T.4 CAPA DE TRANSPORTE

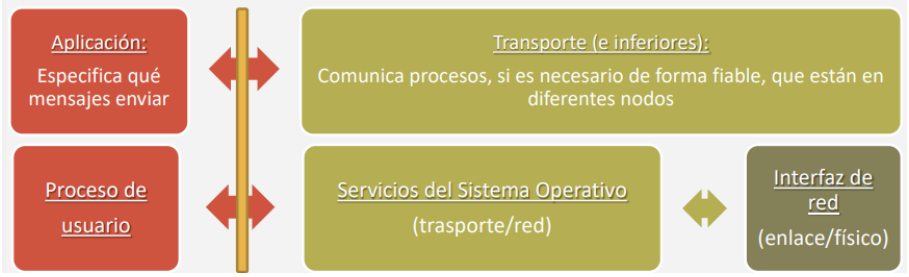
1. Contexto

Tenemos comunicación entre cualquier par de equipos …
¿Para qué? Compartir recursos y/o ofrecer servicios distribuidos (Capa de aplicación)
¿Por qué no es suficiente la capa de red para eso?
-Comunicación host-to-host, ¿cómo distinguir servicios?
Comunicación proceso-proceso. Direccionamiento de procesos.
-Datagramas de tamaño limitado
Flujos. Segmentación y reemsamblado
-Almacenamiento temporal
Colas – buffers
-Es best-effort y no asegura fiabilidad
Orden, control de errores, de flujo y de congestión

2. Protocolos de la capa de transporte

Los protocolos de transporte se ejecutan en los nodos terminales que se están comunicando
Objetivo:
-Ofrece al nivel superior (sesión o aplicación según la arquitectura de red usada) una comunicación extremo-a-extremo fiable
-Suplir (o no) todas las carencias que a nivel de control de errores presenta el protocolo de nivel de red utilizado
Protocolos de transporte disponibles en TCP/IP:
-Clásicos: TCP, UDP
-"Recientes": SCTP, UDP-Lite
-En desarrollo: QUIC
-Auxiliares: DCCP, RSVP, ¿TLS?

3. Interacción Transporte y Aplicación



4. Servicios que se ofrecen

Direccionamiento
Identificación de procesos
Seguimiento de la comunicación entre procesos origen y destino
Protocolos orientados a la conexión (ej: TCP)
- Transporte fiable de datos:
  - Control de orden de los mensajes
  - Confirmación de la recepción
  - Retransmisión en caso de pérdidas/errores
- Control de congestión y control de flujo
- Segmentación y reensamblado de datos (stream)

Protocolos no orientados a la conexión (ej: UDP)
Multiplexación:
- Múltiples comunicaciones en el mismo nodo

5. Orientado (o no) a la conexión

Protocolos orientados a la conexión (TCP)

```
socket.connect(destino);
...
data = socket.receive();
...
socket.send(info1);
socket.send(info2);
...
socket.disconnect(destino);
```

Creamos un socket conectado a un destino. Todos los envíos/recepciones van/vienen de aquí

Ya sabemos quién es el origen

Los envíos consecutivos pueden unirse y es el TCP quién decide cuántos segmentos TCP se envían y con qué información

Envío explícito de fin de conexión

Protocolos no orientados a la conexión (UDP)

```
socket.send(info, destino);
socket.send(info1, destino2);
socket.send(info2, destino2);
...
[data,origen] = socket.receive();
```

- No conexión
- Cada envío/recepción puede ir/venir de un sitio diferente.
- Cada envío genera un datagrama UDP

Orientado a la conexión:

Ventajas: Fiabilidad y controles avanzados (congestión y flujo)

Desventajas: Sobrecarga (tamaño de cabecera, control, más mensajes –conexión y confirmaciones-) y solo mensajes punto a punto

Uso: Aplicaciones que requieran entregas correctas

No orientados a la conexión:

Ventajas: Poca sobrecarga y permite comunicación multicast

Inconvenientes: La fiabilidad/congestión/flujo se deja a la aplicación (si es necesario)

Uso: Aplicaciones con pérdidas asumibles, comunicaciones a grupos o que sean simples pares petición/respuesta cortos

6. Paradigmas de interacción en Internet

Modelo Cliente – Servidor:

Dos roles diferenciados

Servidor:

- Ofrece un servicio
- Proceso en ejecución continua
- Recibe peticiones

Cliente:

- Requiere un servicio
- Proceso temporal
- Envía las peticiones

Modelo P2P (peer to peer):

