Nivel de Transporte. Protocolo TCP. Índice

Introducción

Finalidad de la Capa 4: Capa de Transporte

Conectividad extremo a extremo, transporte de datos fiable.

- Segmentación de los datos de aplicación de la capa superior.
- Establecimiento operaciones extremo a extremo.
- Envio de segmentos de host final a host final.
- Asegurar la fiabilidad de los datos.
- Proporcionar control de flujo.

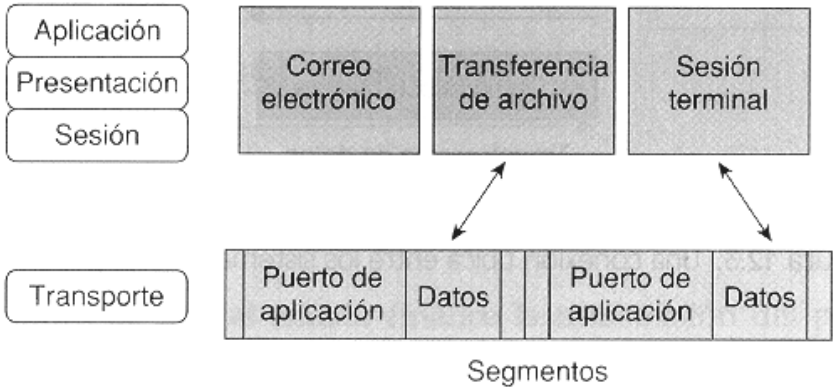
De dos formas:

- **No orientado a la conexión: UDP**.
  - Unidad de transporte: mensaje.

- **Orientado a la conexión: TCP**.
  - Unidad de transporte: segmento.
  - Conexión lógica entre extremos.

Conexión en un sistema de iguales

- Varias aplicaciones pueden compartir la misma conexión de transporte.
- Se realiza una “multiplexación de conversaciones de capa superior”.



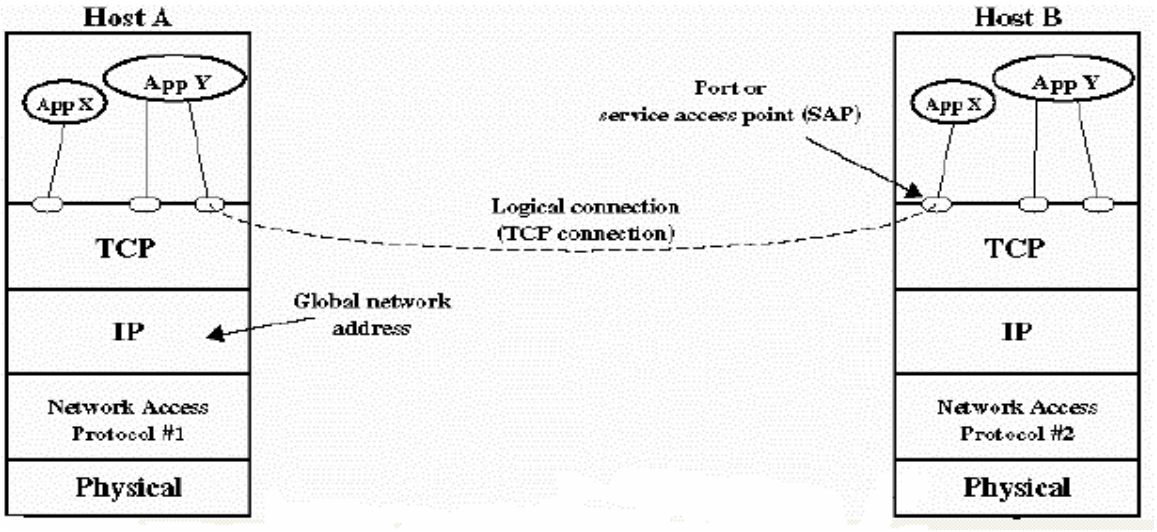
- Un host es el **cliente** que inicia la comunicación y el otro host es el **servidor** que se encuentra a la espera de conexiones entrantes.
- Despues de efectuarse la sincronización, se dice que se establece la conexión y comienza la transferencia de datos. Durante la cual el software de protocolo verifica que los datos se reciben correctamente.
- Cuando se transmiten los datos, puede tener lugar congestión por:
  - Un host no puede aceptar datos a cierta velocidad elevada.
  - Si muchos hosts envian a un mismo destino.

Puertos y Sockets

- Tanto transmisor como receptor crean puntos terminales denominados "**sockets"**.

- En el elemento receptor que espera la conexión, el programa "**servidor**" informa al sistema operativo que espera conexiones a través de un determinado puerto, para que cuando lleguen los datos el programa sea notificado.

- En el elemento transmisor el programa "**cliente**" efectua la conexión desde un puerto local libre hasta el puerto del host remoto que ha sido reservado anteriormente por el servidor.



