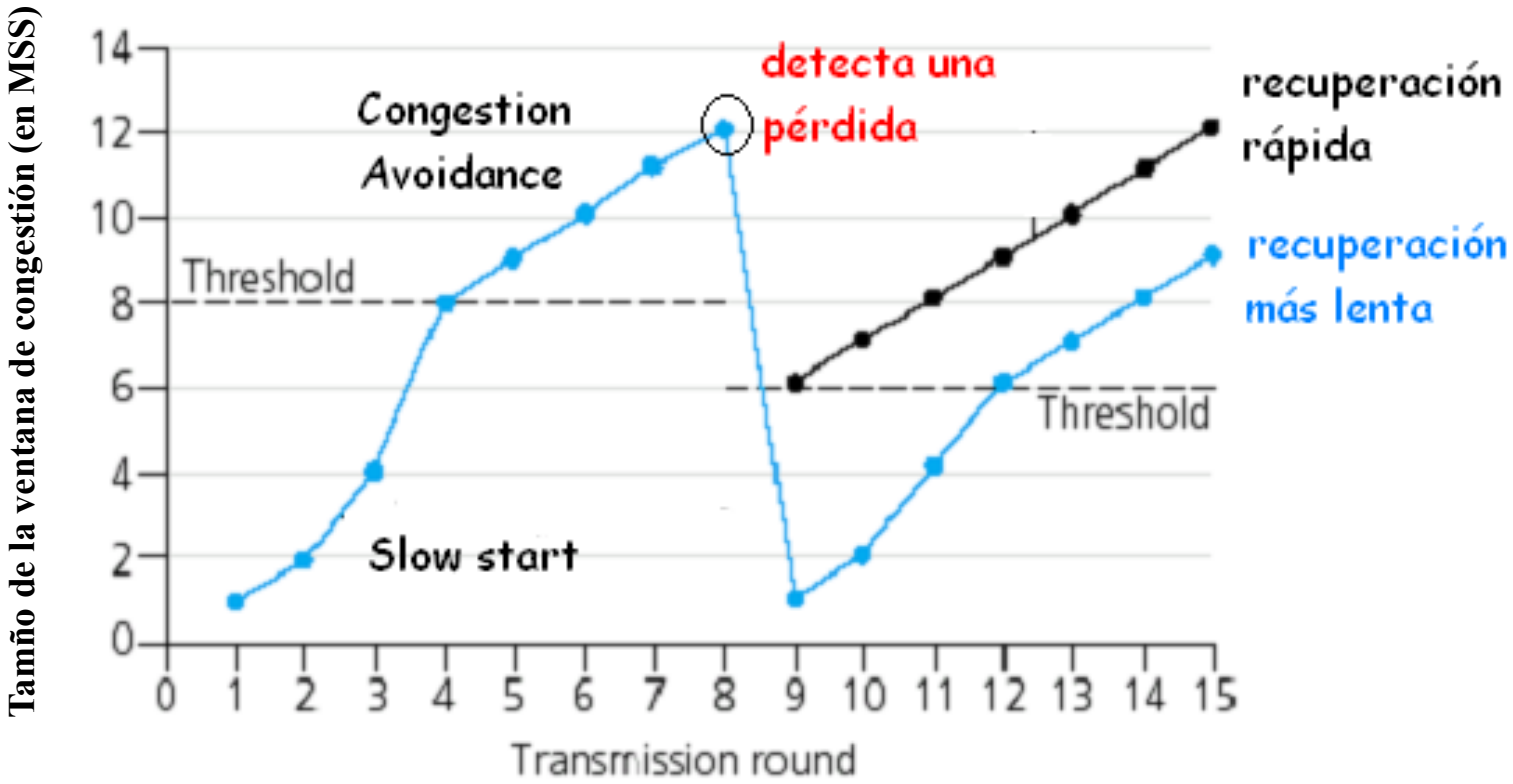


Control de congestión

- Conceptos diferentes
  - Control de flujo:
    - no enviar más paquetes de los que puede recibir el receptor
  - Control de congestión:
    - no enviar más paquetes en una parte de la red (subred)
- Control de congestión en TCP
  - La detección de congestión se basa en observar los siguientes hechos:
    - El aumento del RTT de los segmentos confirmados, respecto a anteriores segmentos.
    - La finalización de temporizadores de retransmisión
  - … son causados por congestión en la red
  - Uso de la ventana de congestión (cwnd) para controlar la velocidad de envío
    - Tamaño de la ventana útil = mínimo(rwnd,cwnd)