Los procesos puede tener cualquiera de los dos roles

Al inicio sí deben tomar roles (quién se conecta a quién), pero sin estar designado a priori y en la comunicación no se diferencian roles

7. Direccionamiento

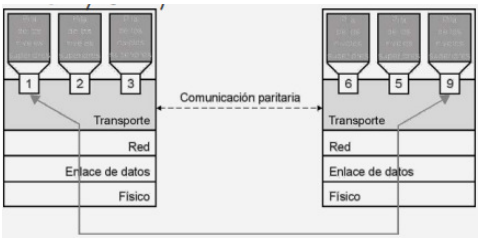
Un proceso de la computadora local –cliente– necesita servicio desde un proceso remoto –servidor–

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

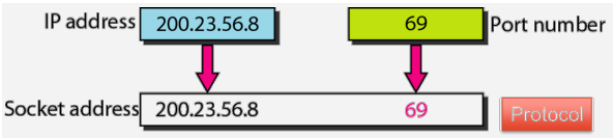
- Puerto efímero (normalmente elegido de entre los dinámicos)

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

La comunicación proceso a proceso necesita 3 identificadores

- Protocolo
- Dirección IP
- Número de puerto

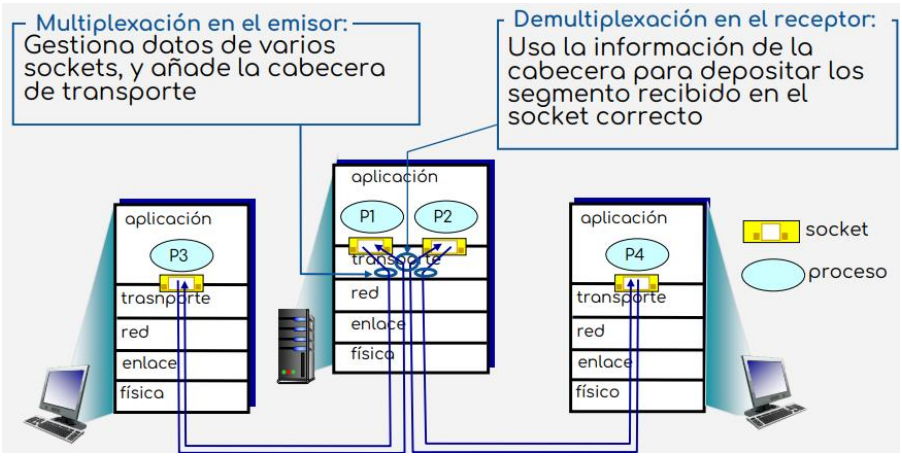


La combinación se denomina dirección de socket:
<protocolo, IP, puerto>

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)

8. Multiplexación / demultiplexación

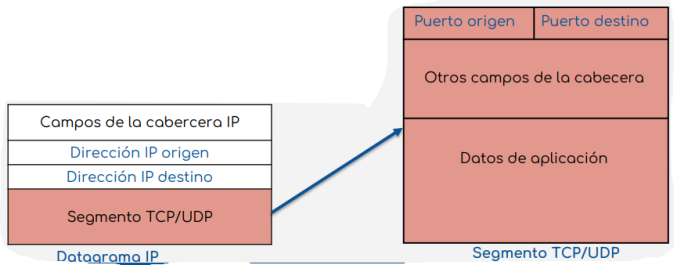


Funcionamiento de la demultiplexación

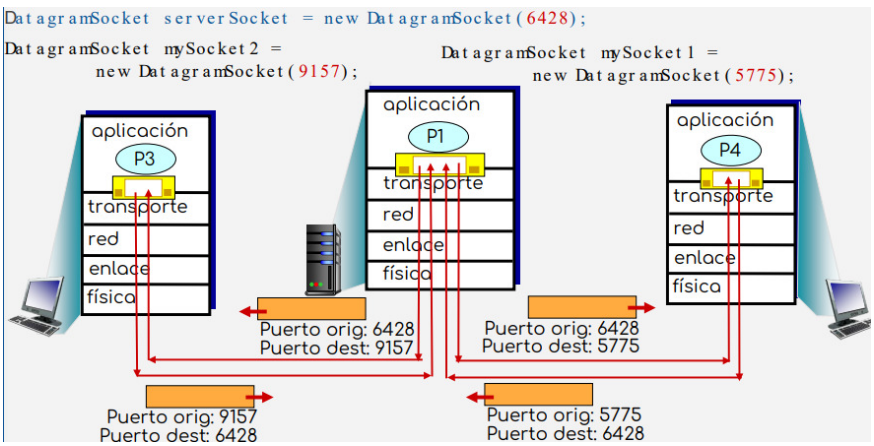
Los host reciben datagramas IP:

