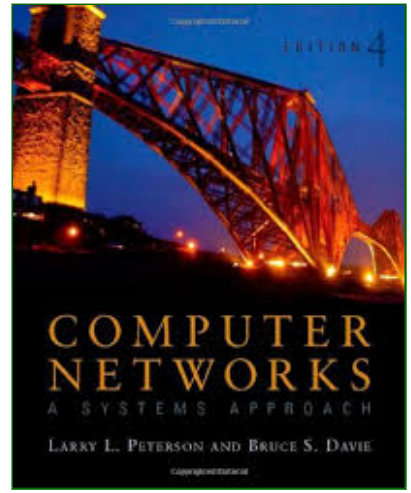
Redes de Computadores

Tema 5 – Capa de transporte

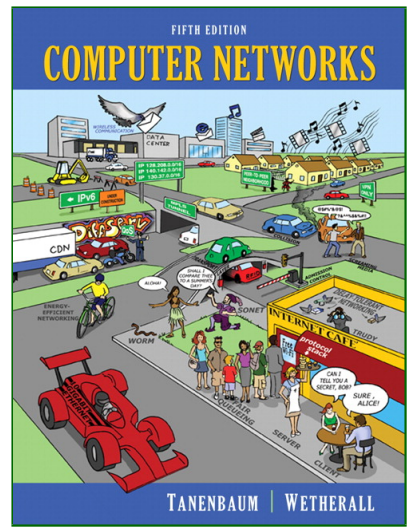
Natalia Ayuso, Juan Segarra y Jesús Alastruey

Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza

Índice



Capítulo 5



Capítulo 6

1 Introducción

- Función: control de la comunicación extremo a extremo

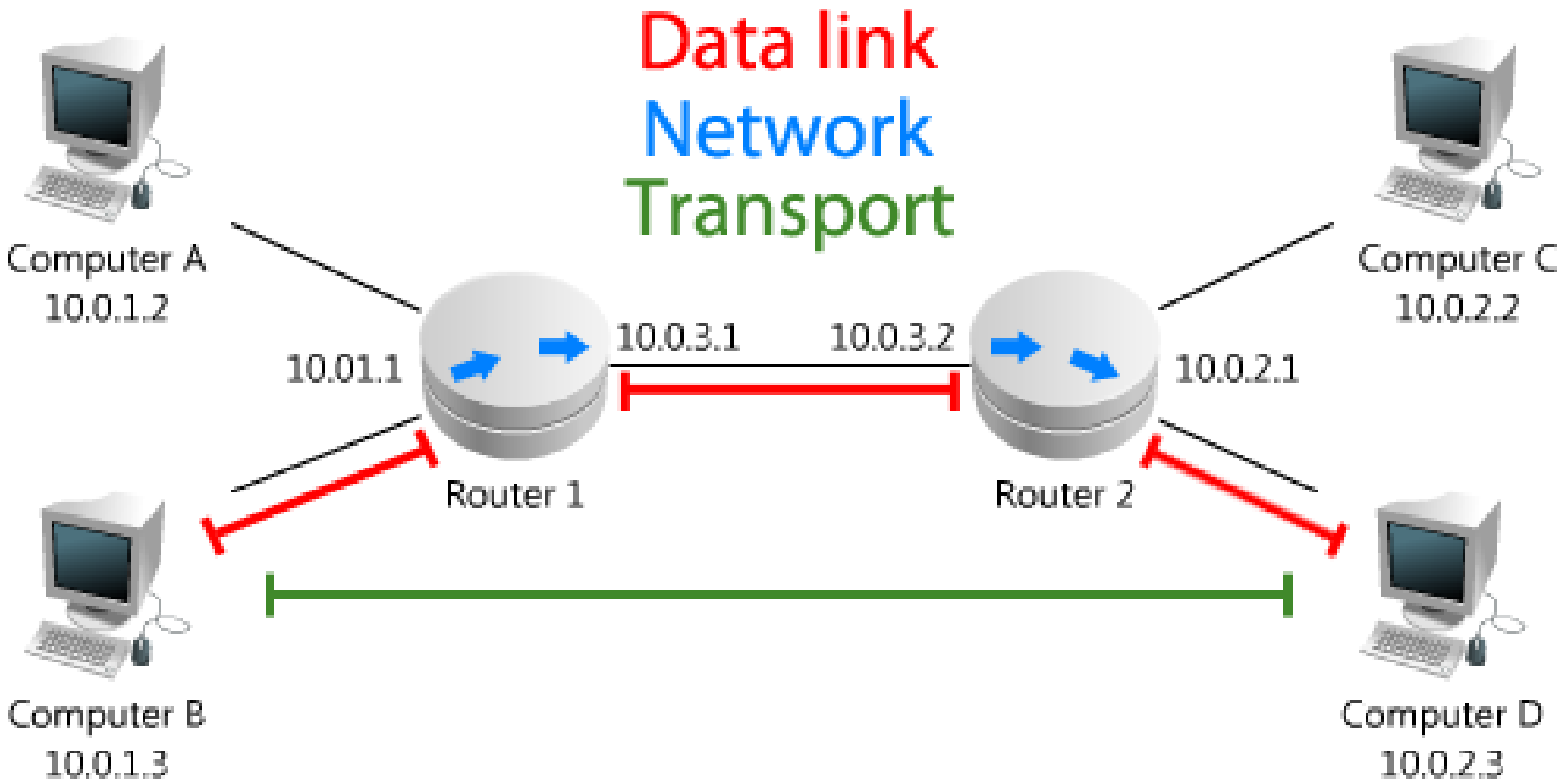


Imagen: OSI model – Layer 4: Transport (TCP and UDP with Scapy)

1 Introducción (II)

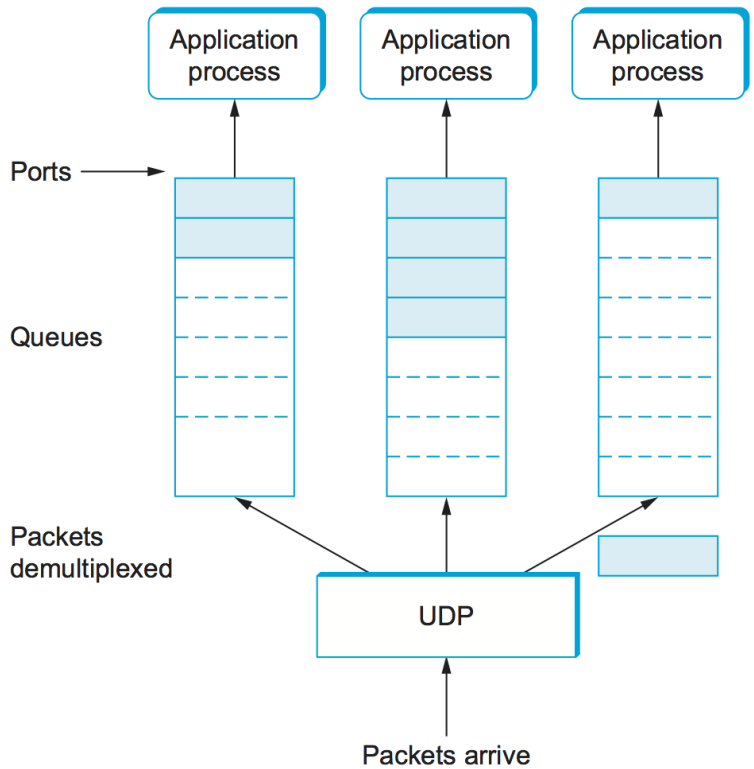
- Se supone un protocolo en la capa de red que puede ser:
  - No fiable: paquetes perdidos, duplicados, desordenados
  - Limita los paquetes a un tamaño finito (MTU)
  - Entrega los paquetes con retardo arbitrariamente largo
- Posibles servicios extremo a extremo:
  - Fiabilidad
  - Permitir mensajes de tamaño arbitrariamente grande
  - El receptor puede controlar el flujo de datos del emisor
  - Reserva de recursos (QoS)