Asignación de puertos (I)

Los puertos se identifican por un número de 16 bits. Los de cada protocolo se manejan por separado. Se emplea el siguiente criterio:

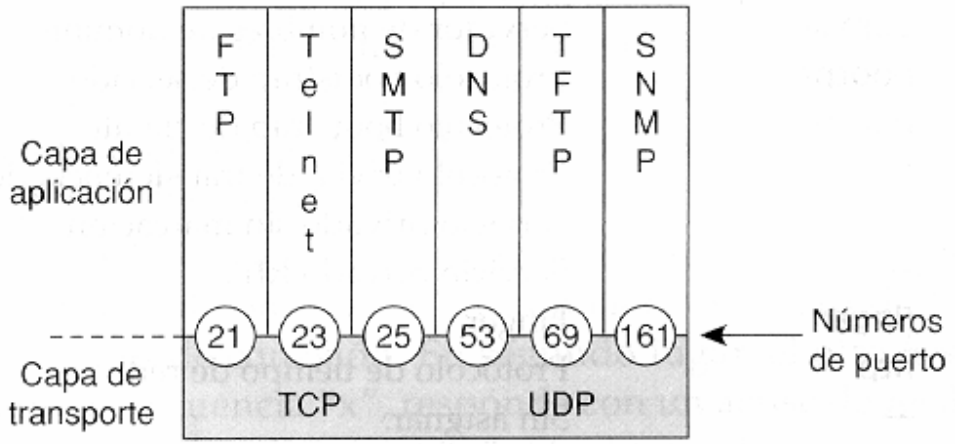
- Puertos **bien conocidos**: Los puertos 1 al 255 están reservados y tienen asignación universal. Los puertos por debajo del 1024 solo pueden gestionarse por el superusuario (root) del sistema y están asignados a aplicaciones. Los números superiores a 1023 no están regulados.

Decimal	Palabra clave	Descripción
0	—	Reservado.
1-4	—	No asignado.
5	rje	Entrada de trabajo remoto.
7	echo	Eco.
9	discard	Descartar.
11	users	Usuarios activos.
13	daytime	Fecha y hora.
15	netstat	Quién está activado o conectado.
17	quote	Cita del día.
19	chargen	Generador de caracteres.
20	ftp-data	Protocolo de transferencia de archivos (datos).
21	ftp	Protocolo de transferencia de archivos
23	telnet	Conexión al terminal.
25	smtp	Protocolo simple de transferencia de correo.
37	time	Hora del día.
39	rlp	Protocolo de localización de recursos.
42	nameserver	Nombre de *host* del servidor.
43	nicname	Quién es.
53	domain	Servidor de nombres de dominio.
67	bootps	Protocolo bootstrap de servidor.
68	bootpc	Protocolo bootstrap de cliente.
69	tftp	Protocolo trivial de transferencia de archivos.
75	—	Servicio privado sin marcación.
77	—	Servicio privado RJE.
79	finger	Finger.
123	ntp	Protocolo de tiempo de red.
133–159	—	Sin asignar.
160–223	—	Reservado.
224–241	—	Sin asignar.
242–255	—	Sin asignar.

Asignación de puertos (II)

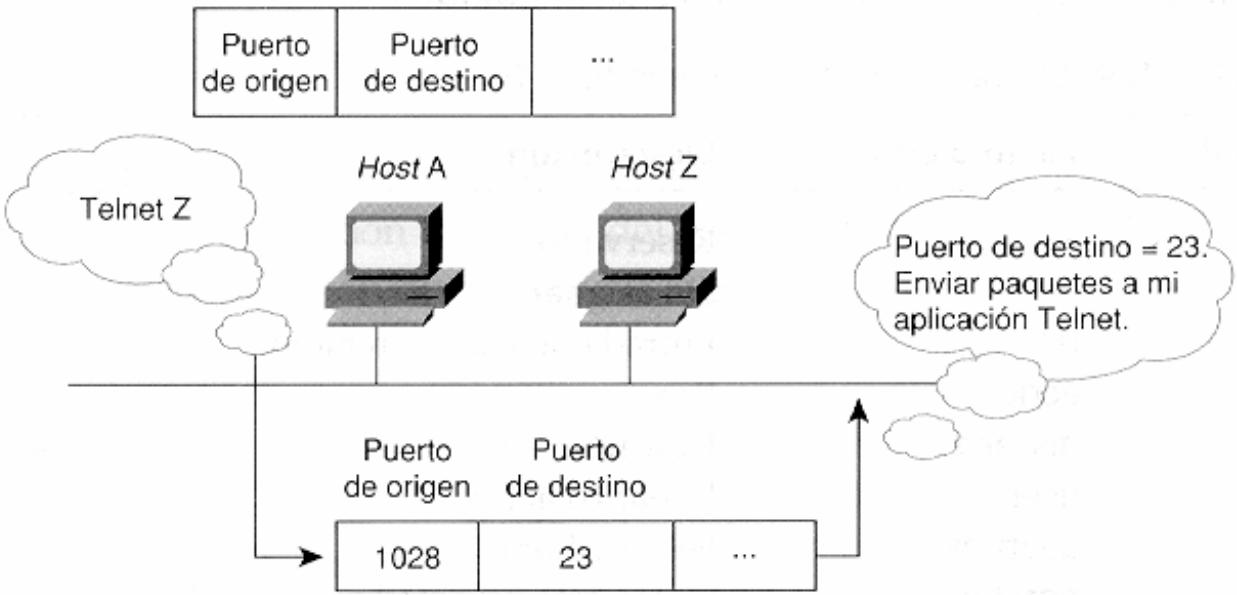
- Puertos de **asignación dinámica**: Los gestiona el software de red, integrado en el sistema operativo y los asigna de forma dinámica según se van solicitando por las aplicaciones. Son los puertos de número mayor de 1024.