- -Una dirección IP origen y una dirección IP destino
- -Un segmento de la capa de transporte
- -Ese segmento tiene un puerto origen y un puerto destino

El sistema utiliza las IPs y los puertos para asignar el segmento al socket



Demultiplexación en sockets sin conexión

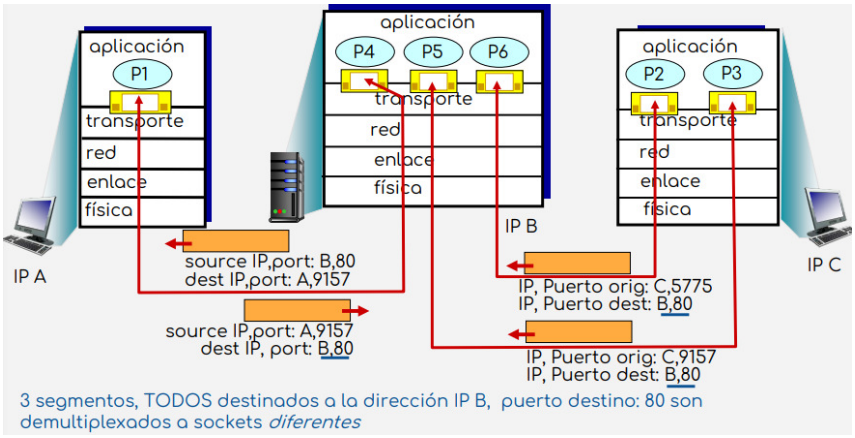


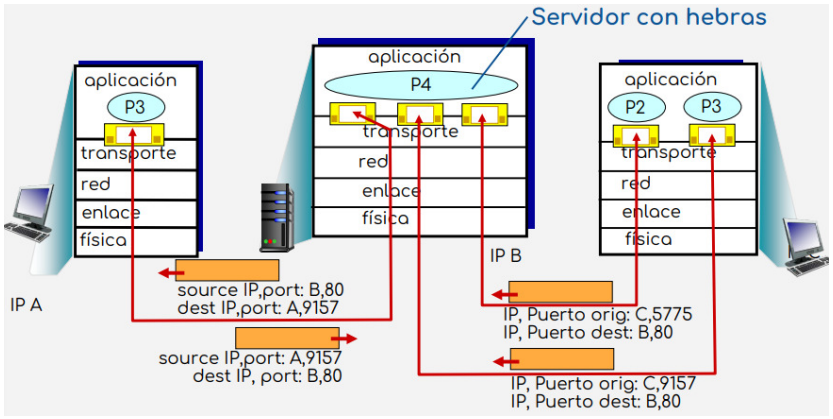
Un socket TCP se identifica con la 4-tupla:

-Dirección IP origen
-Puerto origen
-Dirección IP destino
-Puerto destino

Demultiplexación: El receptor usa los 4 valores para dirigir el segmento al socket apropiado
Los servidores tienen diferentes sockets para cada cliente conectado
Los hosts que albergan servidores pueden soportar muchos sockets TCP simultáneos:

Cada socket se identifica por su propia 4-tupla

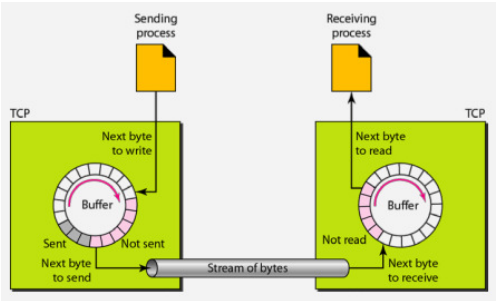




Servicio orientado a la conexión: Segmentación

La capa de aplicación requiere enviar grandes cantidad de datos
Podemos recibir muchos datos y que el servicio destinatario esté ocupado y no puede recoger los datos
Uso de colas (buffers):