1 Introducción (III)

- UDP (User Datagram Protocol)
  - Protocolo de transporte bidireccional «sin conexión»
  - No garantiza un servicio extremo a extremo fiable
  - Usado para transmisión continua de *información actualizada*: juegos en red, DNS, SNMP, etc.
  - Utilizado en multicast
- TCP (Transmission Control Protocol)
  - Protocolo de transporte bidireccional «orientado a conexión»
  - Fiable (checksum + retransmisión)
  - Usado cuando se requiere fiabilidad: FTP, HTTP, Telnet, SMTP, etc.
- RTP (Real Time Protocol)
  - Orientado a aplicaciones de tiempo real: *streaming*, etc.
  - Funciona sobre UDP

2 User Datagram Protocol (UDP)

- Servicio de comunicación entre procesos
- Unidad de transferencia: datagrama
- Destinatario no envía confirmación de recepción



2.1 Cabecera UDP

Checksum: opcional sobre IPv4 y obligatorio sobre IPv6

- pseudo cabecera + cabecera UDP + datos
- pseudo cabecera = IP origen + IP destino + IP protocolo (17) + longitud del segmento UDP

Longitud: tamaño en bytes del datagrama

- cabecera + datos

Puertos origen y destino: números de 16 bits $[0, 65535]$ que asocian comunicaciones de capa de transporte (*sockets*) a procesos en equipos origen y destino

2.2 Puertos software

- Abstracción para identificar procesos en un nodo
- Se usan en UDP y TCP
- Servidores usan puertos asociados al protocolo, por ejemplo: SSH: 22, DNS: 53, HTTP: 80
  - IANA gestiona correspondencia servicio-puerto
  - RFC 6335 define 3 rangos:

    $\leq$1023: sistema, requieren privilegios de administrador

    1024–49151: usuario

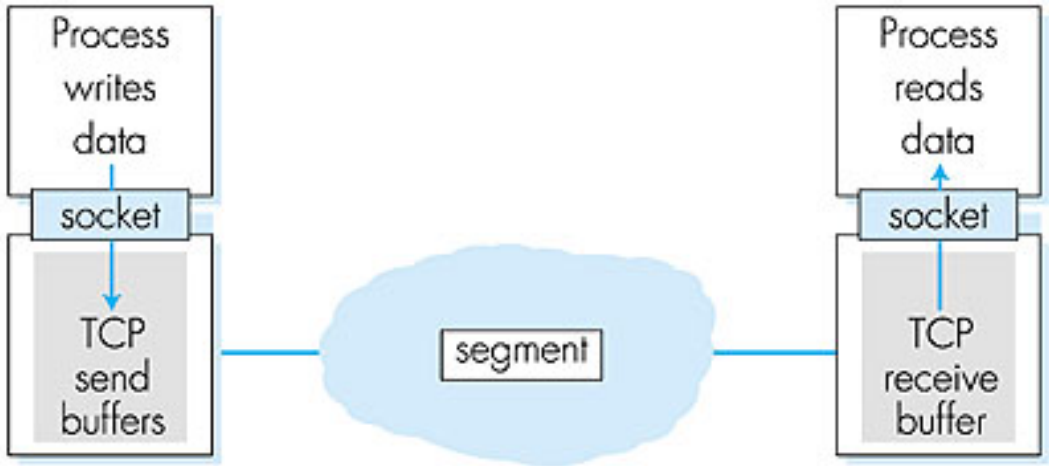
    $\geq$49152: dinámicos/privados
  - Ver fichero `/etc/services`
- Clientes suelen usar puertos locales aleatorios, asignados por el sistema, *sin* usar *bind()*

3 Índice

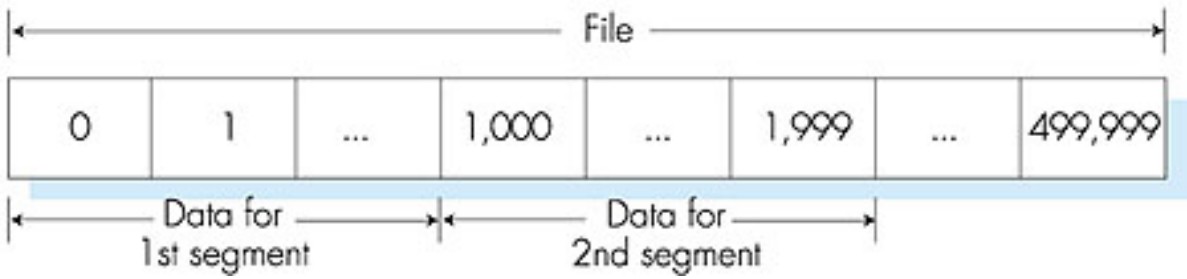
3 Transport Control Protocol (TCP)

- Comunicación entre procesos fiable, orientada a conexión
  1. Establecimiento de conexión
  2. Transferencia de datos
  3. Liberación de conexión
- Control de errores (checksum)
- Control de flujo: evita que receptor se desborde (ventana)
- Control de congestión: evita sobrecarga red (flags)[Tema 6]
- Unidad de transferencia: segmento TCP
- Secuenciación de datos *extremo a extremo*
  - Posición de los datos en el mensaje (nº de secuencia)
  - Confirmación de datos correctos (nº de ACK)
  - Retransmisión de un segmento si el emisor no recibe confirmación transcurrido un tiempo desde su envío (tiempo de expiración/*retransmission timeout*, RTO)

3 Transport Control Protocol (TCP) (II)

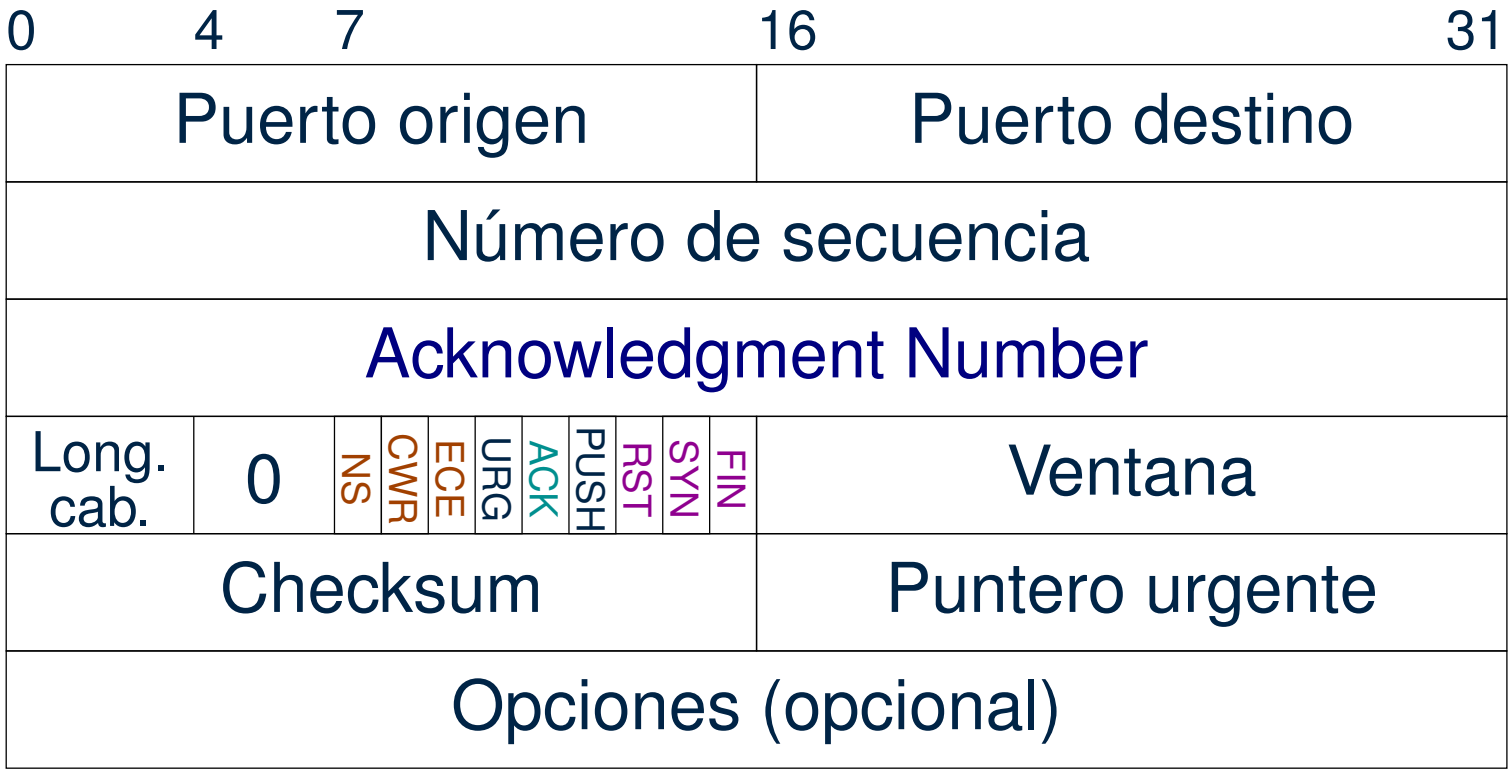


TCP send and receiver buffers



File of 500 KB with MSS of 1 KB

3.1 Cabecera TCP



Puertos origen y destino: identifican extremos de la conexión

Nº de secuencia (SEQ): posición del primer byte de datos del segmento en el mensaje original.