- Los "**puertos registrados"** son los que han sido registrados por aplicaciones comerciales y se situan por encima de 1024.



Asignación de puertos (III)

- El puerto origen es asignado de forma dinámica y es diferente del puerto en el host de destino.

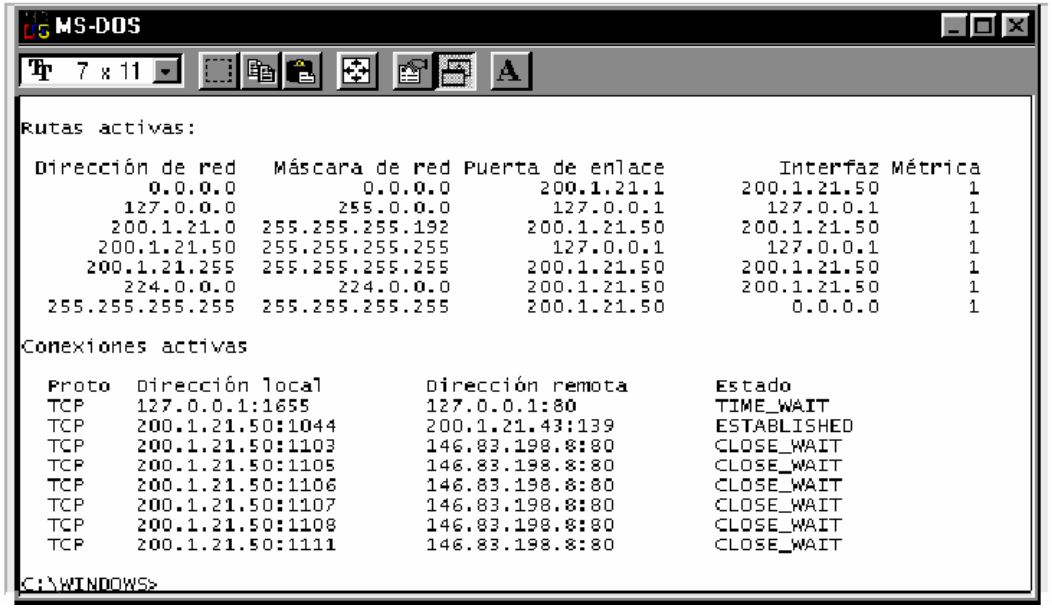


Enlace entre puertos y aplicaciones

La apertura de un puerto puede ser de dos tipos:

- **Apertura pasiva**: Los servidores se conectan a un puerto y lo mantienen abierto esperando una conexión (servidores “monohebra”, “multihebra” y “superserver inetd”).
- **Apertura activa**: Los clientes abren un puerto y se generan una serie de mensajes por la red para conectar con el otro extremo.

netstat -rn



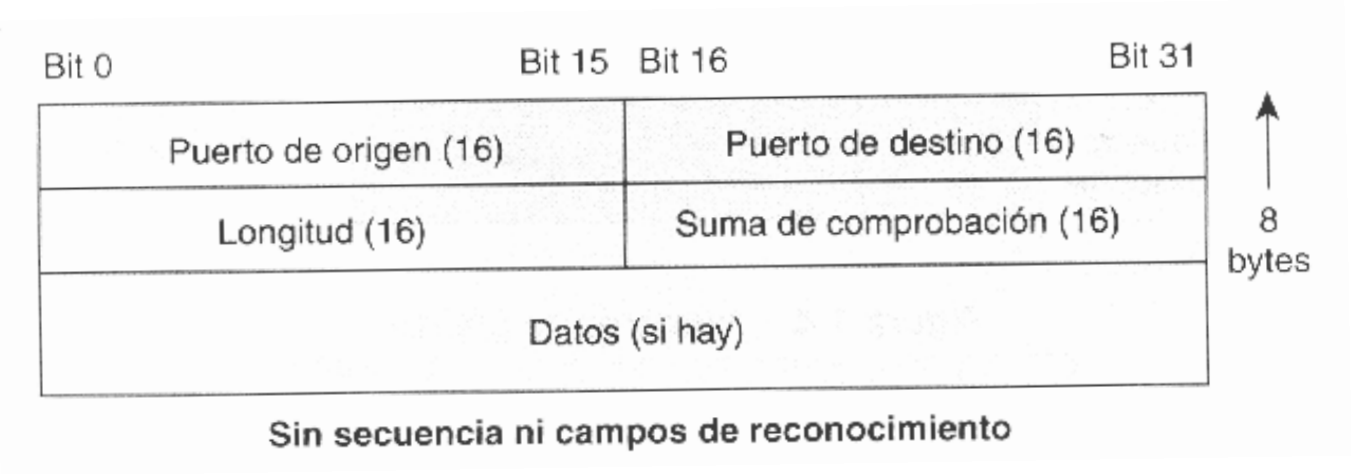
Protocolo UDP

El protocolo UDP es la forma más sencilla de enviar datos:

- Envio de mensajes, no orientado a conexión.
- No fiable.
- No se verifica la entrega de los mensajes.
- No reensambla los mensajes entrantes.
- No utiliza acuse de recibo.
- No proporciona control de flujo.