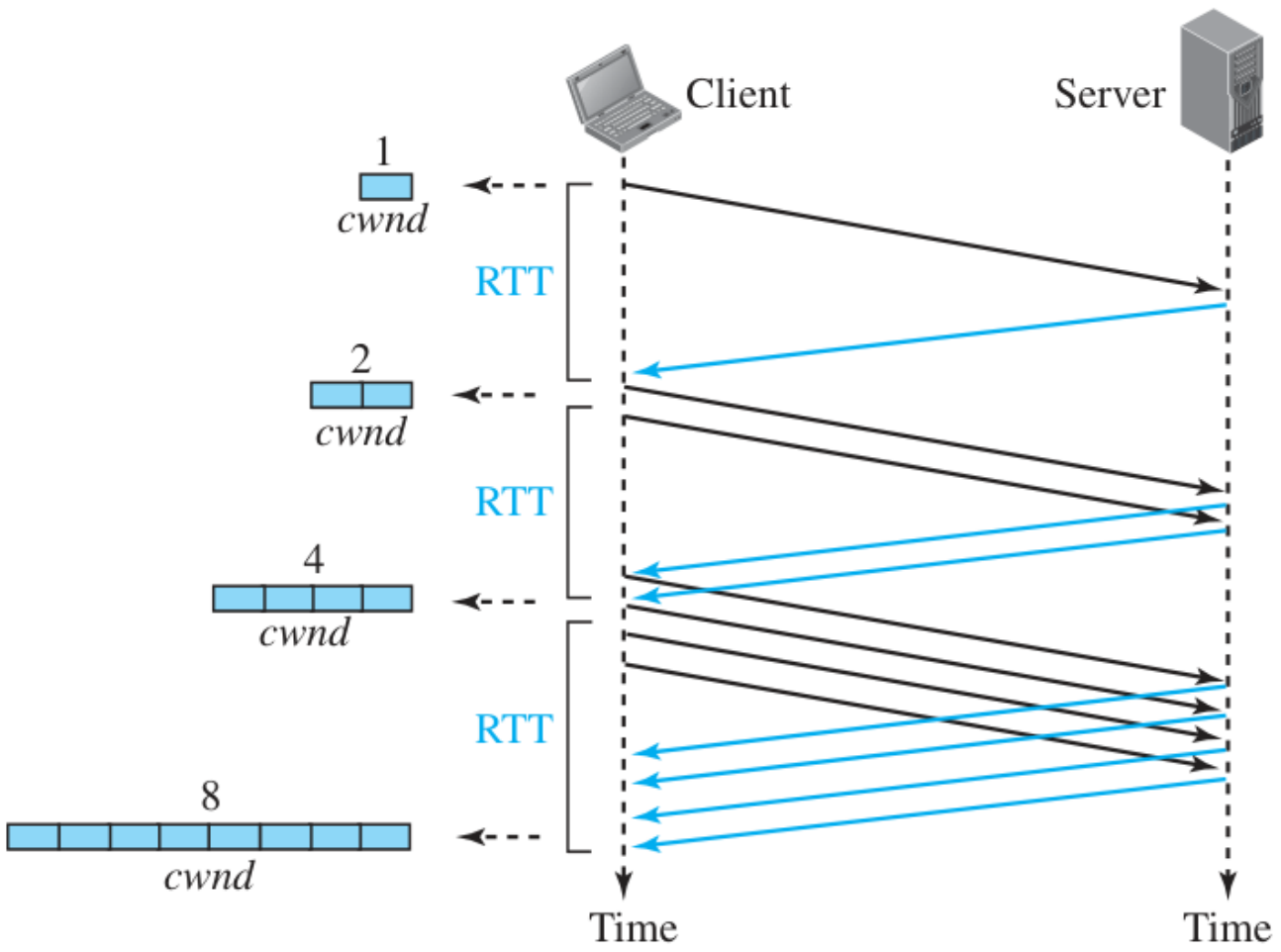
Control de congestión

- Algortimo de congestión de TCP (RFC 5681)
  - Está compuesto de 3 algoritmos: slow start, congestion avoidance (obligatorios) y fast recovery (opcional).