Acknowledgment Number (ACK): siguiente nº de secuencia que se espera recibir.

3.1 Cabecera TCP (II)

- SEQ y ACK son relativos a un valor inicial aleatorio (ISN, Initial Sequence Number)

Longitud de la cabecera: medida en palabras de 32 bits

Flag ACK: validez del campo Acknowledgment Number

Flags SYN, FIN, RESET: establecer/finalizar/abortar conexión

Flag URG: avisa de datos urgentes (puntero urgente)

Flag PUSH: fuerza el vaciado de *buffers* en origen y destino

Flags NS, CWR, ECE: control de *Explicit Congestion Notification* [Tema 6]

Checksum: pseudocabecera (=UDP) + cabecera TCP + datos

Ventana: bytes que el receptor puede aceptar

Opciones: parámetros adicionales de la conexión [p. 26]

3.2 Establecimiento de conexión

- *3-way handshake*

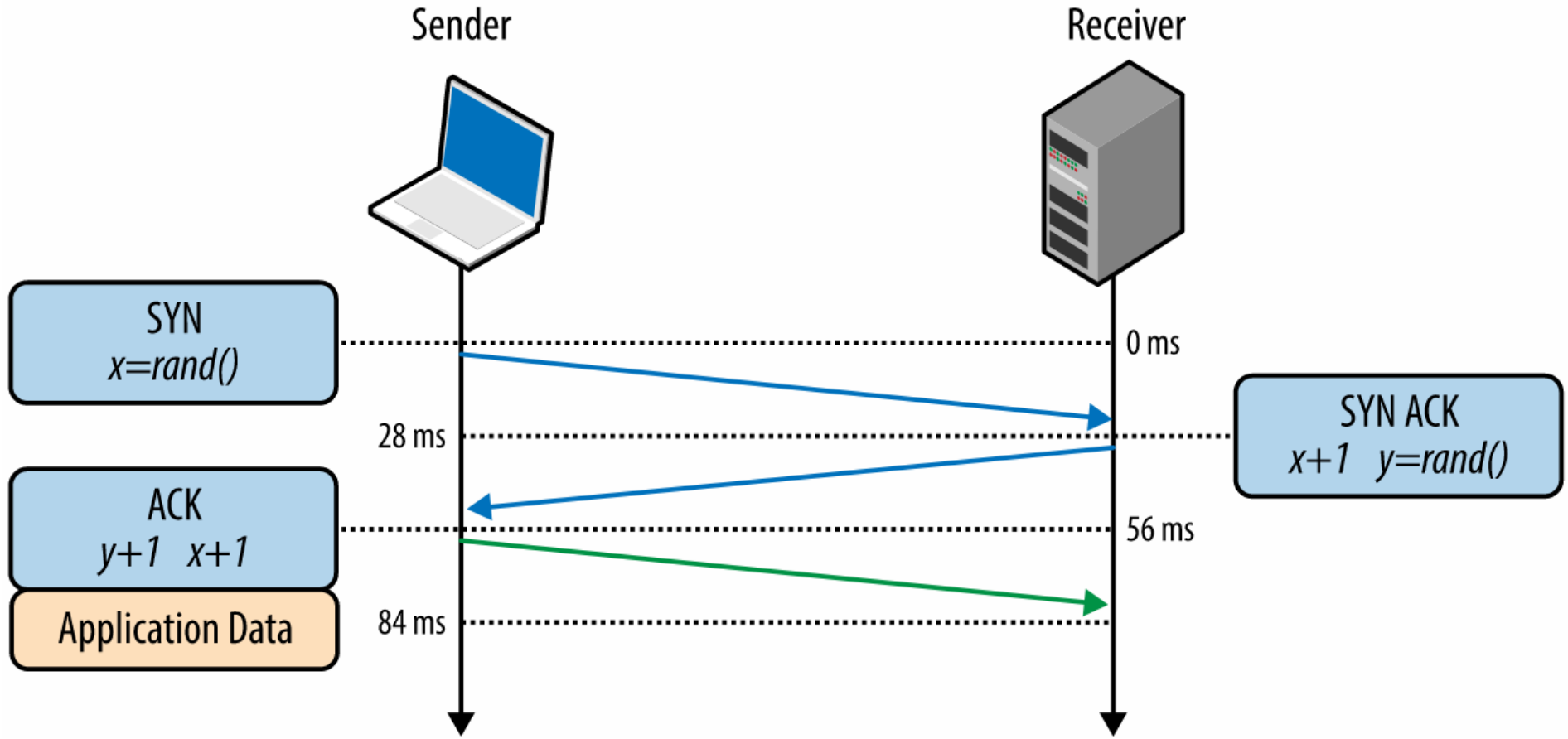
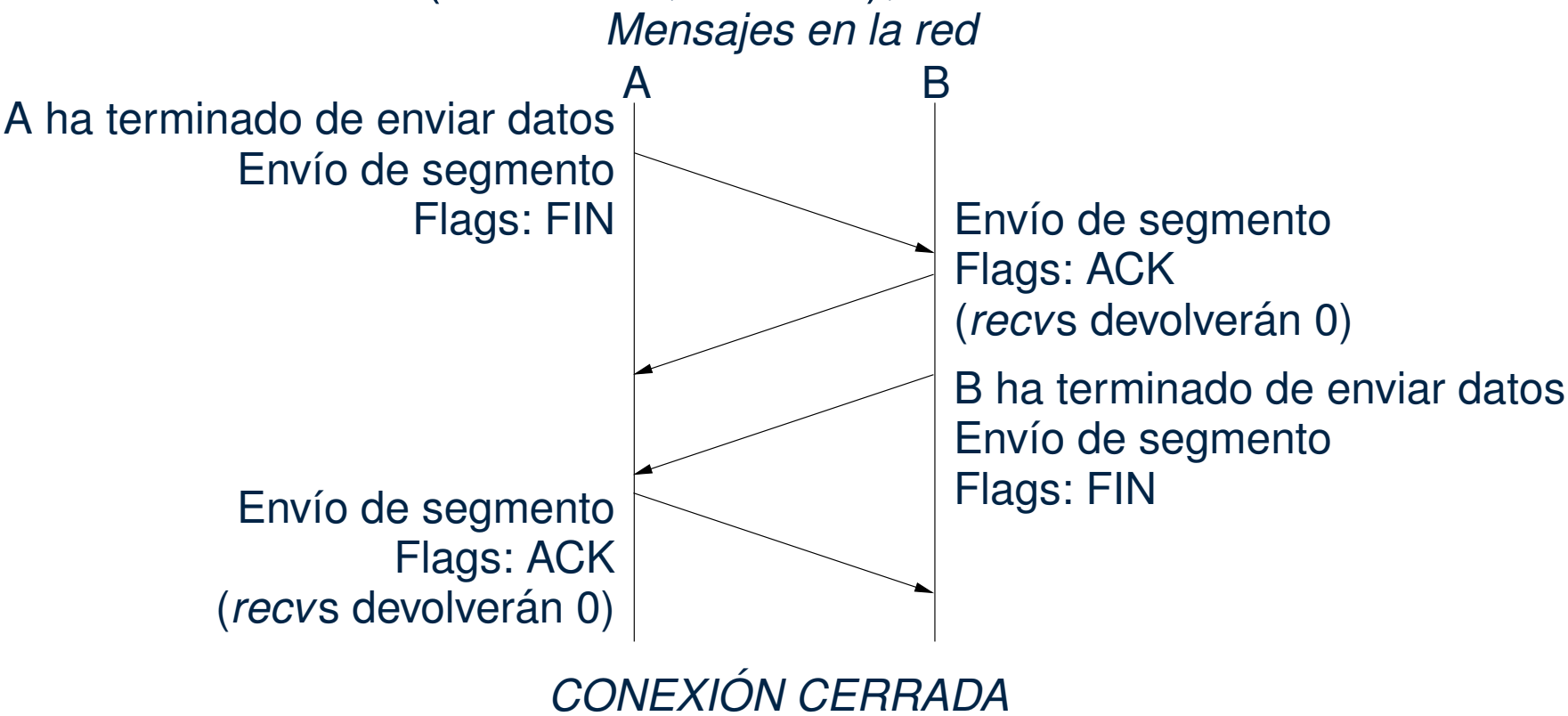