Formato mensaje UDP

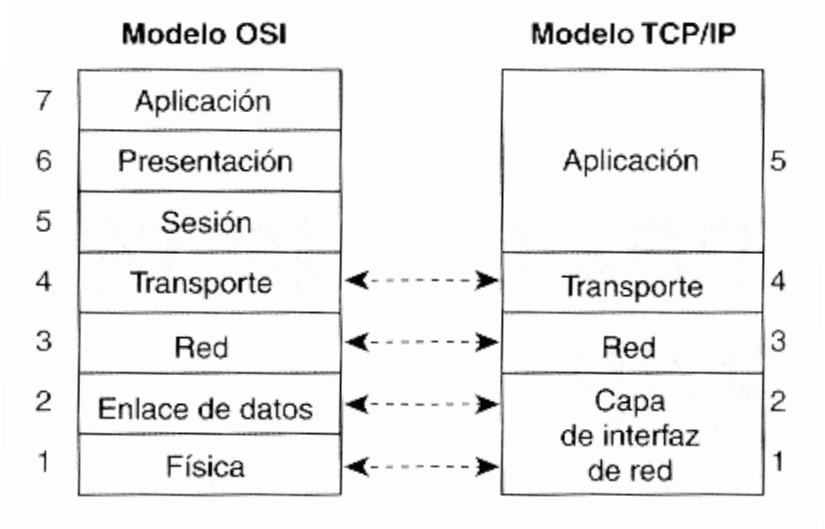
- Se proporciona una pseudo cabecera a la capa IP que no se transmite. Aporta información para formateo del paquete IP.
- La longitud de la cabecera es siempre 64 bits.
- UDP es empleado por TFTP, SNMP, NFS, DHCP, BOOTP y DNS.
- El checksum es opcional.



Sin secuencia ni campos de reconocimiento

Protocolo TCP

- Emplea la técnica de "**acuse de recibo positivo con retransmisión**". Cada vez que el receptor recibe los datos envia un acuse de recibo.
- El transmisor espera que le **confirmen cada segmento** enviado antes de enviar el siguiente.
- El transmisor **inicia un temporizador** cada vez que envia un segmento y lo vuelve a enviar cuando expira.
- Los mensajes de acuse de recibo tienen **numeros de secuencia** para evitar confusiones.



Conceptos Básicos de TCP

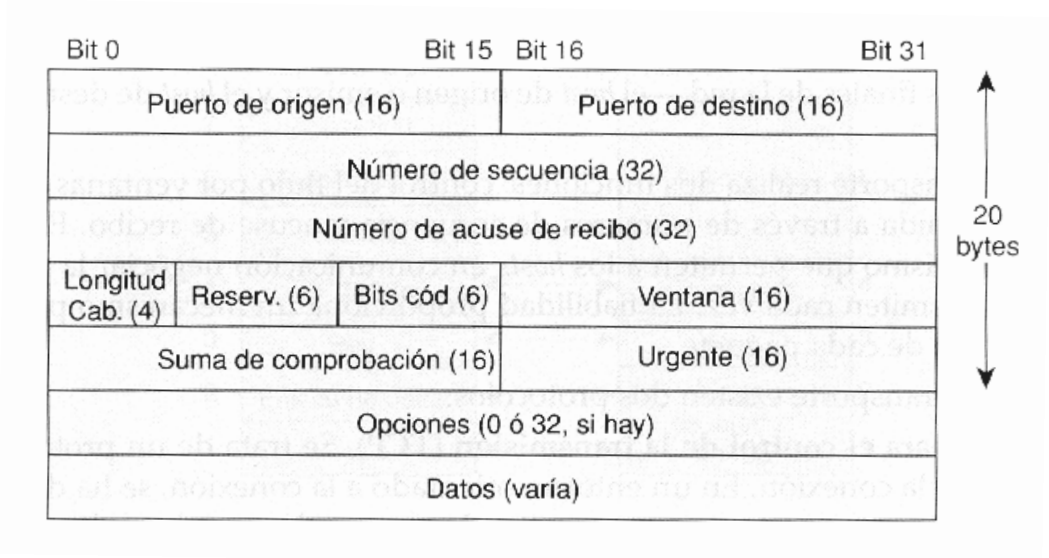
- Orientado a "**stream**". TCP ve los datos como un flujo continuo, no como paquetes independientes.
- Orientado a **conexión**.
- **Fiable**.
- La transferencia se apoya en buffers: **ventanas**.
- Transmisión **full duplex**.
- **Piggybaking**: las confirmaciones de recepción se incluyen en los segmentos de datos en el sentido opuesto para reducir el tráfico.
- **Reensambla** datos en destino a partir de los segmentos entrantes
- **Vuelve a enviar** todo lo que no se ha recibido.

