Teoría de las Comunicaciones

2do. Cuatrimestre de 2017

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

Nivel de Transporte

TCP (Transmission Control Protocol)

http://www.dc.uba.ar/events/vcerf

Bibliografía Básica

- Computer Networks, Fifth Edition: A Systems Approach (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

Dedicaremos además una clase a :

- llman, M., Paxson, V., and Blanton, E., "TCP Congestion Control", RFC 5681, September 2009, http://tools.ietf.org/html/rfc5681.txt

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.
- J. Postel, RFC-793 "Transmission Control Protocol" September 1981.

  (varios errores e inconsistencias)
- RFC 1122 October 1989 (se salvan bugs del 793)
- Extensiones de TCP en el RFC 1323
- Paxon , Allman , Chu and Sargent RFC 6298 “Computing TCP’s Retransmission Timer” June 2011

Referencias

- [RFC1323] Jacobson, V., Braden, R., and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.
- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.
- [RFC2988] Paxson, V. and M. Allman, "Computing TCP's Retransmission Timer", RFC 2988, November 2000.

- Y sigue la lista … una recopilación al 2006 “A Roadmap for Transmission Control Protocol (TCP) Specification Documents” RFC 4614

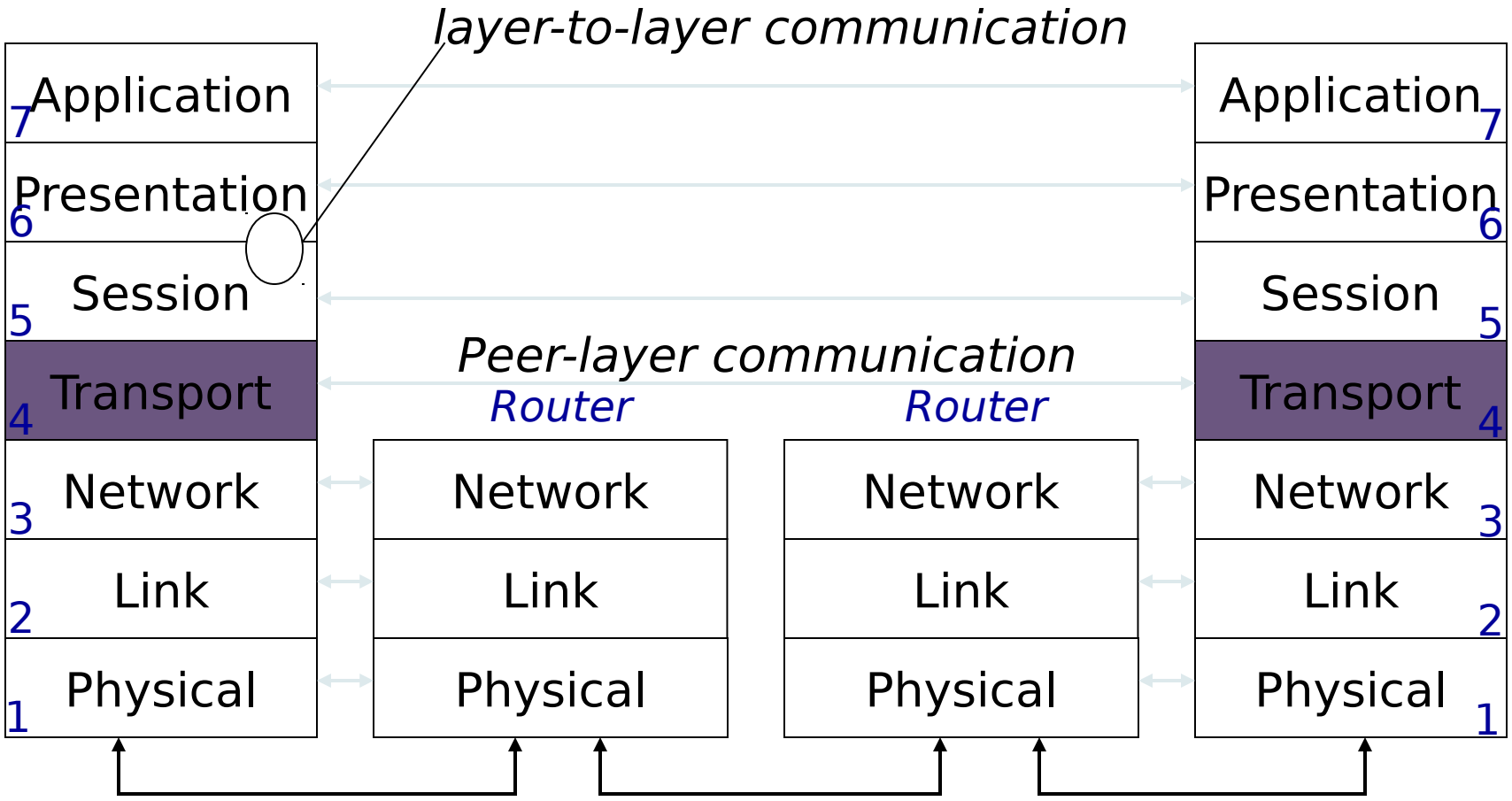
Agenda- 1 Parte

- Revisión Nivel de Transporte
- El Protocolo TCP
  - Características
  - TCP Connection setup
  - Segmentos TCP
  - Números de Secuencia TCP
  - TCP Sliding Window
  - Control de Flujo
  - Timeouts y Retransmisiones (RTX)

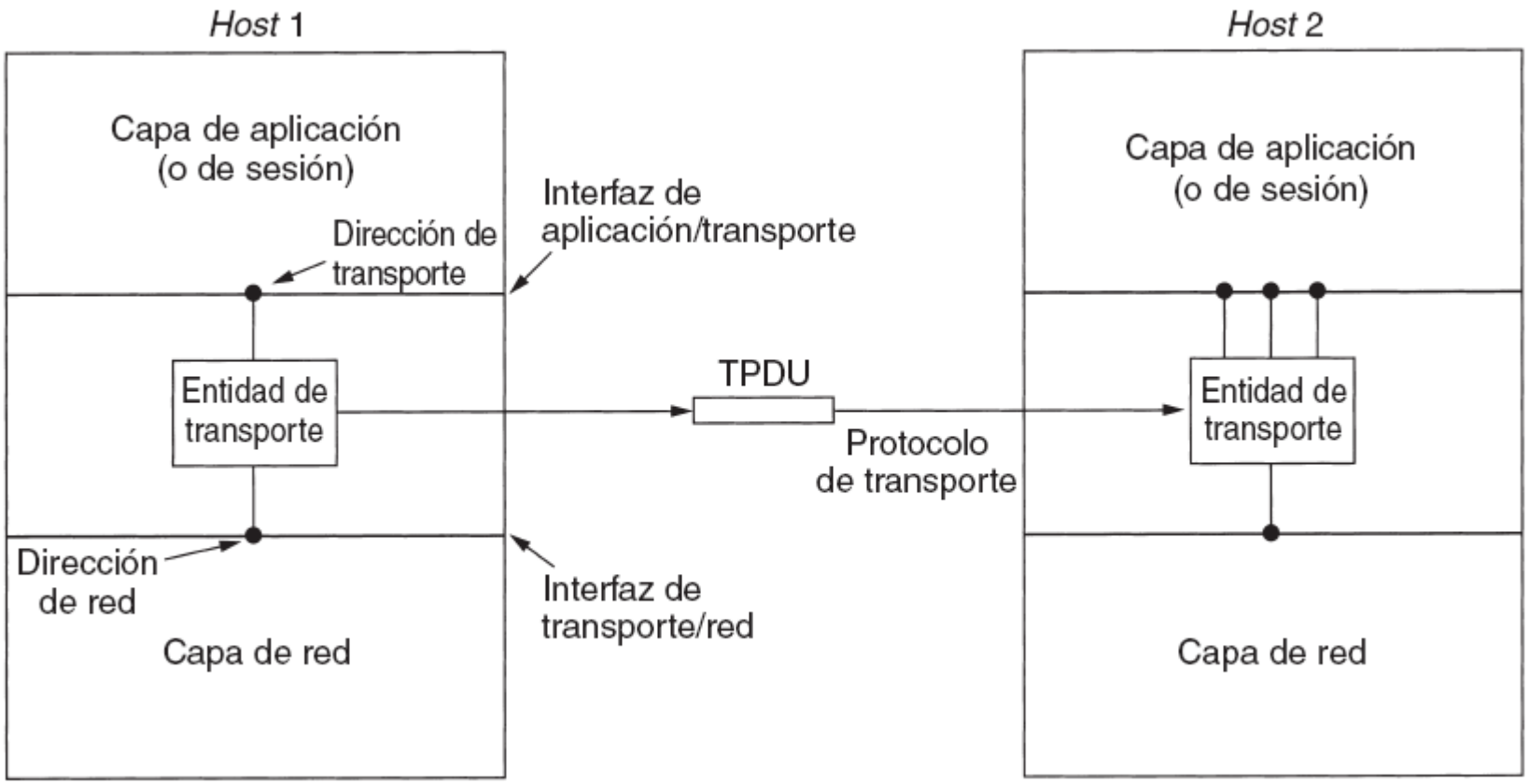
Recordar los modelos de capas

Protocolos E2E

Modelo OSI


layer-to-layer communication
Application
7
Presentation
6
Session
5
Transport
4
Network
3
Link
2
Physical
1
Peer-layer communication
Router
Router
Application
7
Presentation
6
Session
5
Transport
4
Network
3
Link
2
Physical
1

Modelo OSI


Host 1
Capa de aplicación
(o de sesión)
Dirección de
transporte
Interfaz de
aplicación/transporte
Entidad de
transporte
TPDU
Protocolo
de transporte
Dirección
de red
Interfaz de
transporte/red
Capa de red
Host 2
Capa de aplicación
(o de sesión)
Entidad de
transporte
Capa de red

Modelo OSI

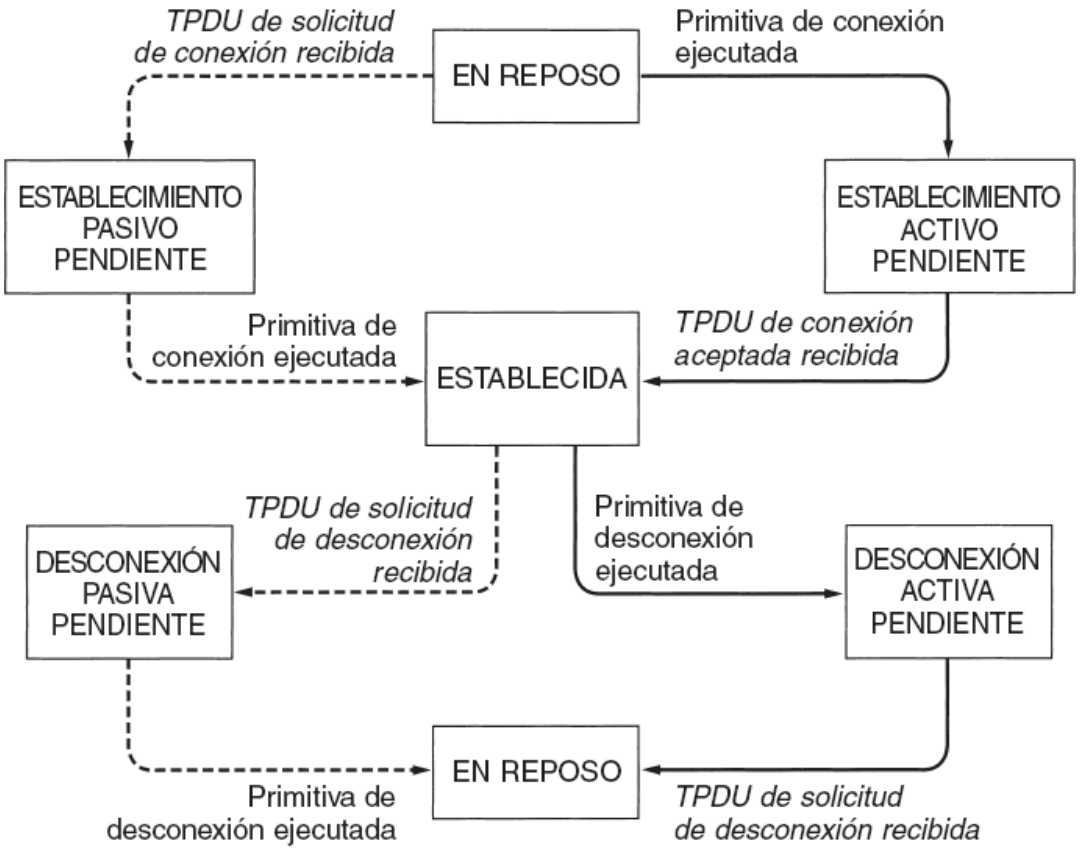
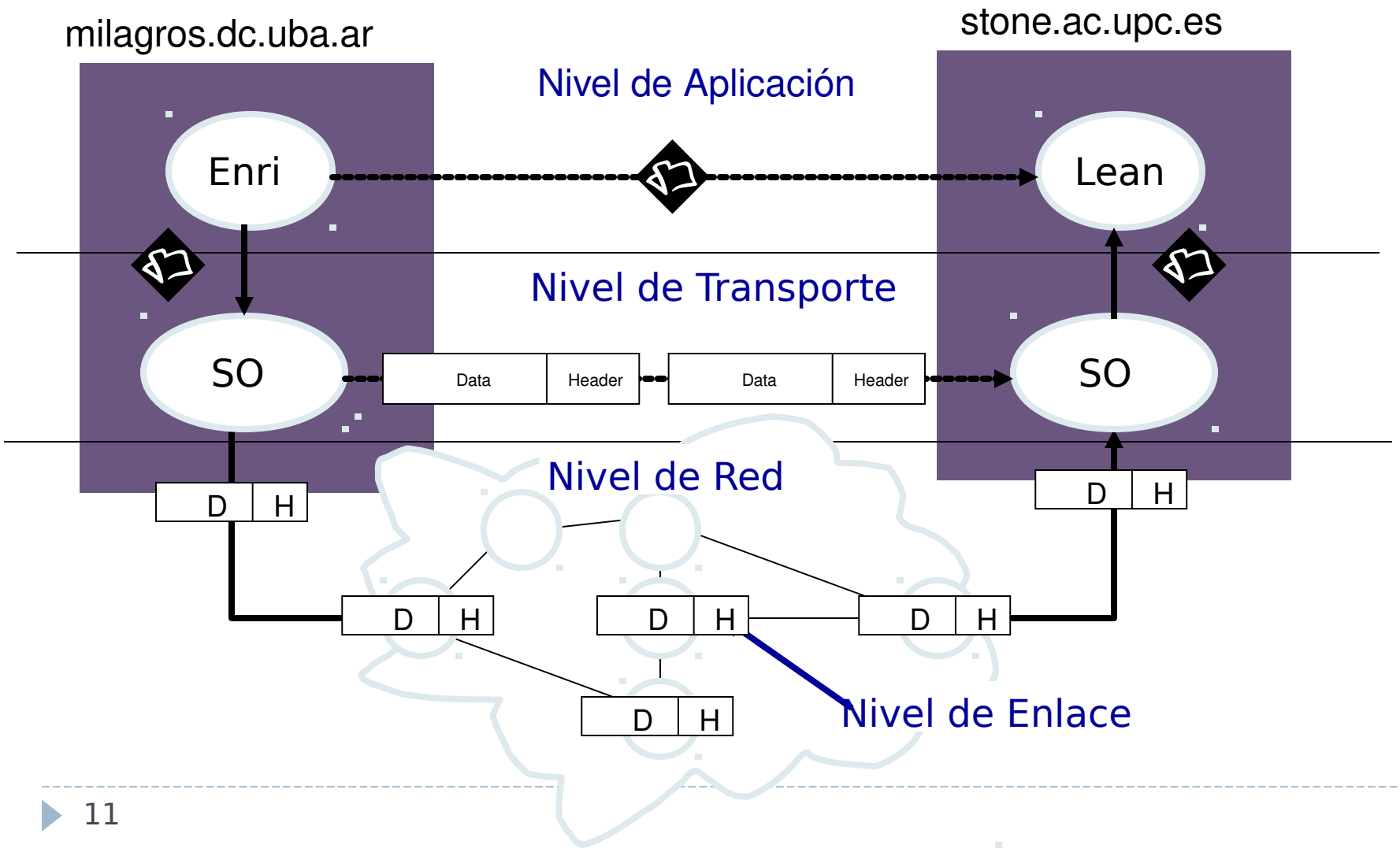


Diagrama de estado de un esquema sencillo de manejo de conexiones. Las transiciones
escritas en cursivas son causadas por llegadas de paquetes. Las líneas continuas muestran la secuencia
de estados del cliente. Las líneas punteadas muestran la secuencia de estados del servidor.