Imagen: High Performance Browser Networking: Building Blocks of TCP

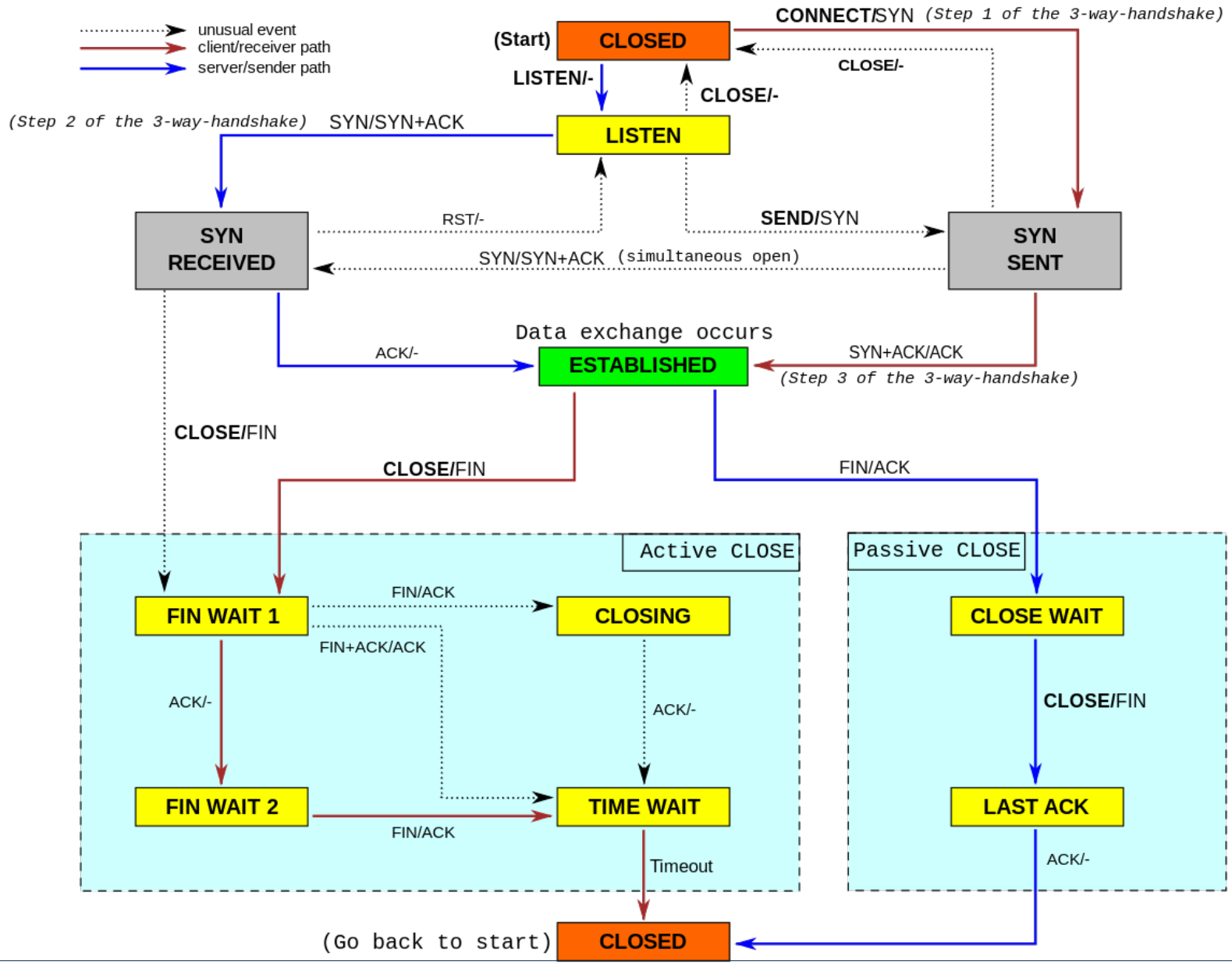
3.2 Finalización de conexión

- Desconexión ordenada o consensuada
  - int close(int fd);
  - int shutdown(int sockfd, int how); //SHUT_RD/WR/RDWR



- Desconexión desordenada o unilateral (flag RESET)

3.3 Estados de una conexión TCP



3.4 Envío de datos

- *Maximum Segment Size (MSS)*: máximo volumen de datos TCP que pueden enviarse en un segmento.
  - Normalmente se inicializa al mayor tamaño que no requiere fragmentación IP: $MSS = MTU - long\_cabeceras\_TCP/IP$
  - Por ejemplo, para Ethernet: $MSS = 1500 - 40 = 1460\ bytes$

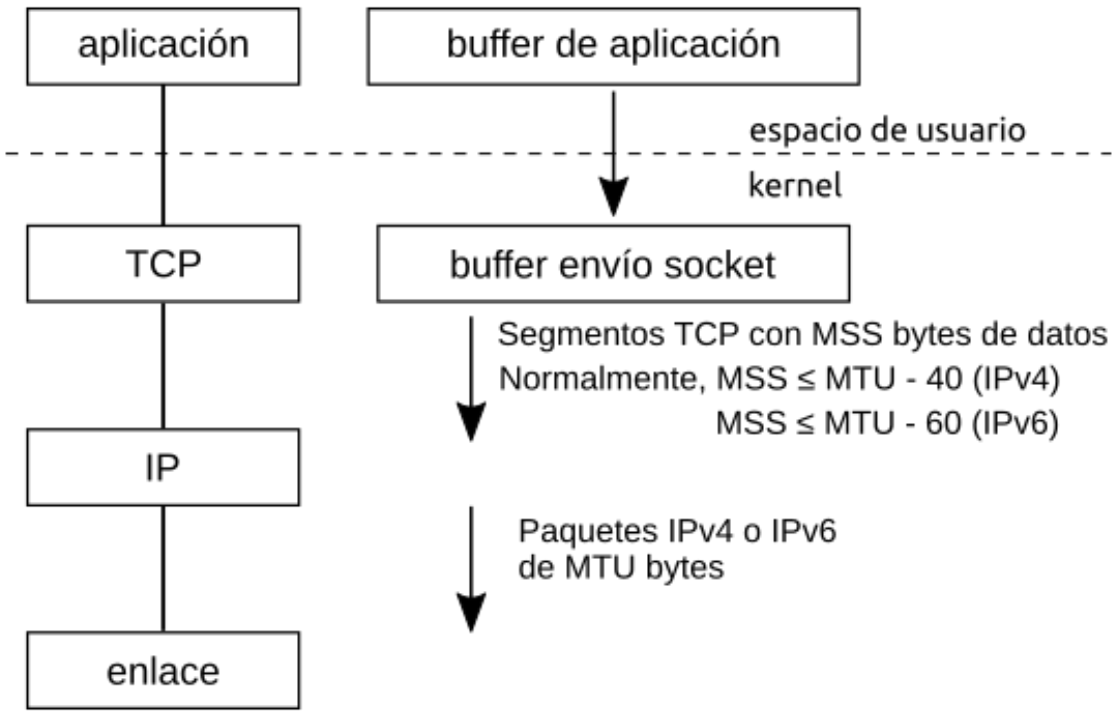


Figura: Buffers involucrados cuando una aplicación escribe en en socket TCP.
Adaptado de Figura 2.11 en Stevens. Unix Network Programming. Vol.1, 2ª ed.