Formato segmento TCP (I)

- Se proporciona una **pseudo cabecera** a la capa de red que no se transmite.

- **Número de secuencia**: indica la posición del primer octeto dentro del stream.

- **Número de ACK**: número de secuencia del siguiente octeto que se espera recibir (confirma llegada de todos los anteriores).



Formato segmento TCP (II)

- **Code Bits**: Indica tipo y contenido de segmento.

Bit (de izquierda a derecha)	Significado
URG	El campo puntero urgente es válido
ACK	El campo acuse de recibo es válido
PSH	El segmento requiere un "push"
RST	"Resetear" la conexión
SYN	Sincronizar números de secuencia
FIN	La fuente ha alcanzado el fin de su secuencia de bytes

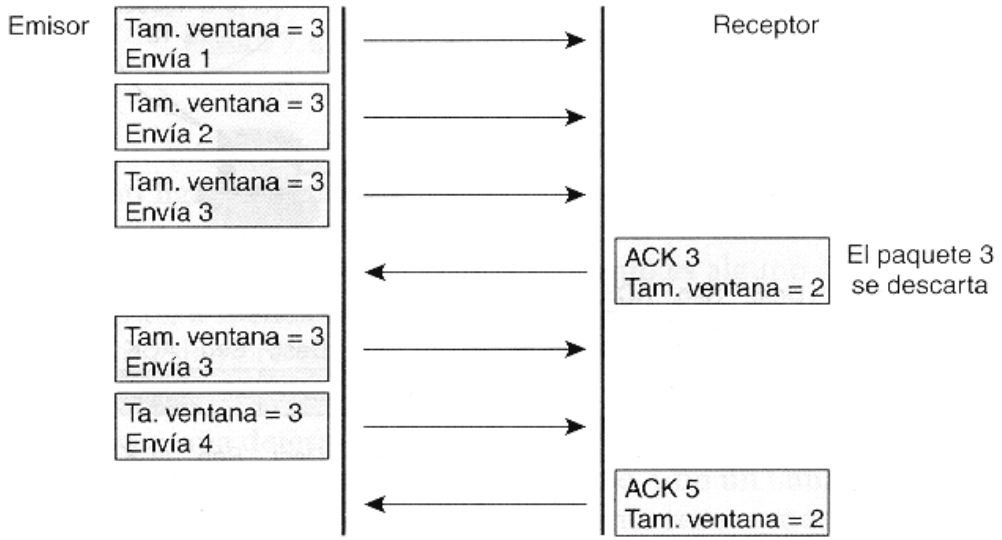
- **Ventana**: numero de bytes que el dispositivo espera aceptar.

- **Urgente**: Cuando el bit URG está a 1, especifica la posición en la que acaban los datos urgentes de la zona de datos.

- **Opciones**: Se emplea para acordar tamaño de segmento y otras opciones entre ambos extremos de la comunicación.

La ventana deslizante (I)

- Para evitar que cada vez que se reciba un segmento se mande un acuse de recibo se emplea la **ventana deslizante**.
- Se **espera** a haber recibido un número determinado de datos para enviar el acuse de recibo.
- El emisor desliza la **ventana del emisor** cada vez que recibe acuses de recibo.
- El receptor desliza la **ventana del receptor** cada vez que se reciben segmentos válidos.

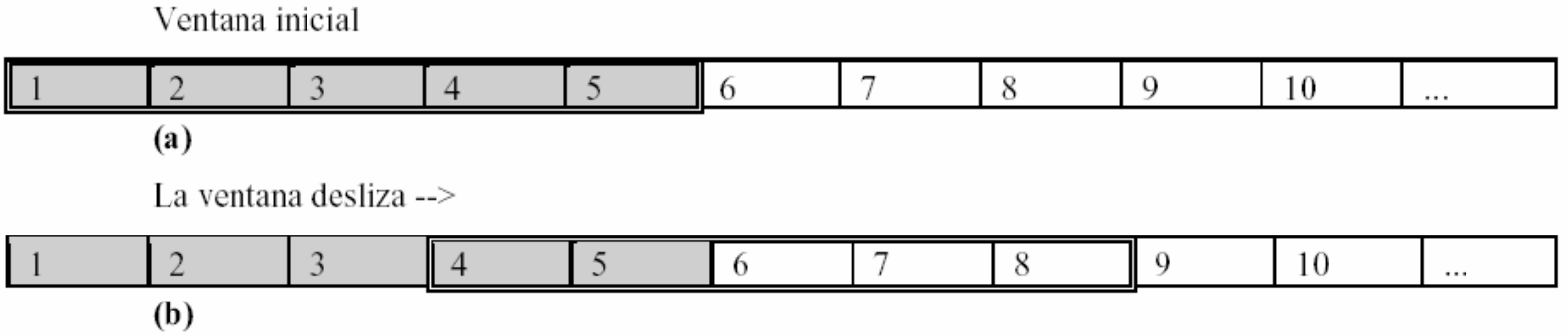


La ventana deslizante (II)

- En TCP se emplea el mecanismo de **ventana deslizante a nivel de byte** para conseguir una transmisión eficiente y un control de flujo extremo a extremo.