Nivel Transporte : “End to End”



Enlace de Datos versus Transporte

- Potencialmente conecta muchas máquinas diferentes
  - requiere de establecimiento y término de conexión explícitos
- Potencialmente diferentes RTT
  - requiere mecanismos adaptivos para timeout
- Potencialmente largos retardos en la red
  - requiere estar preparado par el arribo de paquetes muy antiguos
- Potencialmente diferente capacidad en destino
  - requiere acomodar diferentes capacidades de nodos
- Potencialmente diferente capacidad de red
  - requiere estar preparado para congestión en la red

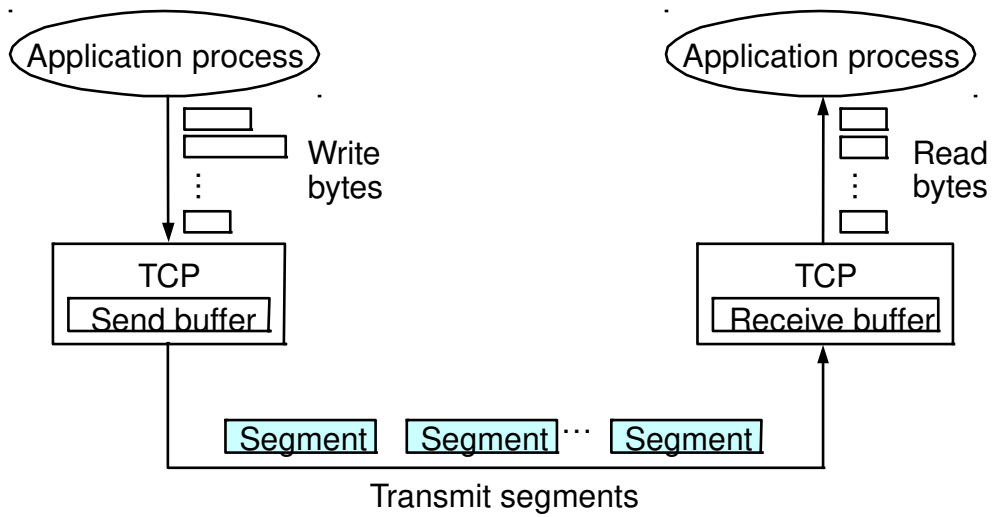
Protocolos E2E en subredes datagramas

- Se apoyan en la capa Red, la cual es de mejor esfuerzo (best-effort)
  - descarta mensajes
  - re-ordena 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 ofrecidos/deseados end-to-end
  - garantía de entrega de mensajes
  - entrega de mensajes en el mismo orden que son enviados
  - entrega de a lo más una copia de cada mensaje
  - soporte para mensajes arbitrariamente largos mensajes
  - soporte de sincronización
  - permitir al receptor controlar el flujo de datos del transmisor
  - soportar múltiples procesos de nivel aplicación en cada máquina

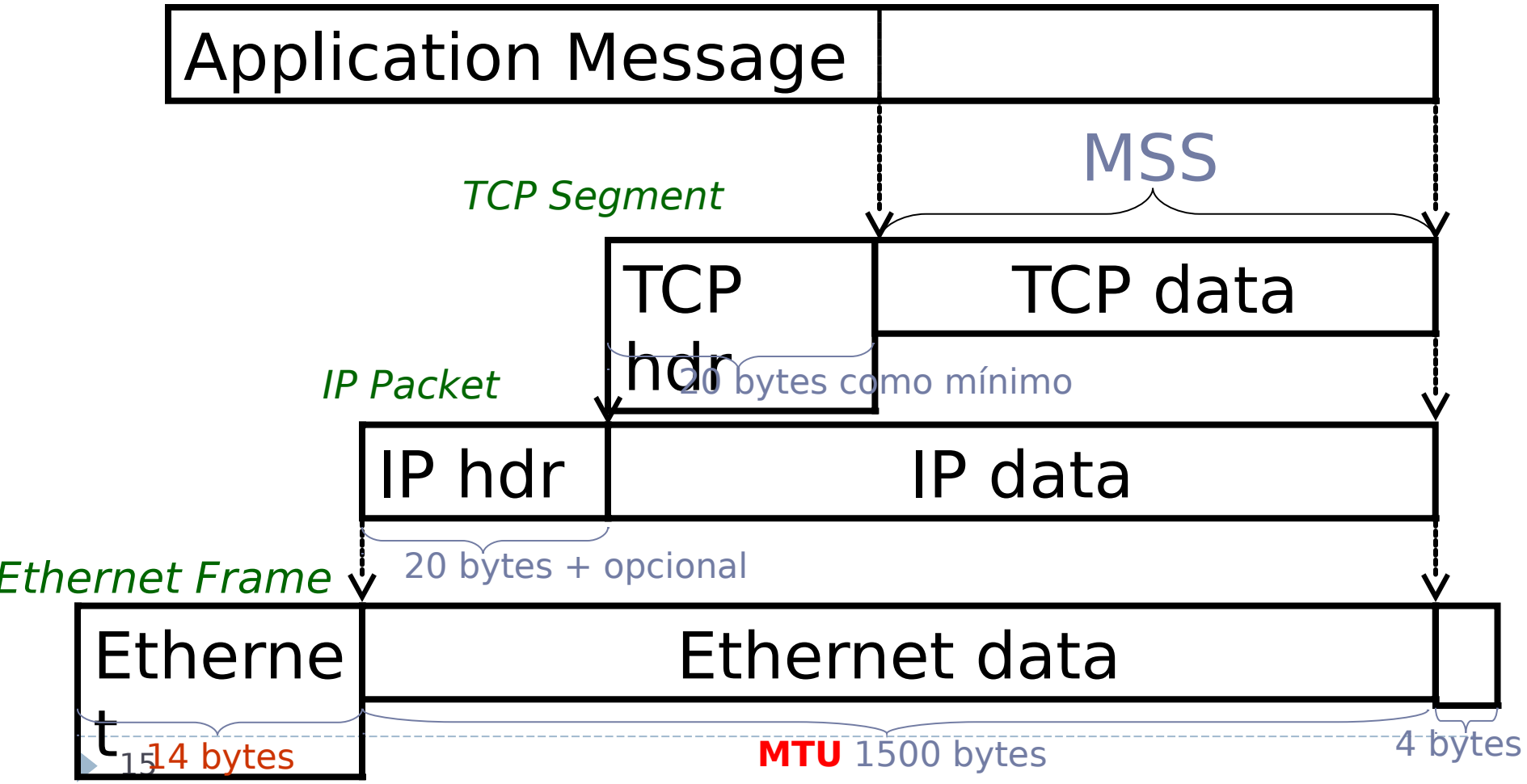
TCP Generalidades

- Orientado a conexión
- flujo de byte
  - app escriben bytes
  - TCP envía *segmentos*
  - app lee bytes

- Full duplex (dos flujos de bytes)
- Control de flujo: evita que el Tx inunde al receptor
- Control de congestión: evita que el Tx sobrecargue la red



MSS :"Maximum Segment Size"



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

SOCKET crea un nuevo punto de comunicación y le asigna espacio en las tablas de la entidad de transporte. Los parámetros de la llamada especifican el formato de direccionamiento
que se utilizará, el tipo de servicio deseado (por ejemplo, flujo confiable de bytes) y el protocolo.
Una llamada SOCKET con éxito devuelve un descriptor de archivo ordinario que se utiliza con las
siguientes llamadas, de la misma manera que lo hace una llamada OPEN

TCP

Características

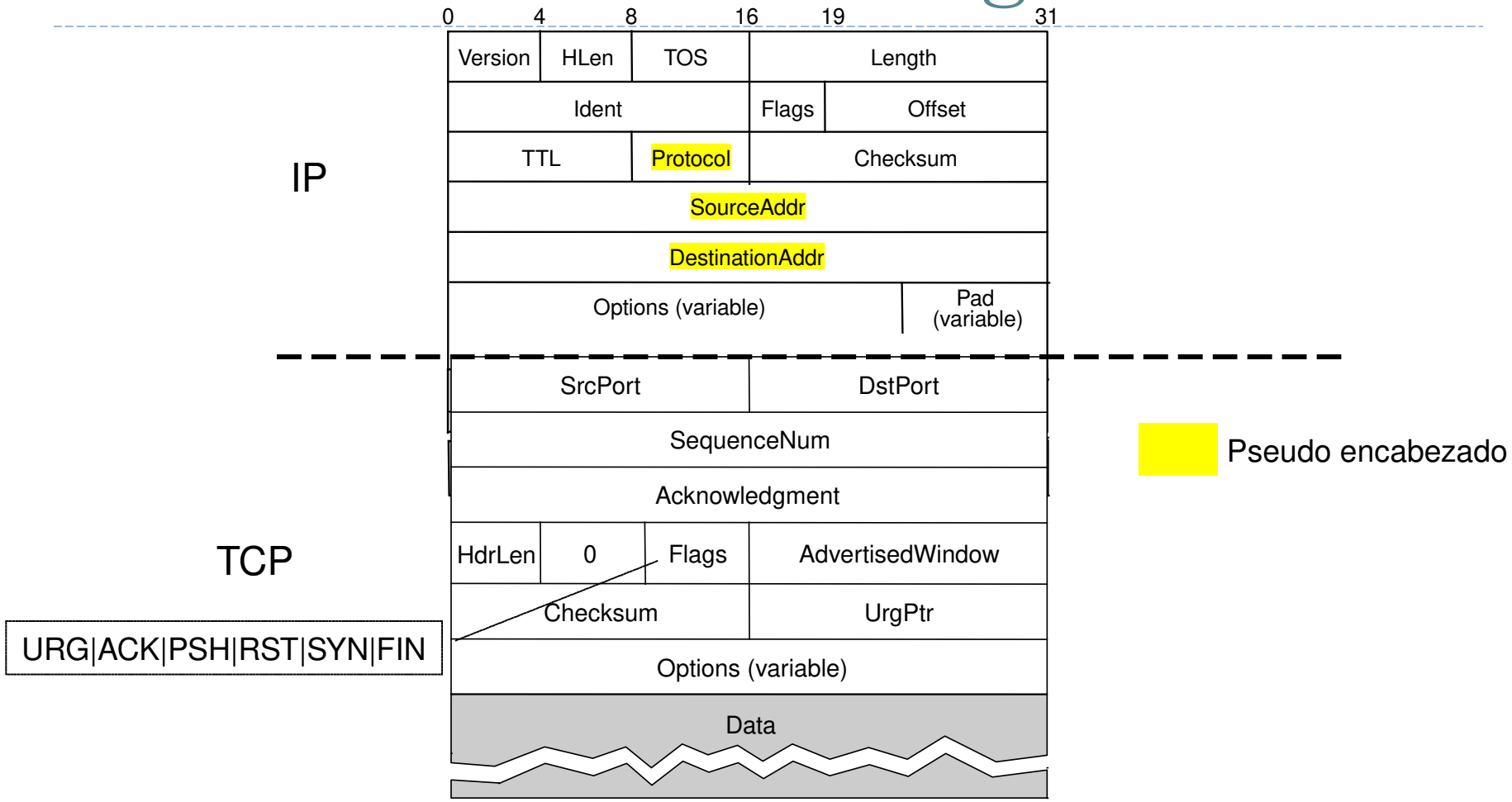
Formato del Header

TCP : Características

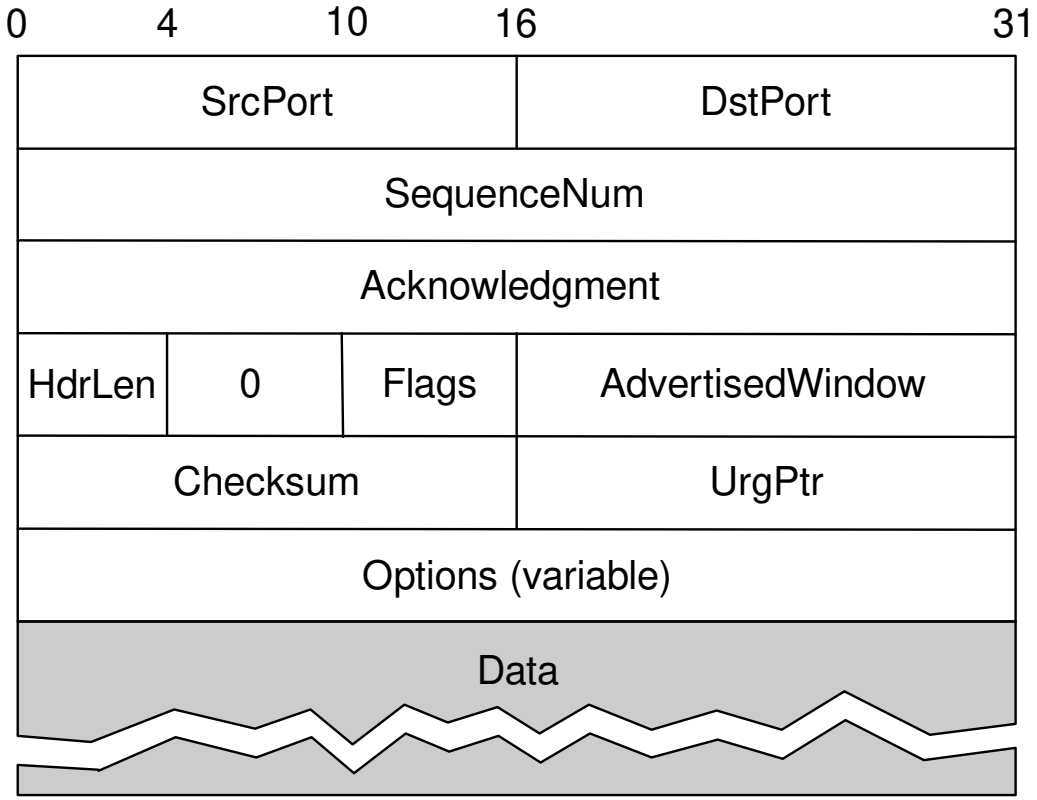
- TCP es orientado a conexion
  - Manejo de la conexión : 3-way handshake usado para setup y 2-2 o 4 way handshake para la liberación (“problema de los dos ejércitos”)