Creación
¿Por cliente o por servicio?
Manejo

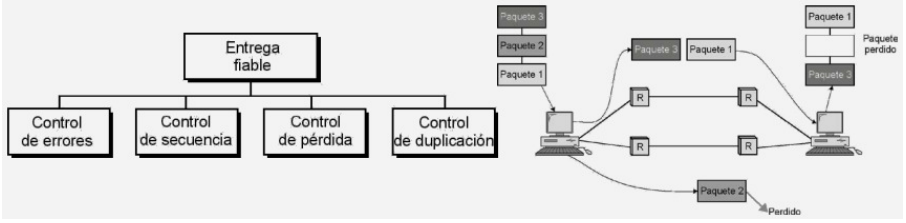


Servicio orientado a la conexión: Errores

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

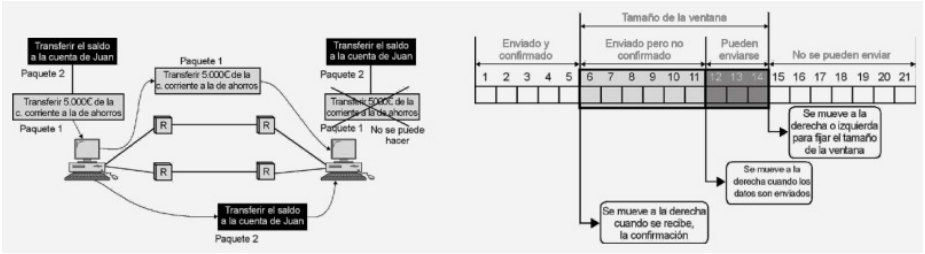


Servicio orientado a la conexión: Flujo

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

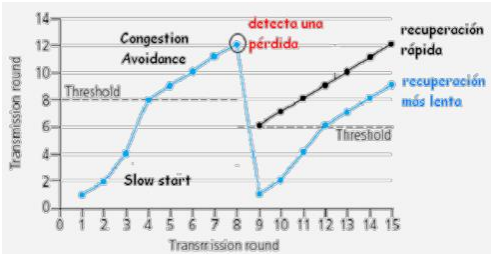


Servicio orientado a la conexión: Congestión

Es posible controlar la congestión extremo a extremo:

Permitir al emisor limitar la cantidad de información enviada para evitar saturar las redes por las que pasa la información

Habitualmente se basan en limitar los envíos cuando se detectan pérdidas u otro tipo de fallo en las comunicaciones



El protocolo UDP

Características

-Orientado al mensaje

-Envía datagramas

-Protocolo no orientado a la conexión

-No fiable

-No incorpora control de flujo ni 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>

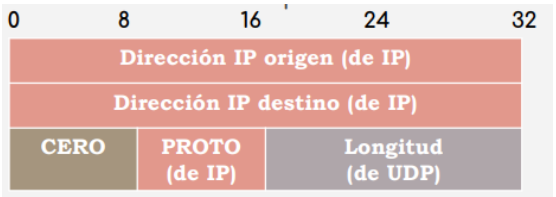
Formato datagrama UDP (8 Bytes)

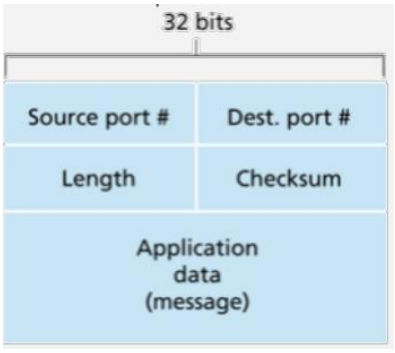
Puerto origen y destino (local –opcional- y remoto)

Longitud total

Checksum aplicado a todo el datagrama UDP + pseudo-cabecera (opcional pero recomendado)

Datos de nivel superior





Pseudo-cabecera para el cálculo del checksum. No se transmite, se crean con los datos existentes en el origen y destino del datagrama

Funcionamiento de UDP

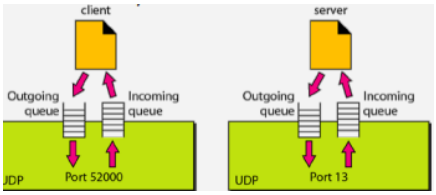
No hay control de flujo / error ni fase de conexión o desconexión El mensaje es encapsulado en un datagrama IP

Uso de 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
- Sí: deposita el mensaje al final de la cola
- No: descarta el datagrama y pide a ICMP que envíe un mensaje de puerto no alcanzable

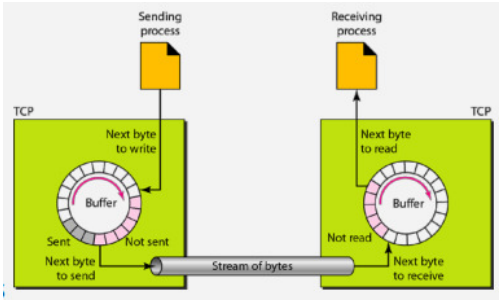
-Todos los mensajes enviados por un mismo puerto son encolados en la misma cola de salida (independientemente del destino)
-Todos los mensajes recibidos por un mismo puerto son encolados en la misma cola de entrada (independientemente del origen)