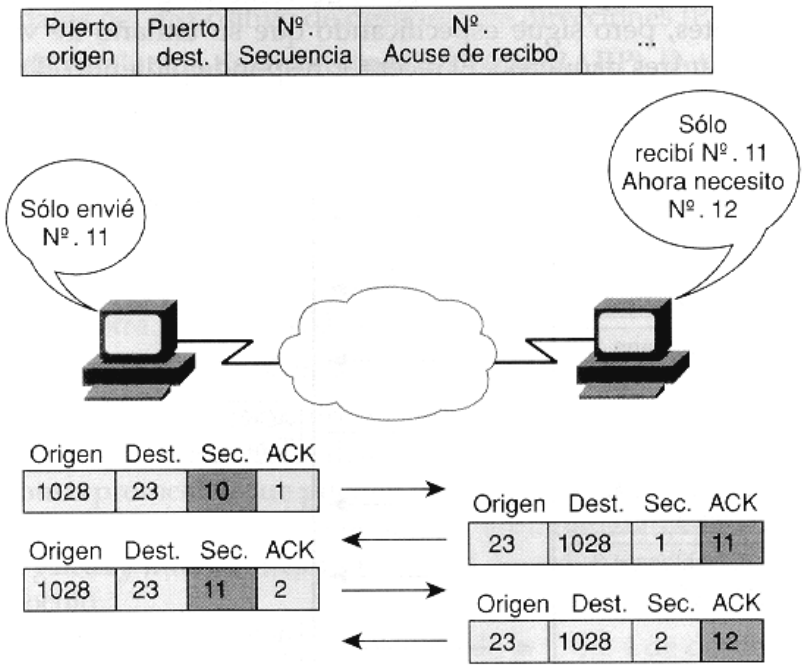
- Los bytes de la secuencia se numeran secuencialmente y el emisor tiene **tres punteros** asociados a cada conexión:

  - **Comienzo de la ventana**: separa confirmados de no confirmados.
  - **Final de la ventana**: indica el byte más alto que puede enviarse sin confirmase.
  - **Intermedio de la ventana**: separa enviados de no enviados.



Control de flujo básico

- Permite que el tamaño de la ventana varie con el campo "**ventana**".
- Esta variación permite controlar el flujo de datos.
- El destino puede dar **tamaño de ventana cero** para detener la transmisión.



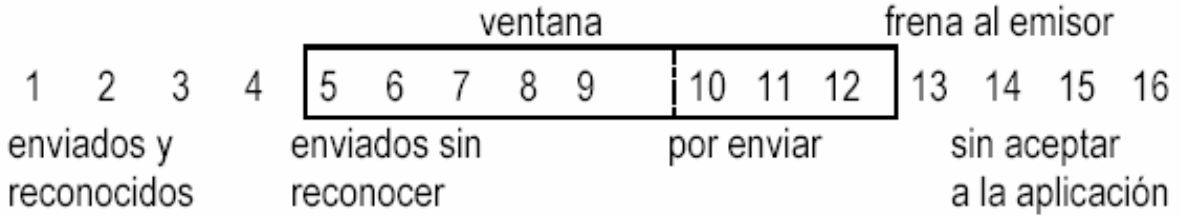
Control de Flujo (I)

- Se puede emplear un bit para envio consecutivo de mensajes: protocolo del bit alternante.

- Para aumentar eficiencia se permite que el emisor emita segmentos sin haber recibido los ACKs de los anteriores.