- TCP provee un servicio de flujo de bytes (***stream-of-bytes*** **)**
- TCP es confiable (estableciendo una suerte de “conexión lógica entre los sockets”)
  - Acknowledgements ACKs
  - Checksums
  - Números de secuencia para detectar datos perdidos o desordenados
  - Datos perdidos o corruptos se RTX después de un timeout.
  - Datos desordenados se podrían reordenar.
  - Control de Flujo evita inundar al receptor .
- TCP implementa mecanismos de ***control* de *congestión (se le dedica una clase la semana próxima).***

Contexto Formato de Segmento

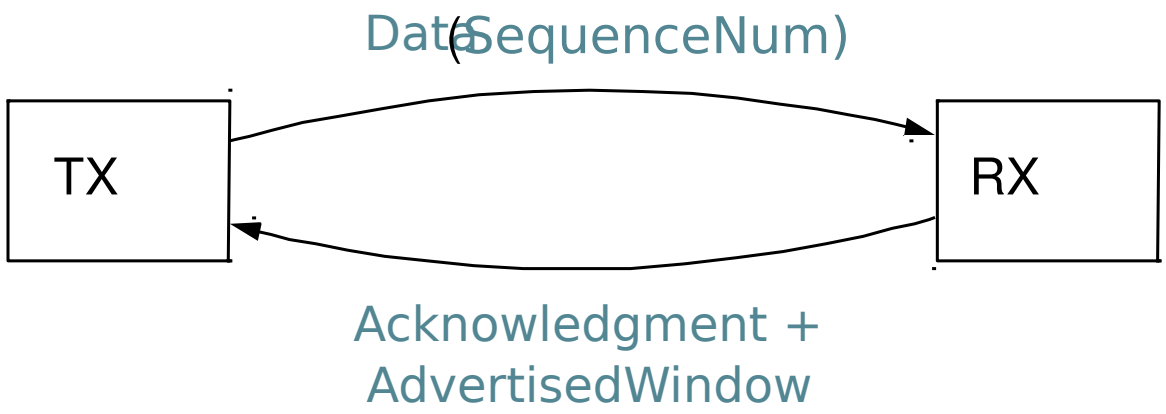


Formato de Segmento



TCP

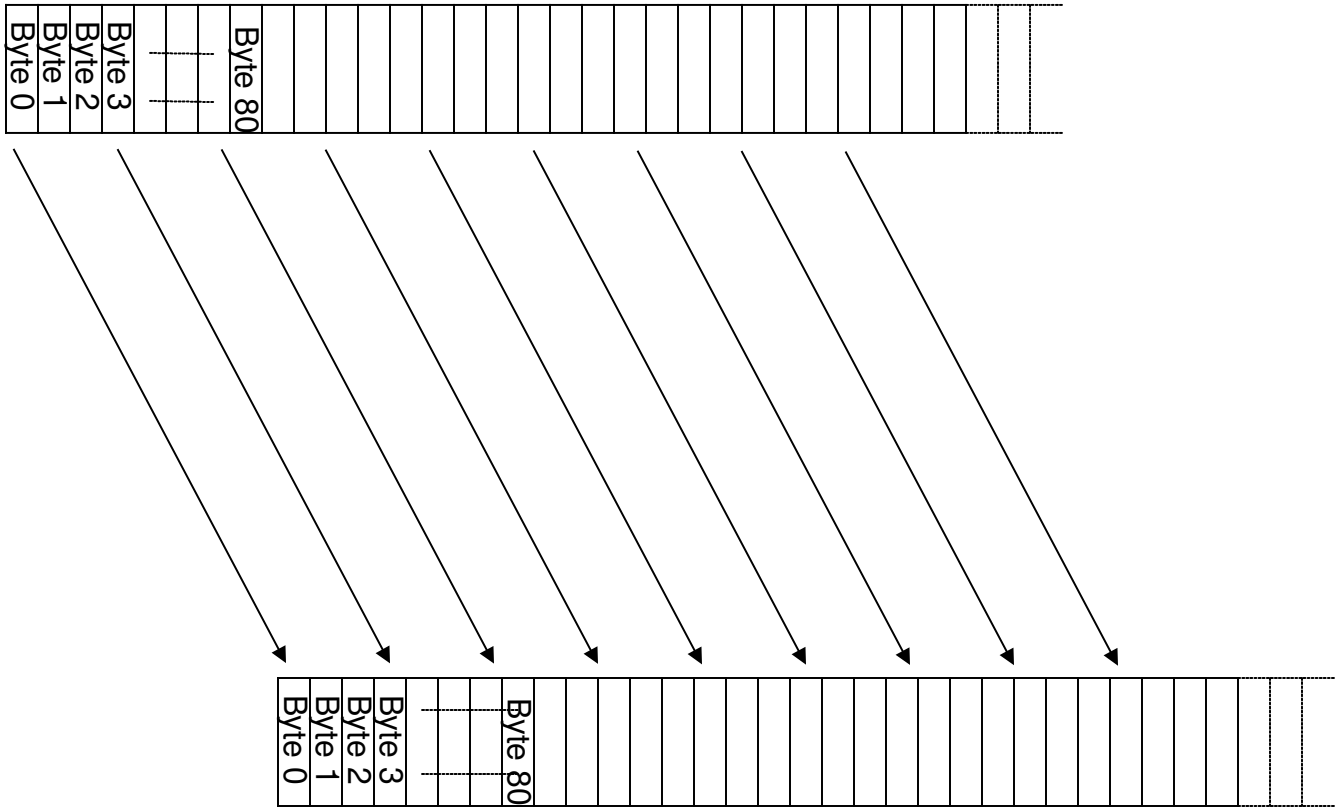
- Cada conexión es identificada por la 4-tupla:
  - **(SrcPort, SrcIPAddr, DsrPort, DstIPAddr)**
- Ventana deslizante + control de flujo
  - **acknowledgment, SequenceNum, AdvertisedWindow**



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

TCP soporta un *"stream de bytes"*

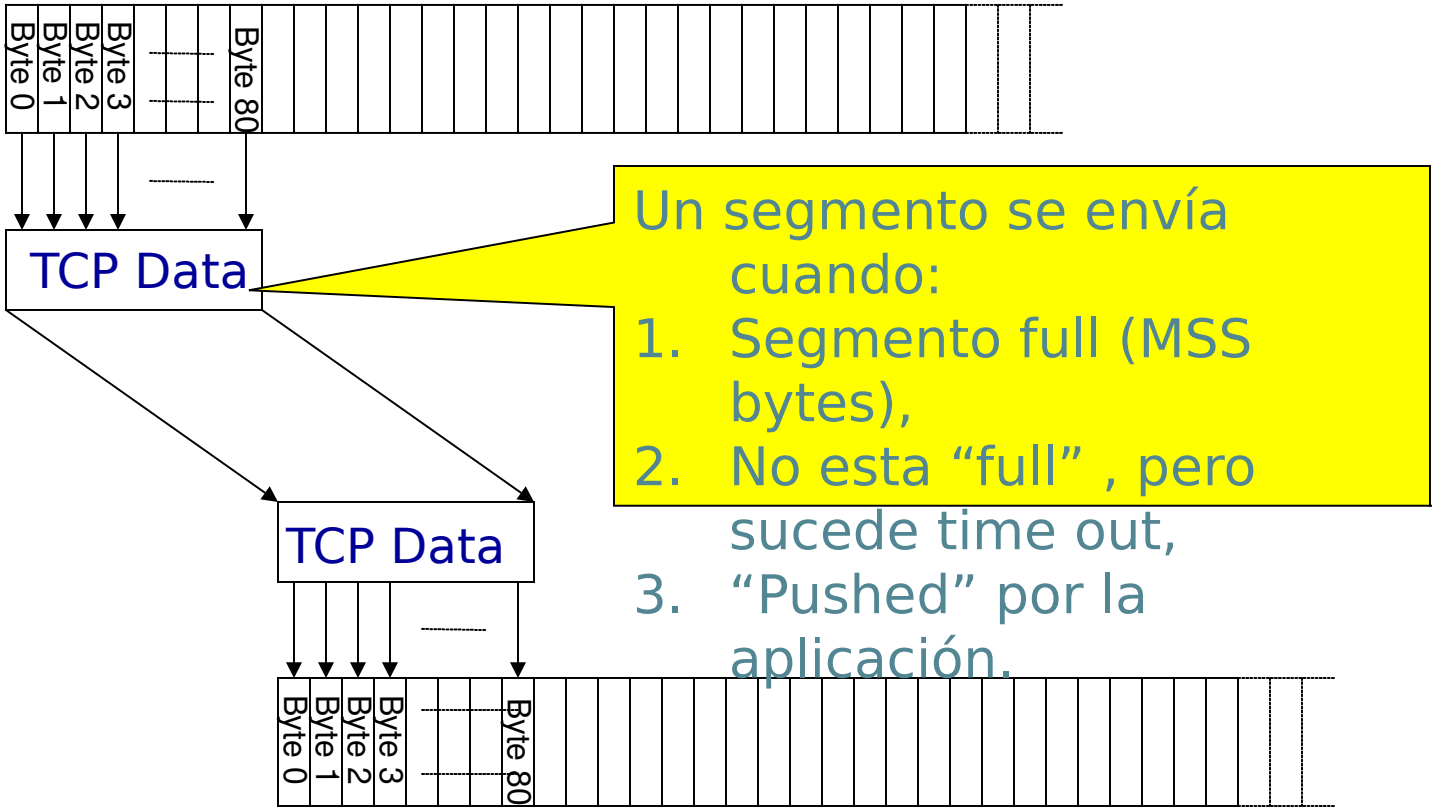
Host A



Host B

…se emula usando segmentos

Host A



Un segmento se envía cuando:

1. Segmento full (MSS bytes),
2. No esta “full” , pero sucede time out,
3. “Pushed” por la aplicación.

Host B

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 inequívoca los datos de aplicación que contiene el segmento TCP. Identifica el primer *byte de datos*
- **Número de reconocimiento (32 bits) :** El indica el siguiente *byte que espera el receptor*
  - Implica la confirmación (ACK) de todos los *bytes recibidos hasta ese momento*.
- **Longitud 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.

La cabecera de TCP

- **Código de redundancia:** Verifica la cabecera y los datos.
- **Puntero a urgente:** Se utiliza para indicar un desplazamiento en bytes a partir del número de secuencia actual en el que se encuentran los datos urgentes. Esta facilidad se brinda en lugar de los mensajes de interrupción.
- **Opciones:** Diseñado para proveer una manera de adicionar facilidades extras no cubiertas por la cabecera regular.

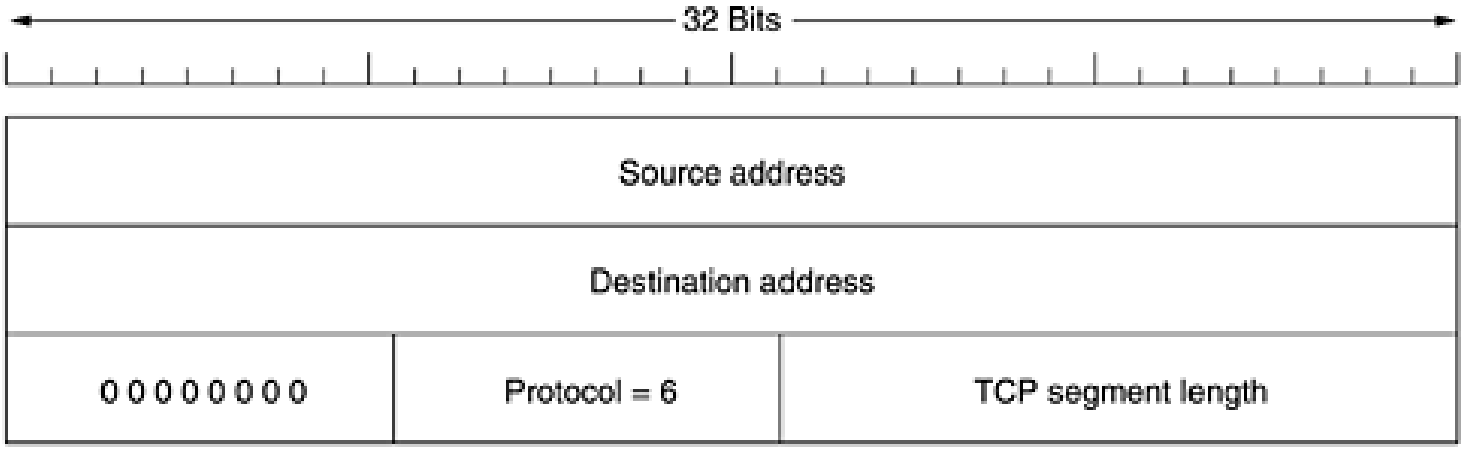
Extensiones de TCP

- Son implementadas como opciones del encabezado
- Almacenar marcas de tiempo en segmentos de salida (con que objeto ??)
- Extender espacio de secuencia con marca de tiempo de 32-bit (PAWS) ?
- Desplazar (escalar) ventana avisada. La idea es medir la ventana en unidades de 2, 4, 8 bytes.