Ventajas frente a TCP
Más rápido (no requiere conexión, sin control de errores)
Menor penalización en la transmisión al tener una cabecera más corta

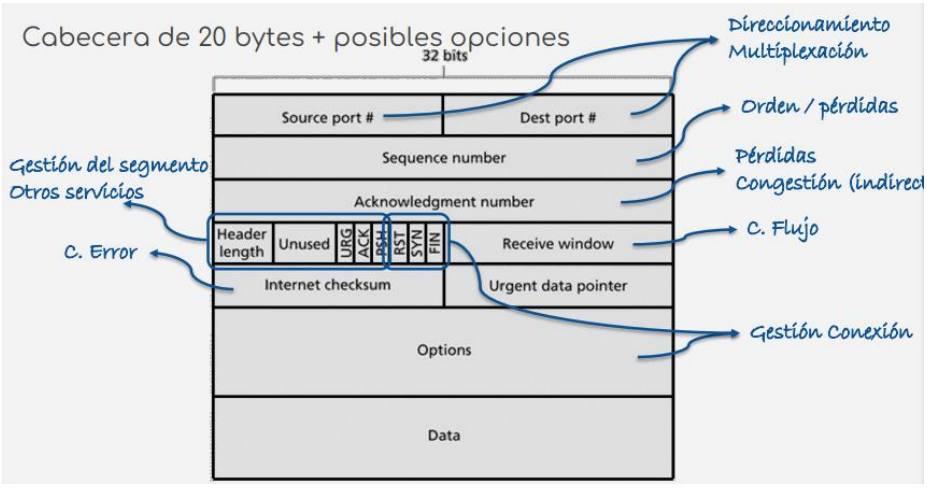
Uso:
Envío de mensajes pequeños
Protocolos de nivel de aplicación tipo mensaje/respuesta (ej: daytime, DNS…)
Protocolos de nivel de aplicación tolerantes a fallos
no preocupa la fiabilidad (ej: envío de información multimedia)
Necesidad de envíos multicast/broadcast (DHCP)

Transmission Control Protocol (TCP)
Envía segmentos TCP
Características del servicio
-Protocolo orientado a la conexión
-Las conexiones son full-dúplex
-Canales punto a punto
-Ofrece un servicio fiable
-Conexión orientada a flujos de bytes



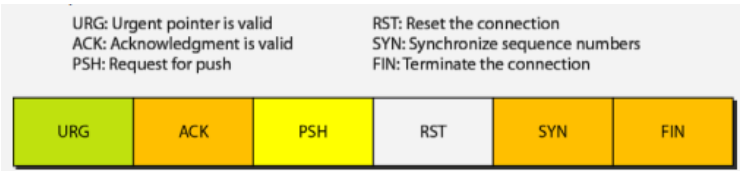
- No mensajes ni datos estructurados (stream de bytes sin tipo)
- Los datos se envían a un buffer de envío
- TCP enviará los datos del buffer de envío cuando lo estime oportuno
- Existe también un buffer de recepción
- MSS: Máxima cantidad de datos en un segmento TCP:
Se negocia durante el establecimiento de la conexión

Segmento TCP



Cabecera TCP: Campos
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

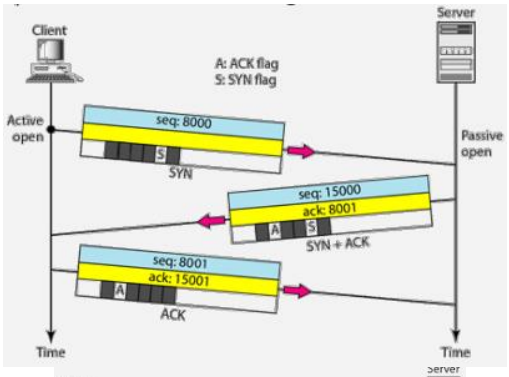


Números de secuencia y ACKs
TCP utiliza el número de byte para numerar los datos
    TCP numera todos los bytes de datos que se transmiten
    La numeración no comienza en 0. Número aleatorio en [0,232-1]
    El número de secuencia es el del primer byte que transporta el segmento:

Ejemplo: Nº aleatorio = 1057
Primer segmento: total de datos = 6000 bytes
→ Los bytes se enumeran de 1057 a 7056
→ Nº de secuencia del segmento: 1057
Segundo segmento: total de datos = 3000 bytes
→ Los bytes se enumeran de 7057 a 10056
→ Nº de secuencia del segmento: 7057