Control de contestión

- Slow start (inicio lento) – Incremento exponencial de ventana
  - El tamaño de la ventana de congestión incrementa en 1 MSS cada vez que llega 1 ACK.

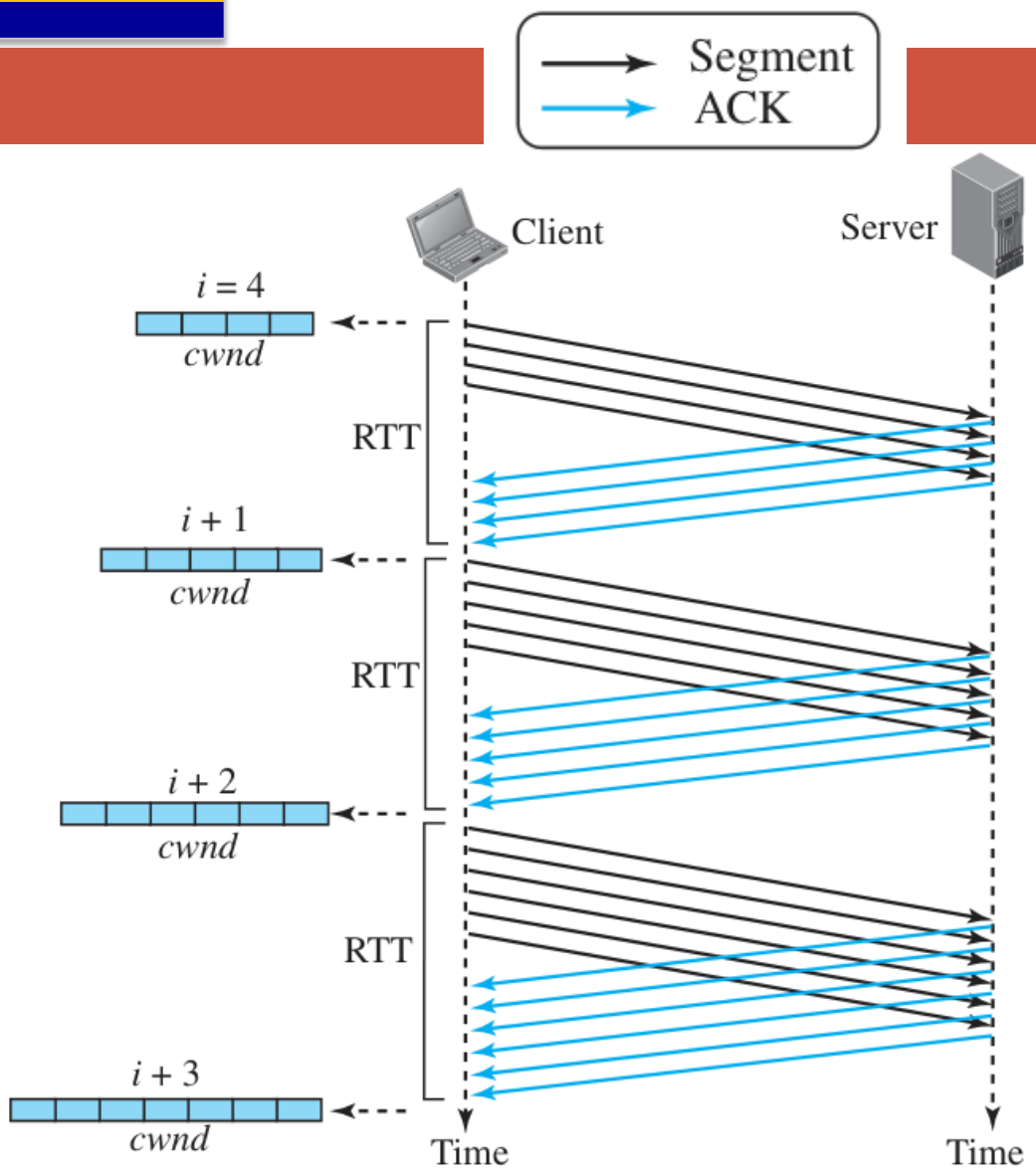


Control de congestión

- Slow start (inicio lento) – Incremento exponencial de ventana
  - TCP comienza con cwnd = 1 MSS
  - TCP envía 1 segmento
  - Se recibe una confirmación
    - Se libera 1 MSS del buffer de recepción
    - Se incrementa cwnd en 1 MSS por cada ACK
    - cwnd = cwnd + 1 MSS = 2 MSS
  - TCP envía 2 segmentos
    - Tantos como le permite su ventana
  - Se reciben 2 confirmaciones
    - Se liberan 2 MSS del buffer de recepción
    - Se incrementa cwnd en 1 MSS por cada ACK
    - cwnd = cwnd + 2 MSS = 4 MSS