3.4 Envío de datos (II)

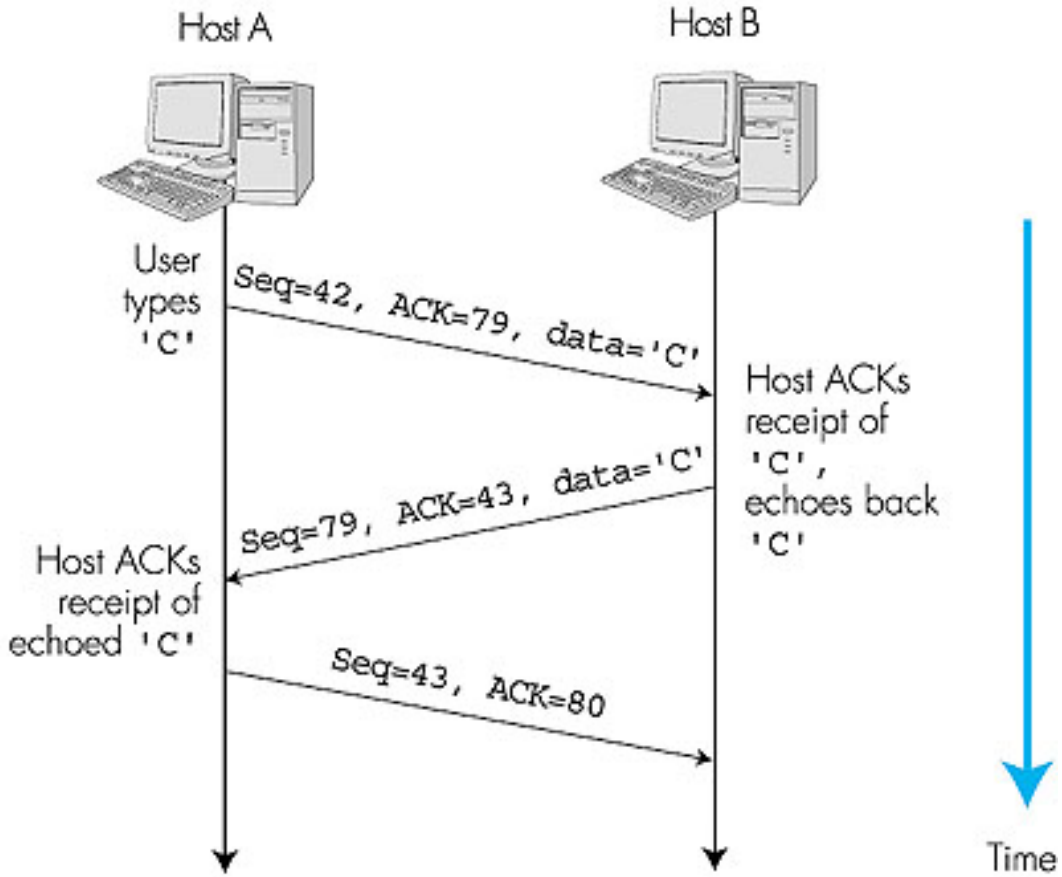


Figura: Envío de datos de cliente a servidor y acuses de recibo en sentido contrario.
Fuente: Sequence and acknowledgement numbers for simple Telnet application over TCP

3.4 Envío de datos (III)

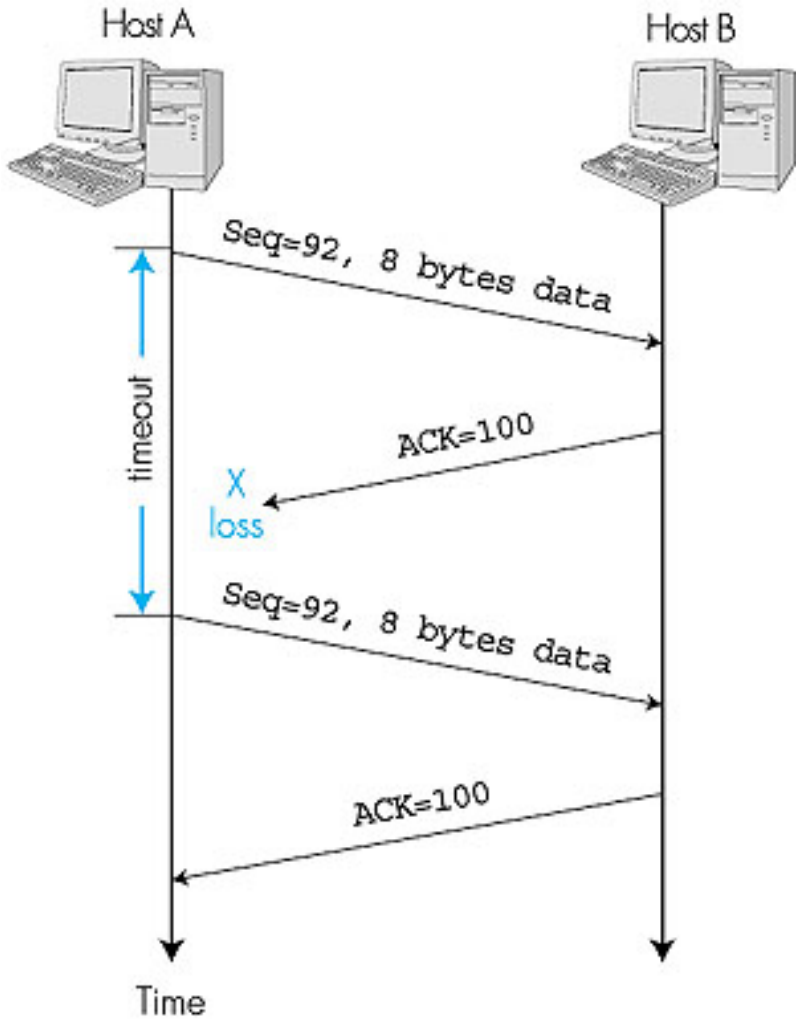


Figura: Envío de datos de cliente a servidor y acuses de recibo en sentido contrario.
Fuente: Retransmisión por pérdida de ACK

3.4 Envío de datos (IV)

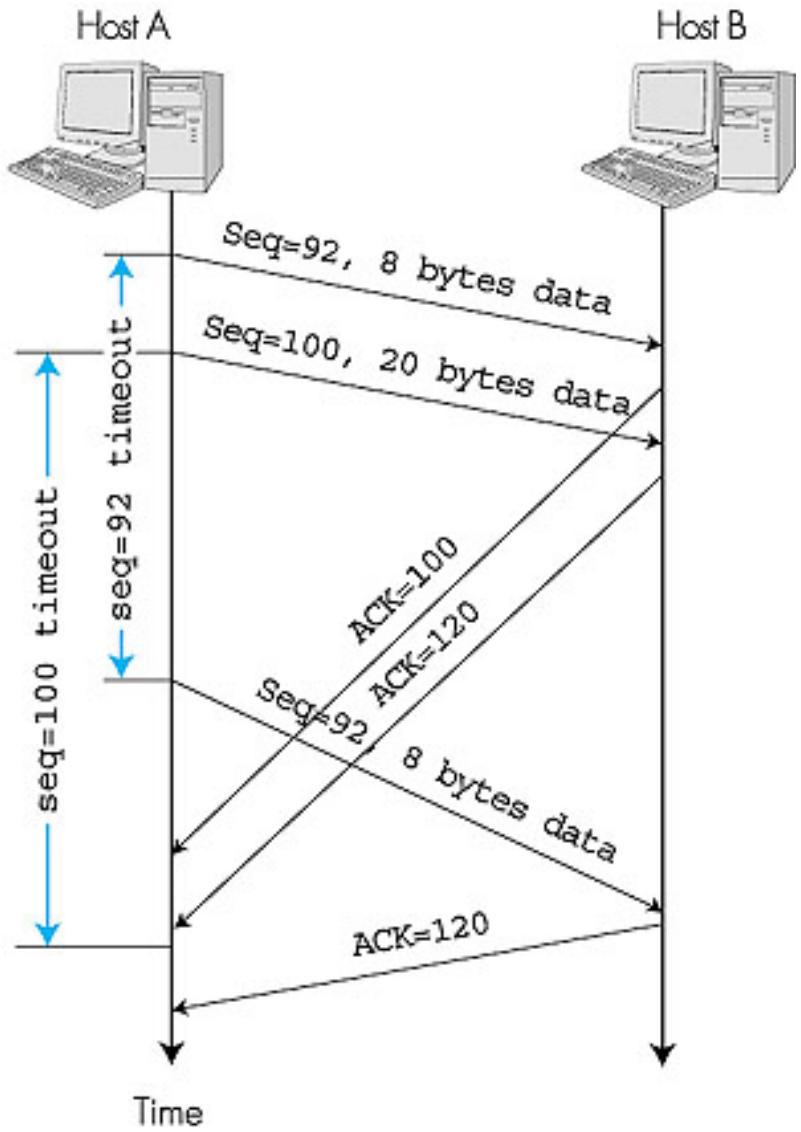


Figura: Envío de datos de cliente a servidor y acuses de recibo en sentido contrario.
Fuente: Retransmisión y Timeout