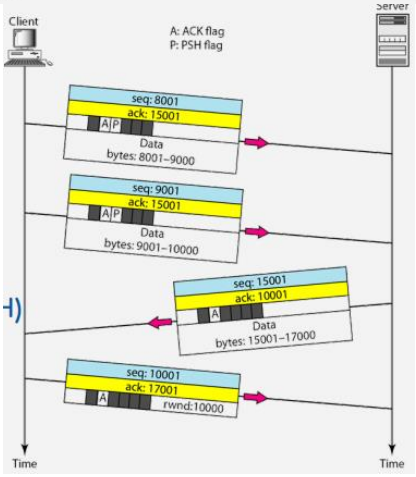
Un segmento sin datos no consume números de secuencia:
    Excepción: segmentos con bits SYN o FIN activos (control de conexión)

Establecimiento de la conexión
Se usa un protocolo de negociación a tres bandas (three-way handshake):

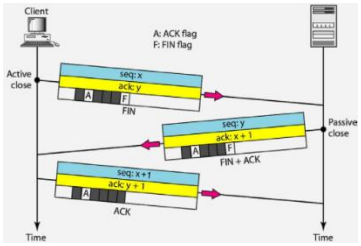


El número inicial de secuencia se elige aleatoriamente
Los segmentos SYN y SYN+ACK no llegan datos pero consumen un número de secuencia

Transmisión de datos
Confirmaciónes (ACKs):
    Número de secuencia del siguiente byte que se espera recibir
    # ACK = # SEQ + longitud(datos)
Transmisión de datos bidireccional:
    Ambos extremos pueden enviar datos y confirmaciones
    La confirmación va con los datos
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 la conexión

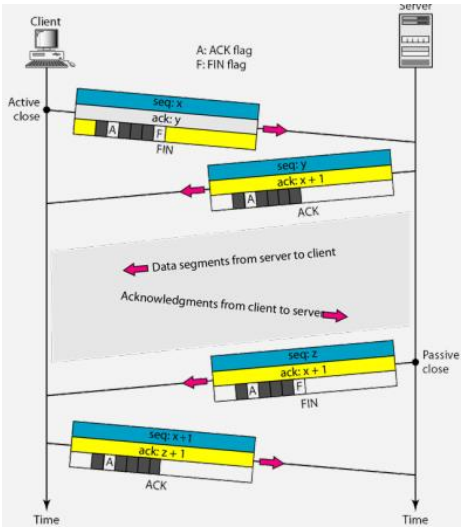


Cierre en tres pasos – three-way handshaking
Cualquiera de los dos extremos puede iniciar la desconexión
Los mensajes con el bit FIN consumen un número de secuencia

Semicierre :

Uno de los extremos deja de enviar datos mientras sigue recibiendo

Incluso sin tener datos pendientes, se suele utilizar este esquema ya que es difícil coordinar los cierres



Control de errores en TCP

TCP incluye mecanismos para detectar y corregir errores:

Corruptos
Perdidos
Duplicados
Fuera de orden

Los mecanismos son:

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

Suma de comprobación (checksum de 16 bits). Detecta segmentos corruptos (se descartan y se consideran perdidos)

Confirmaciones

-Indican la recepción de segmentos de datos
-Los segmentos de control que no llevan datos pero sí consumen número de secuencias (SYN y FIN) se confirman
-Los segmentos que solo sean ACK no se confirman

Retransmisión

Dos posibles escenarios:

- Expira un temporizador
- Se reciben 3 ACKs repetidos para el mismo número de secuencia

Los segmentos ACKs no se retransmiten

RFC 6289: Retransmisión (RTO – retransmission time out)