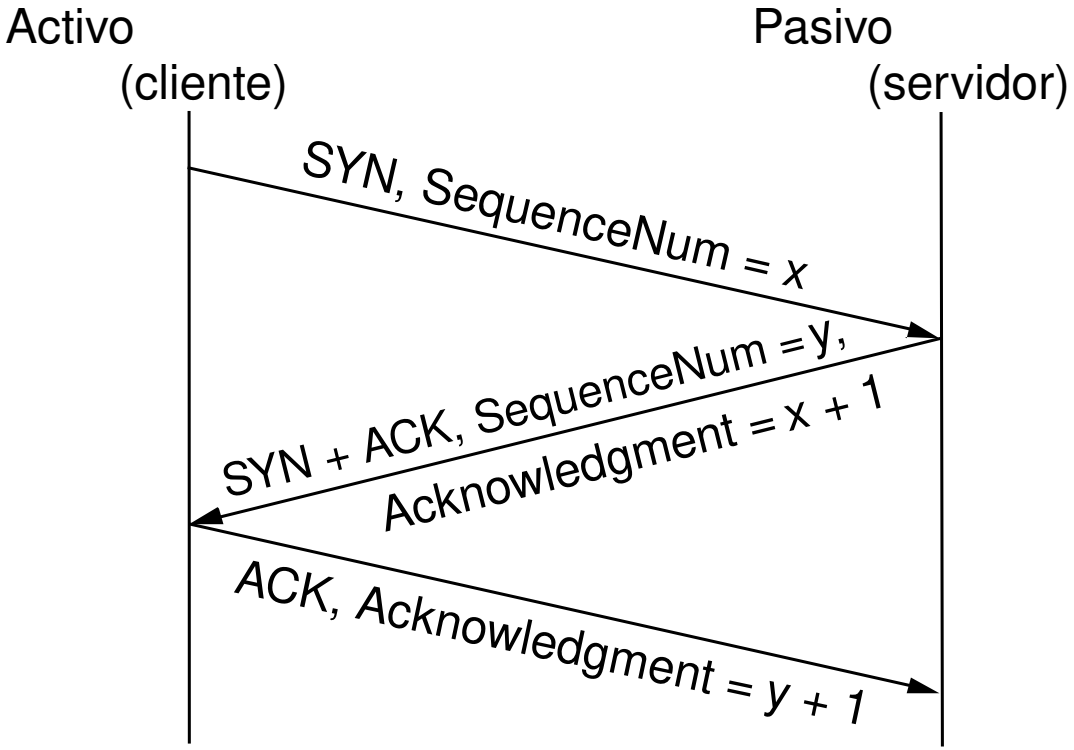
Checksum !

- Se calcula entre el TCP Header , Data y el pseudo header

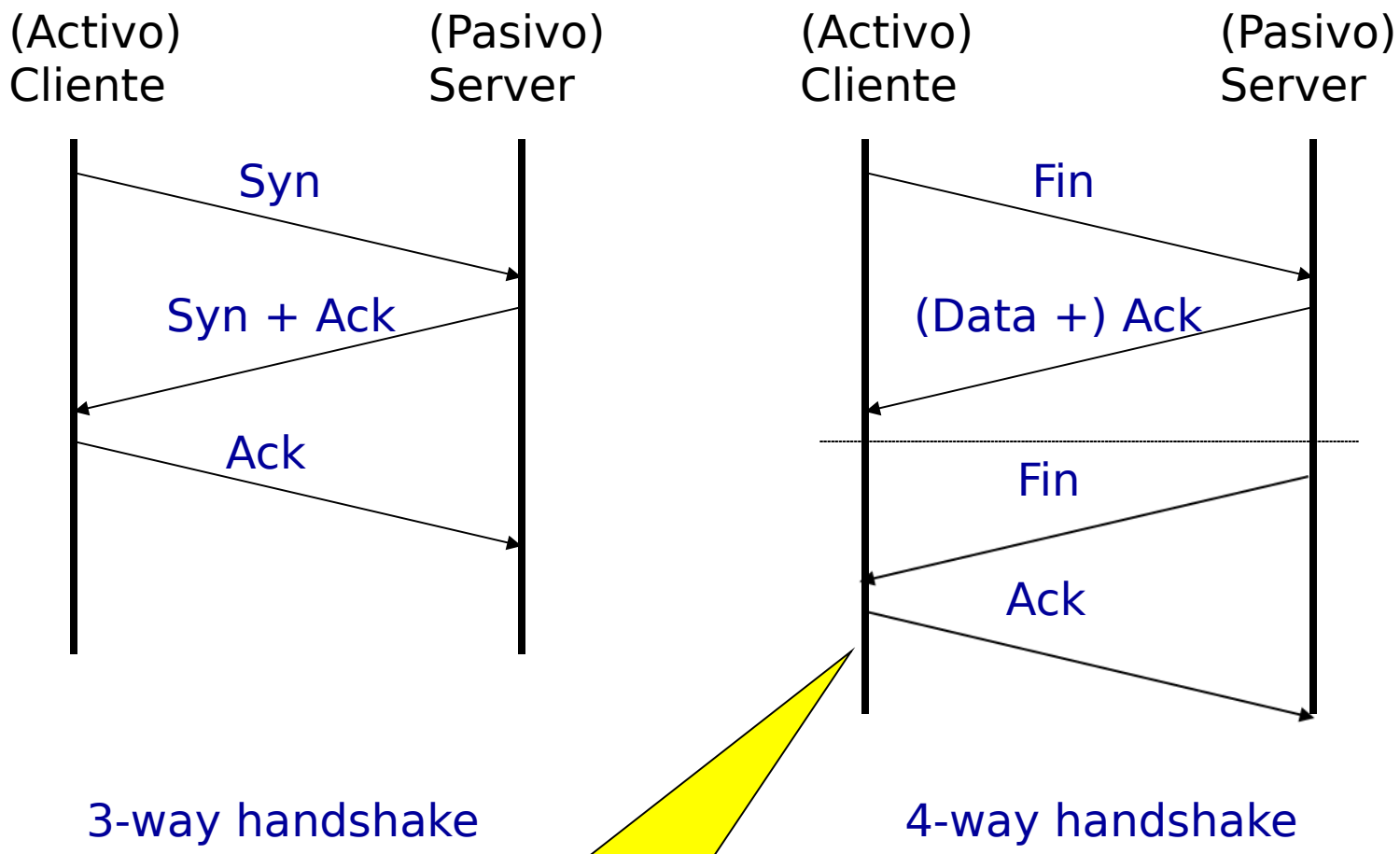
- Como impacta cuando IP es IPv6 ????



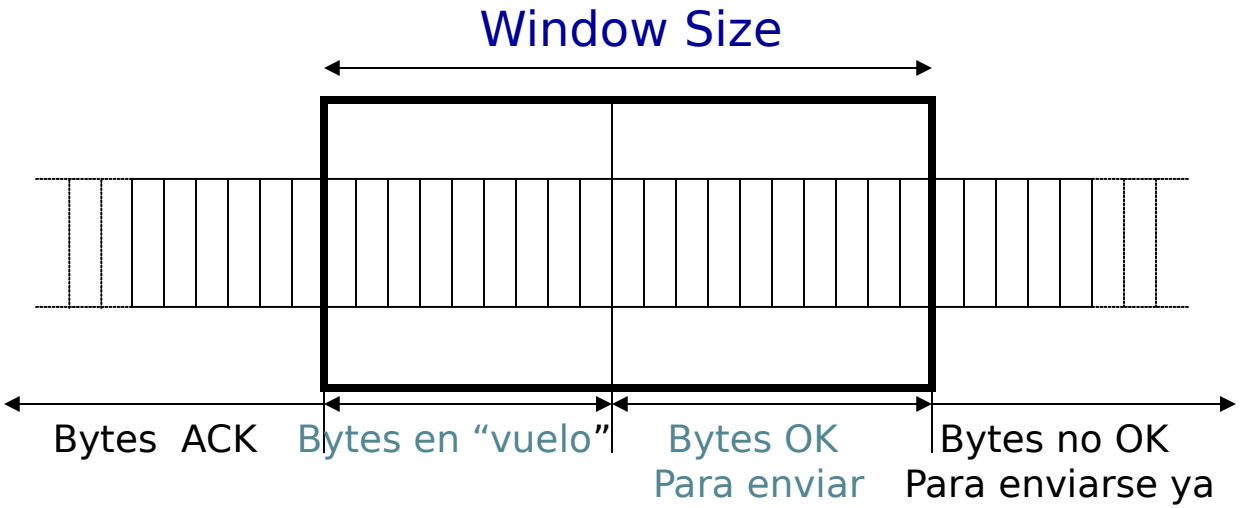
Establecimiento de la Conexión



Liberación de la Conexión



“TCP Sliding Window”



- Política de retransmisión “Go Back N”.
- window size es “advertised” por el receptor (server) (usualmente 4k – 8k Bytes en connection set-up).

TCP

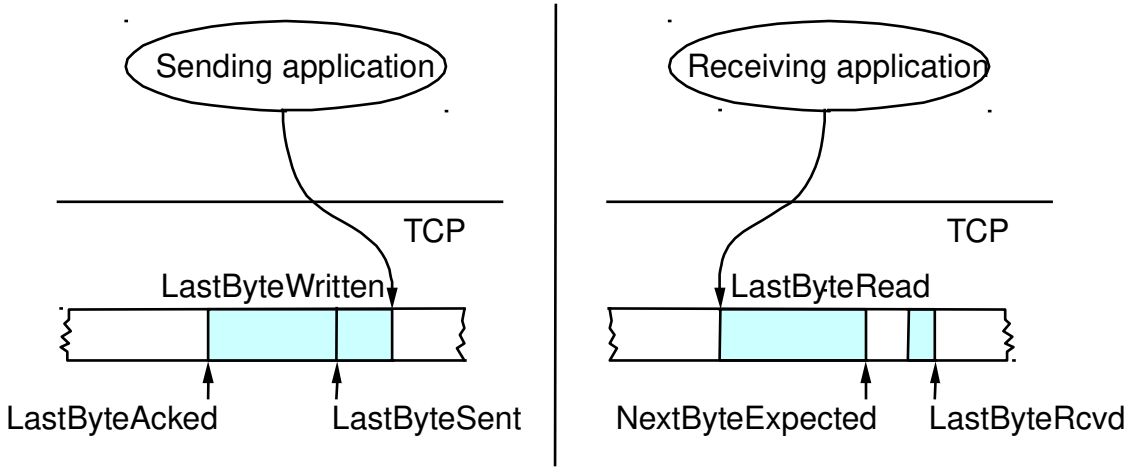
Ventana Deslizante

Control de Flujo

TCP : ventana deslizante

- Se implementa una variante del protocolo de ventana deslizante que estudiamos para los 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



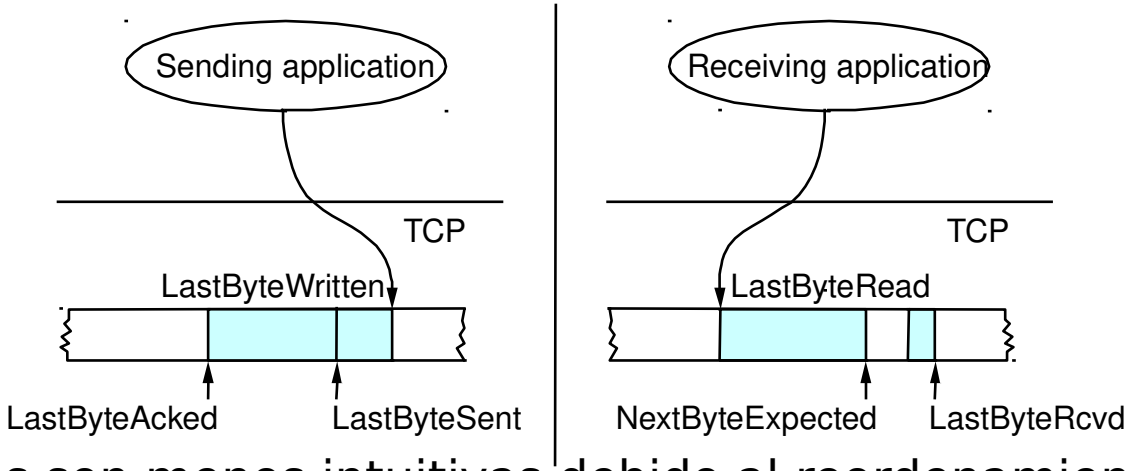
- Lado Transmisor : tres punteros , las relaciones son obvias

LastByteAcked < = LastByteSent

LastByteSent < = LastByteWritten

Se bufferean los bytes entre **LastByteAcked** (los que están a su derecha ya fueron confirmados) y **LastByteWritten** (los que están a su derecha no fueron 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 este se halla recibido y todos su

precedentes

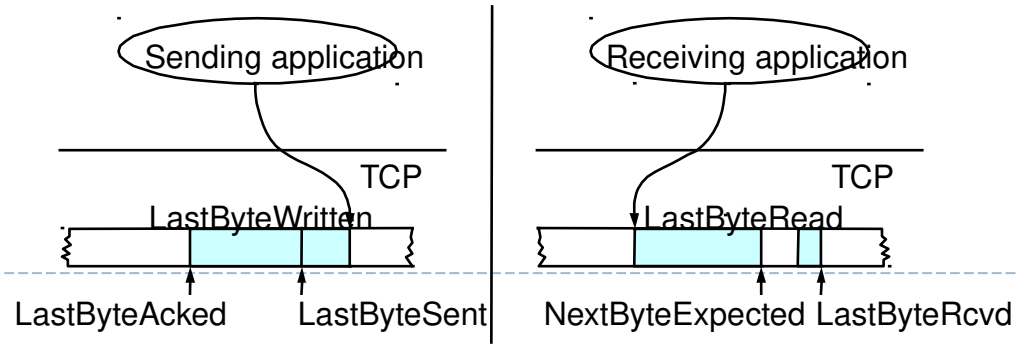
NextByteExpected <= LastByteRcvd +1

Si están en orden se cumple la igualdad . Si están en desorden **NextByteExpected** apunta al gap de la figura **LastByteRead+1**