3.4 Envío de datos (V)

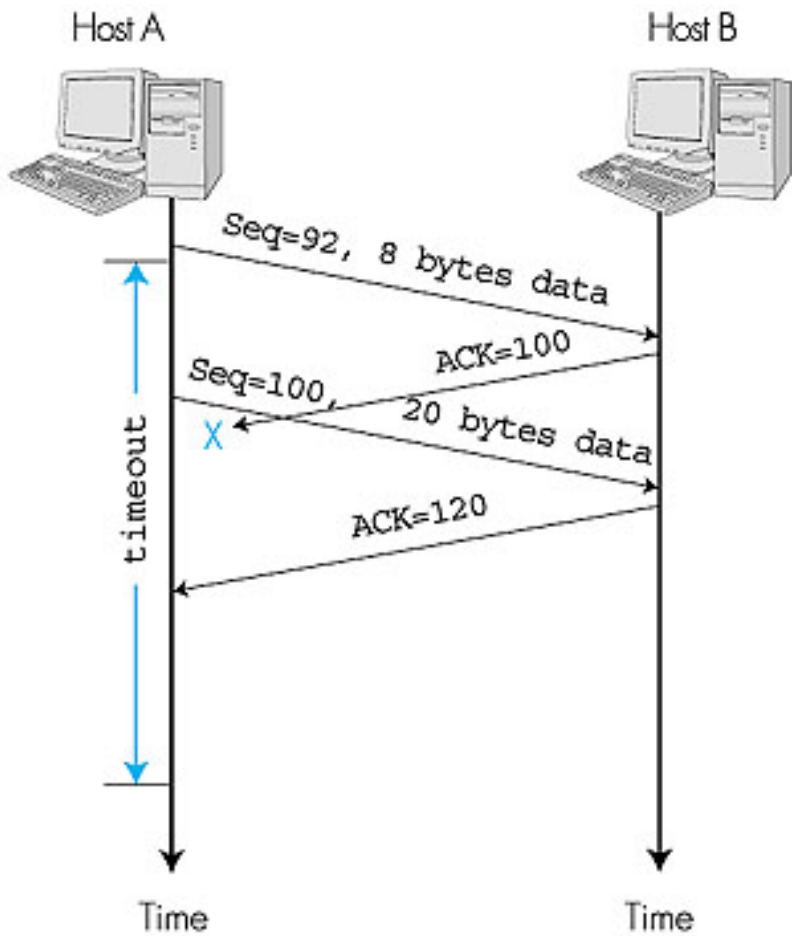
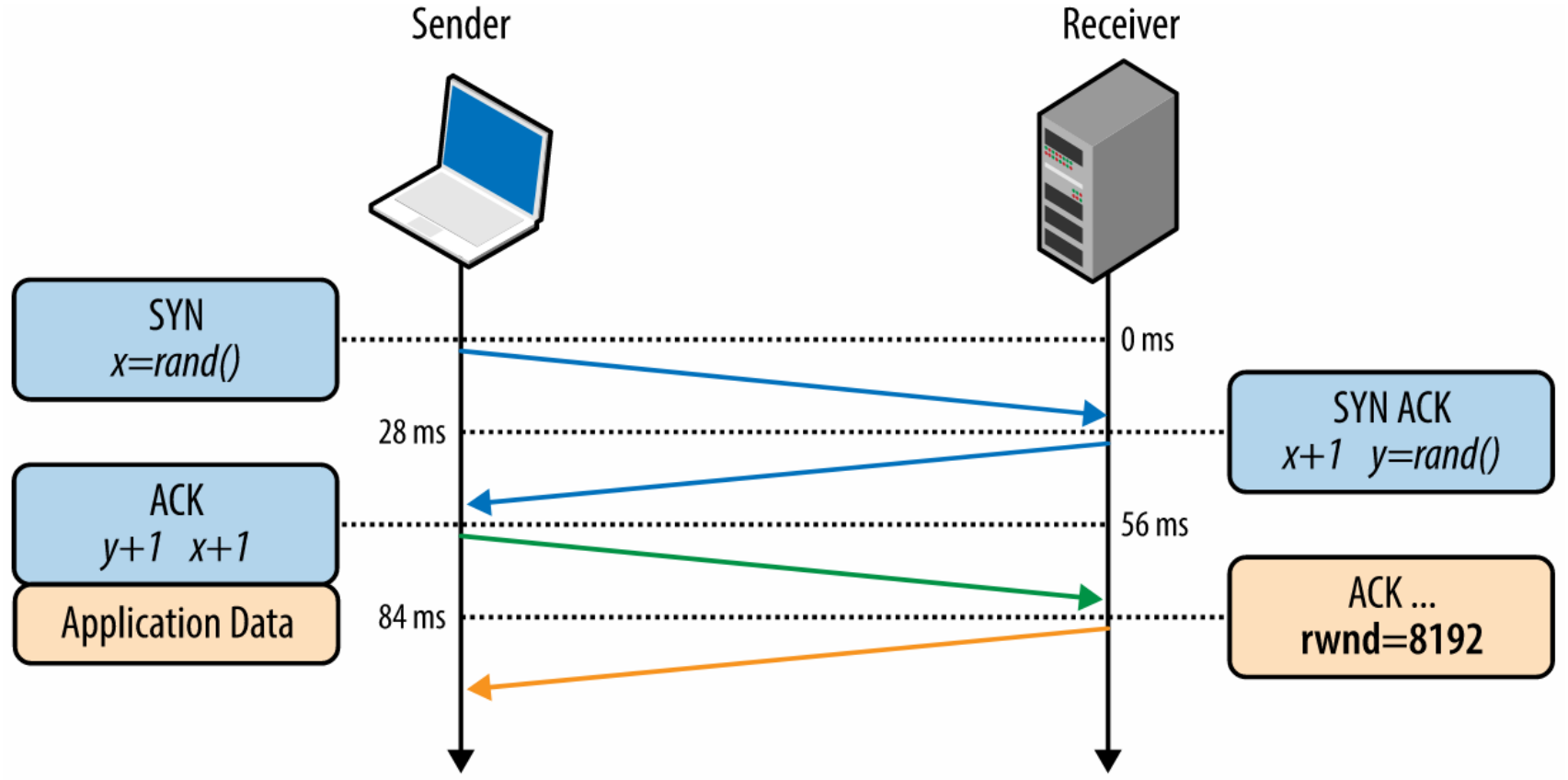


Figura: Envío de datos de cliente a servidor y acuses de recibo en sentido contrario.
Fuente: Retransmisión y ACK acumulado