-Temporizador para todos los segmentos enviados sin confirmar
-Cuando vence el temporizador, el segmento se retransmite
-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-a)*SRTT + a*R' ; a=1/8 y R' es una nueva muestra
RTTVAR = b*|SRTT-R'| + (1-b)*RTTVAR ; b = 1/4
RTO = SRTT + MAX(G, 4*RTTVAR) ; Se sugiere G <= 100ms
Mínimo valor >= 1s y Máximo valor <= 60s
```

Retransmisión rápida después de tres segmentos ACK

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 -> se reenvía el segmento perdido inmediatamente (retransmisión rápida)

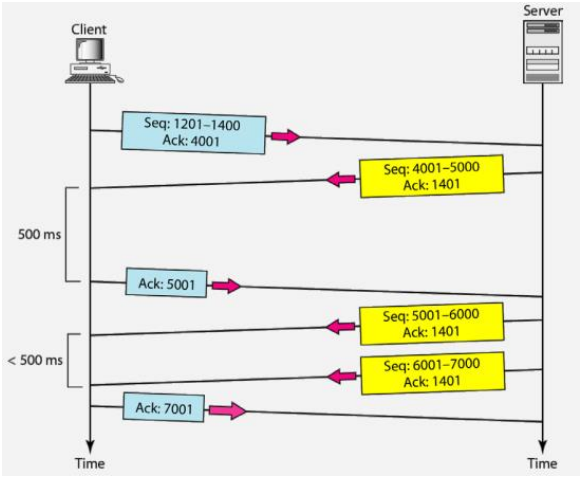
Segmentos fuera de orden

Antes se descartaban los segmentos fuera de orden (saltaría su RTO y se retransmitirían)

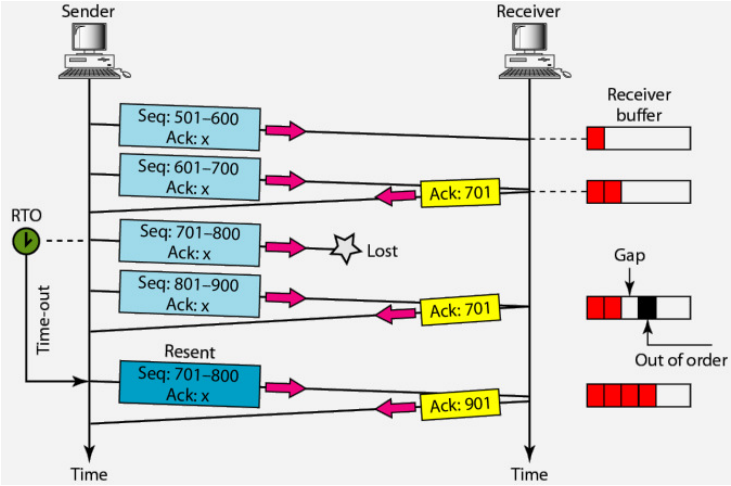
Actualmente se almacenan temporalmente y se marcan como “fuera de orden” hasta que llega el segmento perdido
- Los RST fuera de orden se siguen descartando (TCP RST Attack)

Ningún segmento fuera de orden es entregados

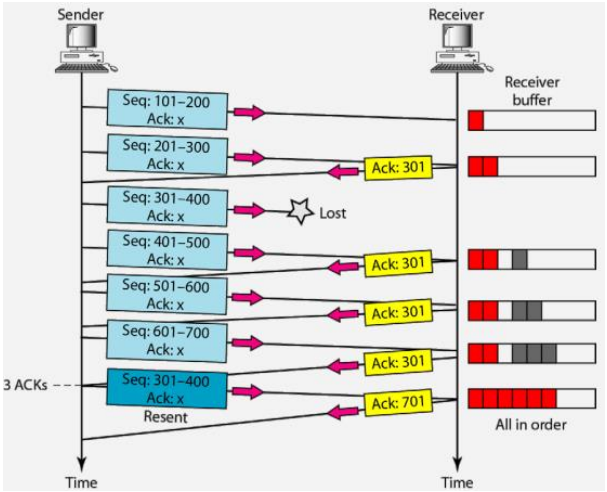
Funcionamiento de TCP: sin errores



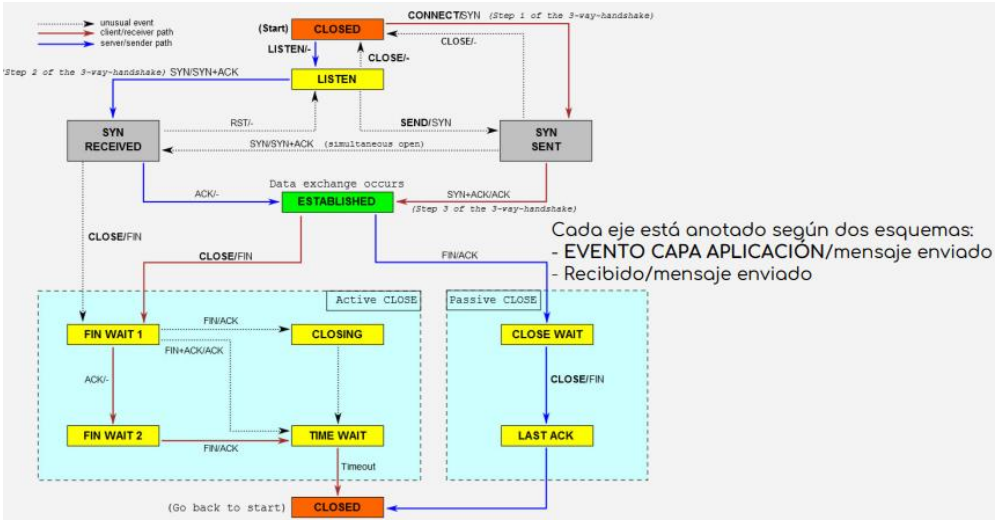
Funcionamiento de TCP: pérdida



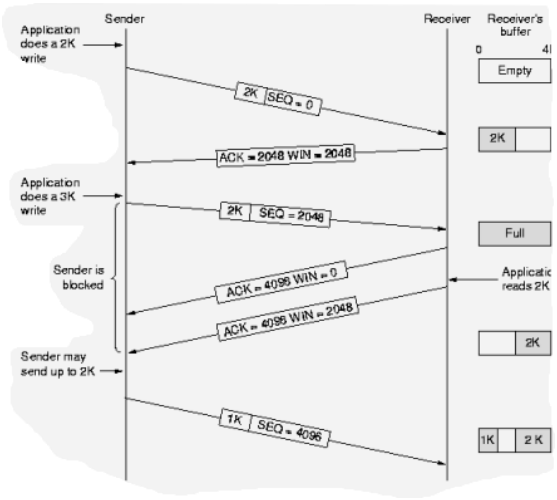
Funcionamiento de TCP: retrans. Rápida



Máquina de estados TCP



Control de flujo TCP: ventana deslizante



Se usa ventana deslizante para hacer la transmisión más eficiente
Ventanas orientadas a byte
Tamaño variable
- Los bytes dentro de la ventana se pueden enviar sin esperar confirmación de los anteriores

Ventana de recepción
- Se negocia en el inicio de conexión
- En cada mensaje se informa de la cantidad de bytes que puede recibir en ese momento el origen

4 ventanas por conexión:
- 2 por extremo: Recepción y envío

Control de flujo TCP: ajuste de la ventana