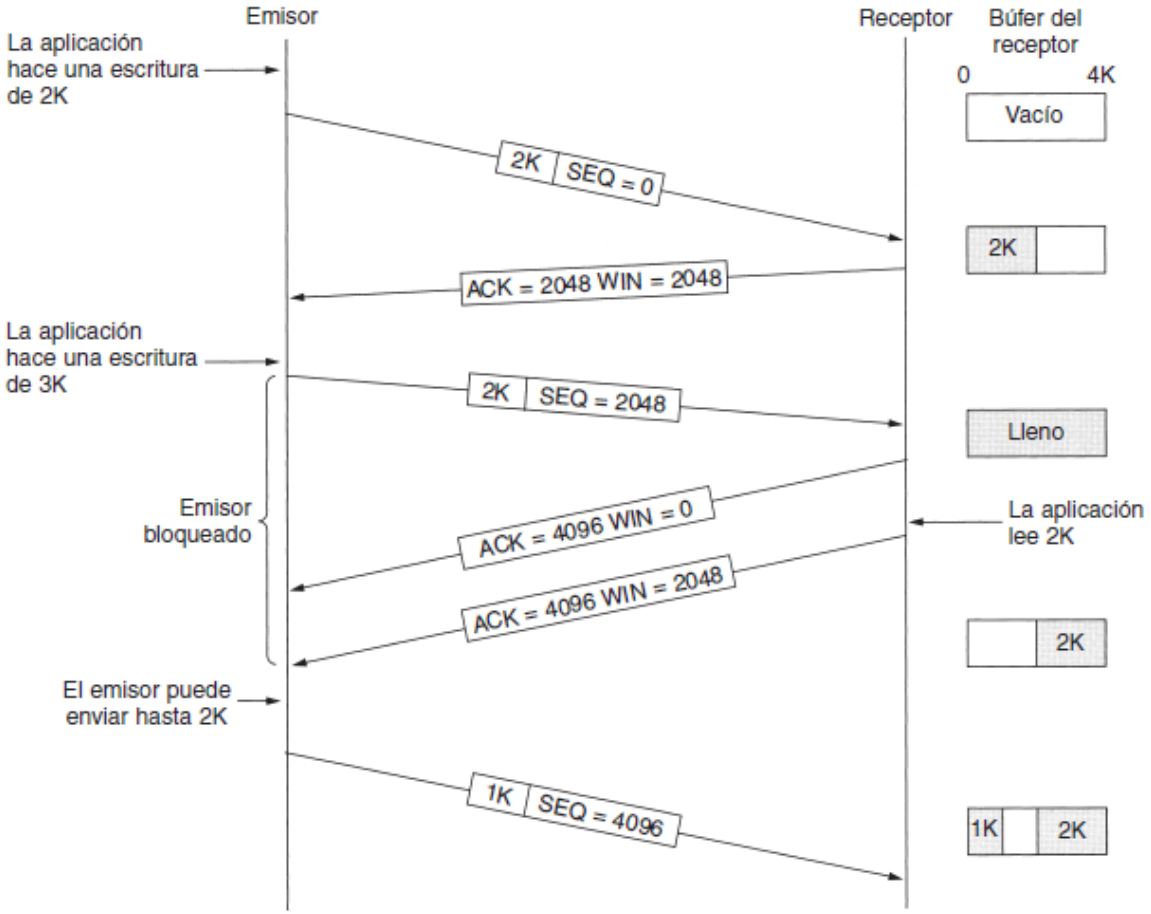
Buffereamos los bytes entre **NextByteRead** y **LastByteRcvd**

Control de Flujo

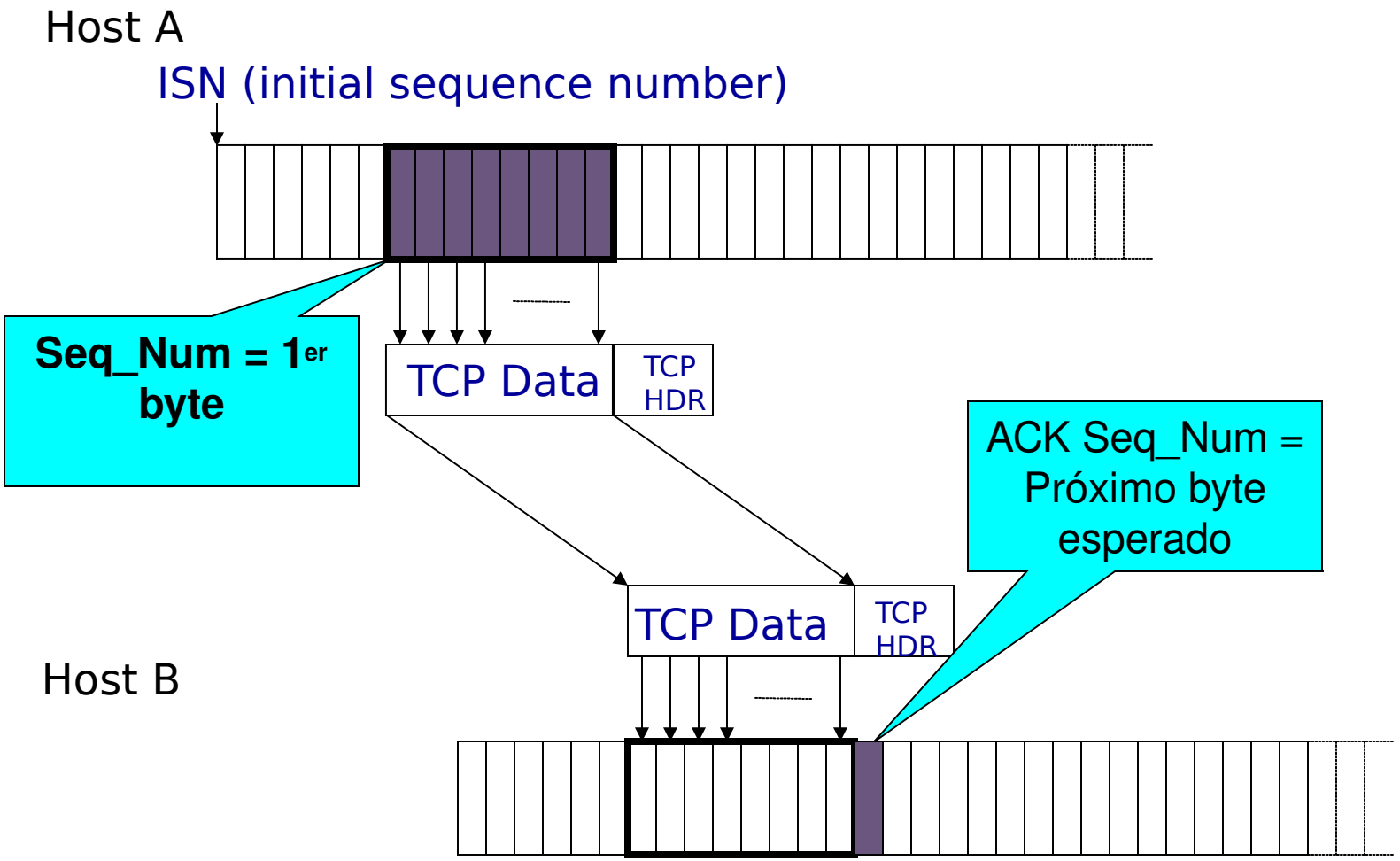
- Tamaño del buffer de envío: **MaxSendBuffer**
- Tamaño del buffer de recepción: **MaxRcvBuffer**
- Lado receptor
  - **LastByteRcvd** - **LastByteRead** < = **MaxRcvBuffer**
  - **AdvertisedWindow** = **MaxRcvBuffer** - (**LastByteRcvd** - **NextByteRead**)
- Lado Transmisor
  - **LastByteSent** - **LastByteAcked** < = **AdvertisedWindow**
  - **EffectiveWindow** = **AdvertisedWindow** - (**LastByteSent** - **LastByteAcked**)
  - **LastByteWritten** - **LastByteAcked** < = **MaxSendBuffer**
  - Bloquear Tx si (**LastByteWritten** - **LastByteAcked**) + *y* > **MaxSenderBuffer,** *y bytes que se desean escribir.*
- Siempre enviar ACK en respuesta a la llegada de segmentos de datos
- Tx persiste enviando 1 byte cuando **AdvertisedWindow = 0**



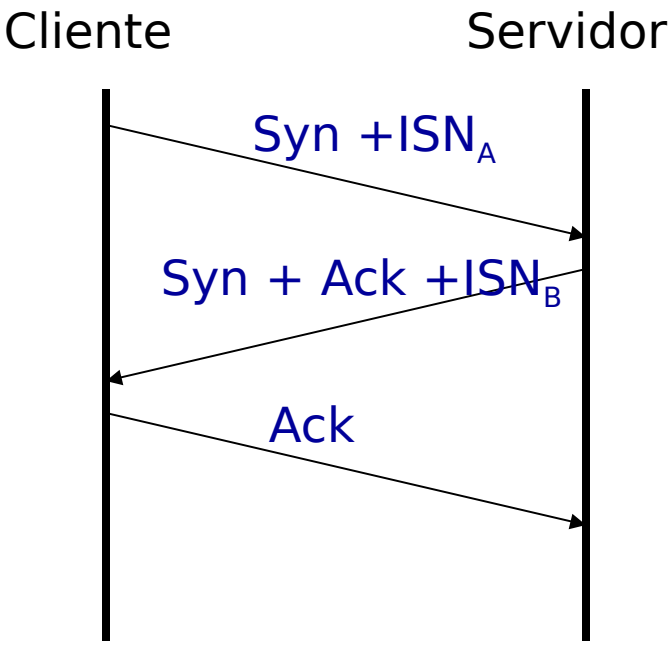
Window Management



Números de Secuencia



ISN: Initial Sequence Numbers



Protección contra reapariciones de igual número de secuencia

- **`SequenceNum de 32 bits`**

Bandwidth	Tiempo hasta tener problema
T1 (1.5 Mbps)	6.4 horas
Ethernet (10 Mbps)	57 minutos
T3 (45 Mbps)	13 minutos
FDDI (100 Mbps)	6 minutos
STS-3 (155 Mbps)	4 minutos
STS-12 (622 Mbps)	55 segundos
STS-24 (1.2 Gbps)	28 segundos

Agenda- 2 Parte

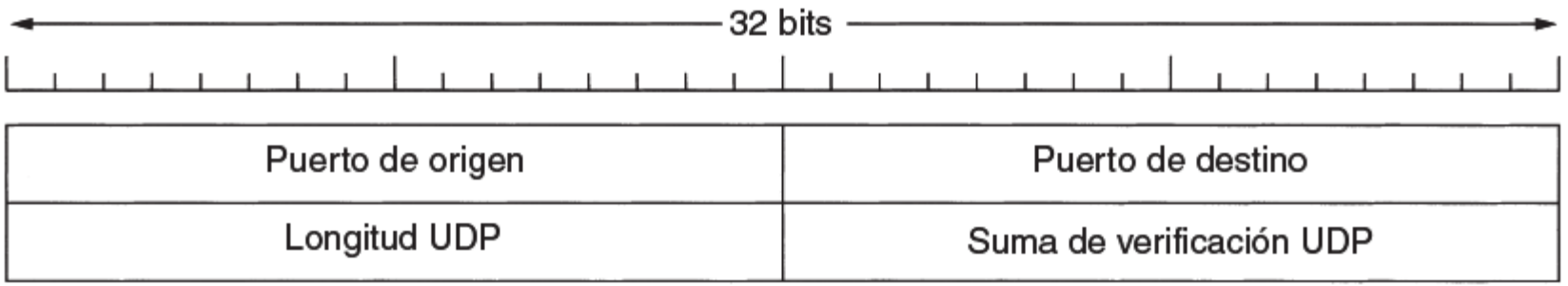
- UDP (User Datagram Protocol)
- El Protocolo TCP
  - Timeouts y Retransmisiones (RTX)
  - Maquina de Estados Finita
  - Ineficiencia por envío de segmentos pequeños y que el receptor los pida

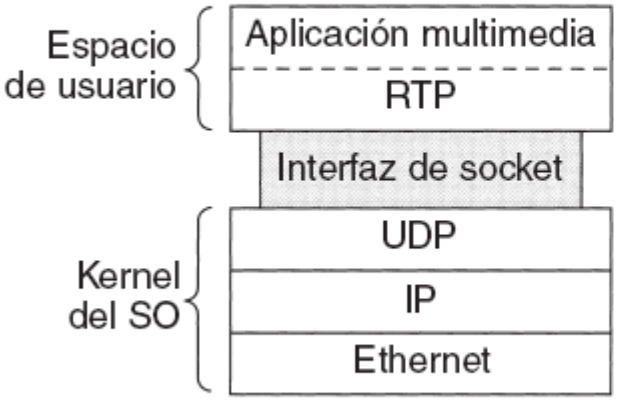
    - Uso del ancho de banda
    - Sockets