3.5 Control de flujo

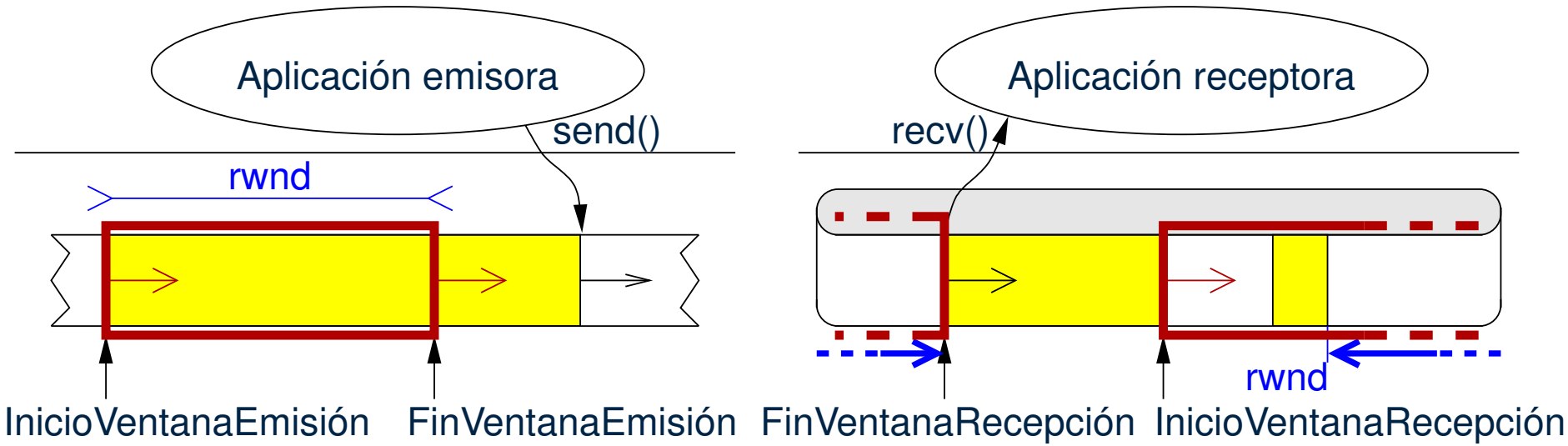
- Objetivo: impedir que el emisor desborde al receptor
- Ventana de recepción, $rwnd$: en cada ACK el receptor indica el número de bytes que puede aceptar
  - Rango números de secuencia: $ackn : ackn + rwnd - 1$



Fuente: High Performance Browser Networking: Building Blocks of TCP

3.5 Control de flujo (II)

- Ventana deslizante de tamaño variable, $rwnd$
- Receptor informa del espacio libre de su buffer en el campo *Ventana*: $rwnd = BufferSize - Ocupado$
- Emisor tiene un límite de datos enviados sin confirmar: $rwnd \geq FinVEmis - IniVEmis$
- Si $rwnd = 0$, envíos de tamaño mínimo cada cierto tiempo



3.5.1 Ejercicios con control de flujo

- Variables en juego:
  - `AdvertisedWindow=MaxRcvBuffer-((NextByteExpected-1)-LastByteRead)`
  - `LastByteSent-LastByteAcked` $\leq$ `AdvertisedWindow`
  - `EffectiveWindow = AdvertisedWindow - (LastByteSent - LastByteAcked)`
  - `LastByteWritten-LastByteAcked` $\leq$ `MaxSendBuffer`

✍ Determina las características de un enlace TCP considerando que el emisor siempre tiene datos a escribir y el receptor lee los datos en cuanto llegan y el RTT se puede aproximar a 0:

- Be=32 KiB, Br=16 KiB: ¿$rwnd$, $ewnd$?

- Be=16 KiB, Br=32 KiB: ¿$rwnd$, $ewnd$?

3.5.1 Algoritmo de Nagle

```
if (hay más de MSS bytes de datos) y
(la ventana lo permite) then
    enviar MSS bytes
else
    if (se espera algún ACK) then
        acumular datos en buffer
    else
        enviar datos del buffer
    end if
end if
```

- Recepción de ACK → activación envío
- Puede desactivarse: opción `TCP_NODELAY` en `setsockopt()`

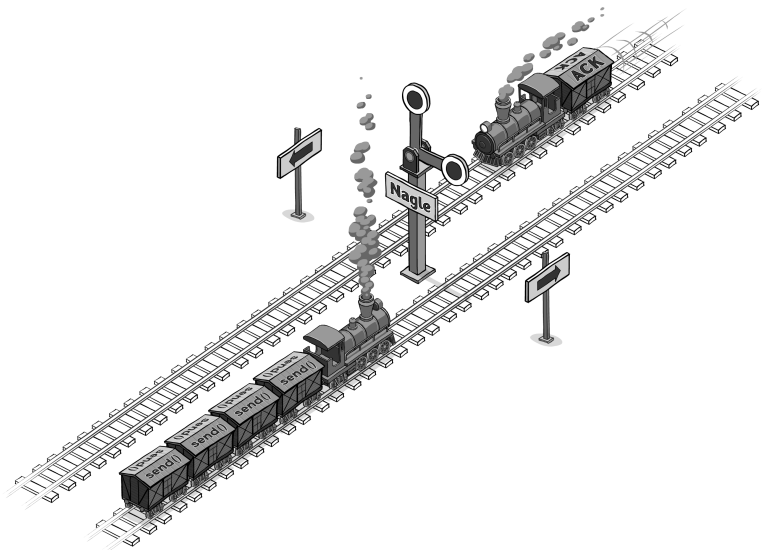


Imagen: http://ithare.com/tcp-peculiarities-as-applied-to-games-part-ii/

3.6 Opciones TCP

- RFC 7323
- Las funcionalidades poco usadas y las nuevas se implementan como opciones en el protocolo básico
- Cada extremo TCP puede incluir opciones en un segmento SYN. Las más comunes son:
- *Escalado ventana de recepción TCP ($rwnd$)*:
  - $rwnd$: 16 bits en cabecera $\rightarrow rwnd_{max} = 64KiB$
  - Conexiones alta velocidad (v_t) o latencia elevada (RTT) requieren ventanas mayores: $V_t \cdot RTT$
    - Por ejemplo, 100 Mbps con RTT de 96 ms $\rightarrow$ $rwnd = 1.2\ MiB$
  - $rwnd = rwnd << nbits = rwnd \cdot 2^{nbits}$, con $nbits = [0 - 14]$, $\rightarrow rwnd_{max} = 65535 \cdot 2^{14} = 1\ GiB$

3.6 Opciones TCP (II)

- *Marca de tiempo TCP (timestamp)*: 32 bits que permiten
  - Medir RTT: *Round-Trip Time Measurement (RTTM)*
  - Extender el rango de números de secuencia ($nseq$) para no reusar valores antes de *Maximum Segment Lifetime* (MSL): *PAWS - Protection Against Wrapped Sequences*
- *PAWS*
  - Con $nseq$ de 32 bits $\rightarrow 2^{32} = 4$ GiB y MSL entre 1 y 2 min
    - 100 Mbps $\rightarrow$ reúso en 5.7 min, 10 Gbps $\rightarrow$ reúso en 3.4 s!
  - Orden de segmentos especificado por combinación de campos <marca de tiempo, número de secuencia>
- Nota: *Maximum Segment Life (MSL)* es el tiempo que un segmento puede existir en la red
  - Según RFC 793: 2 minutos

3.7 Retransmisión adaptativa

- El retardo en la recepción de un ACK es muy variable
- El tiempo de expiración (*retransmission timeout, RTO*) debe ajustarse al RTT:
  - Retransmitir demasiado tarde infrautiliza la red
  - Retransmitir demasiado pronto añade sobrecarga

Objetivo del emisor: para cada envío, establecer su tiempo de expiración (RTO) en función del RTT estimado ($EstimRTT$)

1. Cuando llega ACK, mide el tiempo que ha tardado desde su correspondiente envío: $MuestraRTT$
2. Estima el RTT previsto para el envío: $EstimRTT$
3. Establece el timeout para el envío: RTO

- NO se toma muestra cuando se ha producido un reenvío
- ¿Valor de RTO?

