Teoría de las Comunicaciones

Edición 65 oficial, pero en realidad desde que se comenzó a dictar (a.k.a Redes) es la edición 74. Este es el último cuatrimestre que se dicta.

Dr. Claudio Enrique Righetti

1 octubre 2025

Segundo Cuatrimestre

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Argentina

Agenda : Módulos (1-4)

Módulo 1: Fundamentos de Redes

Historia y evolución de redes, modelos OSI y TCP/IP, señales analógicas vs. digitales, teoría de la información y teoremas de Shannon. Práctica: cálculo de entropía, codificación Huffman y análisis de capacidad de canal.

Módulo 2: Nivel Físico y Enlace

Técnicas de codificación y modulación, medios de transmisión, protocolos de enlace y mecanismos de ventana deslizante. Práctica: implementación de protocolos nivel 2, CRC y algoritmos Stop & Wait.

Módulo 3: Redes de Acceso y LAN

Tecnologías inalámbricas, Ethernet, switching y VLANs. Práctica: configuración de switches y análisis de tráfico con Wireshark.

Módulo 4: Nivel de Red

Redes Orientadas a conexión (Circuitos Virtuales) y sin Conexión (Datagramas). **Protocolo IP**, tablas de ruteo, algoritmos Distance Vector y Link State. Práctica: direccionamiento IP, ICMP y configuración de RIP y OSPF.

Al finalizar estos cuatro módulos, se realizará el primer parcial para evaluar la comprensión de los conceptos fundamentales de redes. Estos módulos establecen las bases necesarias para abordar temas más avanzados en la segunda parte del curso.

Estructura de Módulos (5-8)

Módulo 5: Capa de Transporte

Análisis detallado del protocolo TCP, cálculo del RTO, mecanismos de control de flujo y errores. Práctica: comparación UDP vs. TCP, máquina de estados y análisis de conexiones.

Módulo 6: Control de Congestión

Problema de congestión en redes, curvas de tráfico, taxonomía de soluciones y mecanismos de realimentación. Práctica: implementación de algoritmos Slow Start y Congestion Avoidance.

Módulo 7: Capa de Aplicación

Arquitecturas cliente-servidor vs. peer-to-peer, servicios fundamentales como Web, correo electrónico y DNS. Práctica: herramientas de diagnóstico y protocolos HTTP, SMTP, POP3, IMAP.

Módulo 8: Performance y Seguridad

Análisis de rendimiento, factores que afectan el desempeño, algoritmos criptográficos y conceptos de seguridad. Práctica: configuración de firewalls, certificados digitales y protocolos seguros.

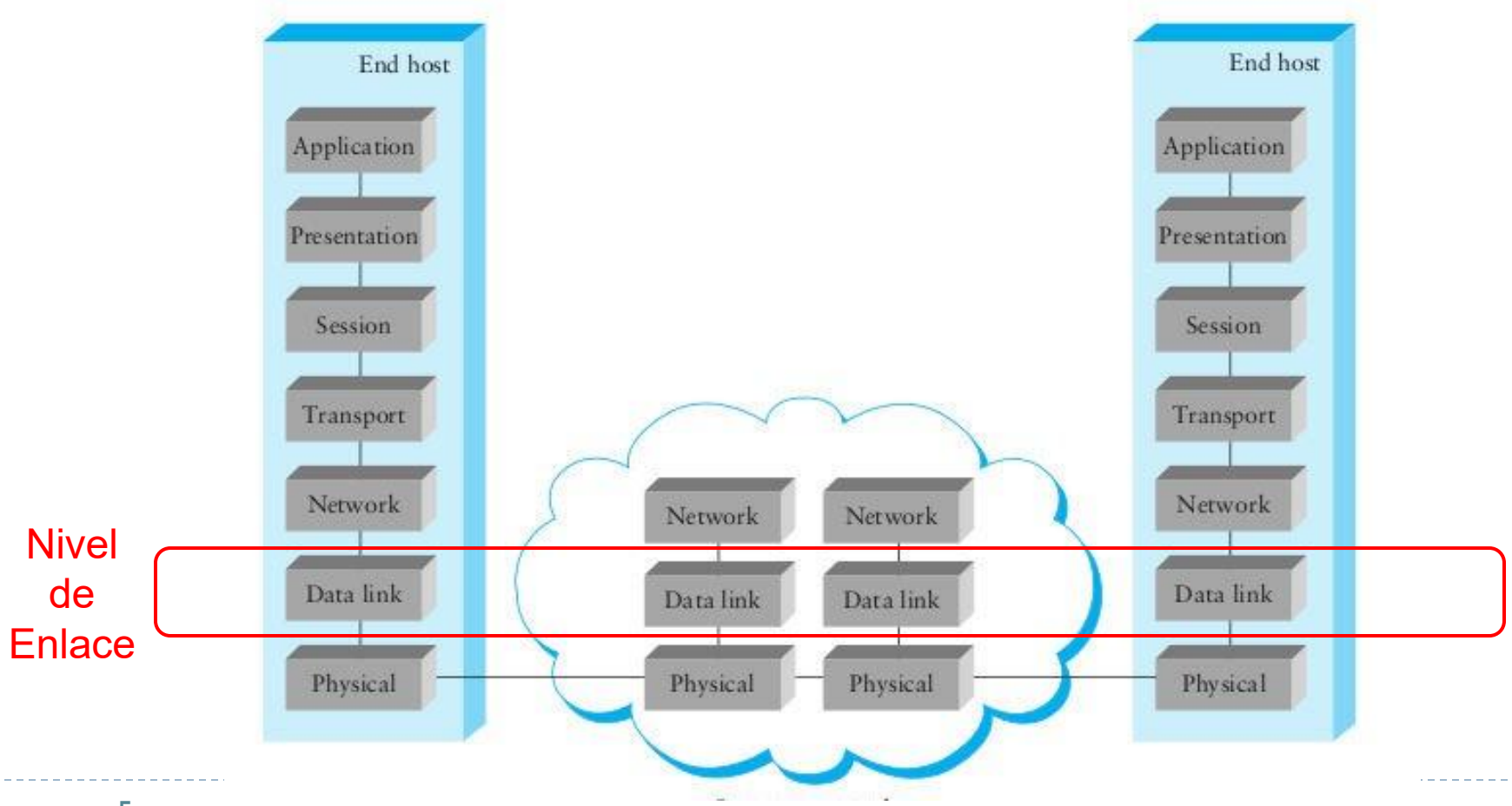
Estos módulos avanzados completan la formación integral en redes, abordando aspectos críticos como el transporte confiable de datos, la gestión de congestión, las aplicaciones de red y la seguridad. Al finalizar el curso, los estudiantes habrán adquirido tanto conocimientos teóricos sólidos como experiencia práctica en la implementación y análisis de redes.

Enlaces punto a punto

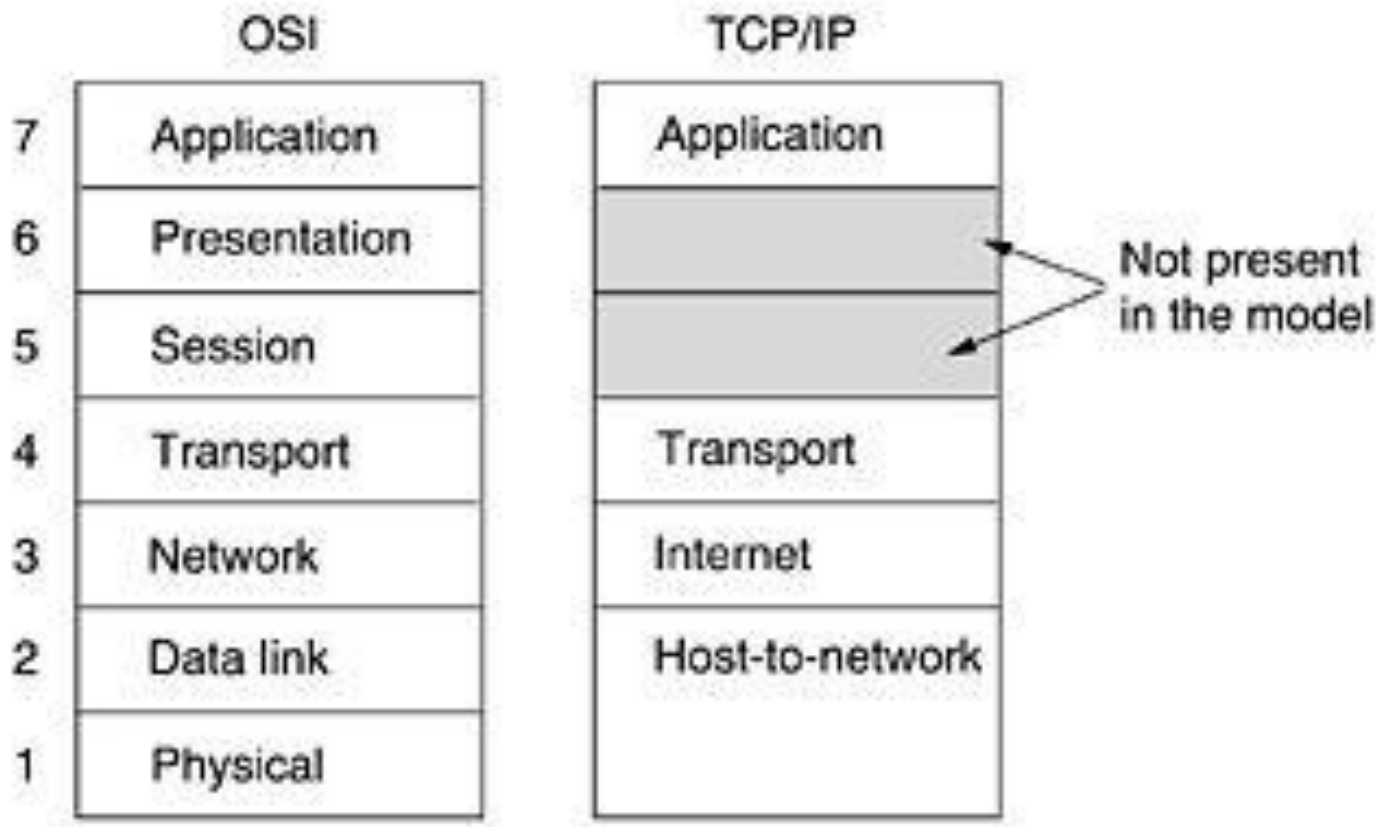
Nivel de Enlace

Paradigma de capas

- Las comunicaciones se dan en capas que se brindan servicios entre sí

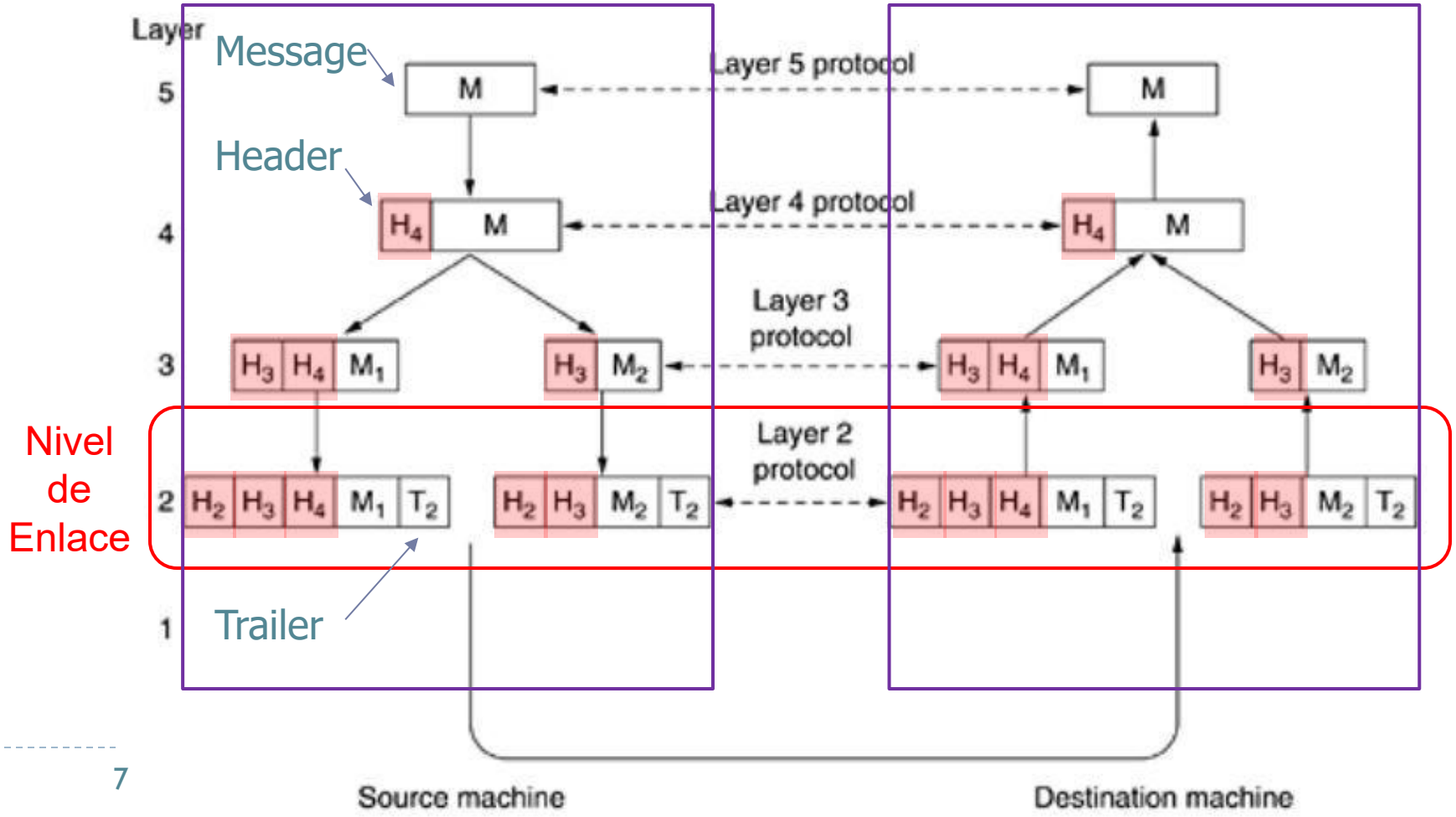


OSI-ISO vs Internet (TCP-IP)



Paradigma de capas

- Cada nueva capa implica el agregado de información de control en forma de encabezados (Headers)



Conceptos de Nivel de Enlace

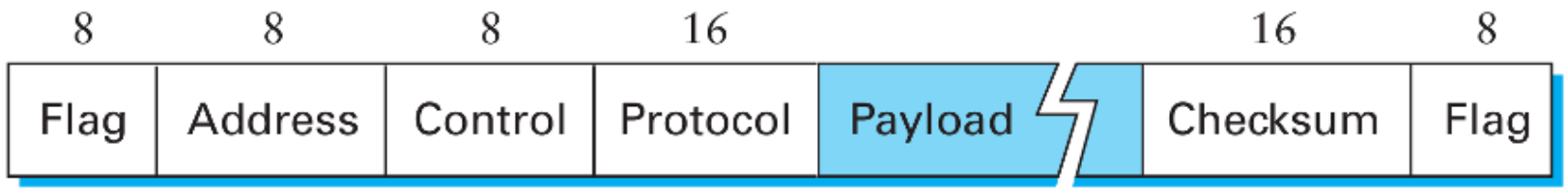
- Tenemos un "caño" serial (no hay desordenamiento)
- Pero: sujeto a ruido y fallas
  - Lo que se recibe puede no ser lo que se envió: "error de transmisión" (algo falló en el Nivel Físico)
- Objetivos
  - Proveer servicio a la capa superior
    - **Confiabilidad**. ¿Confiable o no confiable?
    - **Control de Errores**. ¿Se produjo algún error? ¿Qué hacemos con los errores?
    - **Control de Flujo**. Más adelante: en Nivel de Transporte.
- Estrategia
  - Encapsulamiento o "Framing"
    - Encapsular los bits de Mensaje en Frames
    - agregando Información de Control

Encapsulamiento (Framing)

- ¿Cómo se separan los frames en un tren de bits?

- Opciones:
  - Largo fijo
  - Largo especificado en el encabezado
  - Delimitadores de frame (con bit-stuffing)

- Ejemplo PPP:



Tipo de Servicio

- Sin conexión y sin reconocimiento
  - Los datos se envían sin necesidad de saber si llegan con errors o no.
- Sin conexión y con reconocimiento
  - Los datos se envían y se asegura la correcta recepción sin errores mediante el aviso explícito (ACKs).
- Orientado a conexión
  - Además de asegurar la ausencia de errores en la recepción de los datos, se mantiene un estado de conexión (una sesión).