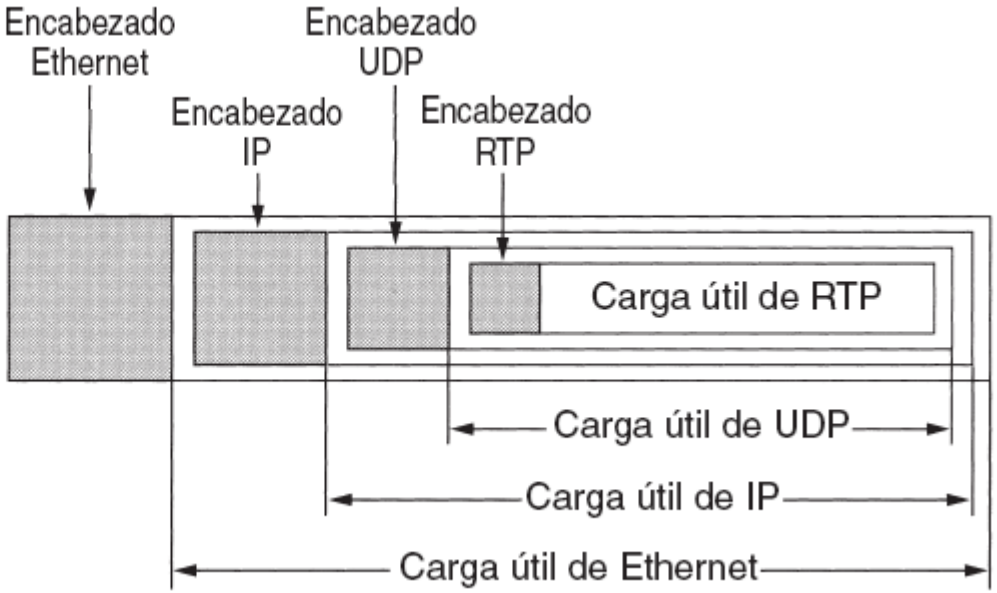
UDP

Servicio sin conexión

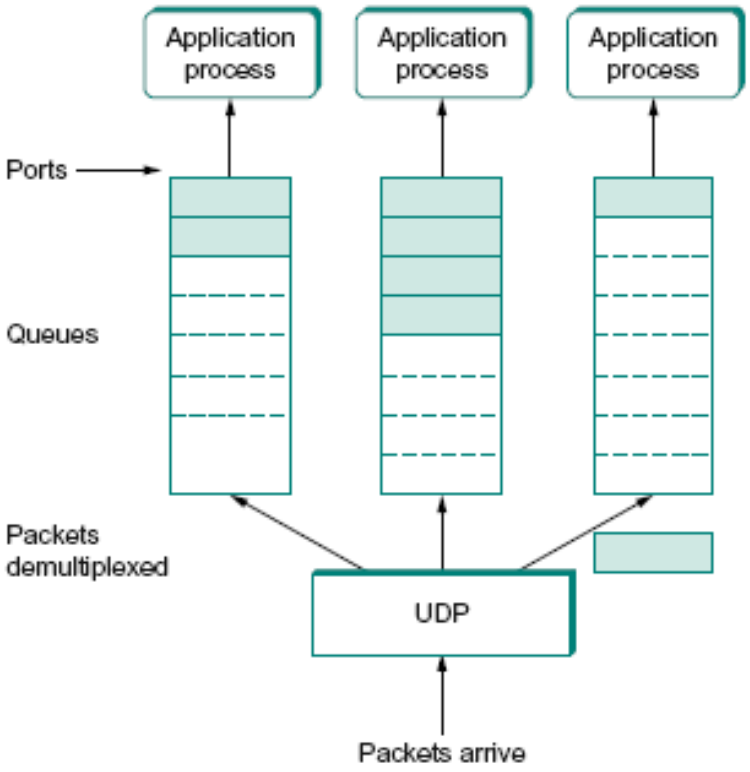
Encabezado UDP y aplicación RTP







Multiplexación



TCP

Retransmisión Adaptativa

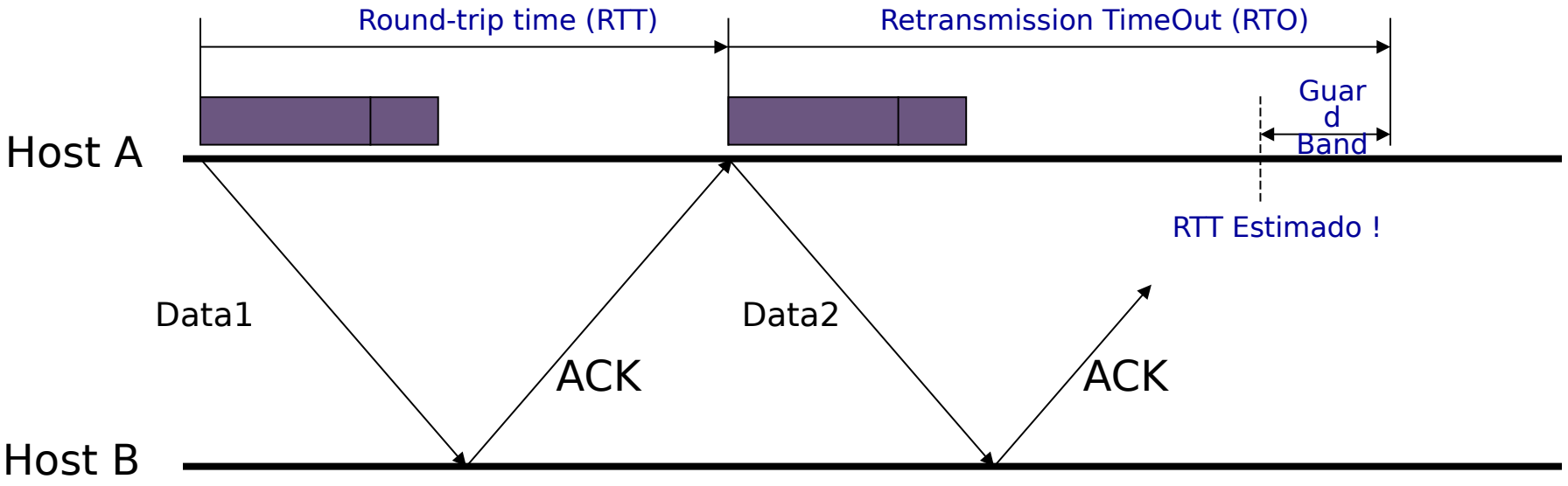
Jacobson (1988)

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

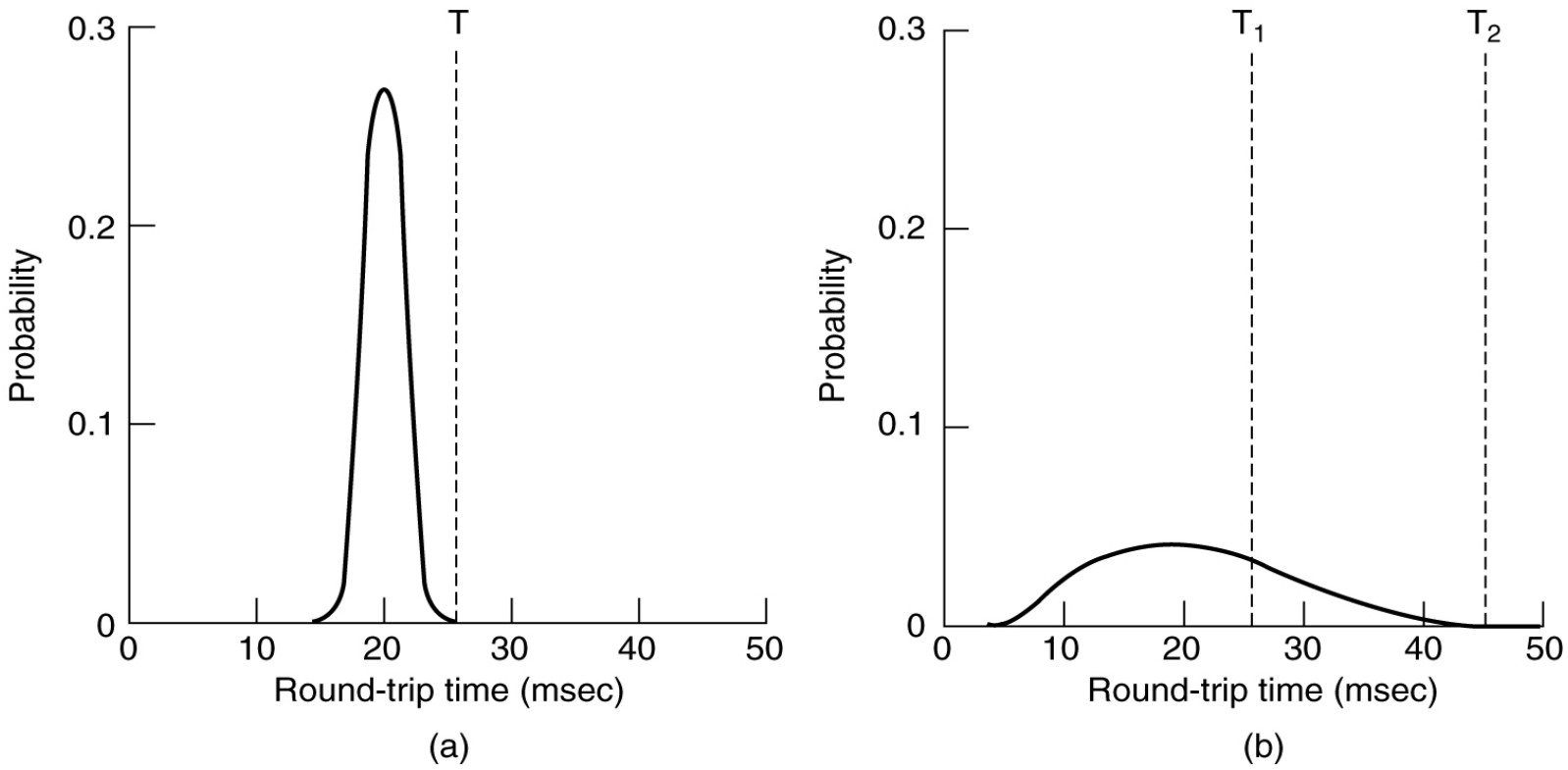


valor adaptativo de timeout de RTX (lazo cerrado !!!!!, teoría de control) :

Congestión
Cambios de ruta

} RTT es muy variable

Concepto :TCP Timer Management



(a) Función densidad de probabilidad para tiempos de llegadas de ACK a nivel 2

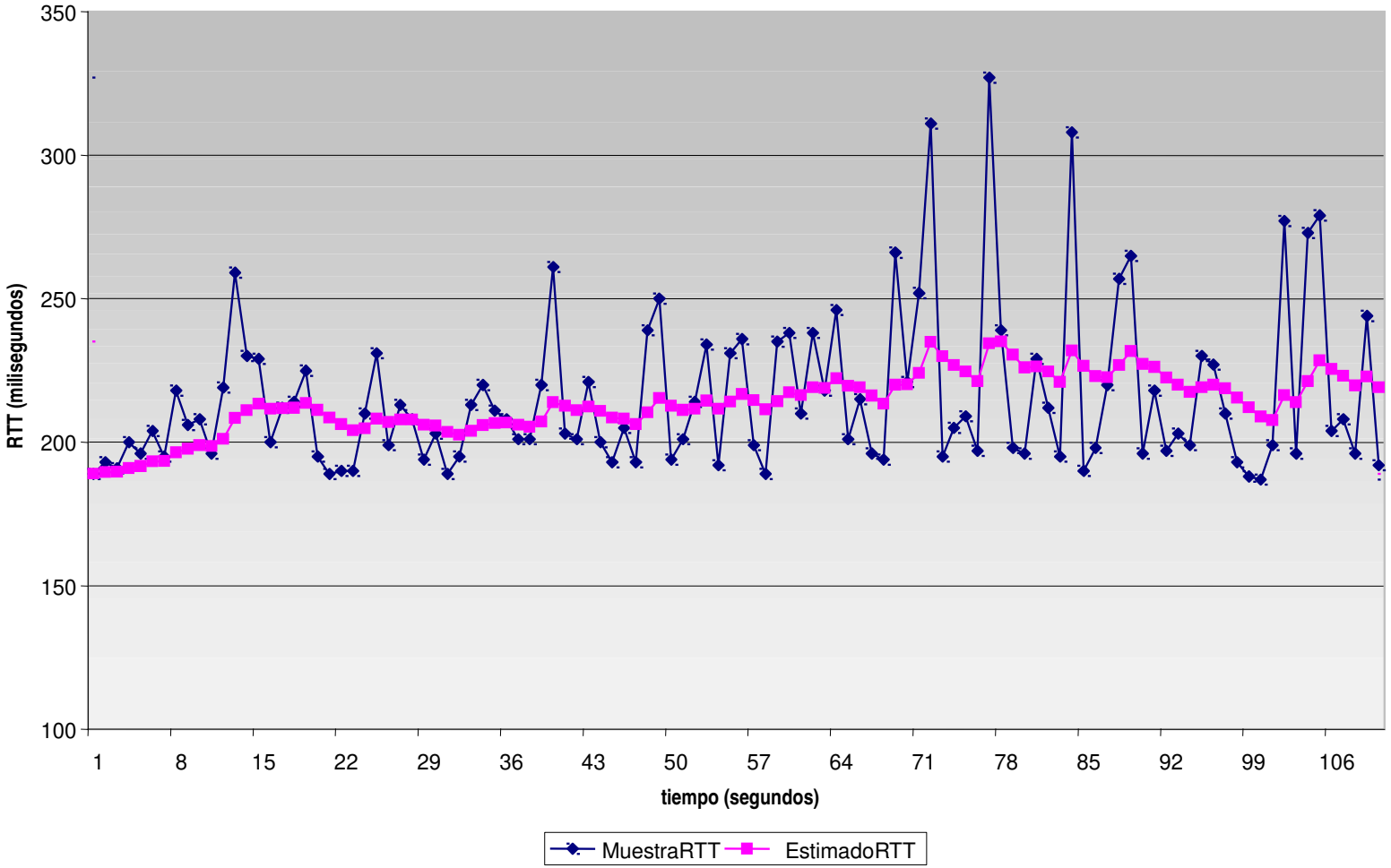
(b) Función densidad de probabilidad para tiempos de llegadas de ACK TCP

Figuras : Tanembaum 4 edicion

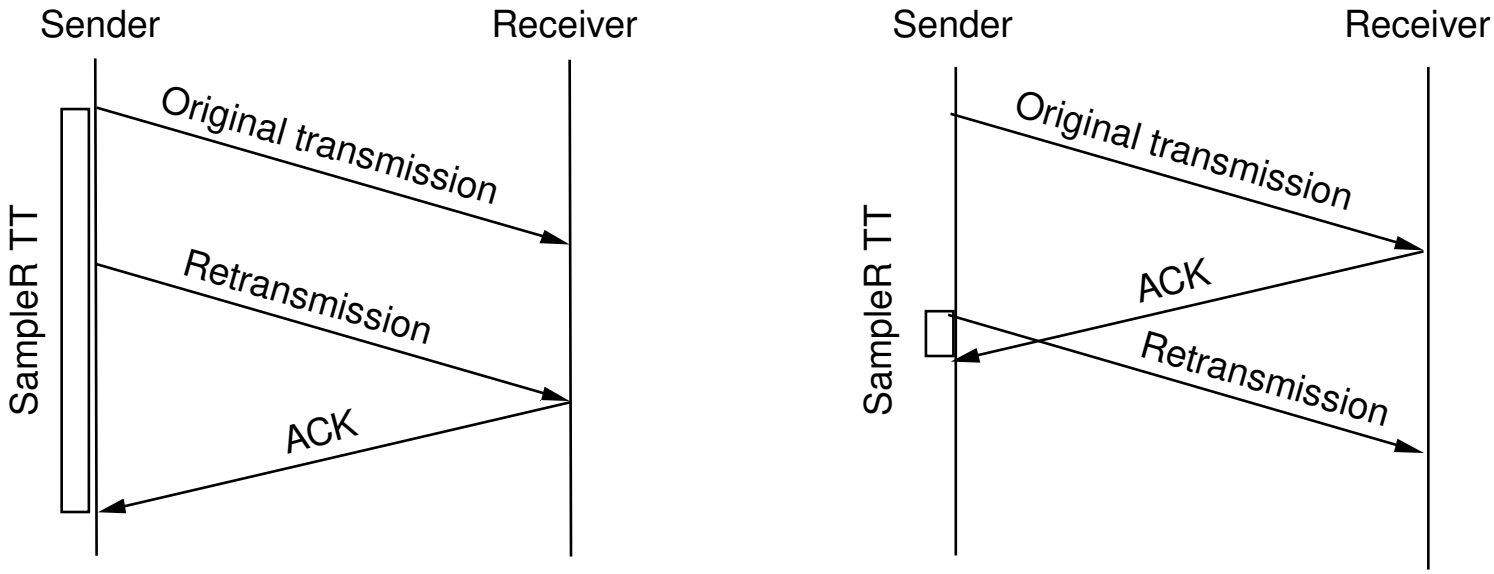
Retransmisión Adaptativa (Algoritmo Original)

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

Ejemplo de estimación de RTT



Algoritmo de Karn/Partridge [KP87]



- No considerar RTT cuando se retransmite
- Duplicar timeout luego de cada retransmisión

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

Algoritmo de Jacobson/ Karels [JK88]

Proponen un nueva forma de calcular Timeout de RTX

- **Diff = sampleRTT - EstRTT**
- **EstRTT = EstRTT + (δ x Diff)**
- **Dev = Dev + δ (|Diff| - Dev)**
  - donde δ es un factor entre 0 y 1 (Por ejemplo 1/8)
- Considerar varianza cuando fijamos el timeout
- **TimeOut** = μ x **EstRTT** + ϕ x **Dev**
  - donde $\mu = 1$ y $\phi = 4$
- *Notas*
  - *Los algoritmos son tan buenos/malos como la granularidad del reloj (500ms en Unix)*
  - *Un preciso mecanismo de timeout es importante para controlar la congestión (como veremos en la próxima clase)*
  - *Además de ayudar a controlar la congestión, la idea es no retransmitir cuando no es necesario.*

[JK88] Jacobson, V. and M. Karels, "Congestion Avoidance and Control", ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z.