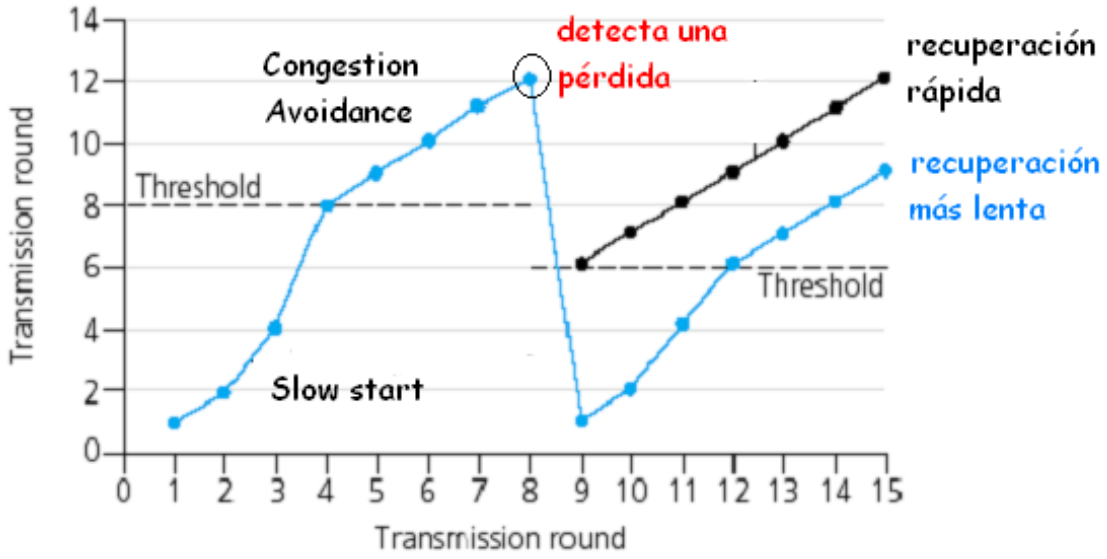
Control de la congestión

- Congestion avoidance (evitación de la congestión) – incremento aditivo
  - El tamaño de la ventana de congestión incrementa en 1 si se confirma toda la ventana enviada.
    - Si llega un ACK, cwnd = cwnd +(1/cwnd)*MSS



Control de la congestión

- Cambios entre algoritmos
  - TCP ejecuta inicialmente **inicio lento**, pero cuando cuanto **cwnd** llega a cierto **umbral** pasa a la fase **evitación de la congestión**
  - En cualquiera de las fases, si hay un evento de pérdida se vuelve a la fase de **inicio lento** con **cwnd = 1 MSS** y el **umbral** se actualiza al **cwnd/2** que provocó la perdida.



Protocolo SCTP

- El Stream Control Transmission Protocol (SCTP) fue definido por SIGTRAN de IETF (2000).
- Características:
  - Orientado al mensaje (como UDP).
  - Orientado a la conexión, confiable, con control de flujo y de congestión y secuenciación (como TCP).
  - Permite opcionalmente envío de mensajes fuera de orden.
- Ventajas:
  - *Multihoming* (posibilidad de varias IP en cada extremo de la comunicación).
  - Selección y monitorización de caminos (múltiples flujos).
  - Mecanismos de validación y protección contra ataques.
- Implementado en diversos sistemas tipo UNIX (Linux, Solaris, FreeBSD, ...).