Síndrome de Silly Window: Una aplicación receptora lenta provoca la aperturas de pocos bytes generando envíos de segmentos muy pequeños.
Solución (lado del receptor): Tras cerrar la ventana, el receptor no anuncia una ventana diferente a 0 hasta que no tenga un tamaña suficiente: T = min(MSS, buffer_recepción/2)
Solución (lado emisor): Algoritmo de Nagle
- Si no existen datos esperando confirmación => Se envía siempre
- Si hay datos pendientes de confirmación => Solo se envía si:
  - Los datos a enviar son >= MSS
  - Recibo la confirmación de todos los datos pendientes
- Respetando que long(datos) <= Tamaño de ventana de recepción

Control de congestión en TCP

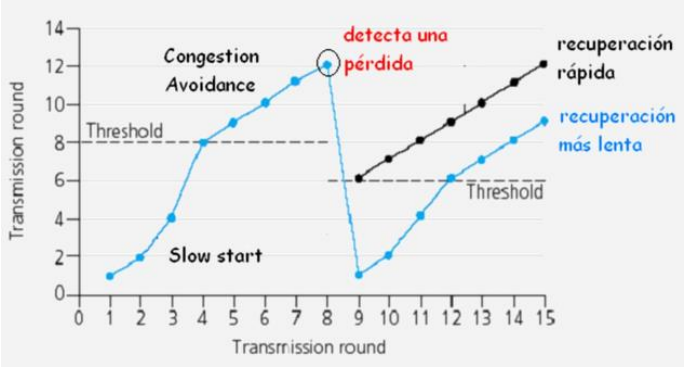
Conceptos diferentes
- Control de flujo: no enviar más paquetes de los que puede recibir el receptor
- Control de congestión: no enviar más paquetes en una parte de la red (subred)

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.

Mecanismos de control de congestión en TCP

Slow start, congestion avoidance (obligatorios)



Slow start

Inicialmente longitud de la ventana de congestión (del emisor) long(ventana_emisor) = MSS*N (N = 1 en el ejemplo)

Cada vez que se confirme un segmento (ACK) se actualiza long(ventana_emisor) = long(ventana_emisor) + MSS

¡Crecimiento exponencial!

Congestion avoidance

Suaviza el crecimiento respecto al slow start

Crecimiento lineal long(ventana) = long(ventana) + MSS/long(ventana)

Detección de la congestión

Se detecta pérdida (también podría ser un incremento en RTT) Threshold = long(ventana)/2

Salta un temporizador (congestión grave) -> Slow start long(ventana) = MSS*N (Valor inicial)

Por ACKs repetidos (congestión leve) -> Congestion Avoidance long(ventana) = Threshold