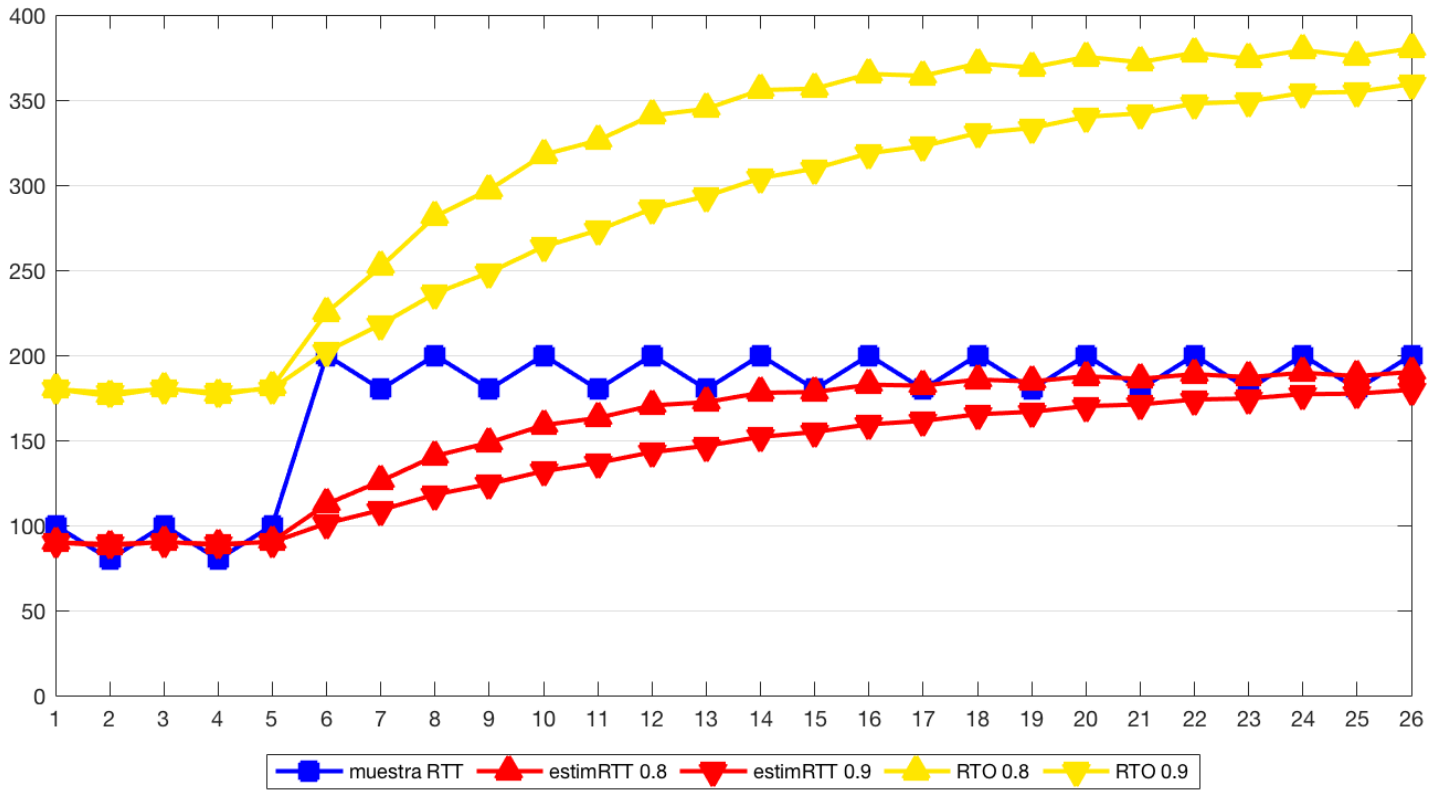
3.7.1 Algoritmo TCP original

- RFC 793: cálculo sencillo del tiempo de expiración

$$EstimRTT = \alpha \cdot EstimRTT + (1 - \alpha) \cdot MuestraRTT$$

con $\alpha \in [0.8,\ 0.9]$

$RTO = \beta \cdot EstimRTT$, con $\beta \in [1.3,\ 2.0]$



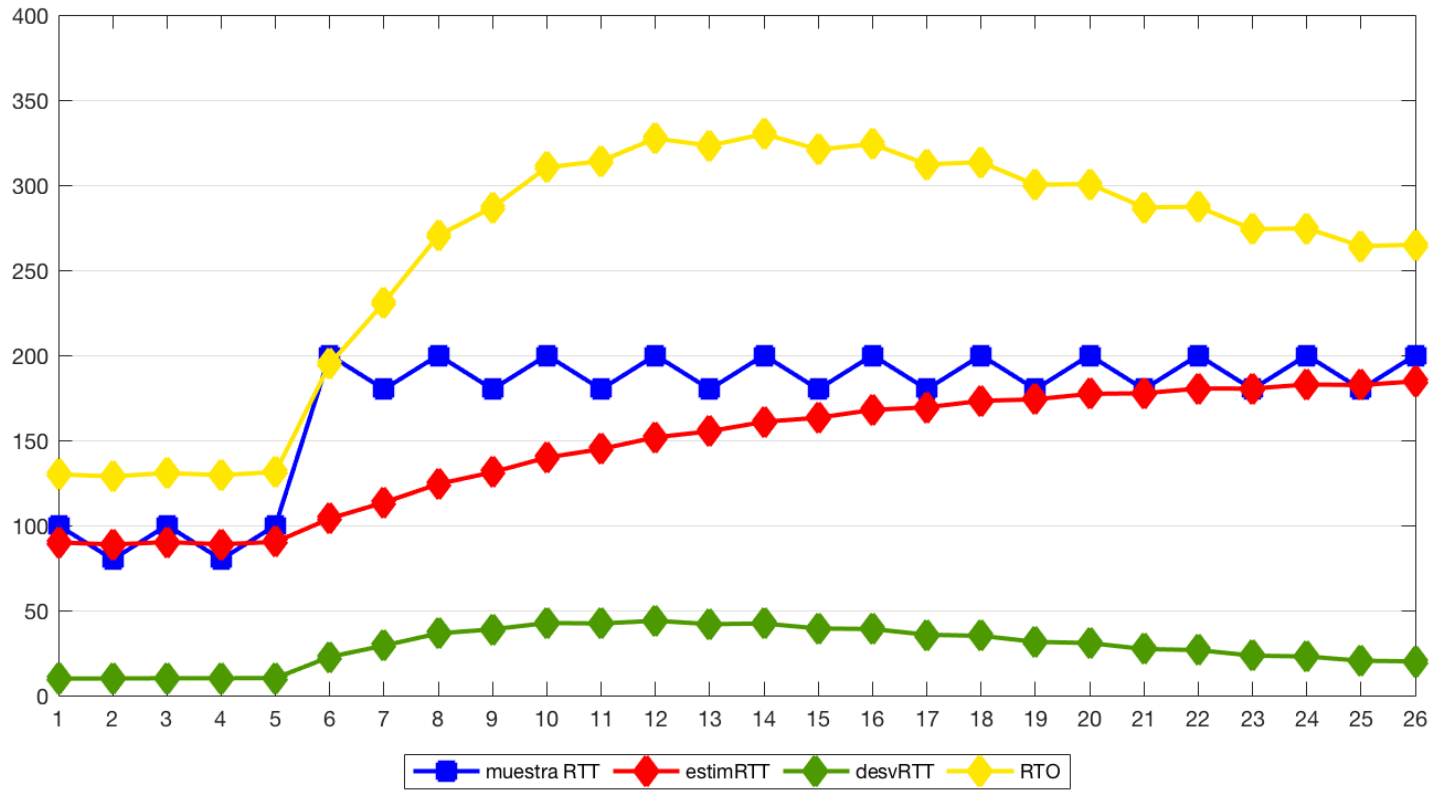
3.7.2 Algoritmo Jacobson/Karels

- Incorpora varianza de RTT a la estimación

$$DifRTT = MuestraRTT - EstimRTT$$
$$EstimRTT += \delta \cdot DifRTT$$
$$DesvRTT += \delta(|DifRTT| - DesvRTT)$$
$$RTO = \mu \cdot EstimRTT + \phi \cdot DesvRTT$$
$$(\delta \in [0,\ 1],\ \mu = 1,\ \phi = 4)$$

3.7.2 Algoritmo Jacobson/Karels (II)

Ejemplo Jacobson/Karels: paso de RTTs de 80-100 ms a 180-200 ms, $\delta = 1/8$. Se ignora el efecto de paquetes perdidos.



3.8 Conclusiones

Caracterísica	UPD	TCP
Fiabilidad		
Velocidad		
Datos acotados		
Conexión		
Orden		
Overhead		
Multicast		

- En una conexión TCP, si el número de secuencia de un segmento de esta conexión es m, entonces ¿el número de secuencia del siguiente segmento será m + 1?