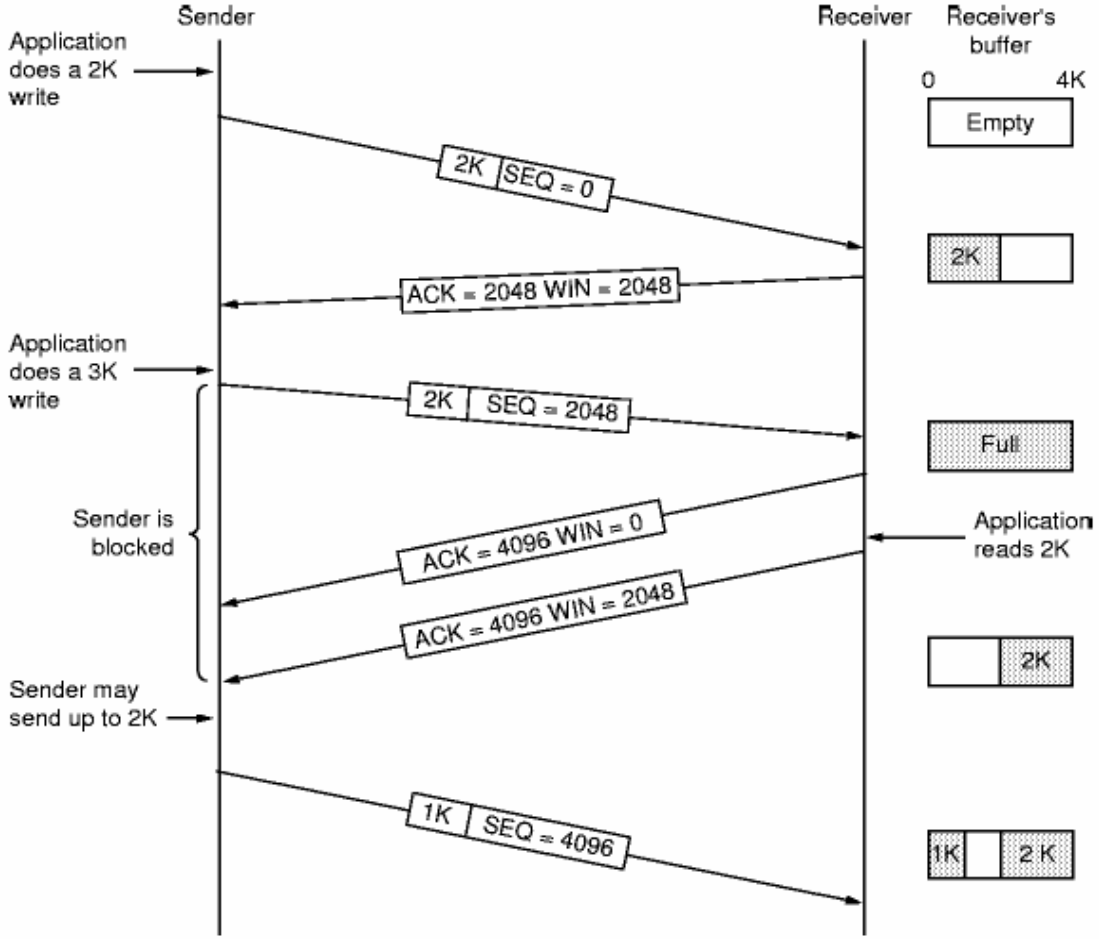
- El tamaño de la ventana permite realizar el control de flujo.

- Aunque el tamaño de ventana sea cero se permite la transmisión de datos “urgentes”.



Control de Flujo (II)

Ejemplo:



Acuses de recibo y retransmisiones (I)

- **Positive Acknowledgement with Retransmision** (PAR): si no se recibe ACK se retransmiten los datos. En el receptor si el checksum es positivo se envia ACK, si no se descarta y espera.

- ACK indica número de orden del **siguiente octeto**.

- TCP envia datos en **segmentos de longitud variable**, por eso los ACKs se refieren a bytes y no a segmentos.

- **La pérdida de ACKs** no fuerza retransmisión, esto puede generar retransmisiones innecesarias.

- TCP espera que el destinatario reconozca datos cada cierto tiempo, si tiene datos que enviar con **piggybacking** y si no con un segmento ACK.

Acuses de recibo y retransmisiones (II)

- El software inicia un **temporizador** y espera el acuse de recibo. Si expira se supone que se perdió y se vuelve a enviar.

- **Problema**: Es imposible conocer a priori lo rápido que van a llegar los acuses de recibo.

- **Solución**: El protocolo calcula el tiempo de retardo y el temporizador se ajusta a estos tiempos de forma automática.

- Se utiliza una **media ponderada** entre el último tiempo de retardo y el tiempo medio anterior. También se usa un factor de peso para ponderar los tiempos.

$$T_{medio\ actual} = (q * T_{medio\ anterior}) + ((1-q) * Nuevo\_Tiempo\_Retardo)$$

Algoritmo de Jacobson

- Para cada conexión se mantiene la **variable RTT**, estimación de tiempo de ida y vuelta. Si ACK se recibe se actualiza:

$$RTT = (a*viejo\_RTT) + ((1-a)*nuevo\_RTT) \qquad (0<\mathbf{a}<1)$$

a es el **factor de amortiguamiento**, si es grande RTT varia con lentitud y si es pequeño varia con rapidez.

- El valor del temporizador es **b*RTT**. Si b es grande el temporizador se hace grande y así el tiempo de espera, si b es pequeño el tiempo de espera también.

- Hay que llegar a un **compromiso** en los valores que dependerá de las caracteristicas del enlace.