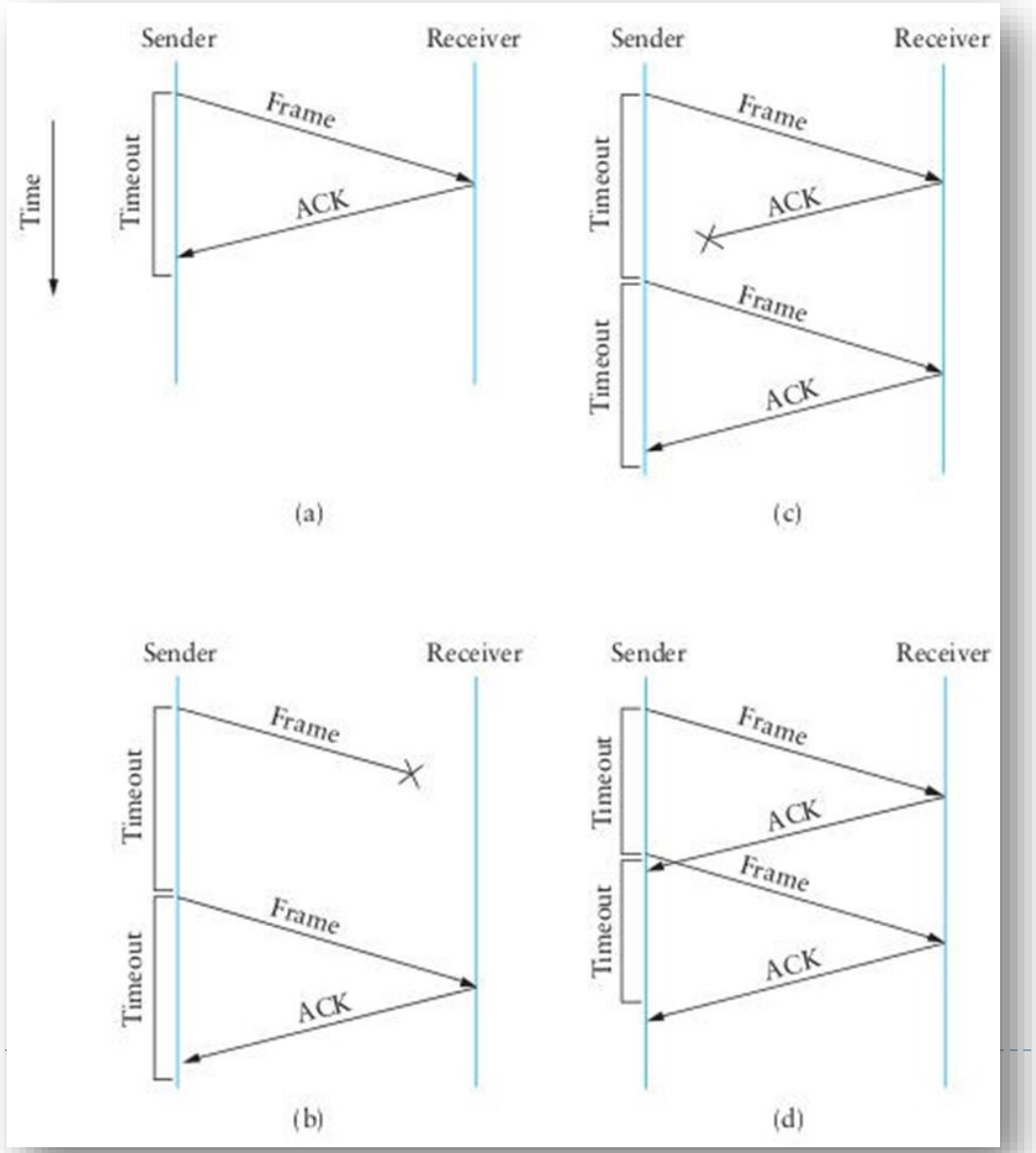
Detección y Corrección de errores

- Redundancia:
  - **m** bits (datos) + **r** bits (redundancia) = **n** bits (codeword)
- Definiendo:
  - **d** la mínima *Distancia de Hamming* entre todas las codewords de un código.
  - **e** la cantidad de bits erróneos en una transmisión dada
- Necesitamos:
  - **e + l** ≤ **d** para poder detectar
  - **2e + l** ≤ **d** para poder corregir

- Para la **confiabilidad**
  - **Surge la nececidad de poder efectuar retransmisiones**
    - Implícitas (cuando ocurre un time-out se asume que el dato se perdió)
    - Explícitas (mensajes de control específicos para pedir repetición de envío de datos)

Transmisión Confiable: Stop & Wait

- Cada Frame debe ser reconocido por el receptor.
- Problema de las reencarnaciones
  - Surge la necesidad de secuenciar (numerar unívocamente)
  - En Stop & Wait se necesita secuenciar al menos 2 frames.
- Existe un tiempo de bloqueo a la espera de confirmaciones.



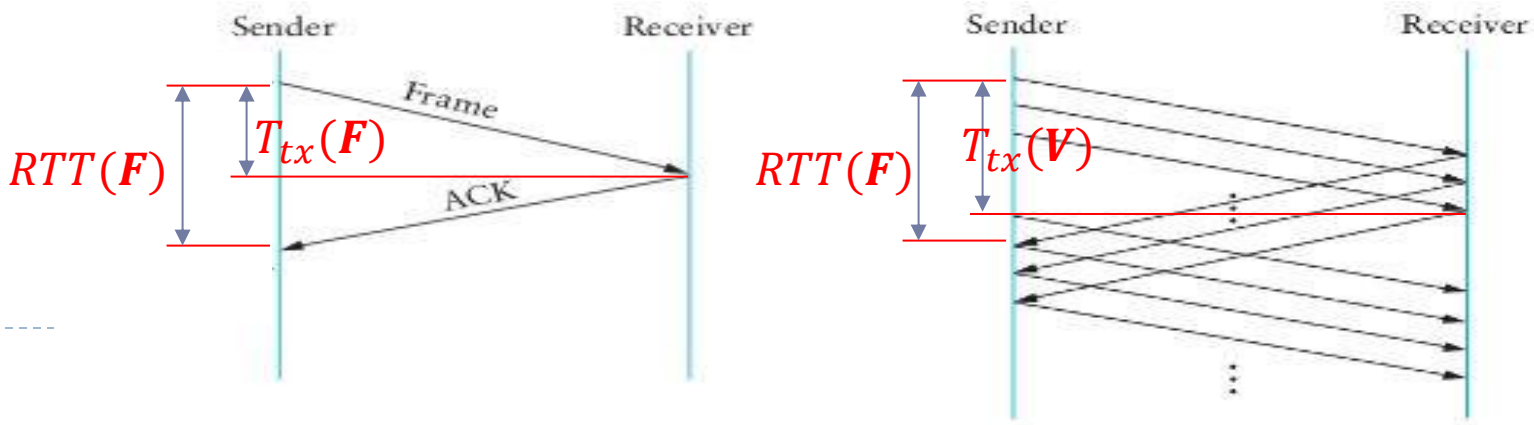
Eficiencia de protocolo

- ¿Cuánto *tiempo se está transmitiendo* con respecto al *tiempo que se está esperando* por las confirmaciones?

$$\eta_{proto} = \frac{T_{tx}}{RTT(\boldsymbol{F})}$$

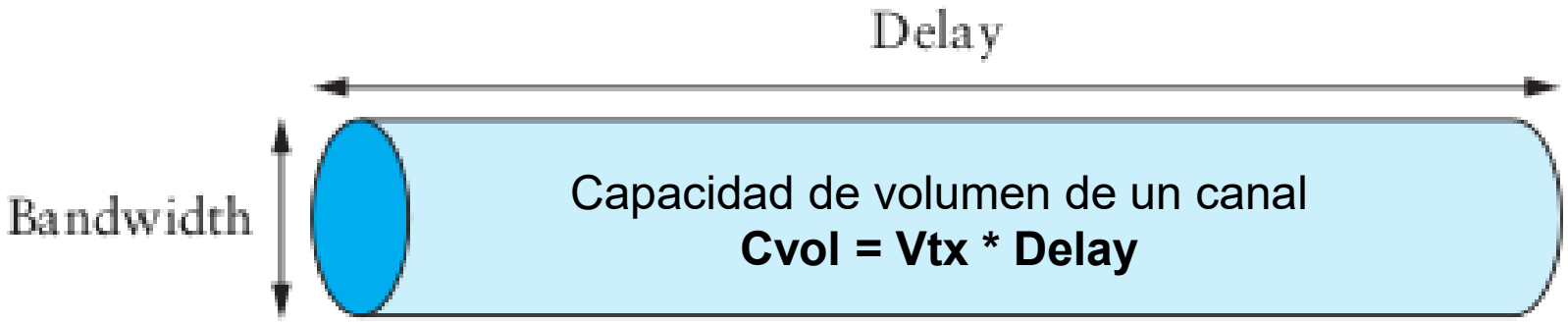
- **Aumentar la eficiencia** es estar lo menos posible bloqueado esperando
- Estrategia: Enviar "varios **F**rames seguidos, sin recibir ACKs"
  - Concepto de "**V**entana de frames" :

$$\eta_{proto} = \frac{T_{tx}(\boldsymbol{V})}{RTT(\boldsymbol{F})}$$



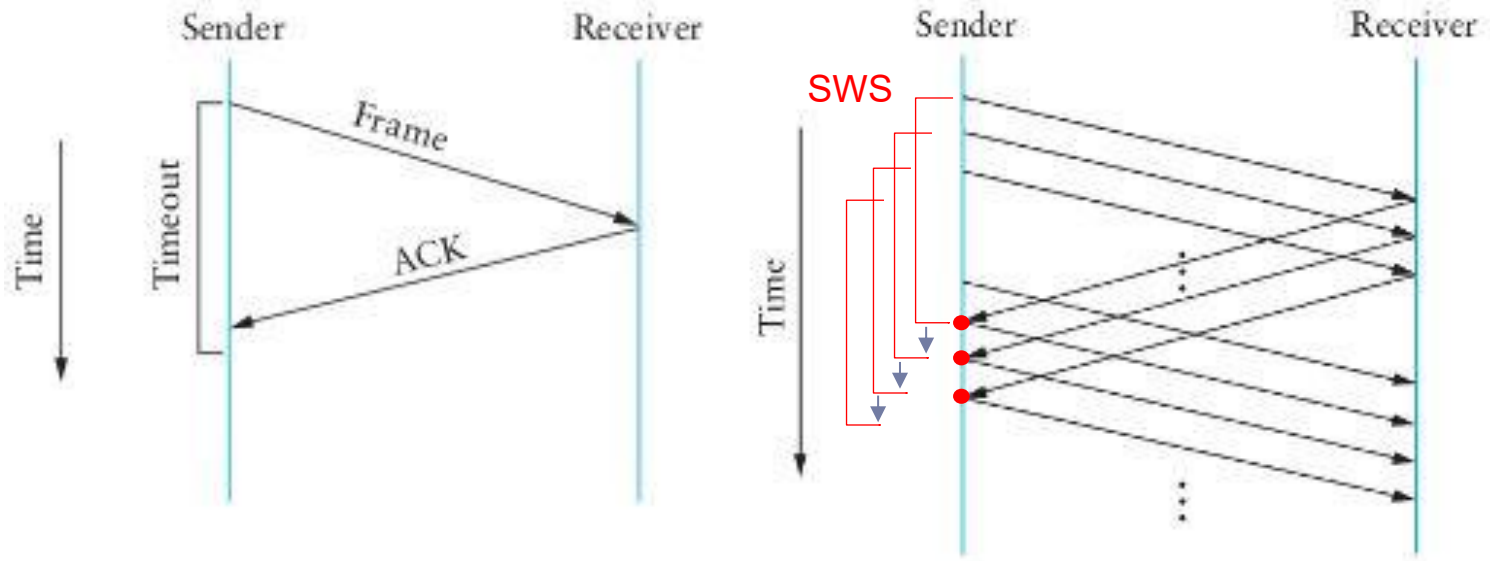
Capacidad de volumen de un canal

- Multiplicando la Velocidad de Transmisión V_{tx} por el Delay se obtiene la cantidad de bits que "entran" en un canal.
  - Para aprovecharlo mejor, deberíamos calcular cuantos bits entran en el canal hasta que llega el primer ACK (es decir, usando **2*Delay=RTT**)



Transmisión confiable: Sliding Window

- Motivación:"mantener lleno el canal"

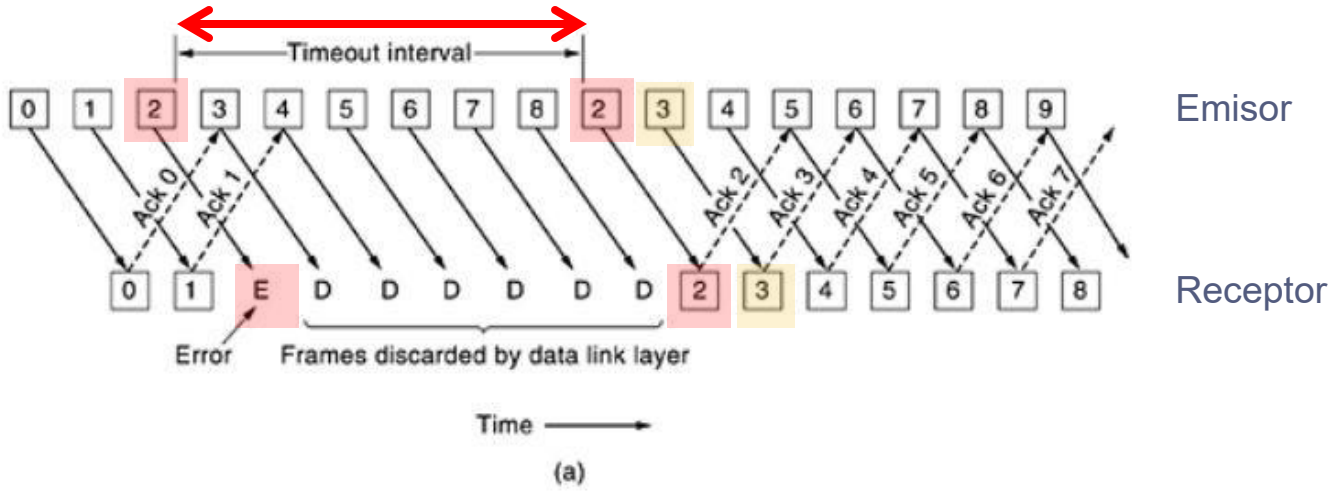


SWS: Send Window Size

- Ventana de emisión: $SWS = \frac{V_{tx}.RTT}{|Frame|}$ frames
- Enviar nuevo frame siempre que se verifique:
  ÚltimoFrameEnviado $\leq$ *ÚltimoFrameReconocido* + *SWS*

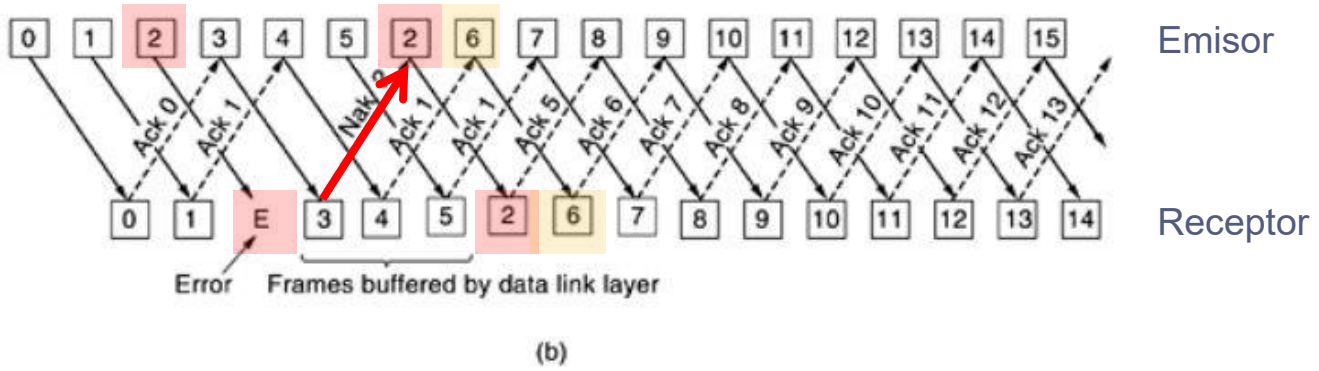
Transmisión confiable: ACKs Selectivos

ACKs **acumulativos**



ACKs **selectivos**

(tipo: Negative ACK)

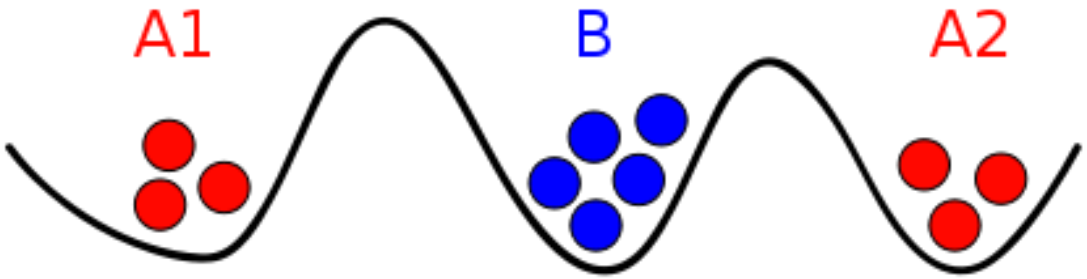


Ventana de recepción: $RWS = \begin{cases} SWS & si\ hay\ Selective\ ACK \\ 1 & si\ no \end{cases}$

Transmisión confiable: Sliding Window

- Resumiendo:
  - Debemos aumentar la ventana de emisión para aprovechar mejor el canal
  - Ventana de emisión: $SWS = \frac{V_{tx}.RTT}{|Frame|}$
  - El receptor puede bufferear o no, dependiendo del esquema de ACKs
  - Ventana de recepción: $RWS = \begin{cases} SWS & si\ hay\ Selective\ ACK \\ 1 & si\ no \end{cases}$
- Y para distinguir reencarnaciones:
  #frames unívocamente identificables ≥ SWS + RWS

Confiabilidad.
El problema de los dos generales (*)



- A1 a A2: ”Atacaremos el 4 de agosto a las 09:00“
- A2 a A1: ”Recibido: Atacaremos 4 de agosto a las 09:00“
- A1 a A2: ”Recibido: Recibido: Atacaremos 4 de agosto a las 09:00“
- . . .

- No existe un algoritmo para la confiabilidad
- Enviamos **un único** mensaje de reconocimiento (ACK)

(*) Video de divulgación sobre este problema planteado desde la óptica de los “algoritmos distribuidos” (por Sebastián Uchitel) https://vimeo.com/141434827

Vinton Cerf
en Exactas-UBA
22-08-2007

Nivel de Transporte

TCP (Transmission Control Protocol)

https://exactas.uba.ar/universo-web/
https://spectrum.ieee.org/vint-cerf-mistakes

Bibliografía Básica

- Computer Networks, A Systems Approach (Fifth Edition) (The Morgan Kaufmann Series in Networking) Larry L. Peterson, Bruce S. Davie (2011)
  - Capitulo 5 Protocolos End to End – TCP págs. 396-431
  - Capitulo 6 Control de Congestión (CC) y Alocación de Recursos – págs. 479-499 y TCP CC págs. 499-530

Conveniente complementar con la lectura de:

- Computer Networks (5th. Edition) Andrew S. Tanenbaum, David J. Wetherall (2010)
  - TCP: págs. 552-571, Algoritmos de Control de Congestión pags 392-404
  - Nivel de Transporte – CC págs. 530-541
  - TCP CC págs. 571-581

Referencias

- Cerf, V., and R. Kahn, "A Protocol for Packet Network Intercommunication", IEEE Transactions on Communications, Vol. COM-22, No. 5, pp 637-648, May 1974.
- [RFC 793] J. Postel "Transmission Control Protocol" September 1981. (varios errores e inconsistencias)
- [RFC 1122] October 1989 (se salvan bugs de la RFC 793)
- [RFC 6298] Paxon, Allman, Chu and Sargent, "Computing TCP's Retransmission Timer", June 2011
- [RFC1323] Jacobson, V., Braden, R., and D. Borman, "TCP Extensions for High Performance", May 1992.
- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", October 1996.
- [RFC2988] Paxson, V. and M. Allman, "Computing TCP's Retransmission Timer", November 2000.
- Una recopilación al 2006: "A Roadmap for Transmission Control Protocol (TCP) Specification Documents" RFC 4614

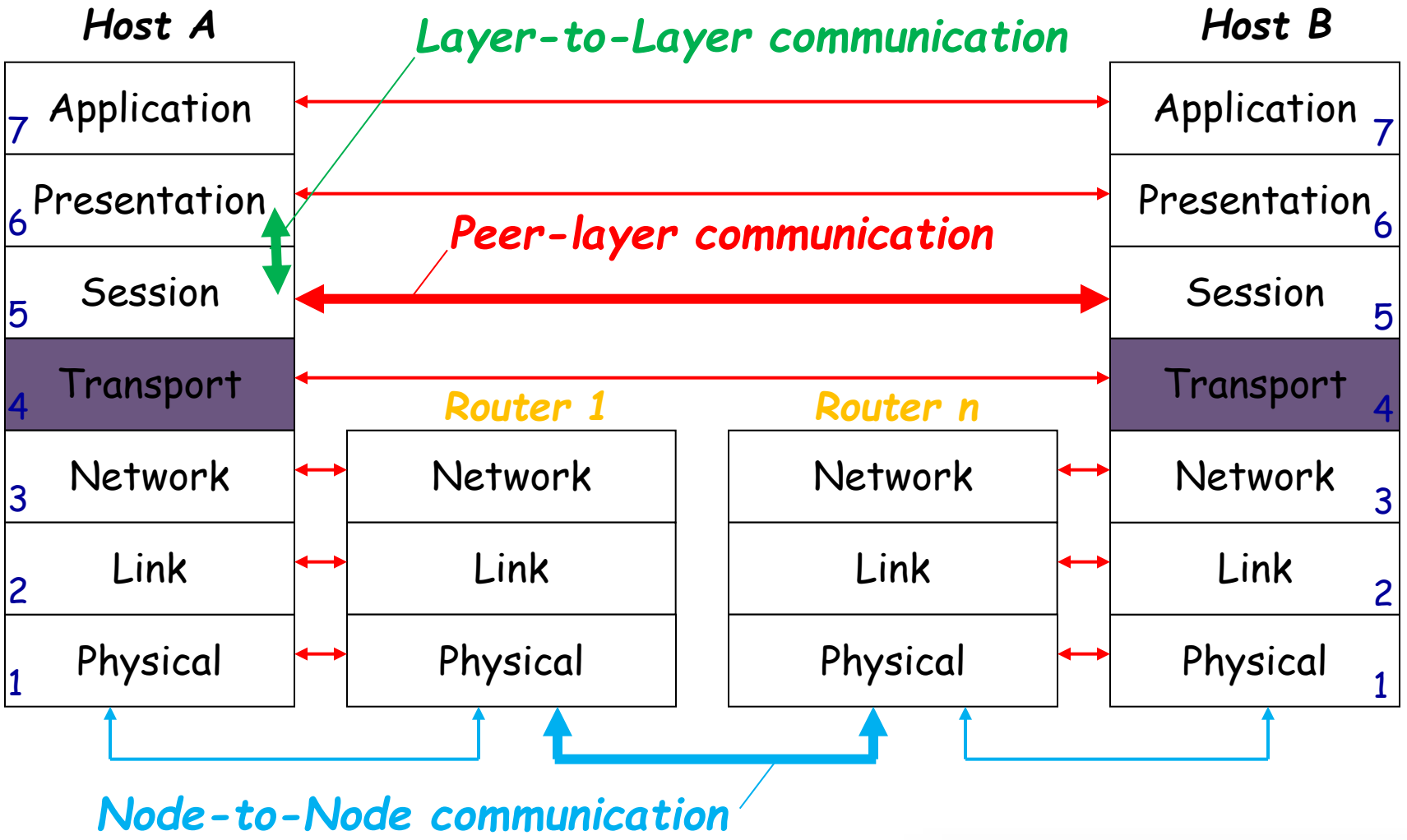
Agenda

- Nivel de Transporte
- El Protocolo TCP (Transmission Control Protocol)
  - TCP Connection setup
  - Segmentos TCP
  - Números de Secuencia TCP
  - TCP Sliding Window
  - Control de Flujo
- El Protocolo UDP (User Datagram Protocol)
- El Protocolo TCP - Dinámicas Especiales
  - Timeouts y Retransmisiones
  - Máquina de Estados Finitos
  - Ineficiencia por envío de segmentos pequeños y que el receptor los pida
  - Uso del ancho de banda
  - Sockets

Repaso: los modelos de capas

Protocolos End to End (E2E)

Modelo OSI



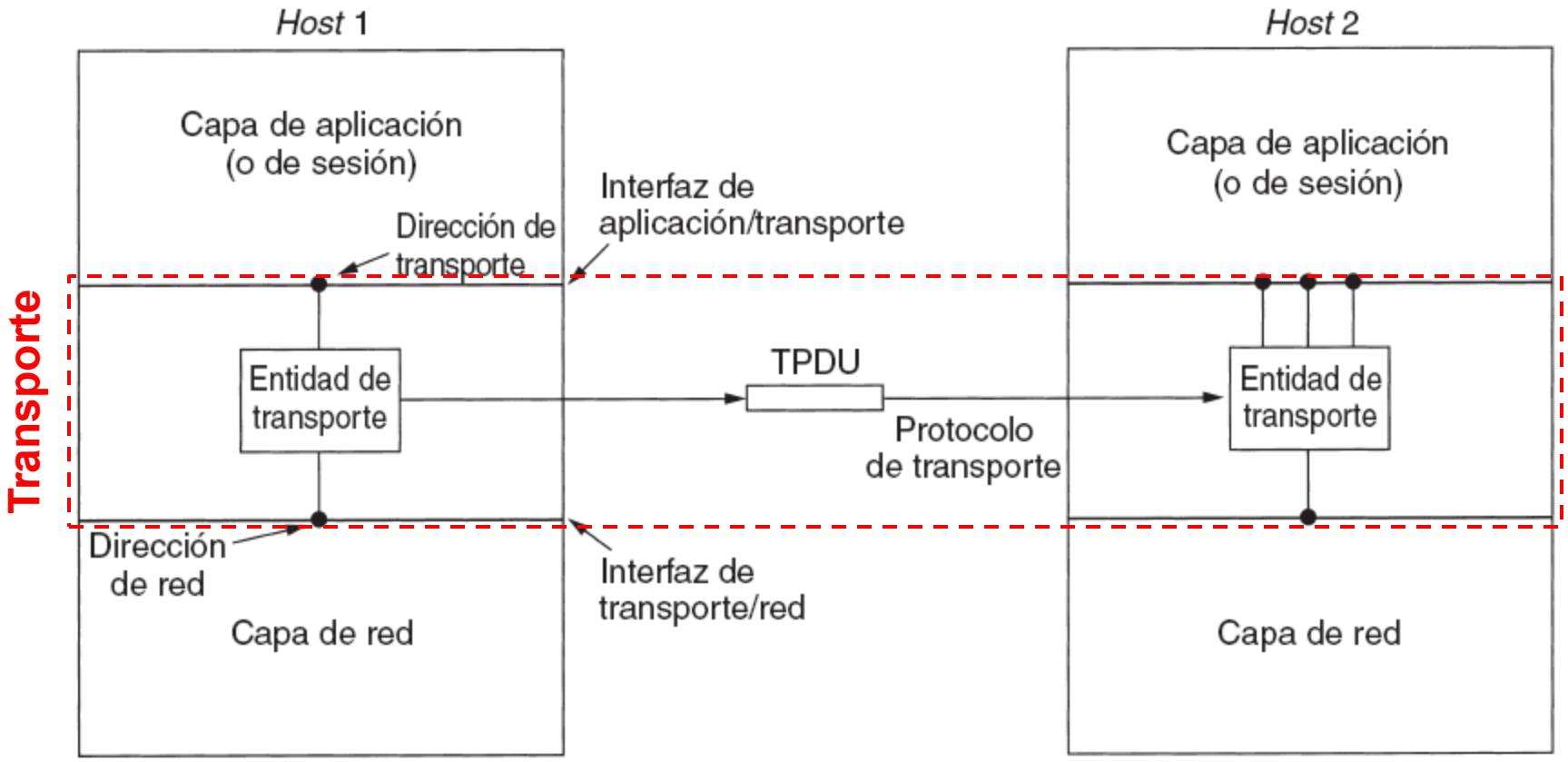
Hubbert Zimmermann, 1980
http://www.cs.cmu.edu/~srini/15-744/F02/readings/Zim80.pdf


IEEE TRANSACTIONS ON COMMUNICATIONS, VOL. COM-28, NO. 4, APRIL 1980 425

OSI Reference Model—The ISO Model of Architecture for Open Systems Interconnection

HUBERT ZIMMERMANN

Modelo OSI



TPDU = **Transport Protocol Data Unit**

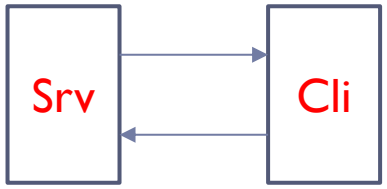
Enlace de Datos versus Transporte

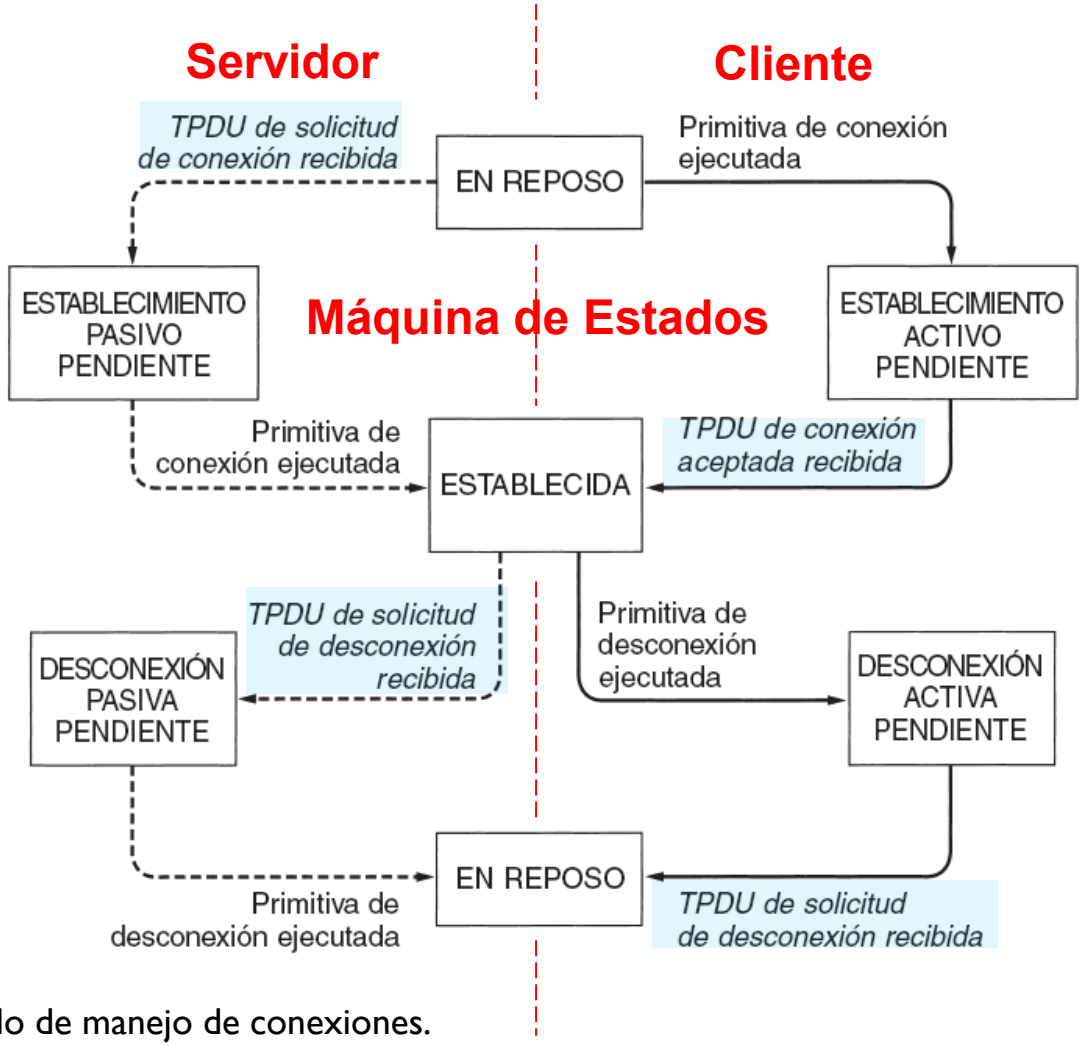
Un servicio de **Transporte**:

- Potencialmente conecta muchas máquinas diferentes
  - requiere de **establecimiento y finalización de conexión** explícitos
- Potencialmente lidia con diferentes RTT
  - requiere mecanismos adaptivos para **timeouts**
- Potencialmente enfrenta largos retardos en la red
  - requiere estar preparado para el arribo de **paquetes muy antiguos**
- Potencialmente existe diferente capacidad de recepción en destino
  - requiere contemplar nodos de diferentes capacidades
- Potencialmente maneja diferente capacidad de red
  - requiere estar preparado para lidiar con **congestión de red**

Modelo OSI

TPDU =
Transport Protocol Data Unit





Máquina de Estados de un esquema sencillo de manejo de conexiones.

- Las **transiciones** ***en cursiva*** son causadas por ***llegadas de paquetes***.
- Las líneas **continuas** muestran la secuencia de **transiciones de estado** del **Cliente**.
- Las líneas **punteadas** muestran la secuencia de **transiciones de estado** del **Servidor**.

Protocolos “End-to-End” en subredes de datagramas

- Se apoyan en la **Capa de Red**, la cual es de **“mejor esfuerzo” (best-effort)**
  - descarta mensajes
  - desordena mensajes
  - puede entregar múltiples copias de un mensaje dado
  - limita los mensajes a algún tamaño finito
  - entrega mensajes después de un tiempo arbitrariamente largo
- Servicios comunes **End-to-End** ofrecidos/deseados
  - **garantía de entrega de mensajes (confiabilidad)**
  - entrega de mensajes en el mismo orden que son enviados
  - entrega de a lo sumo una copia de cada mensaje
  - soporte para mensajes arbitrariamente largos
  - soporte de sincronización
  - permitir al receptor controlar el flujo de datos del transmisor
  - soportar múltiples procesos de nivel aplicación en un receptor

TCP

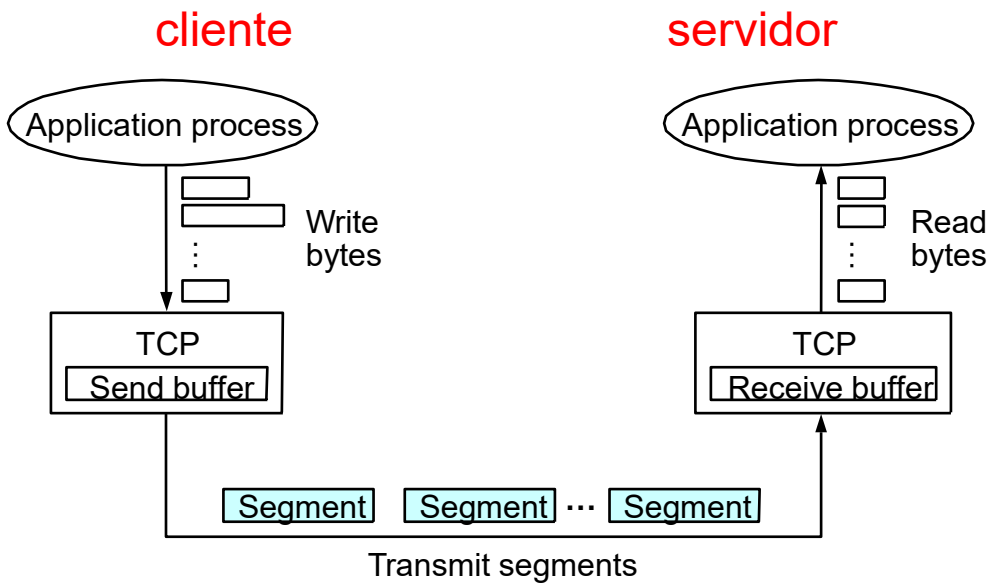
Características

Formato del Header

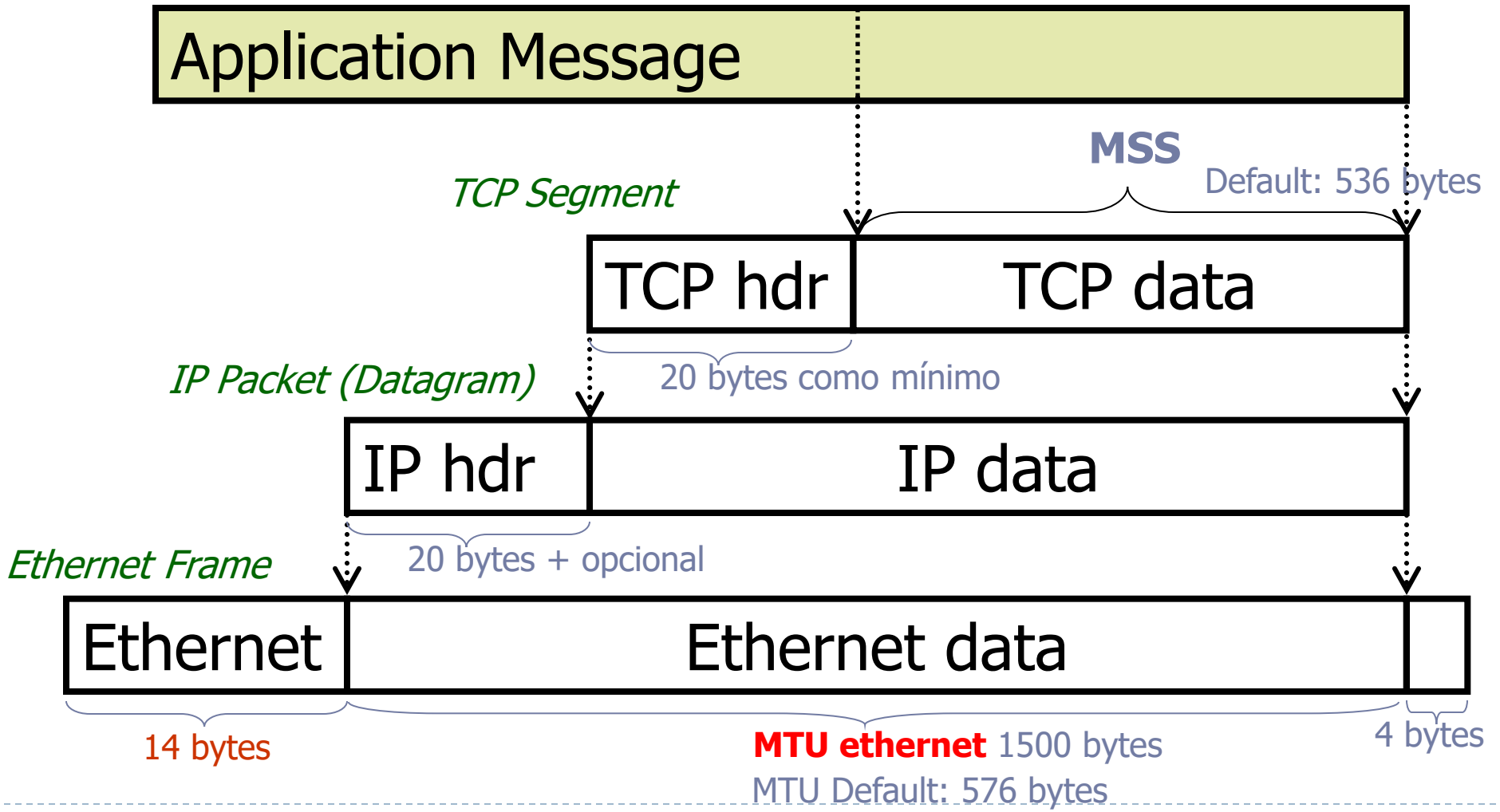
TCP - Generalidades

- Orientado a conexión
- Flujo de bytes
  - App escribe bytes
  - TCP envía *segmentos*
  - App lee bytes

- Full Duplex (dos flujos de bytes)
  - **Control de flujo**: evita que el Tx **inunde al Rx**
  - **Control de congestión**: evita que el Tx **sobrecargue a la red**



MSS: ”Maximum Segment Size”



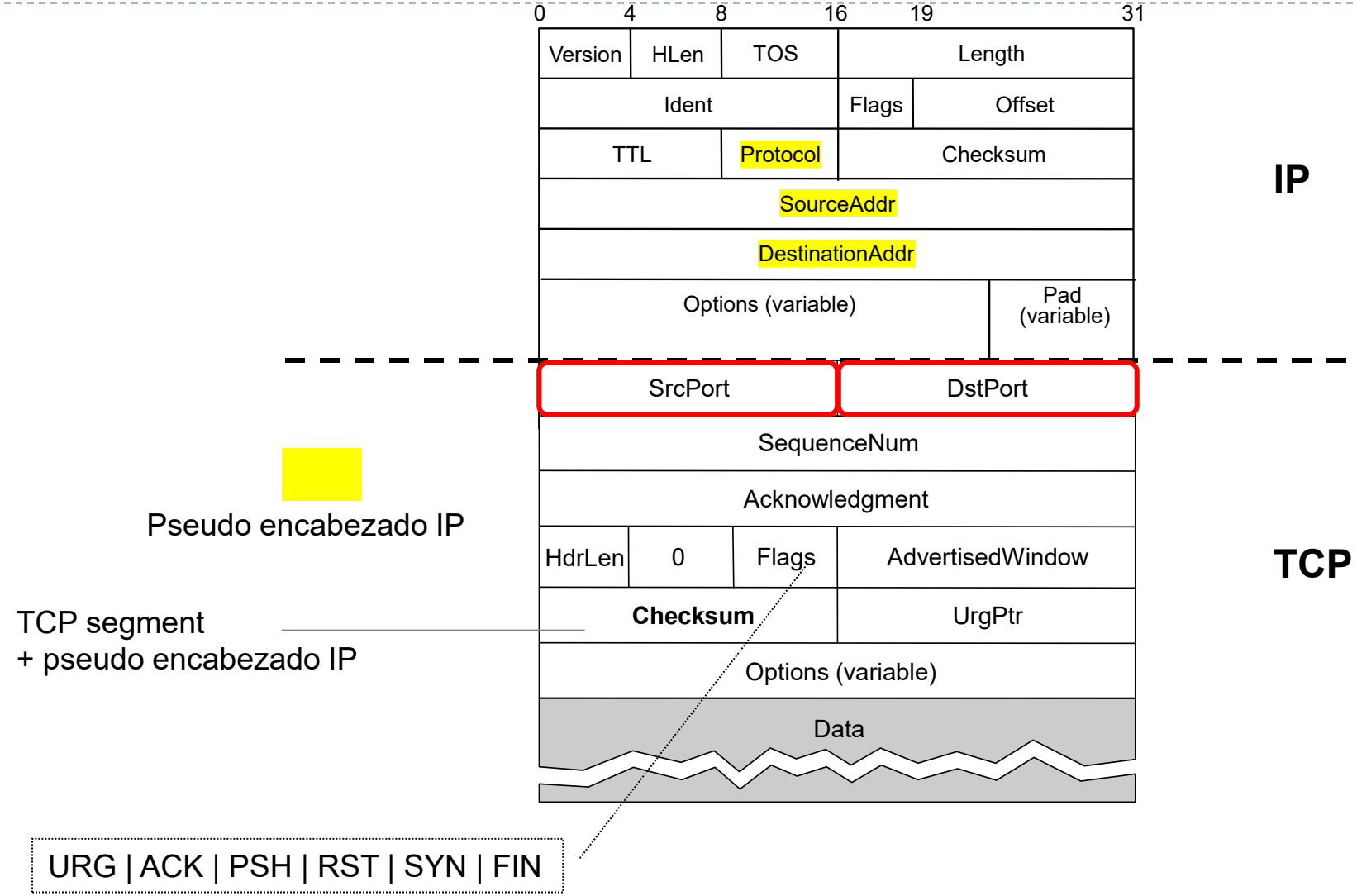
https://tools.ietf.org/html/rfc879.html

TCP: Características

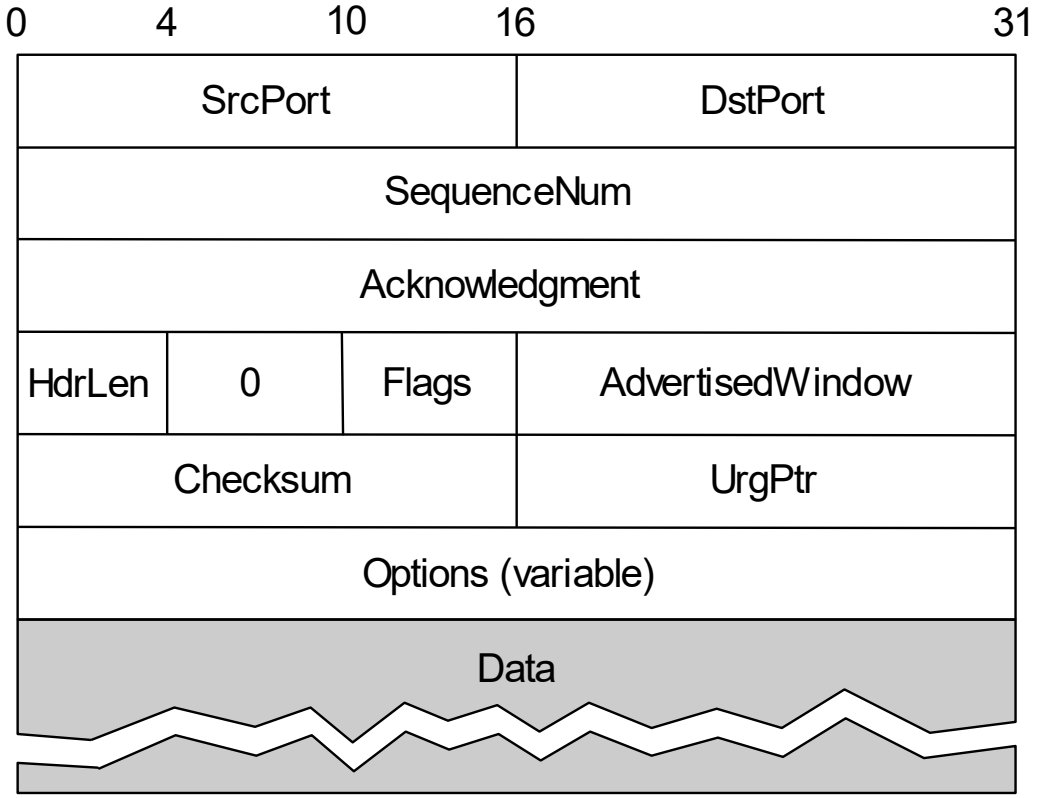
- TCP es **orientado a conexión**
  - Manejo de la conexión:
    - 3-way handshake usado para **establecimiento (setup)**
    - 2-2 o 4-way handshake para la **liberación (release)**
  - TCP provee un servicio de flujo de bytes ***(stream-of-bytes)***

- TCP es **confiable** (establece una "conexión lógica entre sockets")
  - Acknowledgements (ACKs)
  - Checksums
  - Números de secuencia (para detectar datos perdidos o desordenados)
  - Datos perdidos o corruptos se ReTX después de un TimeOut
  - Datos desordenados se podrían reordenar
  - Implementa **Control de Flujo** evita inundar al receptor
  - Implementa **Control de Congestión**
    (lo veremos en detalle una clase especial aparte)

Contexto: Formato de Segmento

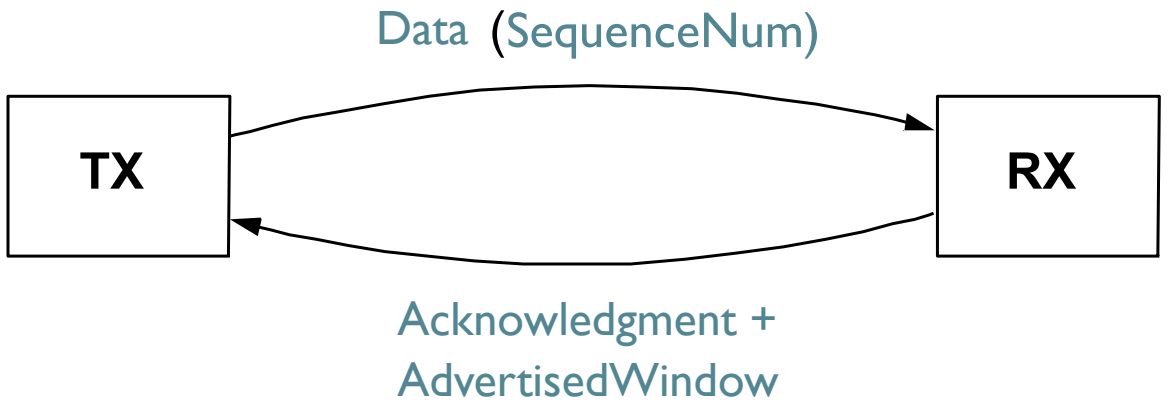


Formato de Segmento TCP



Conexión TCP

- Cada conexión es identificada por la 4-tupla:
  - `SrcPort, SrcIPAddr, DstPort, DstIPAddr`
- Ventana Deslizante + Control de Flujo
  - `Acknowledgement, SequenceNum, AdvertisedWindow`



- Flags
  - `SYN, FIN, RESET, PUSH, URG, ACK`
- Checksum
  - pseudo header (IP) + TCP header + TCP data

La cabecera de TCP

- **Puerto fuente y Puerto destino (16 bits):** Identifican los puntos finales locales de la conexión.
- **Número de secuencia (32 bits):** identifica de forma unívoca los datos de aplicación que contiene el segmento TCP. Identifica el primer *byte de datos*.
- **Número de reconocimiento (32 bits):** Indica el siguiente *byte que espera el receptor.*
  - Implica una confirmación (ACK) de ***todos los bytes recibidos hasta ese momento*.**
- **Longitud de cabecera (4 bits):** Indica cuántas palabras de 32 bits están contenidas en la cabecera de TCP.
  - Es necesario a causa de la longitud variable del campo Opciones.

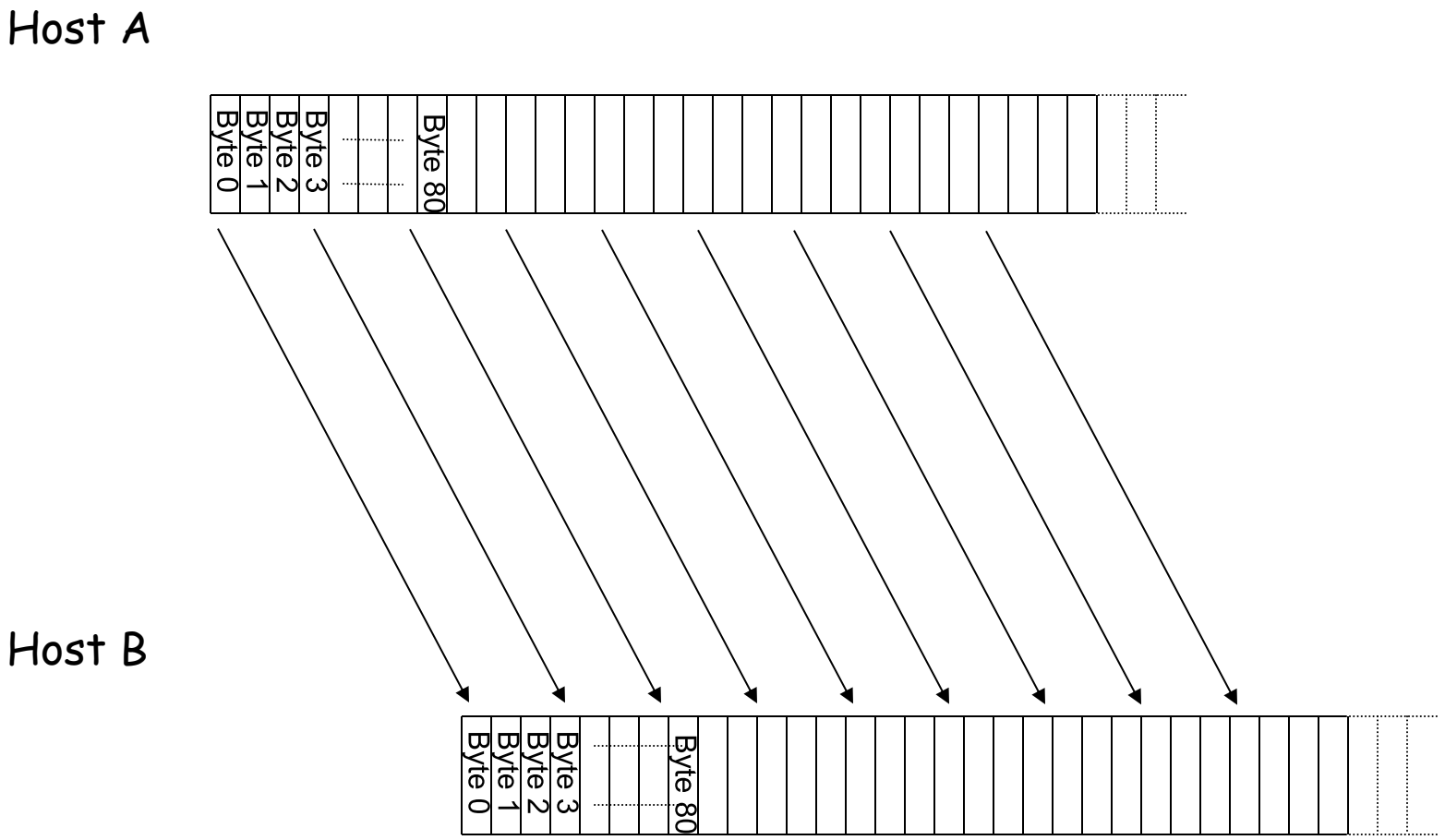
La cabecera de TCP, Flags:

- **URG:** Es igual a 1 si el campo *Puntero a Urgente* está en uso.
- **ACK:** Es igual a 1 para indicar que el número de reconocimiento es válido.
  - Si vale 0, el paquete no contiene un reconocimiento, y entonces el campo *número de reconocimiento* es ignorado.
- **PSH:** "PUSHed data". Indica al receptor que debe entregar los datos a la aplicación inmediatamente después del arribo del paquete.
  - No debe esperar hasta que un buffer total haya sido recibido.

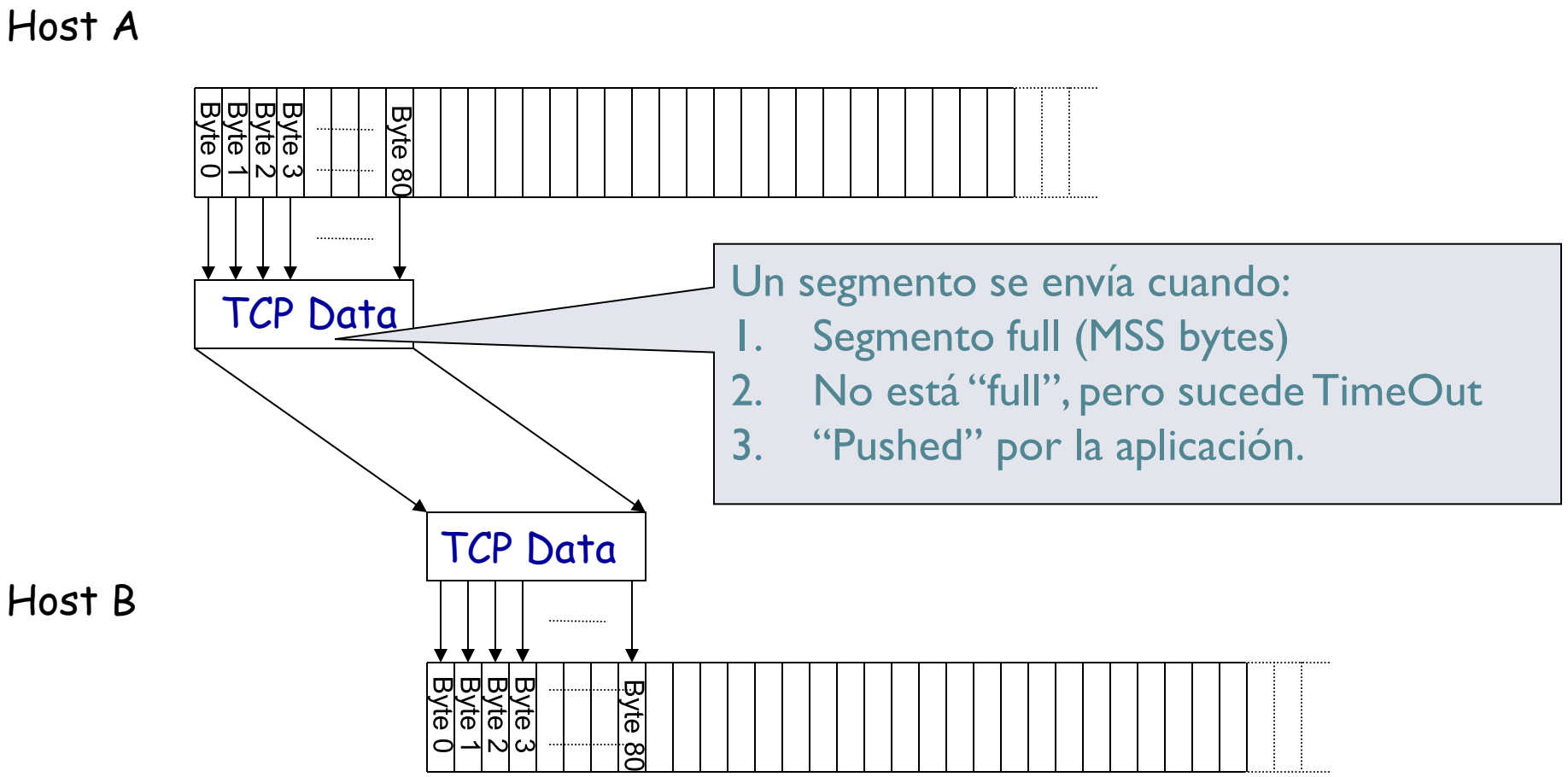
La cabecera de TCP

- **RST:** Se utiliza para resetear una conexión que se ha vuelto confusa debido a la caída de un host o alguna otra razón.
- **SYN:** Es utilizado para **establecer** conexiones.
  - En esencia es usado para indicar **Requerimiento de Conexión** y **Conexión Aceptada**.
- **FIN:** Es utilizado para **liberar** conexiones.
  - Especifica que el emisor no tiene más datos para transmitir.
- **Tamaño de la ventana:** Indica cuántos bytes pueden ser enviados comenzando desde el último byte reconocido.
  - Para propósitos de control de **flujo**.

TCP soporta un *"stream de bytes"*

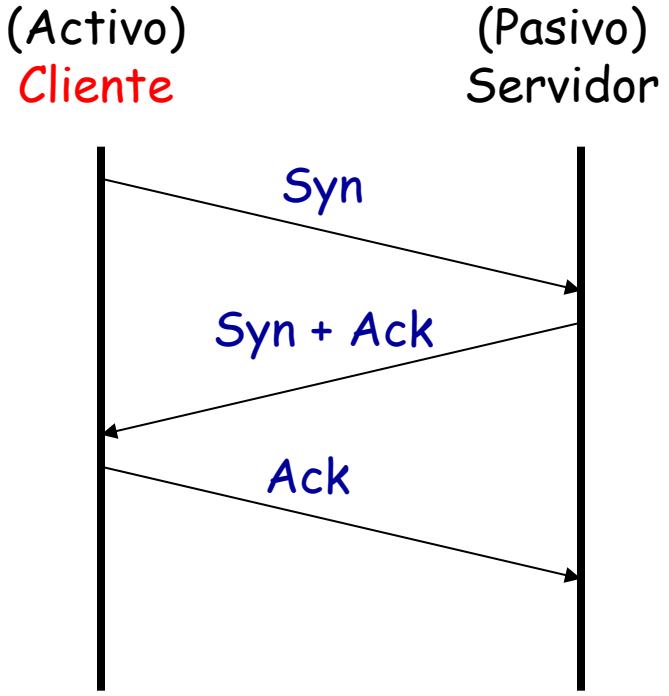


…se emula usando segmentos



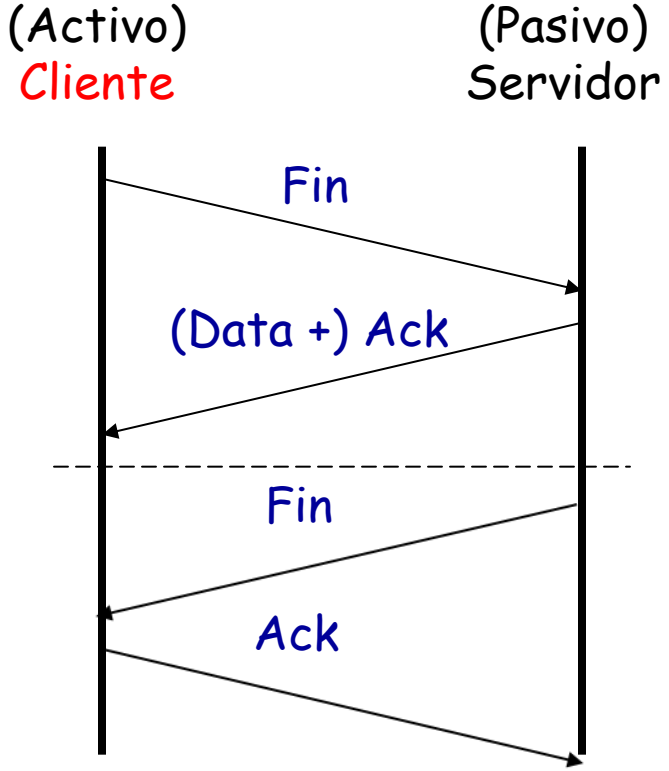
Establecimiento y Liberación de la Conexión TCP

Establecimiento



3-way handshake

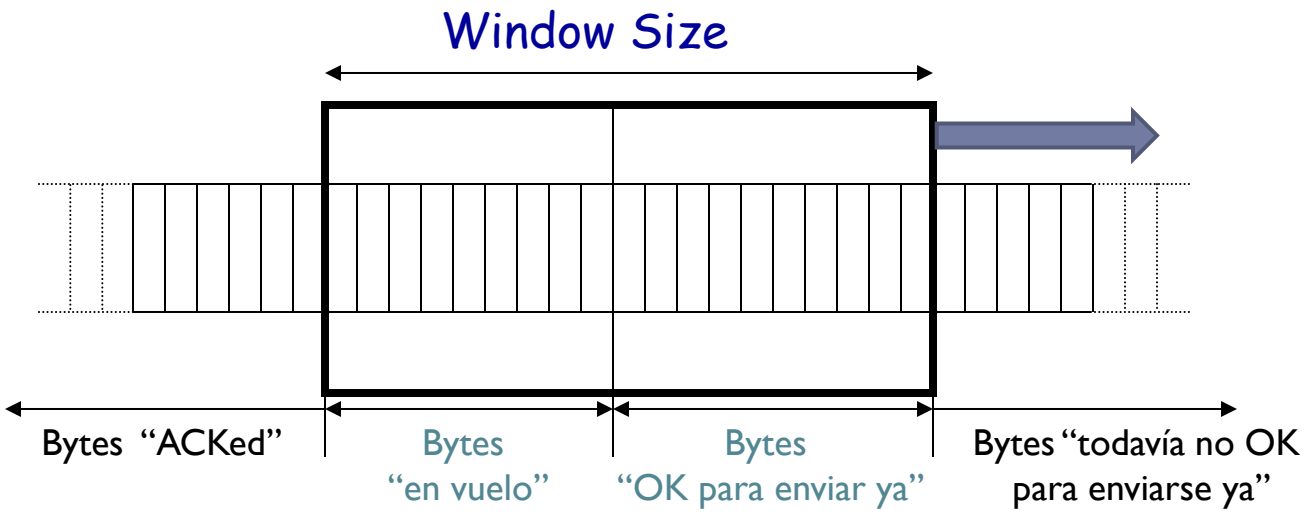
Liberación



4-way handshake

Caso ideal
En el RFC 793 se plantean varios escenarios

TCP: ventana deslizante

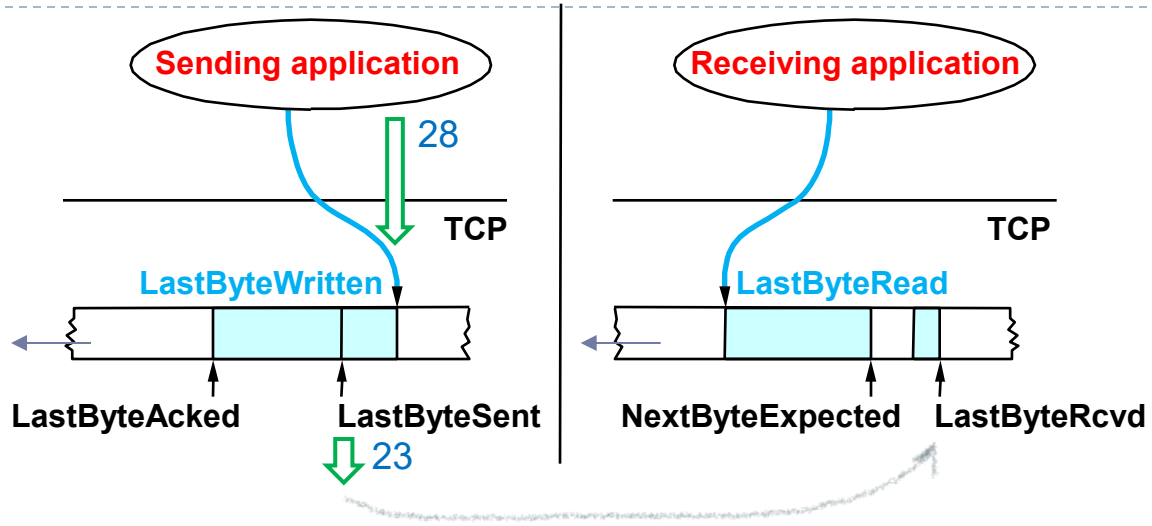


- Política de retransmisión "Go Back N"
- Window Size es "advertised" por el receptor (server)
  - (usualmente de 4KB a 8KB durante la fase de *connection set-up*)

TCP: ventana deslizante

- Se implementa una variante del **protocolo de ventana deslizante** usado en protocolos de Nivel de Enlace
  - Garantiza **confiabilidad**
  - La aplicación recibe los datos en orden
  - Fuerza el control de flujo entre receptor y transmisor

TCP: Ventana Deslizante **Emisor**

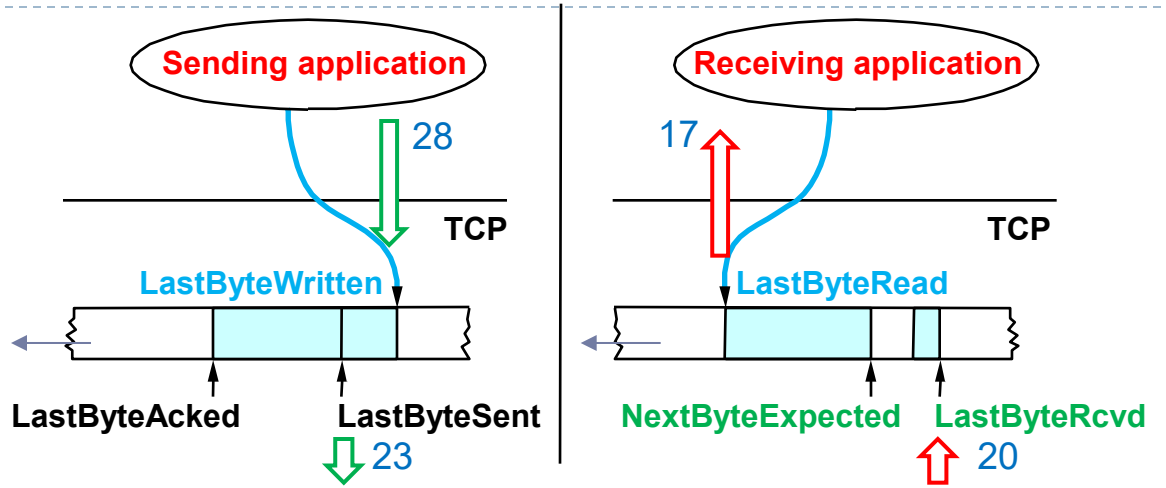


- Lado **Emisor**: tres punteros, las relaciones son obvias:

```
LastByteAcked <= LastByteSent
 LastByteSent <= LastByteWritten
```

- Se bufferean los bytes entre `LastByteAcked` (los que están a su izquierda ya fueron confirmados) y `LastByteWritten` (los que están a su derecha no fueron aun generados)

TCP: Ventana Deslizante **Receptor**



- Las relaciones son menos intuitivas debido al reordenamiento

  `LastByteRead < NextByteExpected`

- Un byte no puede ser **leído por la aplicación** a menos que éste se haya recibido *y también que todos sus precedentes se hayan recibido*

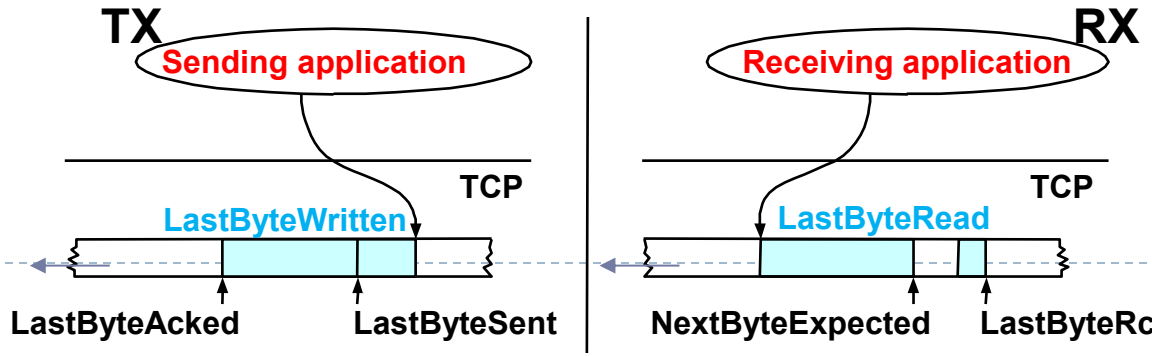
  `NextByteExpected <= LastByteRcvd+1`

- Si están en orden se cumple la igualdad. En cambio, si están en desorden `NextByteExpected` apunta al gap de la figura.
- Se encolan los bytes entre `NextByteExpected` y `LastByteRcvd`

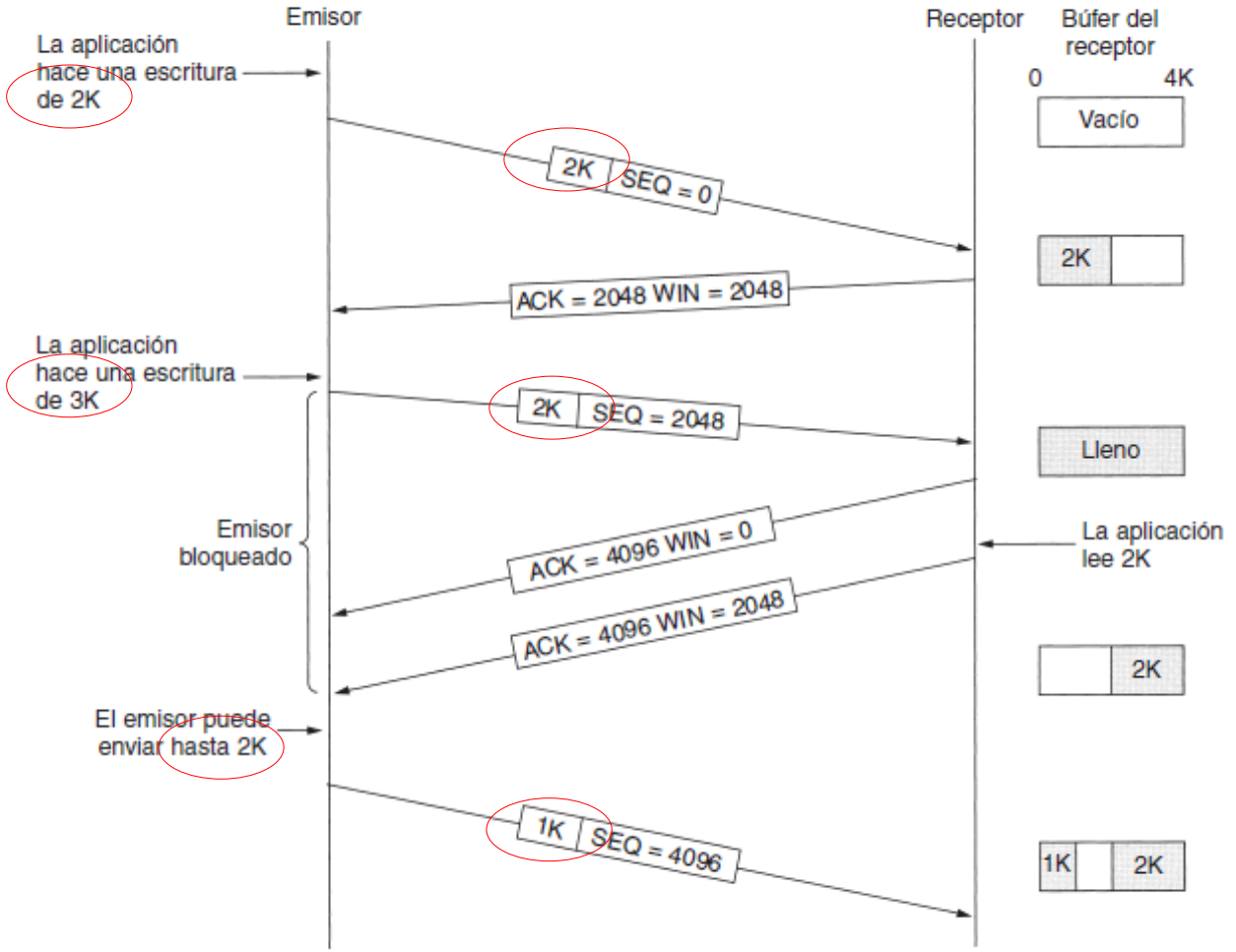
TCP: Control de Flujo

- Tamaño del buffer de **Envío**: `MaxSendBuffer`
- Tamaño del buffer de **Recepción**: `MaxRcvBuffer`
- **Lado RX**
  - `(LastByteRcvd - LastByteRead) <= MaxRcvBuffer`
  - `AdvertisedWindow = MaxRcvBuffer - (LastByteRcvd - LastByteRead)`
- **Lado TX**
  - `(LastByteSent - LastByteAcked) <= AdvertisedWindow`
  - `EffectiveWindow = AdvertisedWindow -(LastByteSent- LastByteAcked)`
  - `LastByteWritten - LastByteAcked <= MaxSendBuffer`

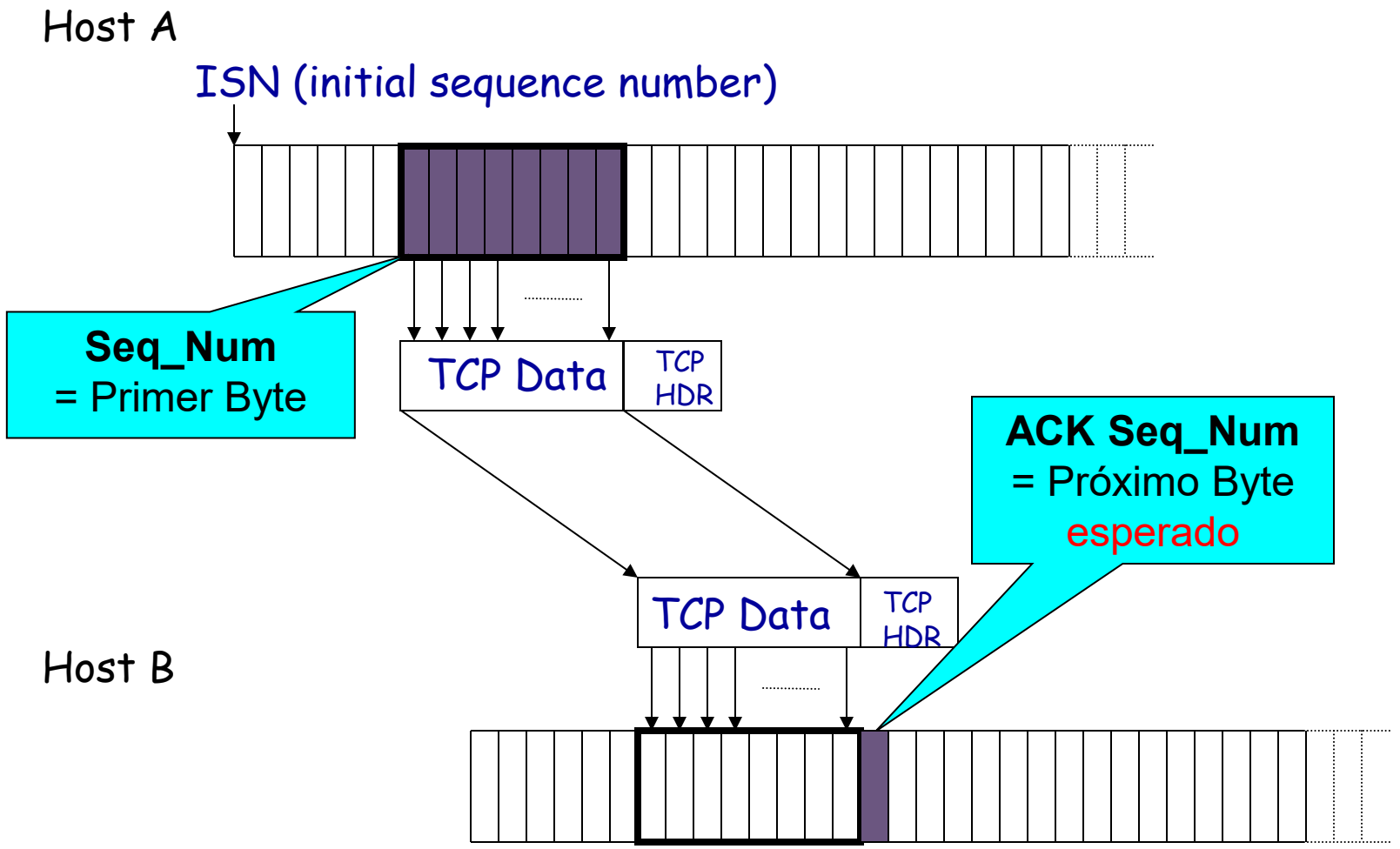
- **Bloquear TX** si (`LastByteWritten - LastByteAcked`) `+ y > MaxSendBuffer`, con *`y` = bytes que se desean escribir.*
- **Enviar desde RX** segmentos ACK en respuesta a la llegada de segmentos de datos
- **Persistir en TX** enviando 1 byte cuando `AdvertisedWindow = 0`



Advertised Window Management



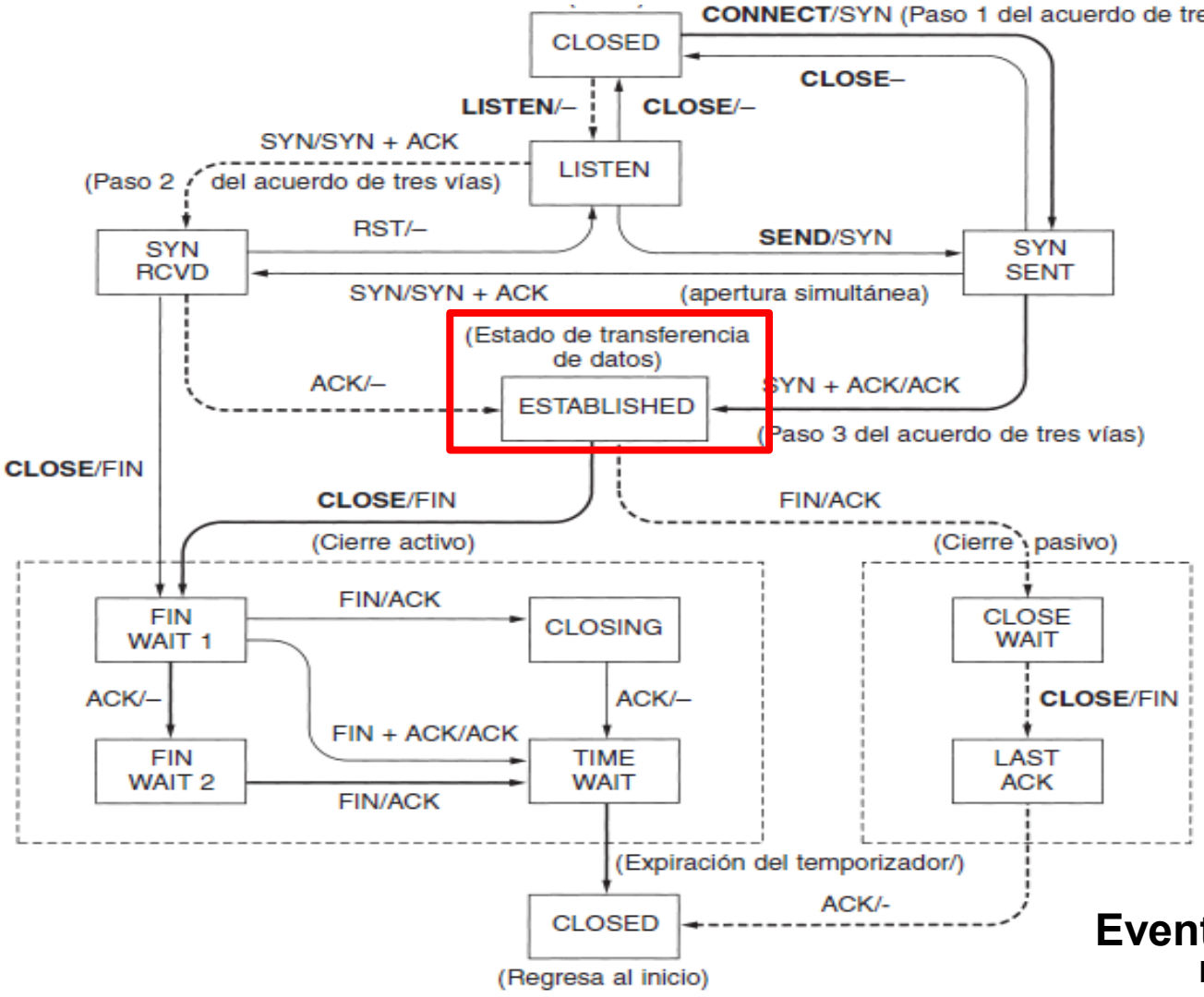
Números de Secuencia



TCP

Máquina de Estados Finitos

Administración de conexiones TCP



Cambios de estado:

Cliente

Servidor

Evento / Reacción

Evento: desde la Aplicación
LISTEN, CONNECT, CLOSE, SEND

Evento: desde un Segmento
SYN, ACK, FIN, RST

Estados del Modelo TCP

Estado	Descripción
CLOSED	No hay conexión activa ni pendiente
LISTEN	El servidor espera una llamada
SYN RCVD	Llegó solicitud de conexión; espera ACK
SYN SENT	La aplicación comenzó a abrir una conexión
ESTABLISHED	Estado normal de transferencia de datos
FIN WAIT 1	La aplicación dijo que ya terminó
FIN WAIT 2	El otro lado acordó liberar
TIMED WAIT	Espera que todos los paquetes mueran
CLOSING	Ambos lados intentaron cerrar simultáneamente
CLOSE WAIT	El otro lado inició una liberación
LAST ACK	Espera que todos los paquetes mueran

Administración de conexiones

- El caso común de un cliente que se conecta activamente a un servidor pasivo se indica con líneas gruesas (continuas para el cliente, punteadas para el servidor).

- Las líneas delgadas son secuencia de eventos poco comunes.
- Cada línea de la figura se marca mediante un par ***evento****/acción.*

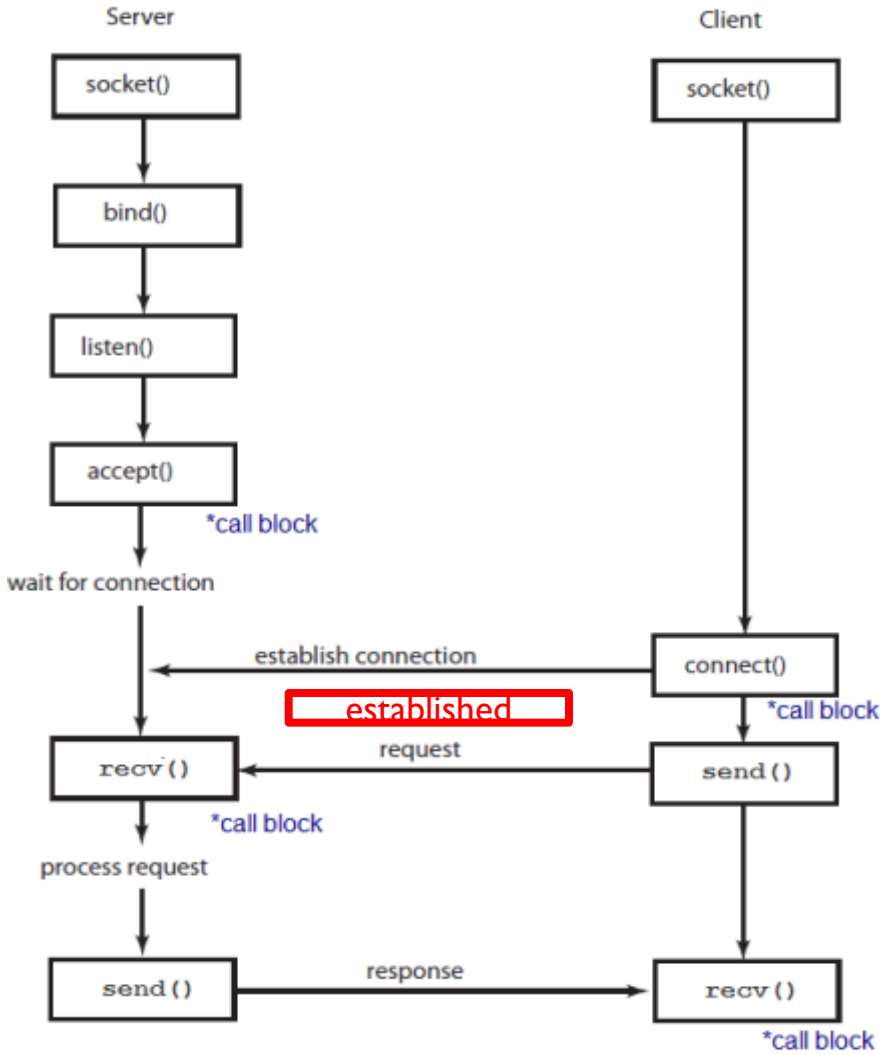
- El **evento** puede ser una **llamada de sistema** iniciada por el usuario (CONNECT, LISTEN, SEND o CLOSE), o la **llegada de un segmento** (SYN, FIN, ACK o RST) o, en un caso, una expiración de temporizador del doble del tiempo de vida máximo del paquete.

- La *acción* es el envío de un segmento de control (SYN, FIN o RST), o nada, indicado por “-”.

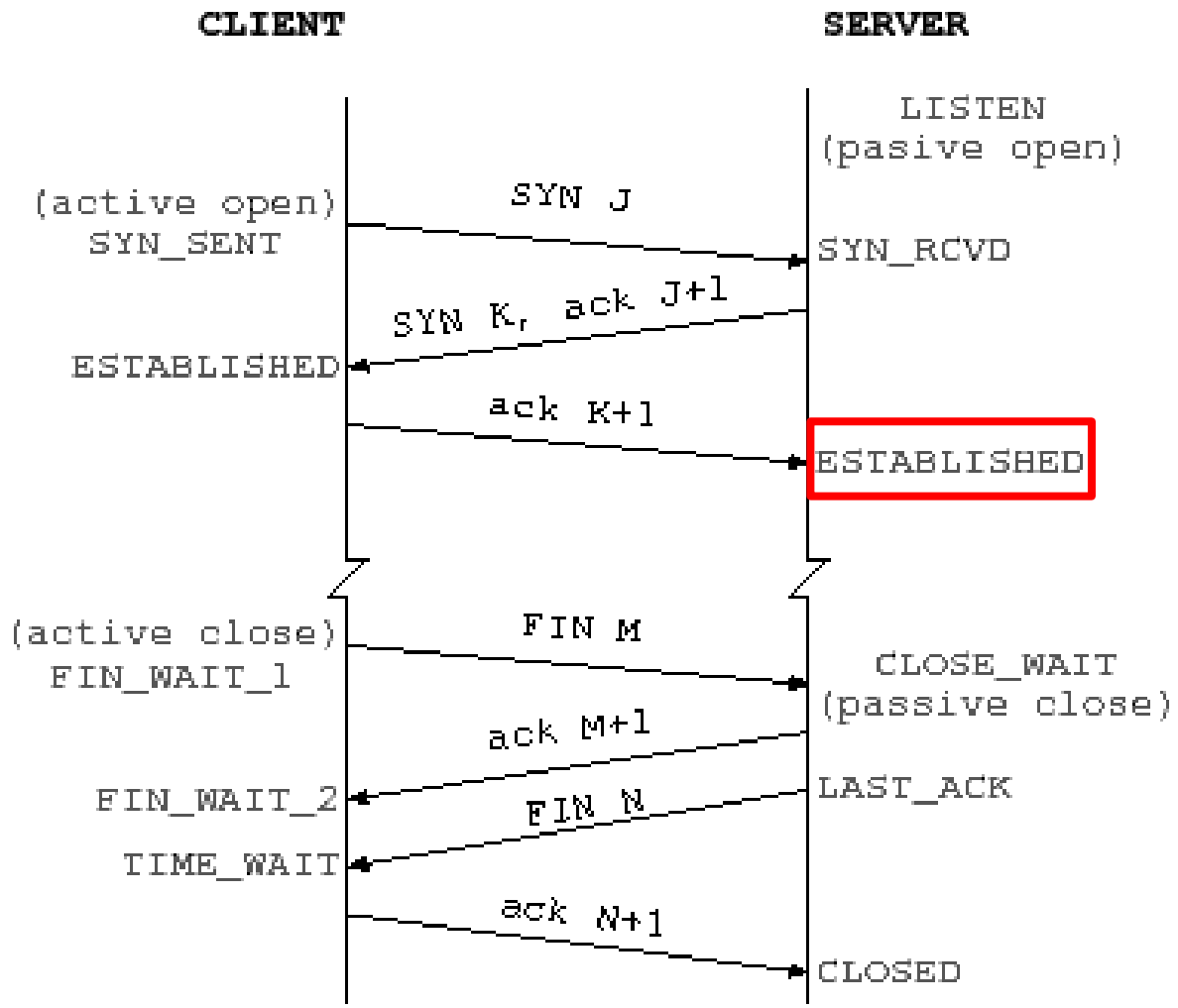
- Los comentarios aparecen entre paréntesis.

Primitivas – sockets berkeley TCP

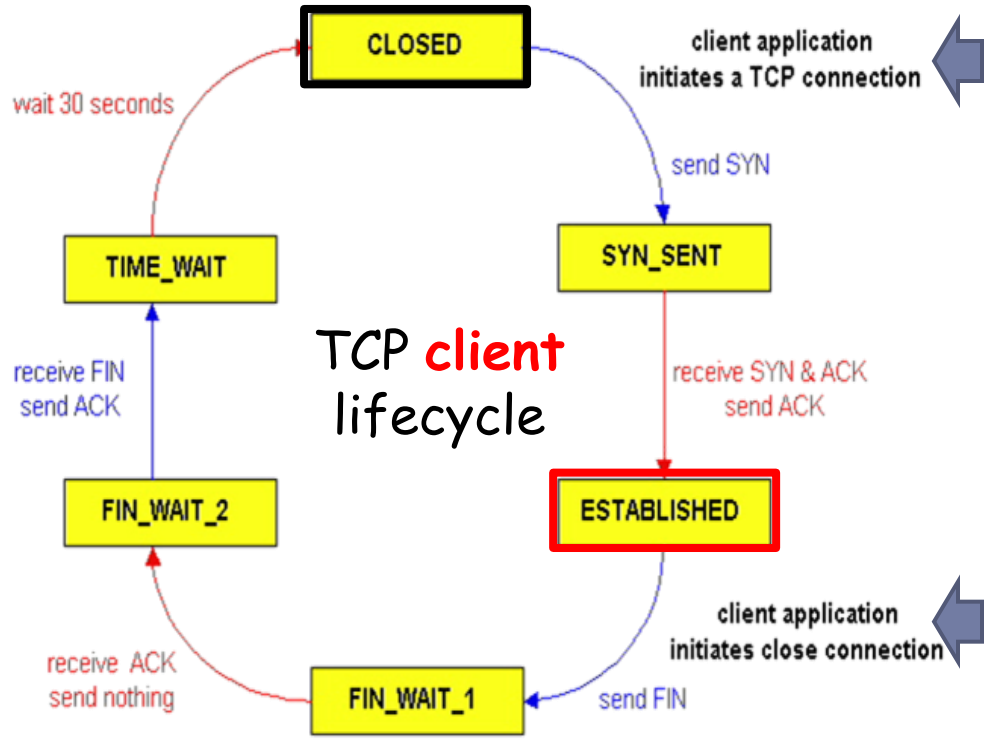
Primitiva	Significado
SOCKET	Crea un nuevo punto terminal de comunicación
BIND	Adjunta una dirección local a un *socket*
LISTEN	Anuncia la disposición a aceptar conexiones; indica el tamaño de cola
ACCEPT	Bloquea al invocador hasta la llegada de un intento de conexión
CONNECT	Intenta establecer activamente una conexión
SEND	Envía datos a través de la conexión
RECEIVE	Recibe datos de la conexión
CLOSE	Libera la conexión

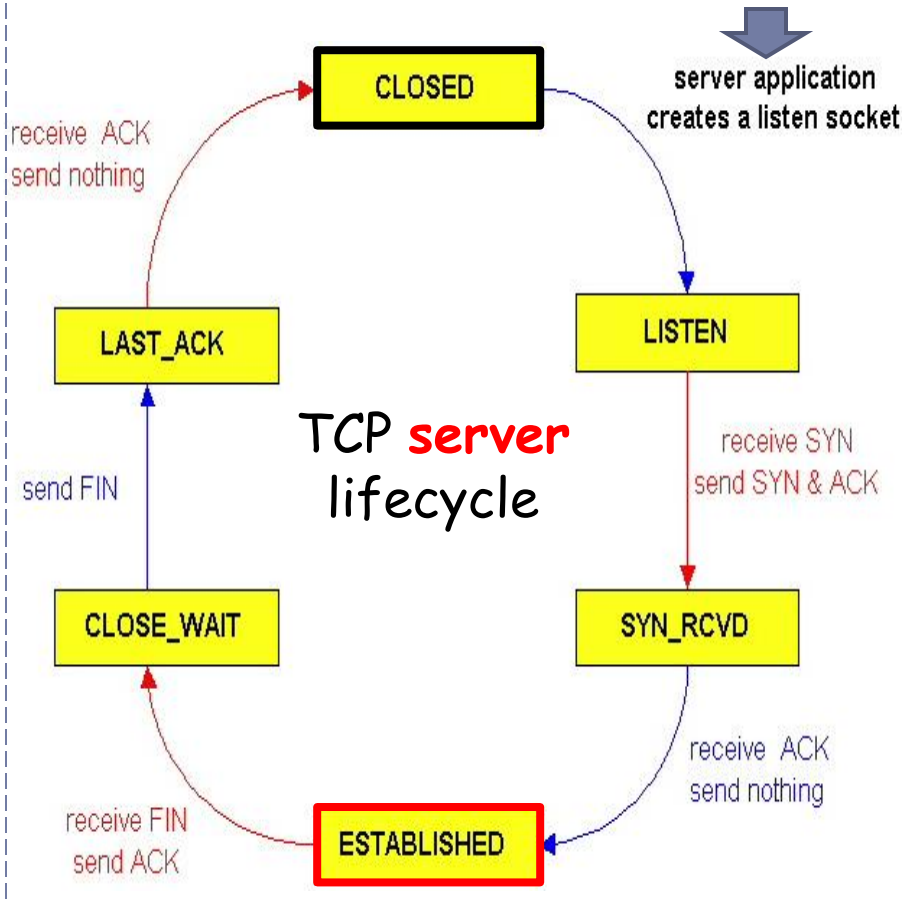


Establecimiento y liberación de una conexión

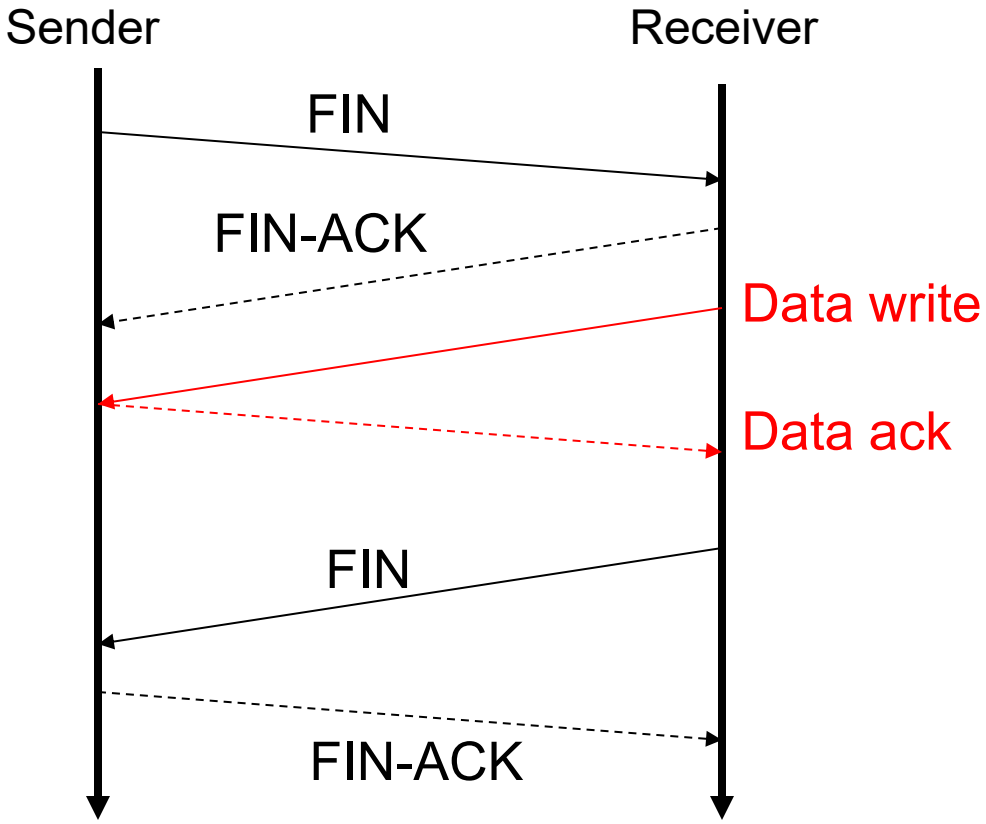


Ciclos en la Máquina de Estados TCP





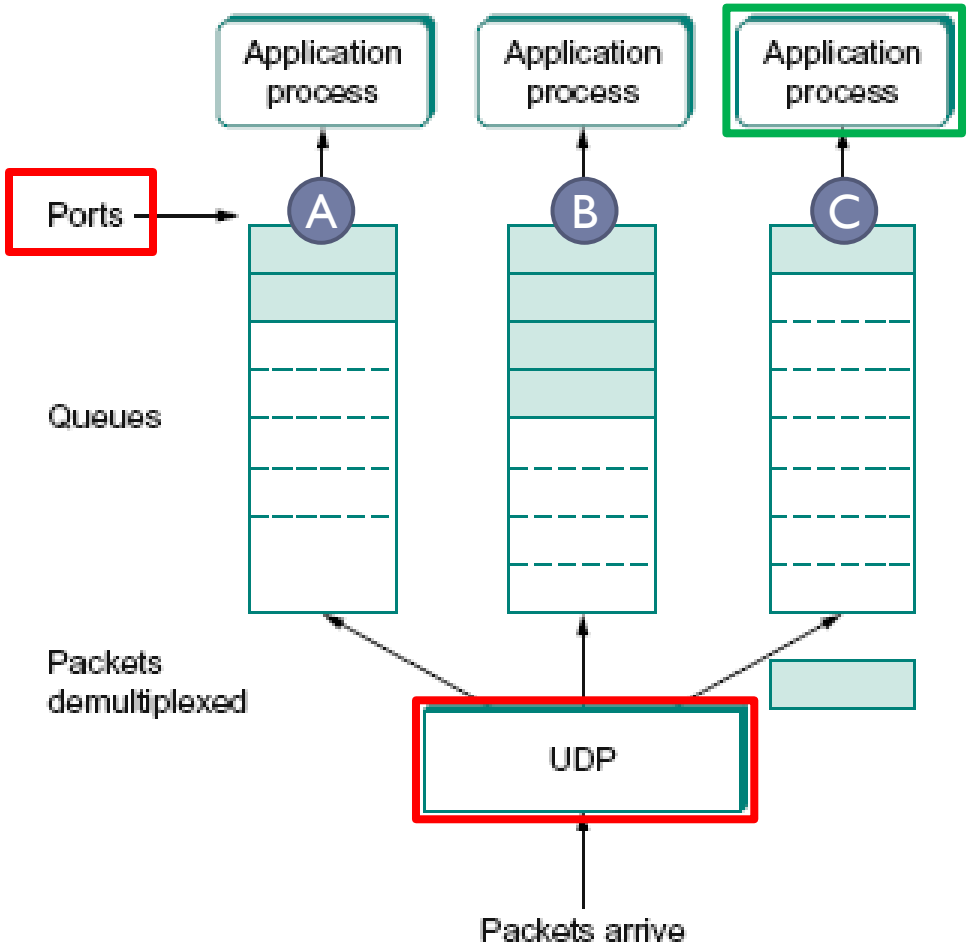
Liberación “Tear-down”



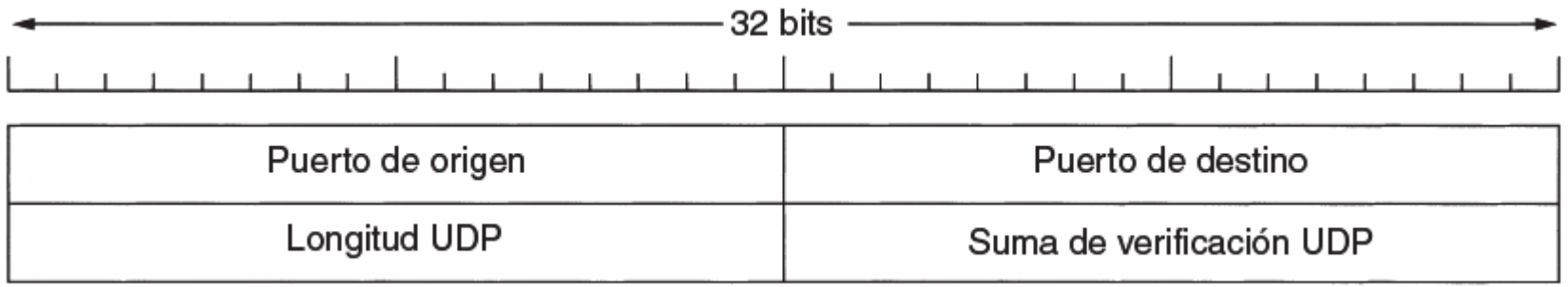
UDP

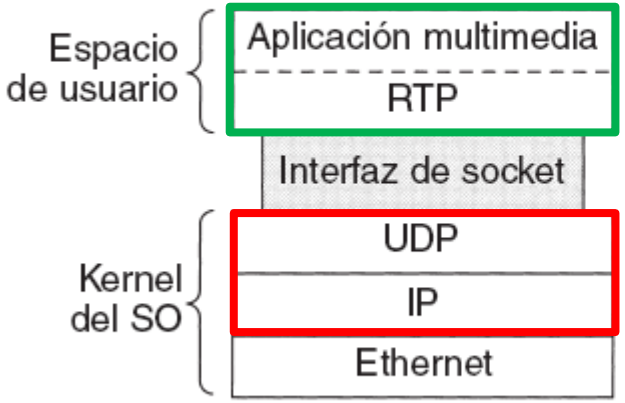
Servicio sin conexión

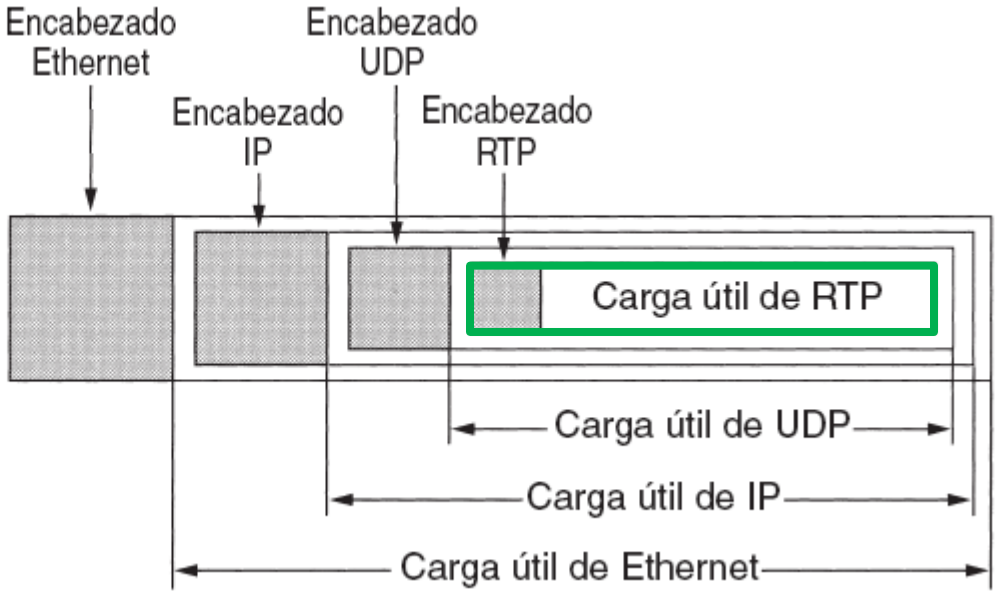
Multiplexación mediante Puertos



Ejemplo: Transporte UDP y Aplicación RTP







TCP

Retransmisión Adaptativa

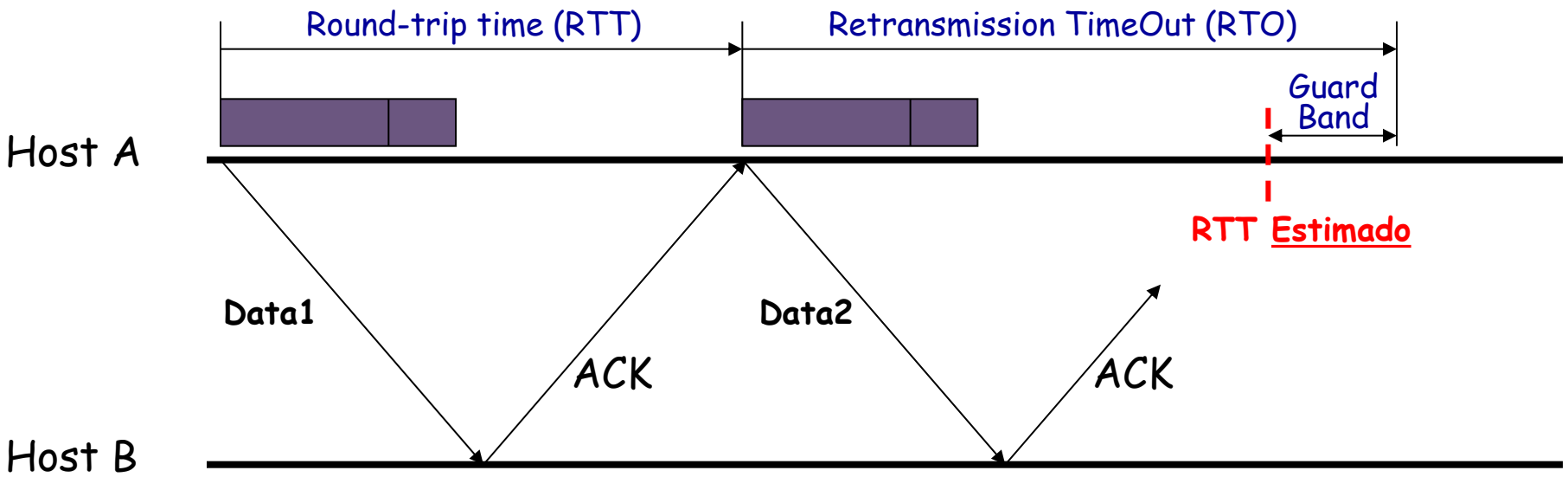
Jacobson (1988)

A A fast algorithm for rtt mean and variation

A.1 Theory

The RFC793 algorithm for estimating the mean round trip time is one of the simplest examples of a class of estimators called *recursive prediction error* or *stochastic gradient* algorithms. In the past 20 years these algorithms have revolutionized estimation and control theory[14] and it's probably worth looking at the RFC793 estimator in some detail.

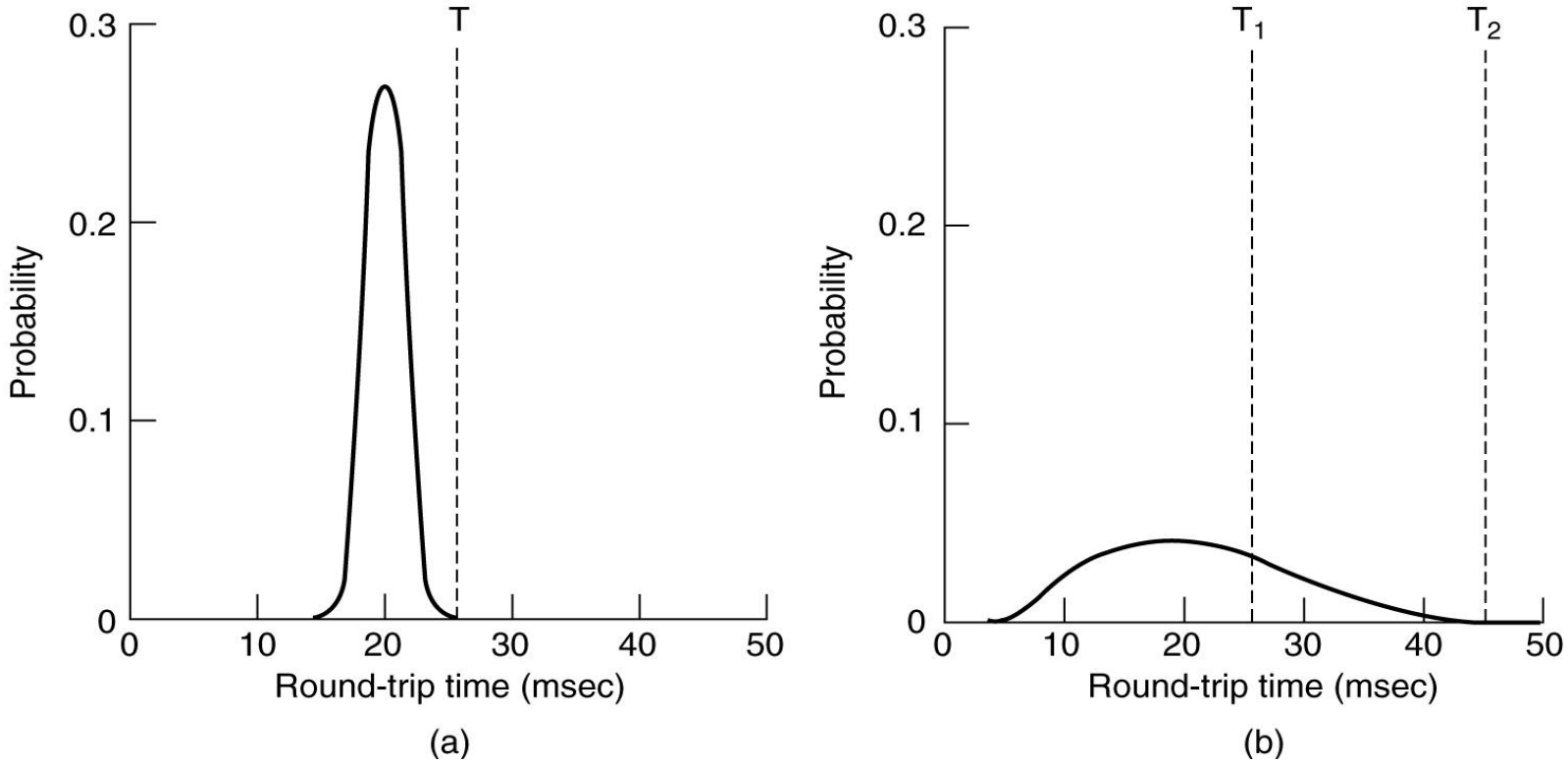
Retransmisión y Timeouts



Valor adaptativo de timeout de ReTX (lazo cerrado → teoría de control):

Congestión
Cambios de ruta
} RTT es muy variable

Concepto: TCP Timer Management



(a) Función densidad de probabilidad para tiempos de llegadas de ACK **en Nivel 2**

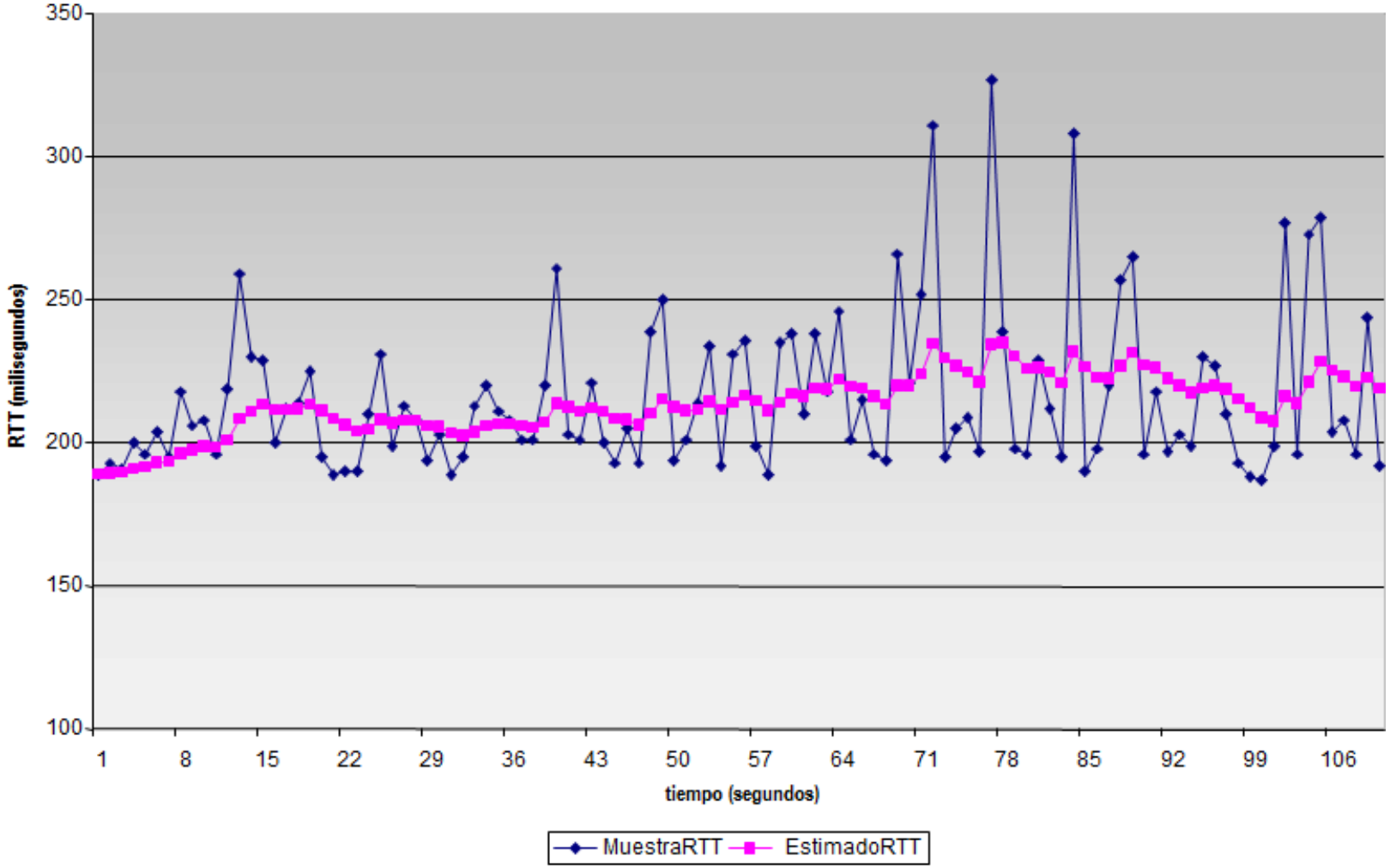
(b) Función densidad de probabilidad para tiempos de llegadas de ACK **en TCP**

Figuras : Tanembaum 4ta edición

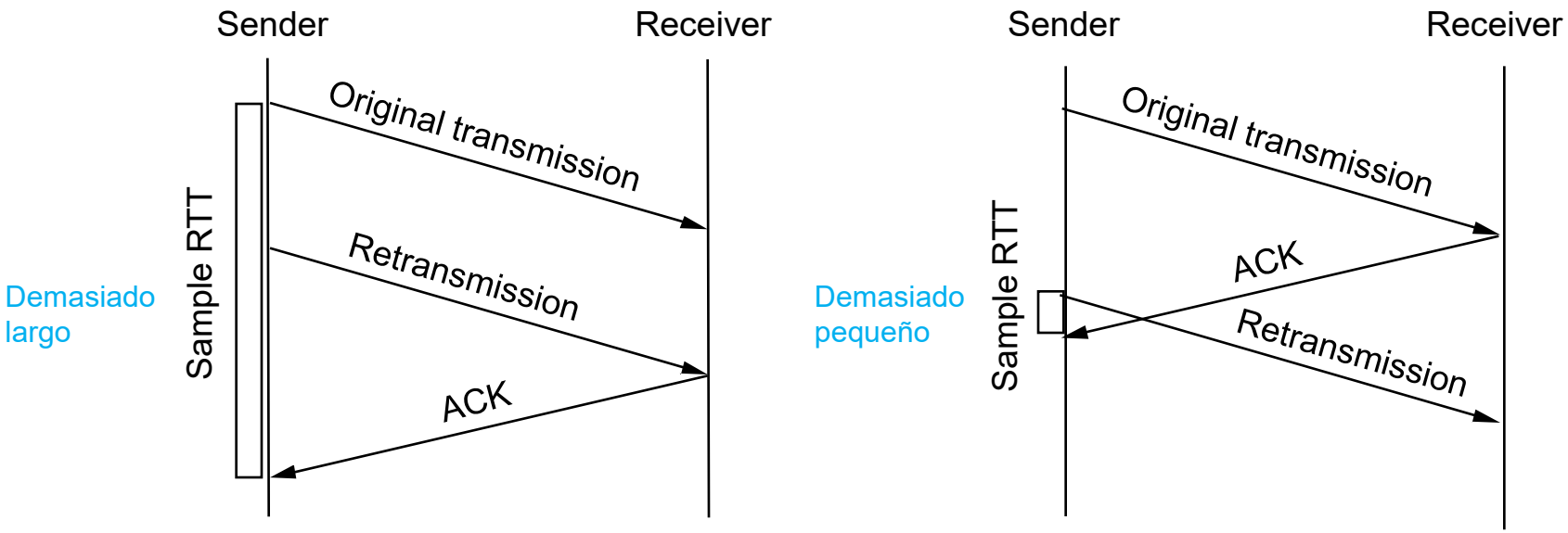
Retransmisión Adaptativa (Algoritmo Original)

- Mide **`SampleRTTi`** para cada par segmento / ACK
- Calcula el promedio ponderado de RTT
  - **`EstimatedRTT`** = α * **`EstimatedRTT`** + β * **`SampleRTTi`**
  - donde $\alpha + \beta = 1$
    - $0.8 <= \alpha$ <= 0.9
    - $0.1 <= \beta$ <= 0.2
- Fija TimeOut basado en **`EstimatedRTT`**
  - **`TimeOut`** = 2 x **`EstimatedRTT`**

Ejemplo de estimación de RTT



Algoritmo de Karn/Partridge [KP87]



- Problemas obvios: incertidumbre en los ACKs
- K/P: No considerar RTT cuando se retransmite
  - Además: Duplicar TimeOut luego de cada retransmisión (exponential backoff, un viejo conocido)

[KP87] Karn, P. and C. Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocols", SIGCOMM 87.

Algoritmo de Jacobson/Karels [JK88]

Nueva forma de calcular TimeOut de ReTX considerando la varianza

- **`Diff = SampleRTT - EstRTT`**
- **`EstRTT = EstRTT + (δ * Diff)`**
- **`Dev = Dev + δ *(|Diff| - Dev)`**
  - donde δ es un factor entre 0 y 1 (por ejemplo 1/8)
- Considerar varianza, fijar el timeout
- **`TimeOut`** = μ * **`EstRTT`** + ϕ * **`Dev`**
  - donde $\mu = 1$ y $\phi = 4$
- **`TimeOut`** se acerca a **`EstRTT`** o a **`Dev`** dependiendo del valor dinámico de la varianza
- ***Observaciones***:
  - *Los algoritmos son tan buenos/malos como la granularidad de su reloj (500ms en Unix-like)*
  - *Un mecanismo preciso de timeout es importante para controlar la* ***congestión*** *(como veremos en una próxima clase)*
  - *Además de ayudar a controlar la congestión, la idea es no retransmitir cuando no es absolutamente necesario.*

[JK88] Jacobson, V. and M. Karels, "Congestion Avoidance and Control" 1988

RFC 6298 "Computing TCP's Retransmission Timer"

- El algoritmo básico del emisor TCP mantiene dos variables de estado para computar el RTO
  - **SRTT** (smoothed round-trip time) and
  - **RTTVAR** (round-trip time variation)

  Hasta que el emisor realice la medición para el primer segmento:
  - "SHOULD set" **RTO <- 1** segundo (versiones anteriores 3 segundos o más)

Cuando se realiza la primera medida R; RTT el host "debe" setear
- **SRTT <- R**
- **RTTVAR <- R/2**
- **RTO <- SRTT + max (G, K*RTTVAR)** , siendo K=4, G=1 seg (granularidad)

https://tools.ietf.org/html/rfc6298

RFC 6298 "Computing TCP's Retransmission Timer"

- En las subsiguientes mediciones R' del RTT el host debe setear:
  - **RTTVAR <- (1 - beta) * RTTVAR + beta * |SRTT - R'|** (1)
  - **SRTT <- (1 - alpha) * SRTT + alpha * R'** (2)

  Con alpha=1/8 y beta=1/4 (sugeridos en [JK88]).

Nota : el valor de SRTT usado para actualizar RTTVAR en (1) es el valor antes de su propia actulizacion en (2). RTTVAR y SRTT debe calcularse en el orden precedente.

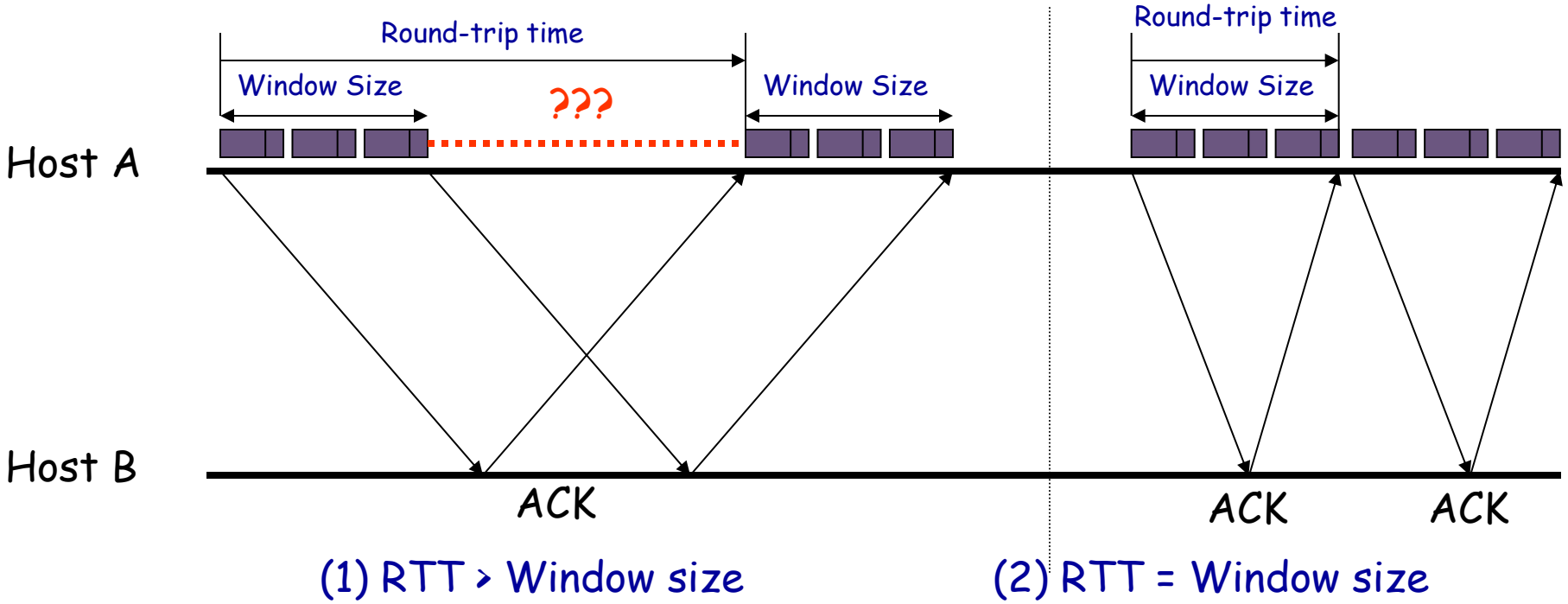
- Luego se debe setear el RTO:
  - **RTO <- SRTT + max (G, K*RTTVAR)**
- Si el RTO calculado es menor a 1 segundo entonces debería redondear a 1 siguen las consideraciones para evaluar cual es un valor conservativo y evitar retransmisiones espúreas.

TCP

Uso del ancho de banda

Sockets

TCP Sliding Window



Mantener el “caño” lleno

- **`AdvertisedWindow de 16 bits`** (64 KB)

Bandwidth	Delay x Bandwidth Product
T1 (1.5 Mbps)	18KB
Ethernet (10 Mbps)	122KB
T3 (45 Mbps)	549KB
FDDI (100 Mbps) 1.2MB	
STS-3 (155 Mbps)	1.8MB
STS-12 (622 Mbps)	7.4MB
STS-24 (1.2 Gbps)	14.8MB

64 KB

Asumiendo RTT de 100 ms