Se verá en prácticas

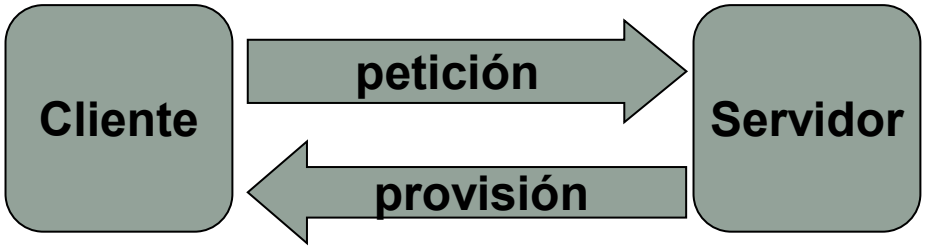
Programación básica de Aplicaciones Distribuidas (Sockets)

Esquemas de programación en TCP y UDP

PROGRAMACIÓN DISTRIBUIDA

Paradigma Cliente/Servidor

- La programación de aplicaciones distribuidas sigue el paradigma cliente/servidor.
- Vamos a diferenciar entre procesos clientes y servidores
- La estructura general de estas aplicaciones es:



Cliente:	**Servidor:**
... Petición_servicio() Servicio(...)	WHILE TRUE DO Espera_petición(...) Servicio(...) END

Paradigma Cliente/Servidor

- Los procesos servidores:
  - Son procesos permanentemente activos que ofrecen un servicio concreto siempre disponible para los usuarios del servicio
  - El servidor tiene una dirección fija y conocida
- Los procesos clientes
  - Piden un servicio en un momento dado
  - Se comunican sólo con el servidor
- Clientes y servidores de una misma aplicación se comunicarán mediante el intercambio de mensajes utilizando los servicios del nivel de Transporte
- El envío y recepción de mensajes se realiza a través de sockets

Programación con sockets

- Definición de socket
  - Un socket es un punto final (origen o destino) de comunicación entre procesos que se ejecutan en ordenadores diferentes
    - El canal que se establece es bidireccional (dúplex)
  - La interfaz socket es una API (*Application Programming Interface*) para realizar aplicaciones distribuidas sobre una red
- Dos tipos básicos de sockets
  - Sockets TCP: comunicación orientada a la conexión, fiable
  - Sockets UDP: comunicación no orientada a la conexión, no fiable

Alternativas para programa con sockets

- C/C++
  - La interfaz socket original estaba diseñada para ser usada desde C sobre sistemas UNIX
  - Tiene el inconveniente de ser compleja de usar
  - Problemas de portabilidad
  - Muy buen rendimiento
  - Posibilidad de acceder a toda la pila TCP/IP desde código
    - Ej: modificar una tabla de encaminamiento, manipulación de memoria caché ARP, configuración de las interfaces de red, etc.
- Java
  - Funciones de manejo de sockets simplificadas respecto a C
  - Código más corto y más legible
  - Código portable
  - "Problemas de rendimiento inherentes al lenguaje"
  - No es posible acceder a toda la pila TCP/IP desde código
    - Uso de "C" a través de JNI (*Java Native Interface*)
- Otras alternativas
  - C# (.NET), Python, etc.

Programación con sockets TCP en Java

- Características
  - Se utiliza el modelo cliente/servidor para establecer las conexiones
  - Una vez creado el canal de comunicaciones, los roles del servidor y del cliente dependen del programador
  - El servidor tiene que estar activo para que el cliente pueda establecer la conexión
  - Proporcionado en el paquete java.net
- Conceptos
  - ServerSocket: clase que usa el servidor para aceptar conexiones de los clientes
  - Socket de conexión: socket que se crea cuando se establece una conexión entre un cliente y un servidor

Programación con sockets TCP

- Acciones del servidor
  - Crear un socket Servidor
  - Esperar una solicitud de conexión
  - Crear el socket de conexión
  - Enviar/recibir usando el socket de conexión
  - Cerrar la conexión
- Acciones del cliente
  - Conectarse con el servidor (crea el socket de conexión)
  - Enviar/recibir usando el socket de conexión
  - Cerrar la conexión

Programación con sockets UDP

- Características
  - UDP proporciona un servicio sin conexión
  - No existe un canal de comunicaciones preestablecido
  - No existen los roles de servidor y cliente
  - Los procesos envían mensajes (datagramas)
  - Los datagramas pueden perderse, llegar en desorden, etc., por lo que el programador es responsable de implementar un protocolo de gestión de este tipo de errores
    - No adecuado para redes con probabilidad de error alta
  - Es más eficiente que TCP

Programación con sockets UDP en Java

- Acciones del servidor
  - Crear un socket
  - Recibir un mensaje (datagrama)
  - Enviar un mensaje de respuesta
- Acciones del cliente
  - Crear un socket
  - Enviar un mensaje (datagrama)
  - Recibir mensaje de respuesta
  - Cerrar el socket