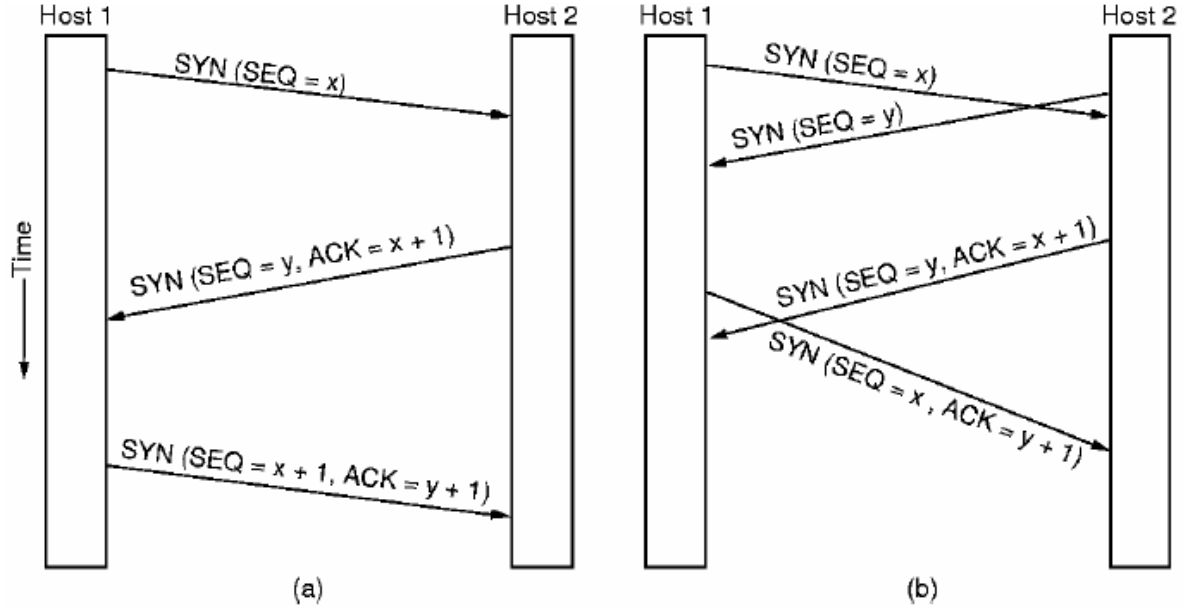
Algoritmo de Karn

- Igual que el anterior, pero además los segmentos para los que se ha realizado retransmisión no deben considerarse a la hora de actualizar el RTT.

- Siempre que se retransmite un segmento, se dobla el temporizador, hasta que un segmento llegue sin necesidad de retransmisión y reduzca el RTT.

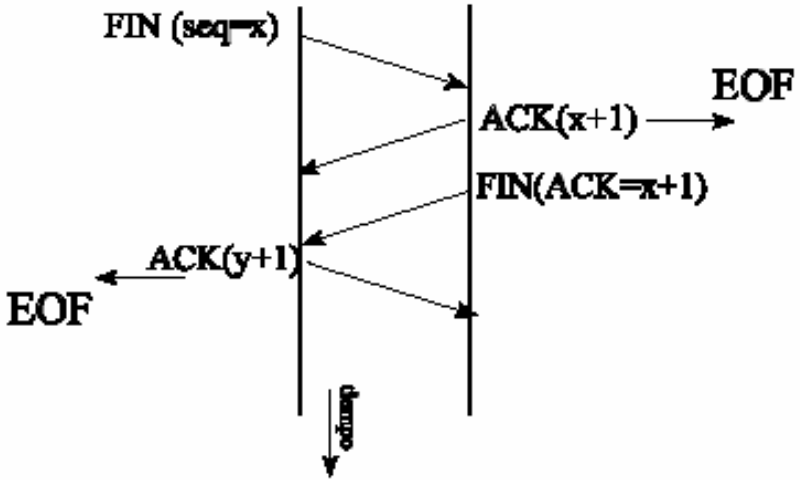
Establecimiento y liberación (I)

- Utiliza el "**intercambio de señales de tres vias**". Consiste en enviar un segmento SYN, recibir la respuesta SYN y ACK. Por último se envia un acuse de recibo para indicar conexión establecida.
- Tres fases en el diálogo TCP:
  - o **Establecimiento de conexión**: se reservan los recursos.
  - o **Transferencia**: datos son transmitidos secuencialmente.
  - o **Terminación de conexión**: finaliza la conexión.



Establecimiento y liberación (II)

- Como la red no es fiable, el intercambio de tres vias asegura el establecimiento de la conexión casos extremos.
- Al inicio se ponen de acuerdo los extremos en los números de secuencia iniciales.
- El proceso de liberación comienza con un **segmento FIN**.
- El otro extremo, reconoce el FIN con un ACK.
- Hay que liberar cada conexión de forma independiente.
- TCP proporciona mecanismo para **fallos de conexión**: segmento RST, indica la finalización inesperada de la conexión.



Envio forzado de datos

- El **uso de buffers** incrementa el rendimiento, pero para los datos urgentes ralentiza el envío.

- Para **aplicaciones "interactivas"**, TCP proporciona una operación "push" para obligar a transmitir datos inmediatamente. Estos datos serán entregados al otro extremo inmediatamente.

- Se dispone también de **envio de "datos urgentes"**. Este mecanismo es diferente y emplea el bit URG y el campo de "puntero de urgencias".

Control de la congestión

- Grandes retardos causados por el exceso de tráfico: se guarda lo que no se puede transmitir, hasta que se desbordan los buffers y se pierden los datos a transmitir. Se termina colapsando la red.

- Para evitarlo se baja la tasa de transmisión. Se mantiene una ventana de congestión que se reduce cuando hay congestión de su valor nominal, de forma que :

  ventana_permitida = min(ventana_anunciada, ventana_congestión)

- Hay dos técnicas:
  - **Multiplicative Decrease**: cuando se pierde un segmento, se reduce la ventana de congestión a la mitad (hasta un mínimo), y se aumentan los timers de los segmentos que quedan para reducir el tráfico exponencialmente.
  - **Slow Start**: cuando acaba la congestión la ventana de congestión se aumenta en un segmento cada vez que se recibe un ACK, cuando llega a la mitad de su valor original, se frena la tasa de aumento.

Ejemplo de diálogo TCP