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 mas)

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 = 1seg (granularidad)

RFC 6298 "Computing TCP's Retransmission Timer", continuación

- 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]).

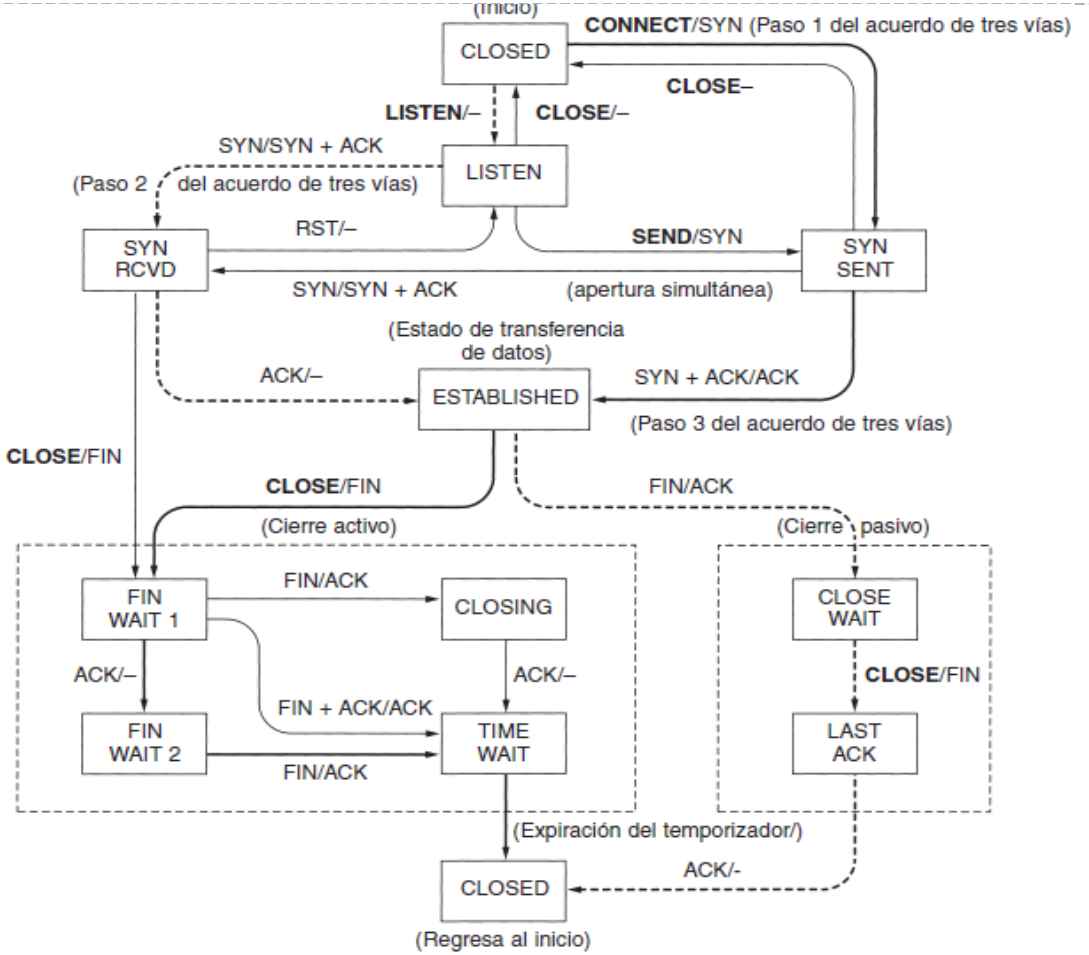
Nota : el valor de SRTT usado para actualizar RTTVAR en (1) es el valor antes de su propia actulizacion en (2). RTTVAR y SRTT debe calcularse en el orden precedente .

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

TCP

Maquina de estado finitos

Administración de conexiones



Máquina de estados finitos de administración de conexiones TCP. La línea continua gruesa es la trayectoria normal de un cliente. La línea punteada gruesa es la trayectoria normal de un servidor. Las líneas delgadas son eventos poco comunes. Cada transición está indicada por el evento que la ocasiona y la acción resultante, separada por una diagonal.

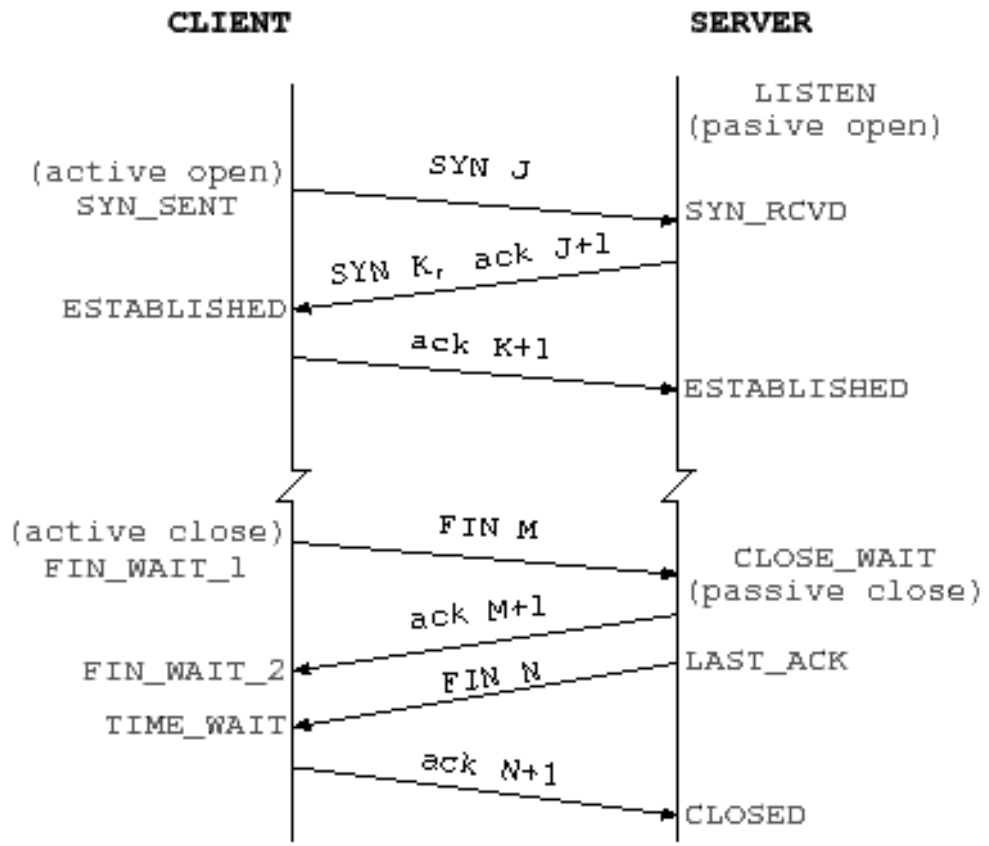
Administración de conexiones

La máquina de estados finitos del slide anterior . 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.*

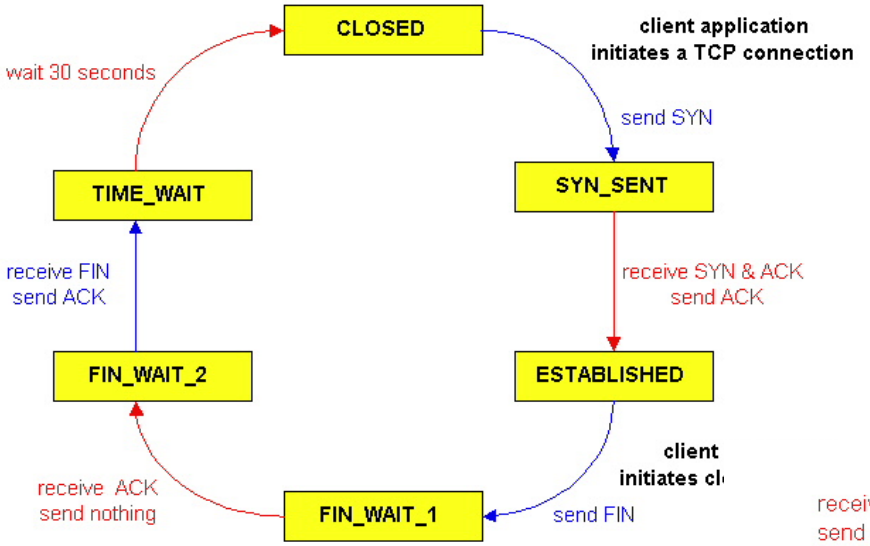
El evento puede ser una llamada de sistema iniciada por el usuario (CONNECT, LISTEN, SEND o CLOSE), la llegada de un segmento (SYN, FIN, ACK o RST) o, en un caso, una expiración de temporizador del doble del tiempo de vida máximo del paquete.

La acción es el envío de un segmento de control (SYN, FIN
o RST), o nada, indicado por —. Los comentarios aparecen entre paréntesis.

Liberación y Establecimiento de una conexión

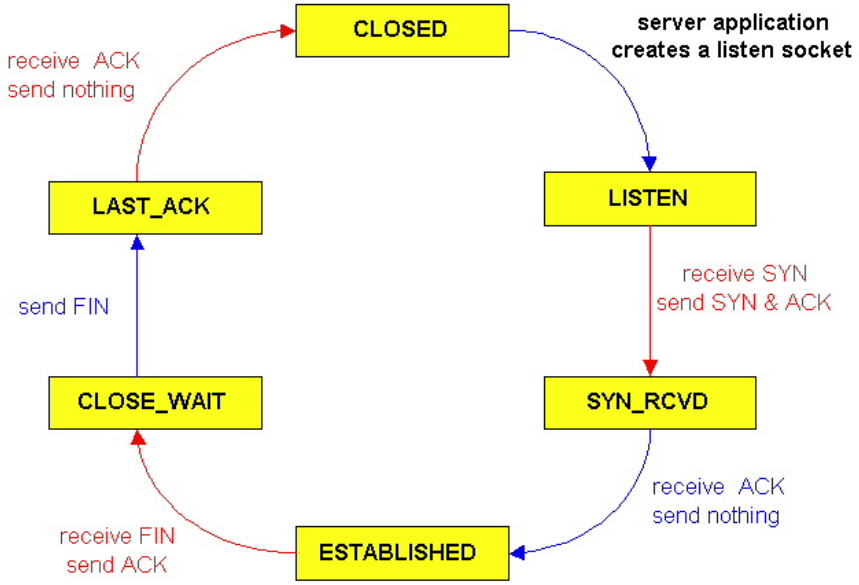


TCP: Finite State Machine



TCP client lifecycle

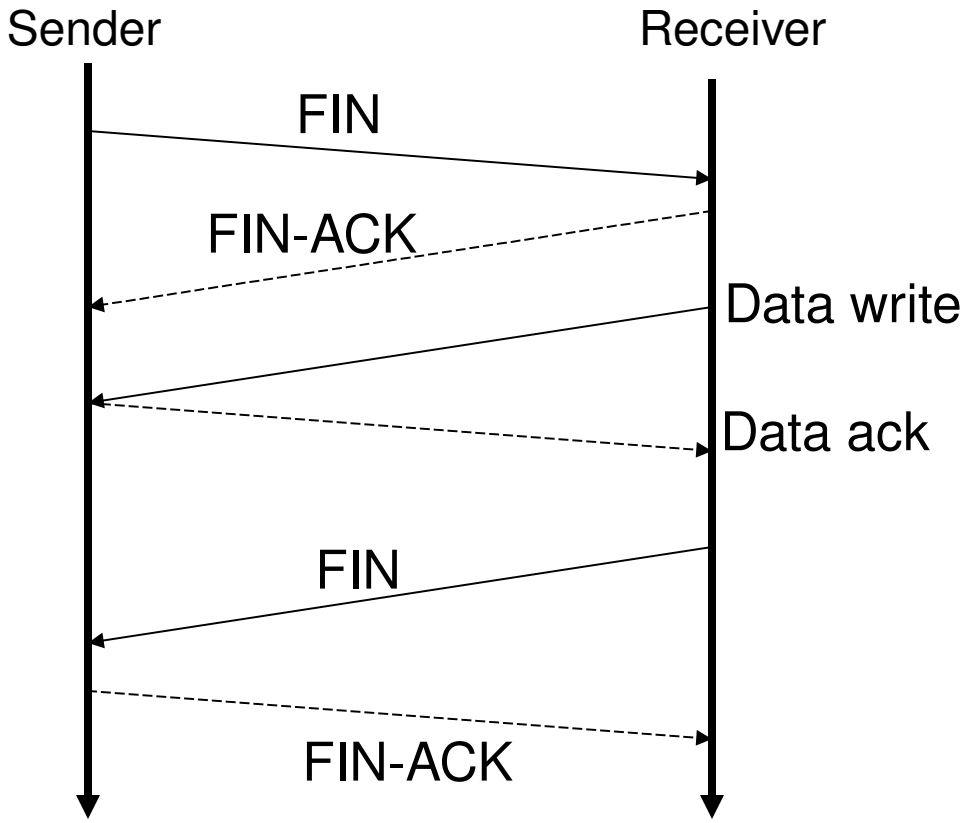
TCP server lifecycle



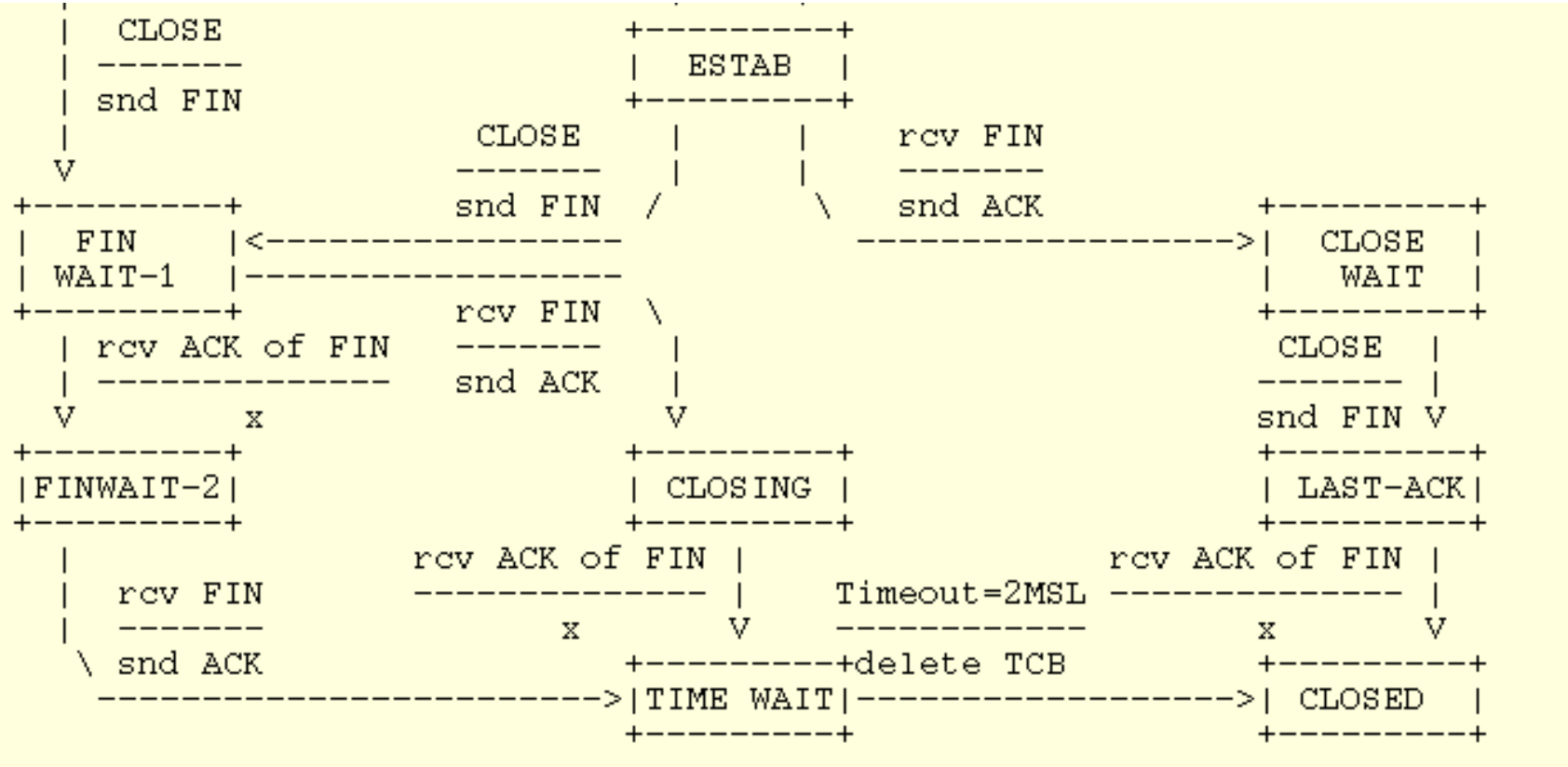
Estados del Modelo TCP

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

Liberación “Tear-down”



Conexión “Tear-down”



Detectando “Half-open Connections”

```
    TCP A                                                TCP B

1.  (CRASH)                                  (send 300,receive 100)

2.  CLOSED                                            ESTABLISHED

3.  SYN-SENT --> <SEQ=400><CTL=SYN>                --> (??)

4.  (!!)     <-- <SEQ=300><ACK=100><CTL=ACK>     <-- ESTABLISHED

5.  SYN-SENT --> <SEQ=100><CTL=RST>                --> (Abort!!)

6.  SYN-SENT                                          CLOSED

7.  SYN-SENT --> <SEQ=400><CTL=SYN>                -->
```

TIME-WAIT Assassination

```
     TCP A                                                TCP B

 1.  ESTABLISHED                                          ESTABLISHED

     (Close)
 2.  FIN-WAIT-1  --> <SEQ=100><ACK=300><CTL=FIN,ACK>  --> CLOSE-WAIT

 3.  FIN-WAIT-2  <-- <SEQ=300><ACK=101><CTL=ACK>      <-- CLOSE-WAIT

                                                          (Close)
 4.  TIME-WAIT   <-- <SEQ=300><ACK=101><CTL=FIN,ACK>  <-- LAST-ACK

 5.  TIME-WAIT   --> <SEQ=101><ACK=301><CTL=ACK>      --> CLOSED

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

 5.1. TIME-WAIT   <--  <SEQ=255><ACK=33> ... old duplicate

 5.2  TIME-WAIT   --> <SEQ=101><ACK=301><CTL=ACK>    -->  ????

 5.3  CLOSED      <-- <SEQ=301><CTL=RST>             <--  ????
    (prematurely)
```

TCP

Algoritmo de Nagle

Silly Window

Cuando las app leen y/o escriben de forma “ineficiente”

- Envío de pocos bytes
  - Considerar una conexión telnet a un editor en el servidor que reacciona con cada letra tipeada (echo) , al llegar el carácter a la entidad TCP emisor , se crea un segmento de 21 bytes , a IP 41 bytes , del lado receptor un segmento TCP , mas IP 40 bytes de ACK .
  - El edito leyó el carácter envía una actualización de ventana de 40 bytes por
  - Una vez que lo proceso echo del carácter , otros 41 bytes
  - 162 Bytes y cuatro segmentos por cada letra tipeada
- Algunas soluciones del lado del receptor: retardar el ACK y de las actualizaciones de ventana
  - Sin embargo cuando envío paquetes con un Byte sigue siendo ineficiente

Algoritmo de Nagle (1984)

- Cuando la aplicación tiene datos para enviar
  - If el tamaño de la ventana y los datos disponibles en el buffer >= MSS
    - Enviar un segmento full
  - Else
    - If los datos en vuelo están sin reconocer
      bufferear el nuevo dato hasta que llegue el ACK
      Else
      enviar todo los datos nuevos ahora

TCP: persist timer

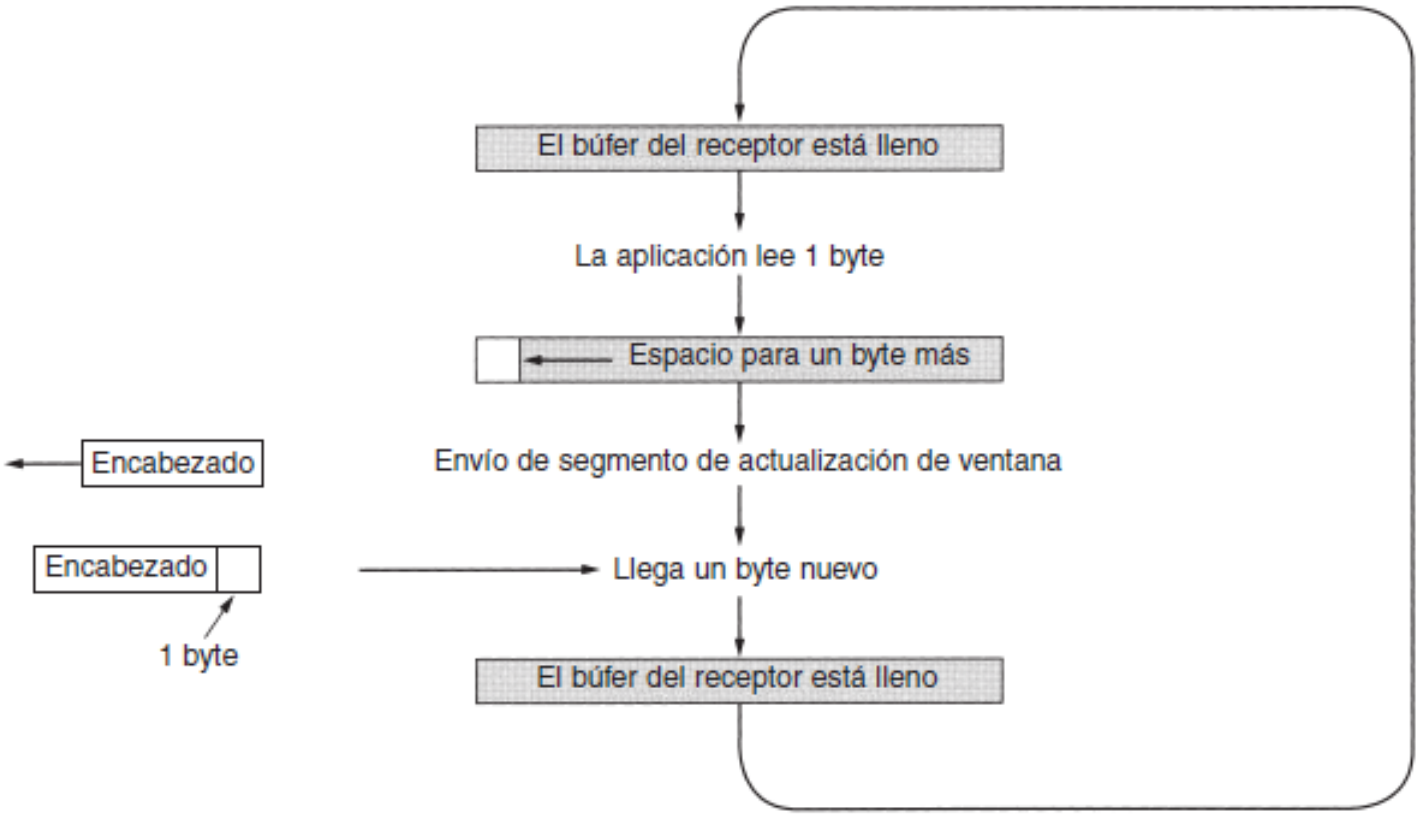
- Cuando la aplicación lee despacio
  - si la app receptora lee muy despacio el buffer se puede llenar y anuncia W=0 tamaño de la ventana
  - Cuando la ventana vuelve a aumentar el receptor envia un segmento ACK con la nueva ventana
  - Que pasa si se pierde ?Else
    - el emisor no puede enviar datos
    - El receptor no sabe que el ACK se perdió

Solución : TCP envía periódicamente windows probes con 1 byte de dato

Síndrome de Ventana estúpida (Silly)

- Se envían datos en bloques y el receptor su app lee muy despacio , por ejemplo un 1 byte a la vez
  - Tomemos el siguiente ejemplo donde inicialmente el buffer del rx esta lleno y el emisor lo sabe ya que recibió w=0

Silly Window Syndrome (*)



"El búfer ahora está lleno, por lo que el receptor confirma la recepción del segmento de 1 byte pero establece la ventana en 0. Este comportamiento puede continuar indefinidamente"

Solución de Clark

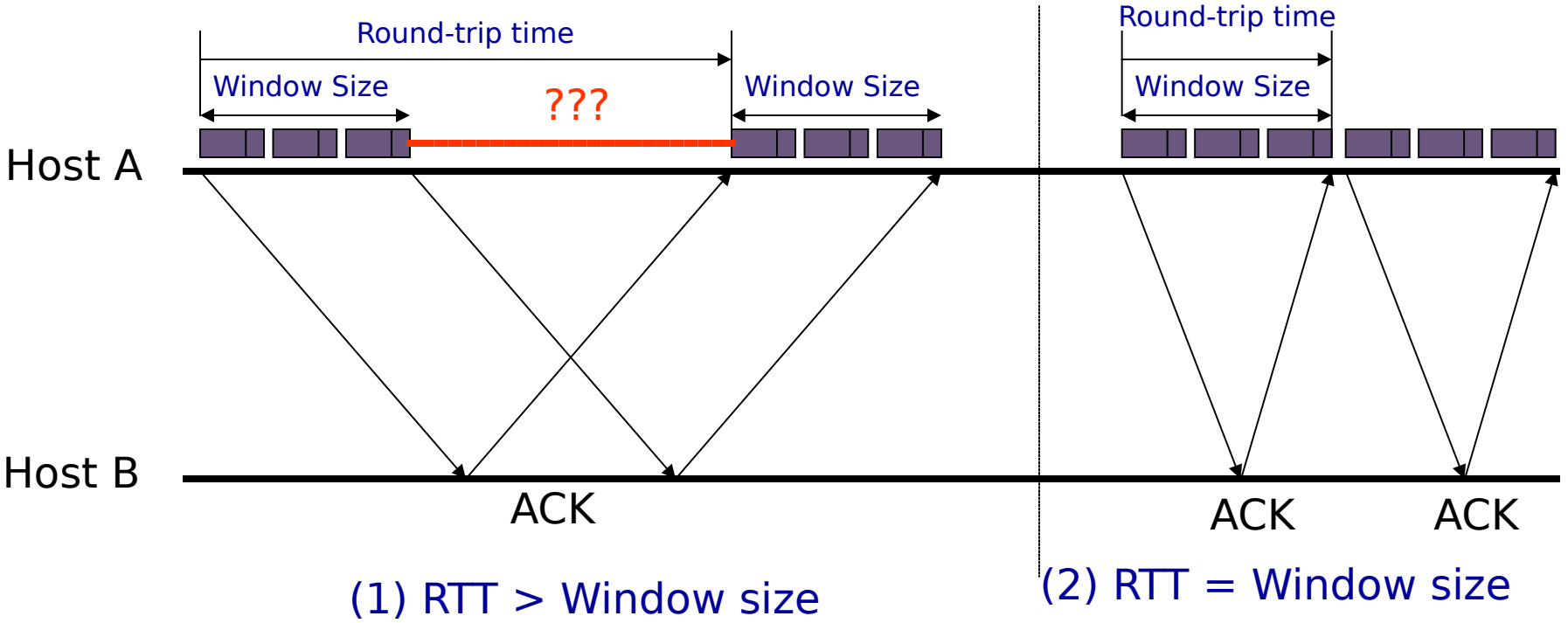
- No enviar aviso de ventana para 1 Byte
- Esperar hasta tener una cantidad de buffer considerable
  - Buffer = MSS que se anuncio en la conexión
  - O a la mitad de la capacidad del buffer , el valor mas pequeño de los dos

TCP

Uso del ancho de banda

Sockets

TCP Sliding Window



Mantención del “caño” lleno

- **AdvertisedWindow de 16 bits**

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

Berkeley Sockets

primitivas de los “sockets” TCP.

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection