UNCUYO
UNIVERSIDAD NACIONAL DE CUYO
FACULTAD DE INGENIERÍA
Licenciatura en Ciencias de la Computación

Redes de Computadoras

Unidad 4

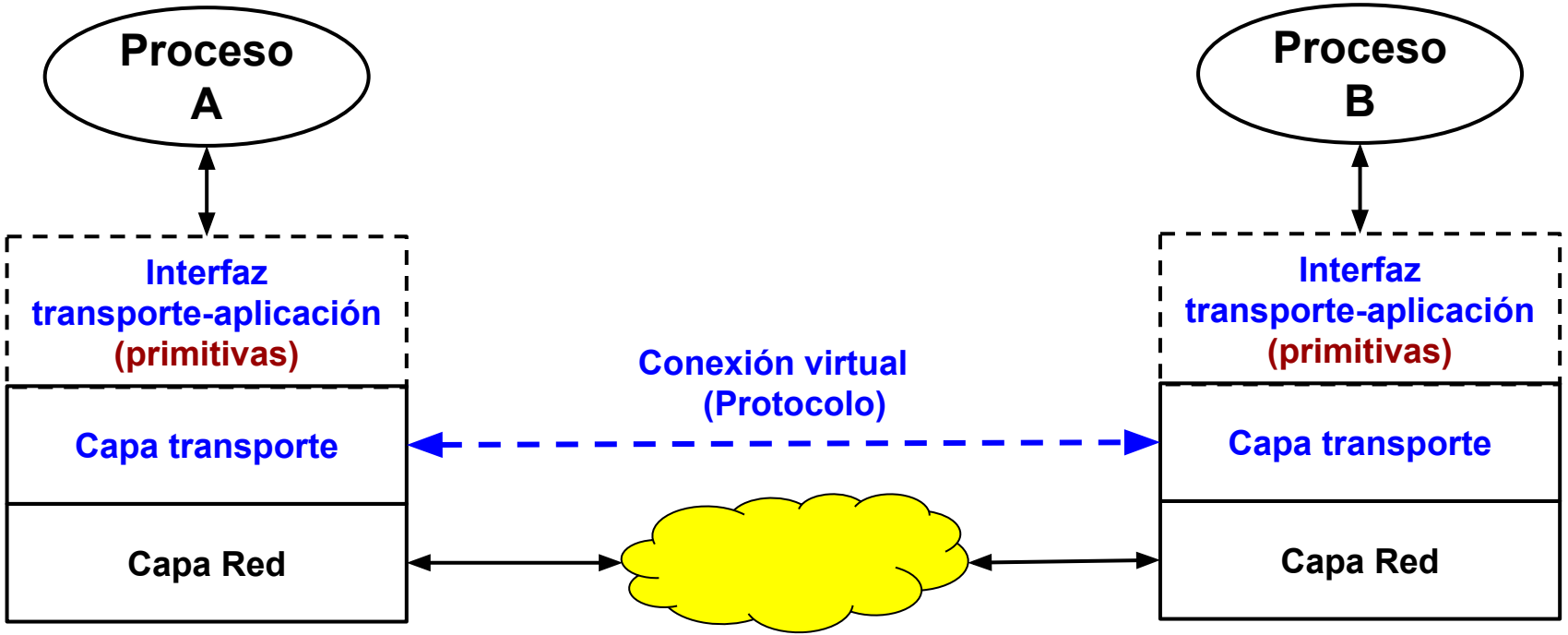
Capa de Transporte

Dr. Ing. Pablo Daniel Godoy (pablo.godoy@ingenieria.uncuyo.edu.ar)

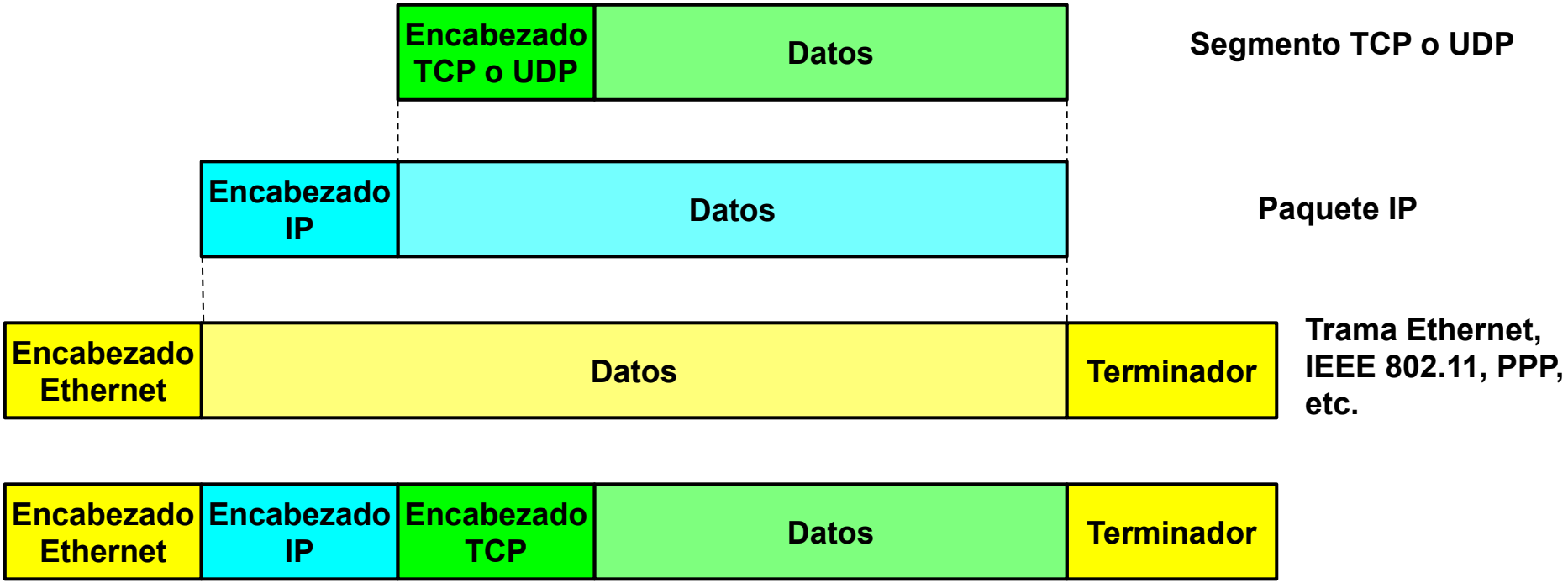
Capa de Transporte

- Servicios: Transporte de datos desde un **proceso** en la máquina origen hacia un **proceso** en la máquina destino.
- Dos tipos diferentes de servicios:
  - Orientado a conexión: **TPC** (Transmission Control Protocol)
    - Entrega confiable (con confirmación de recepción)
    - Control de flujo.
  - Sin conexión: **UDP** (User Datagram Protocol)

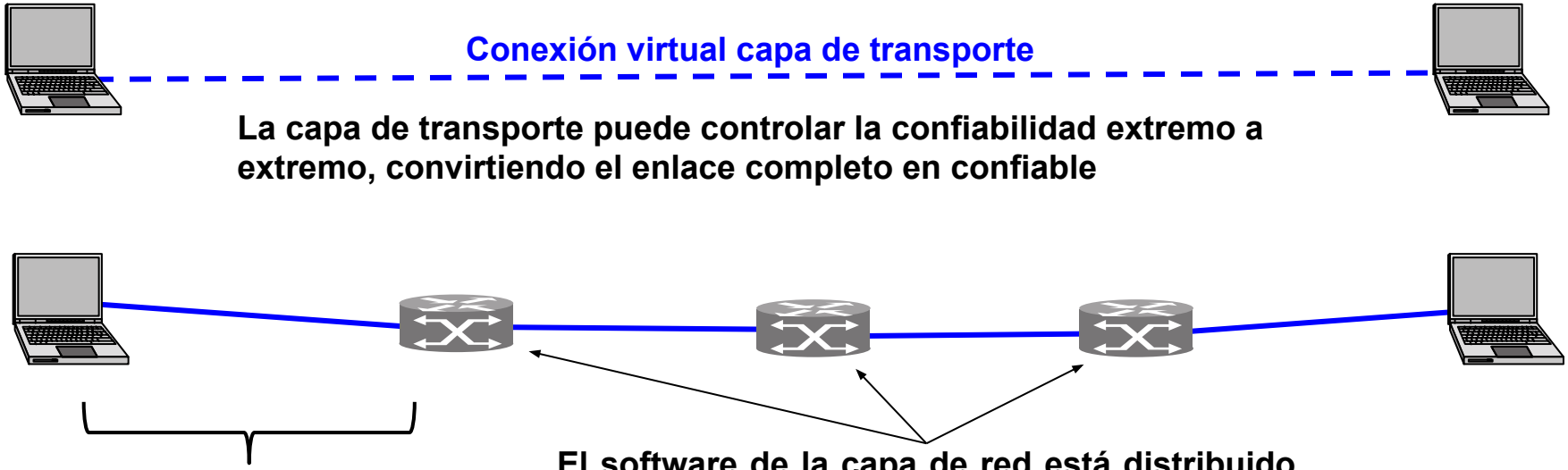
Capa de transporte



Encapsulamiento



Confiabilidad en las capas de enlace y transporte



La capa de transporte puede controlar la confiabilidad extremo a extremo, convirtiendo el enlace completo en confiable

La capa de enlace de la máquina del usuario solo puede controlar la confiabilidad en este enlace

El software de la capa de red está distribuido entre las máquinas de usuario y routers. Las máquinas de usuarios no tiene control sobre el buen funcionamiento de los elementos de la capa de red.

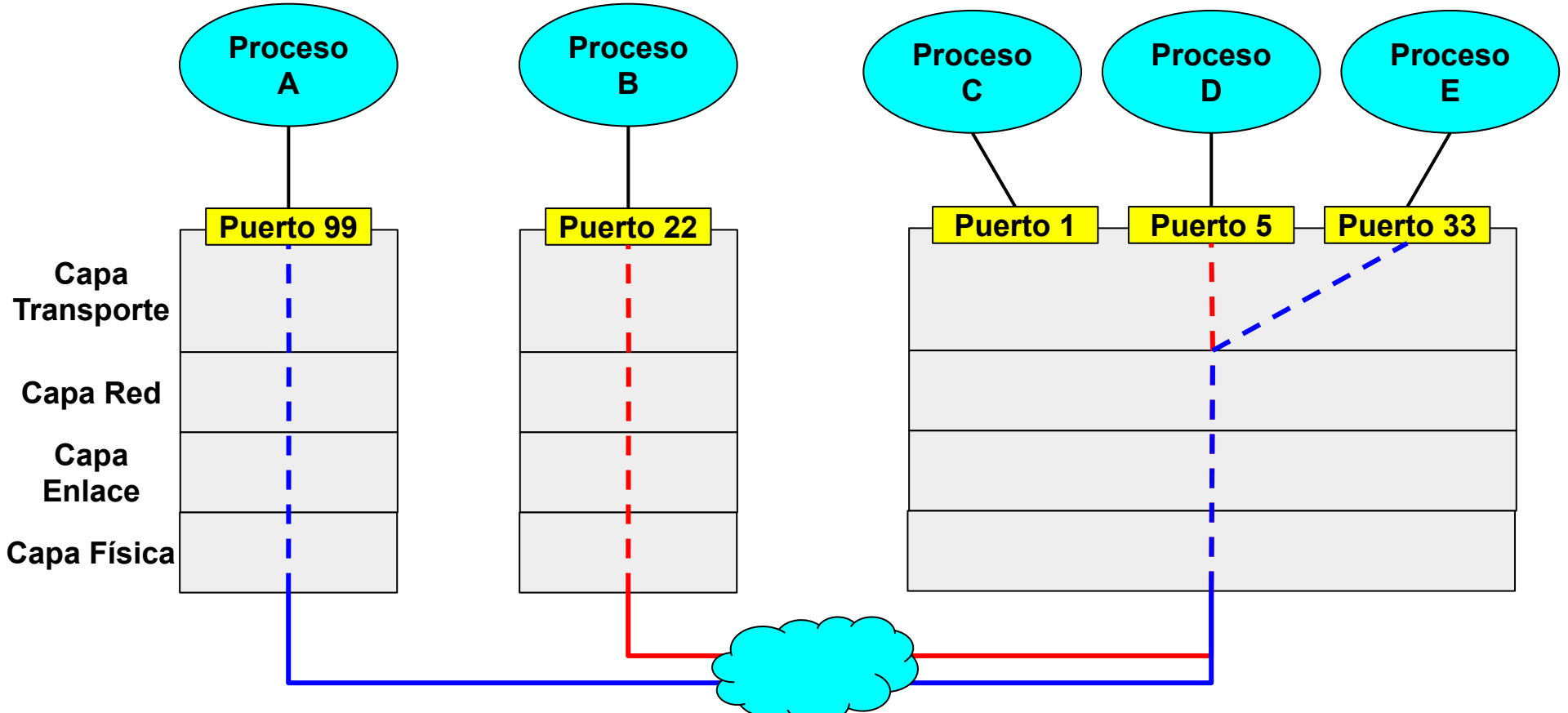
Diferencias entre servicios orientados a conexión en la capa de enlace y en la capa de transporte

	Capa enlace (punto a punto, ej. PPP)	Capa transporte
Elementos entre las máquinas a conectar	Un enlace (cable)	Internet (redes)
Direccionamiento	No necesario	Obligatorio
Dificultad en el establecimiento de la conexión	Simple	Complejo
Posibles problemas con los “paquetes”	Perderse o duplicarse	Perderse, duplicarse, demorarse o llegar fuera de orden
Latencias	Pequeña	Muy variable. Pequeña o grande

Esquema de direccionamiento capa transporte

- El direccionamiento es mediante **Puertos** (también conocido como TSAP o Transport Service Access Point).
- Todo proceso que desee utilizar la pila de protocolos TCP/IP necesitará un **número de puerto**, para distinguirse de otros procesos en la misma máquina.
- Un proceso A que quiere conectarse a un proceso B, necesita conocer la **dirección IP** (en qué máquina está), y su **número de puerto**.
- Puertos reservados: para aplicaciones típicas. Ejemplos:
  - Puerto TCP o UDP 80 para servidor http. 443 para https.
  - Puerto TCP 22: ssh
  - Puerto TCP 25: Correo electrónico.
  - Puerto TCP 43: Whois.
  - Puerto TCP 631: CUPS (sistema impresión de Unix)
  - Puerto 5400: VNC
  - Puerto 6881: BitTorrent
  - En Linux, ver “/etc/services” para más ejemplos.

Esquema de direccionamiento TCP



Esquema de direccionamiento Capa transporte

- ¿Cómo obtiene un proceso el N° de puerto del proceso al cual se quiere conectar?
  - Conoce su puerto porque está reservado.
  - Están de acuerdo. Complicado: hay cientos de aplicaciones compiten por los puertos, y no siempre puede asegurarse disponibilidad.
- ¿Los procesos que escuchan conexiones deben estar siempre activos (y consumiendo recursos todo el tiempo)?
  - **Servidor de procesos**: Escucha en varios puertos y si en alguno hay un pedido de conexión, genera o invoca el proceso asociado al puerto.
    - Ejemplo: **demonio inetd** en Unix o servidor de Internet. Escucha en varios puertos asociados a servicios de Internet.
      - Lanza los procesos asociados a cuando llegan peticiones.
      - Atiende procesos simples.

Esquema de direccionamiento TCP: servidor de procesos

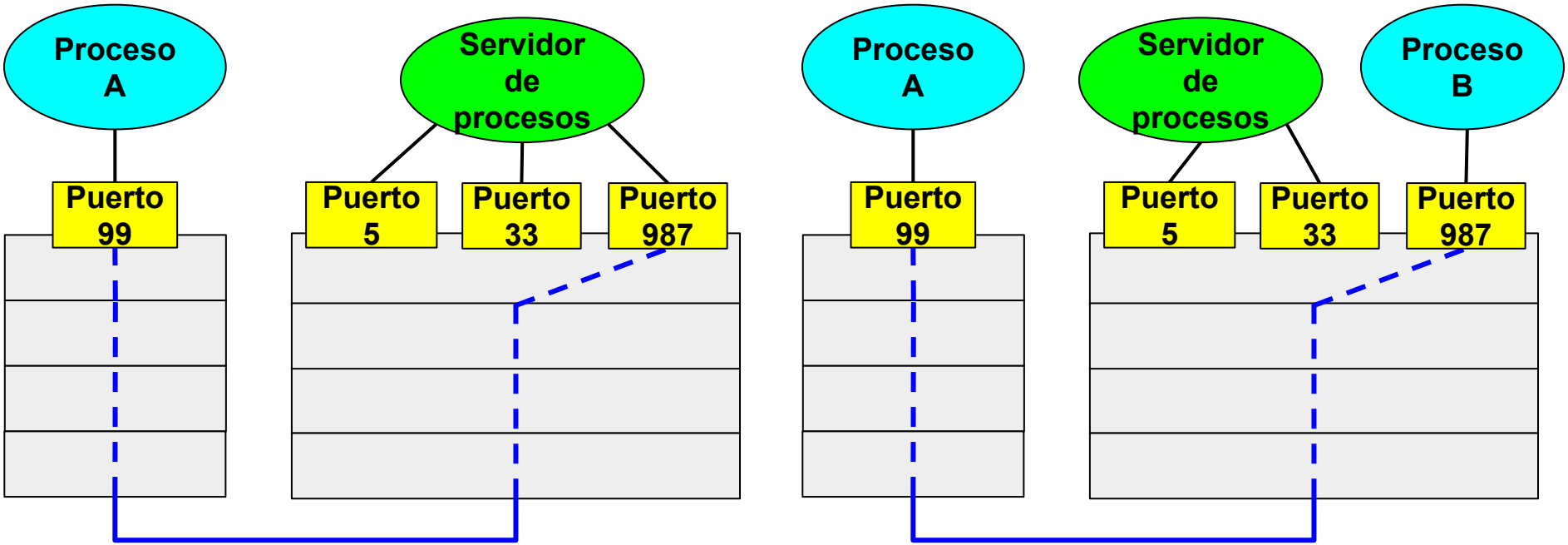


Figura obtenida de: A. Tanenbaum, D. Wetherall, "Redes de computadoras", Quinta edición (2012), pag. 439

Protocolo UDP (User Datagram Protocol)

- Servicio **sin conexión**. RFC 768.
- Permite transportar paquetes IP encapsulados, multiplexarlos y demultiplexarlos entre diferentes procesos.
- Puertos origen y destino: identifican a los procesos origen y destino
- Longitud UDP: Longitud total (encabezado y datos).

Encabezado UDP

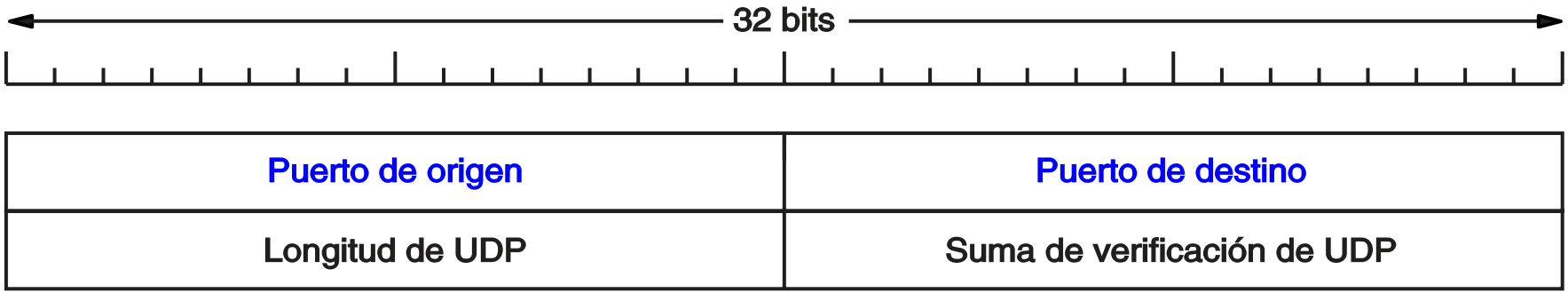
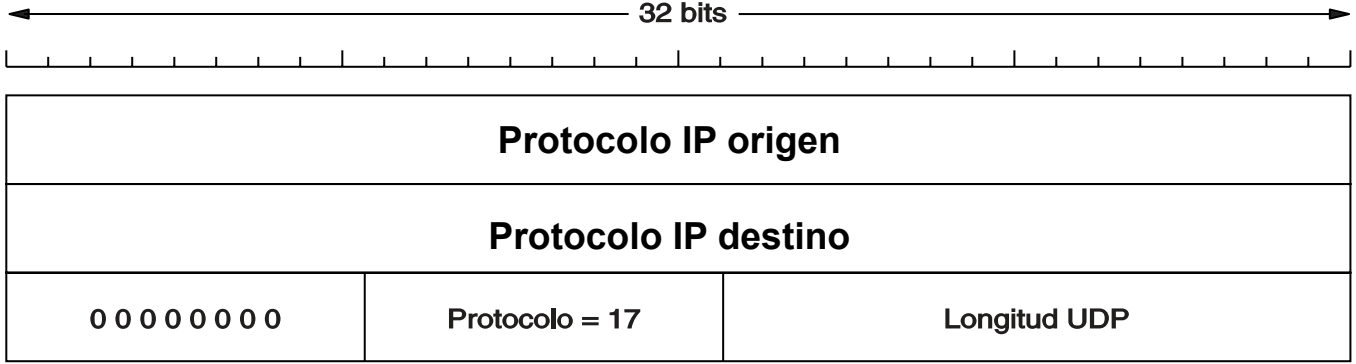


Figura obtenida de: A. Tanenbaum, D. Wetherall, "Redes de computadoras", Quinta edición (2012), pag. 465

Encabezado UDP

- Suma verificación: Incluye en la suma de verificación: el encabezado UDP, los datos y un “encabezado IP conceptual” o “pseudo-encabezado” [1].
  - Realiza una suma de palabras de 16 bits en complemento a 1, y saca el complemento a 1 (cuando el receptor realice el cálculo, debe dar 0).
  - Si se detecta un error, el segmento se descarta, sin tomar otra medida.



[1] El pseudo-encabezado no es parte de los datos, ni del paquete UDP, ni del IP. Solo se utiliza para el cálculo de la suma de verificación.

Figura obtenida de: A. Tanenbaum, D. Wetherall, "Redes de computadoras", Quinta edición (2012), pag. 466

Protocolo UDP (Use Datagram Protocol)

- ¿Que hace UDP?
  - **Multiplexar y demultiplexar paquetes IP entre procesos**.
  - Detección de errores opcional.
- Aplicaciones UDP:
  - Aplicaciones que intercambien solo pocos paquetes (DNS, RPC)
  - Aplicaciones donde paquetes retransmitidos no tienen valor (Voz).

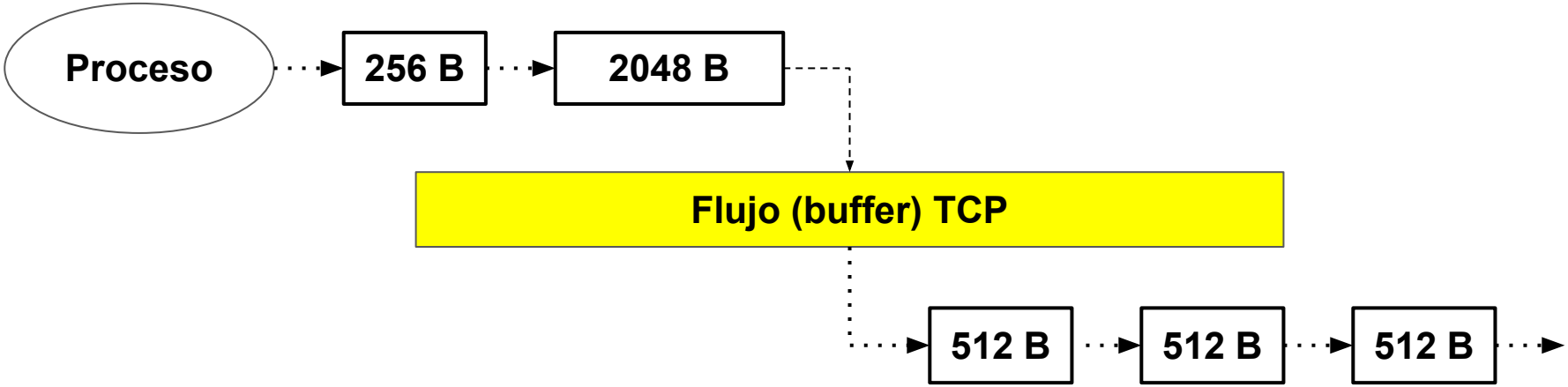
Nota: Los sniffers muestran un campo “Stream Index”. El mismo no es parte del encabezado UDP ni TCP. Los sniffers lo utilizan para distinguir “conversaciones”. A cada par IP-Puerto origen y IP-Puerto destino le asigna un “Stream Index” diferente que permite seguir conversaciones. En Wireshark se puede usar el filtro “udp.stream”

Protocolo TCP

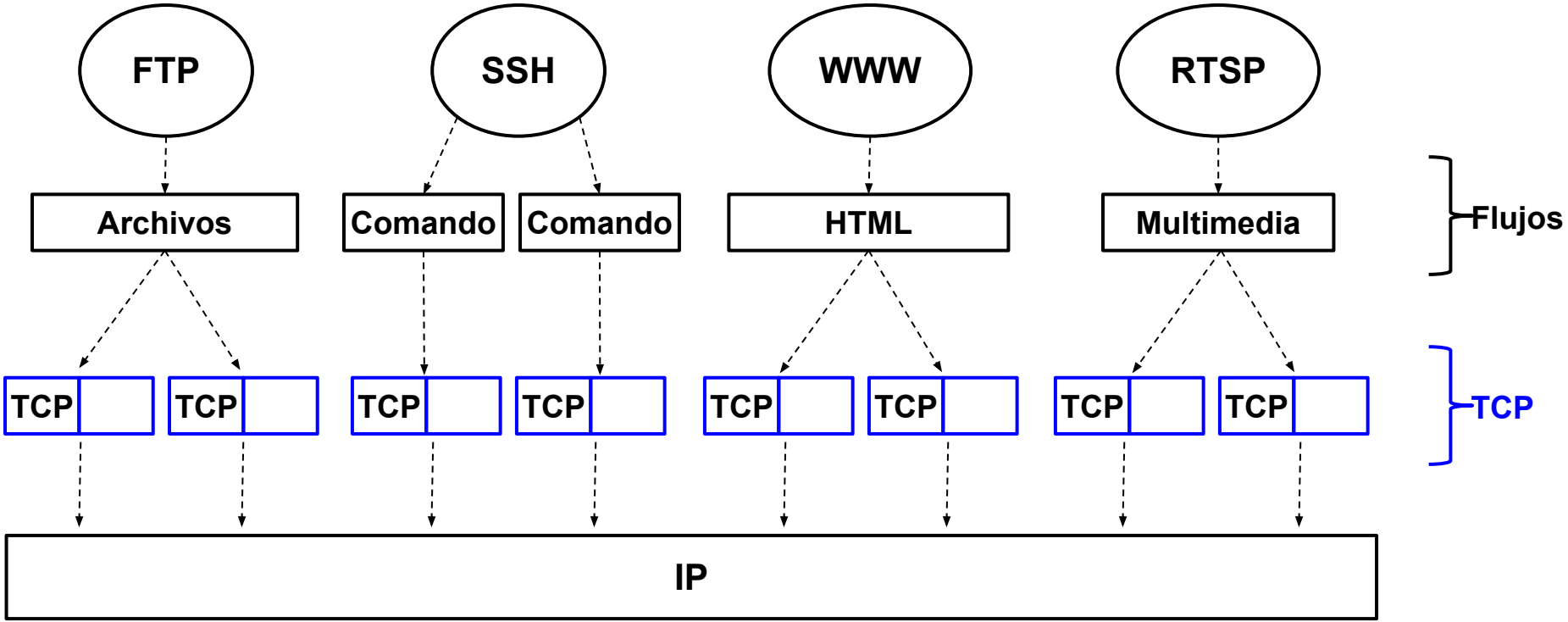
- Servicio **orientado a conexión**, **confiable** (control de errores, confirmaciones de recepción y retransmisión), **control de flujo extremo a extremo** y **control de congestión**. **Full duplex**. **No soporta multidifusión o difusión**.
- Primera RFC 793. Actualmente hay una gran colección de RFCs dedicadas a TCP (la RFC 4614 es una Roadmap o guía de las RFCs dedicadas a TCP).
- Objetivos:
  - Proporcionar un **flujo de bytes** confiable extremo a extremo sobre una internet no confiable.
  - Adaptarse **dinámicamente** a diferentes topologías, anchos de banda, retardos, tamaños de paquete, etc. de las redes que componen la internet.
- Implementación:
  - Proceso de usuario (primera implementación).
  - Biblioteca de algún lenguaje o biblioteca común (posteriores implementaciones).
  - Parte de Kernel (implementaciones actuales).

Protocolo TCP - Flujo de bytes

- Una conexión TCP es un **flujo de bytes**, no de mensajes.
  - El proceso puede escribir datos de cualquier longitud en el flujo TCP, y TCP puede tomar fragmentos de cualquier longitud para enviar (iguales o no).
  - Los **límites** de los mensajes escritos por el proceso emisor **no se preservan**.
  - Los **ACKs** están en función de **bytes**, no de segmentos.



Protocolo TCP: flujo de bytes



Protocolo TCP

- El proceso **puede decidir** si almacenar los datos en el buffer o enviarlos sin demora (aplicaciones interactivas pueden requerir el segundo caso).
- TCP decide que tan grandes deben ser los segmentos.
- El tamaño máximo del segmento TCP está limitado por:
  - La MTU de la ruta (se evita la fragmentación ya que causa degradación).
  - Tamaño máximo de paquetes del protocolo IP.

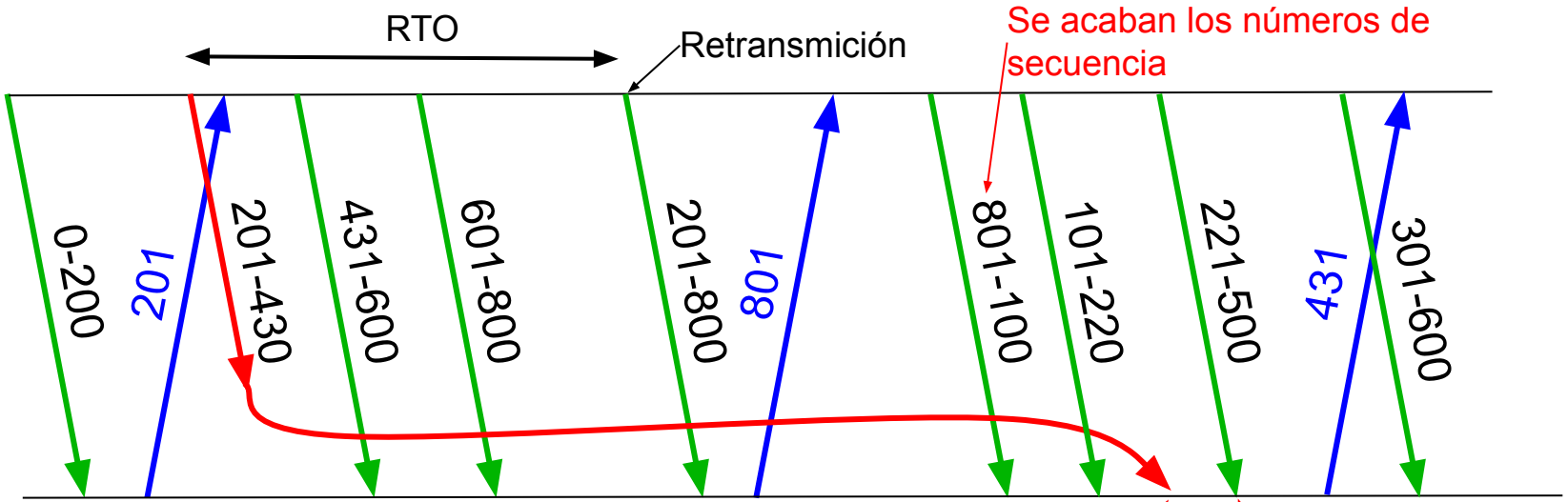
Números de secuencias

	Capa Enlace	IPv4	IPv6	TCP
Se asignan a:	Cada trama	A cada paquete	No se usa	**Cada Byte**
Objetivo	Que el receptor detecte tramas repetidas.	Asociar fragmentos a paquetes	No se usa	Asociar ACKs a bytes enviados. Detectar bytes repetidos Ordenar bytes

- **Problema 1: Los retardos pueden ser muy grandes y muy variables**.
  - La cantidad de números de secuencia debe ser suficientemente grandes (**muchos bits para número de secuencia**), para que cuando un número de secuencia k se reutilice, los paquetes y ACKs anteriores con número de secuencia k ya se hayan extinguido.

Problema de cantidad de número de secuencias pequeño.

Ejemplo: se supone un sistema cuyo número de secuencia máximo es 999



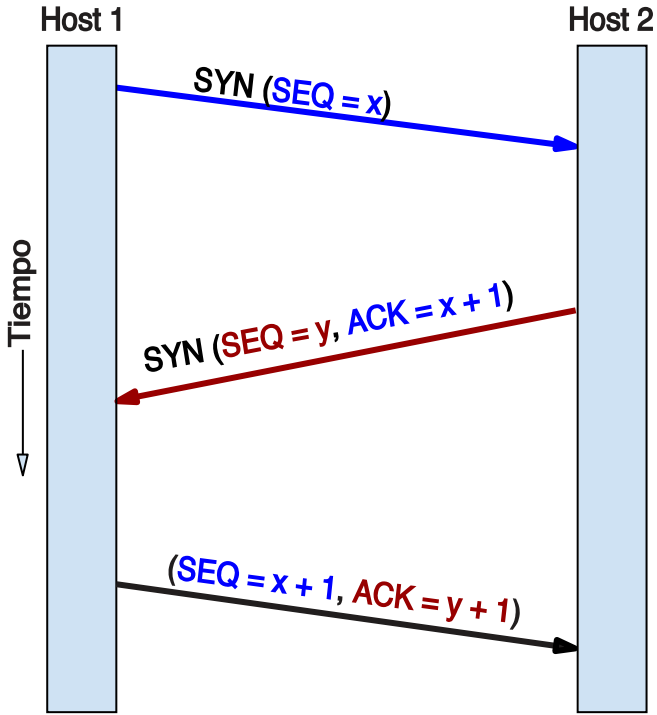
Introducción a la Interfaz Sockets

- Conjunto de primitivas que son ejecutadas por aplicaciones en la capa de aplicación para acceder a los servicios de la capa de transporte.

Algunas primitivas de la Interfaz socket usadas en aplicaciones cliente

- **Connect**: Comienza el procedimiento de conexión. La ejecuta el cliente.
- **Listen**: Anuncia la disposición para aceptar conexiones (asigna espacio de memoria para almacenar peticiones de conexión). La ejecuta el servidor
- **Accept**: Queda en espera de conexiones entrantes. La ejecuta el servidor.
- **Send** y **Receive**: Enviar y recibir datos a través de una conexión ya establecida.
- **Close**: Se libera la conexión (debe liberarse en ambos lados).

Protocolo TCP - Establecimiento normal de una conexión

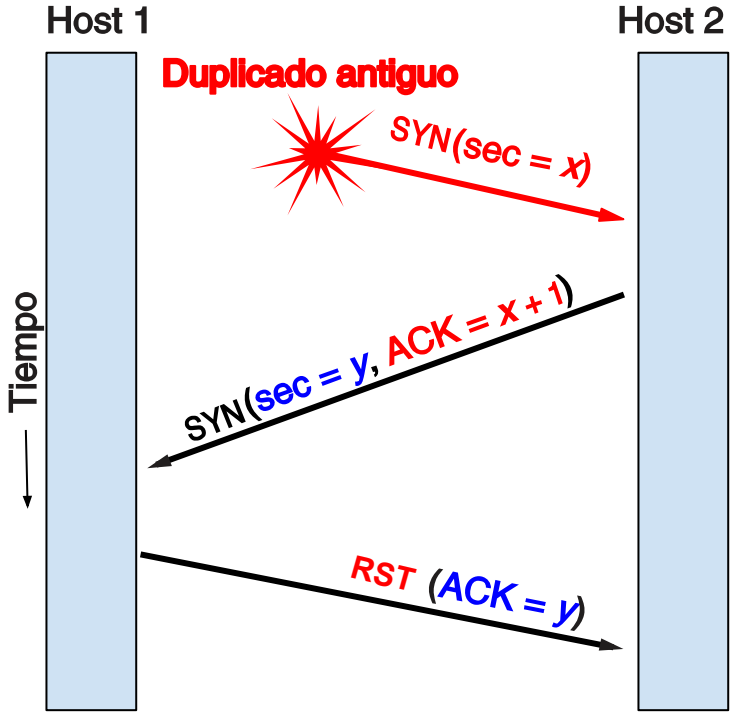


Establecimiento correcto de una conexión

- Las conexiones son full duplex, formadas por dos conexiones half duplex.
- El Host 1 envía una petición de conexión (mensaje SYN) indicando su número de secuencia inicial.
- El Host 2 reconoce la petición del Host 1, indicando su propio número de secuencia inicial (mensaje SYN).
- El Host 1 reconoce el mensaje del Host 2 y su número de secuencia inicial.

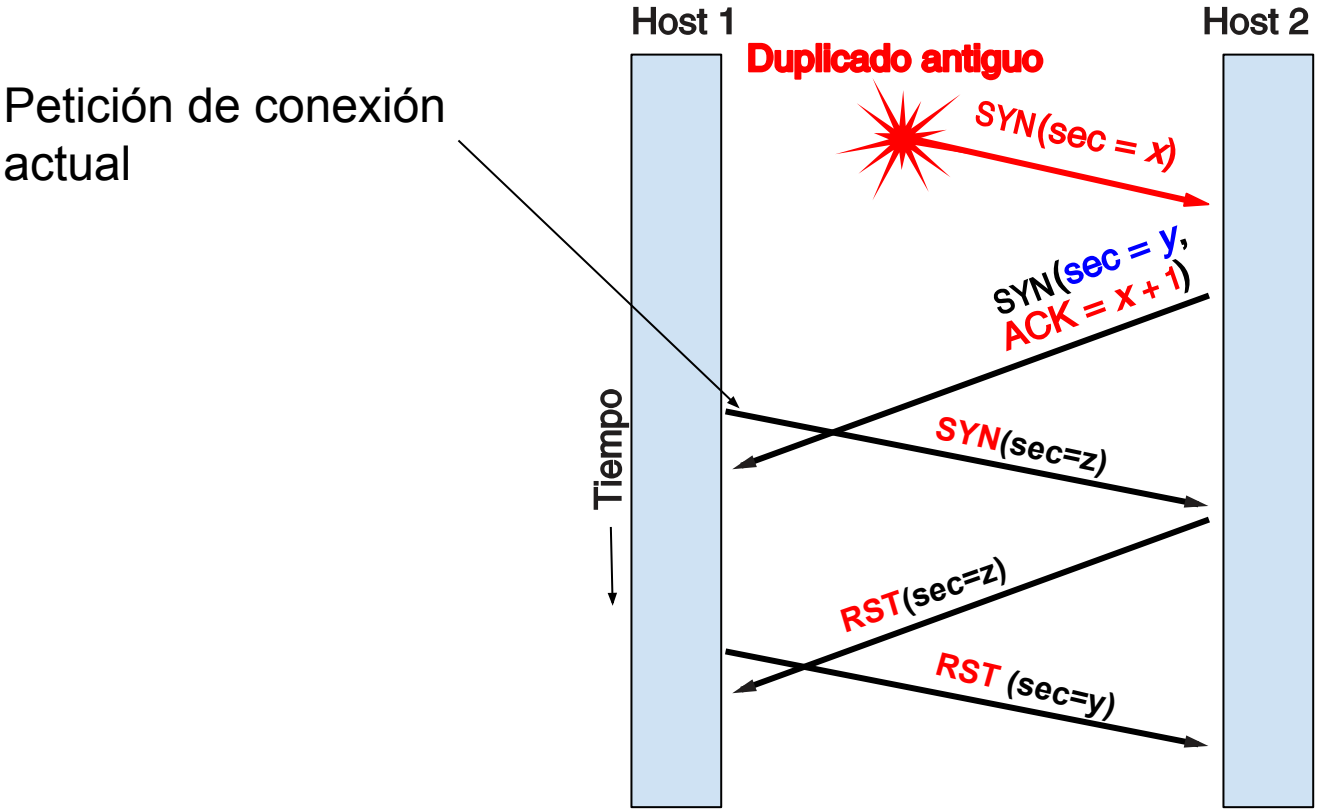
Inicio de una conexión: Acuerdo de 3 vías - situaciones no normales
Problemas con los SYN viejos

- El emisor podría enviar una petición de conexión (mensaje SYN), que **se pierde**.
- Luego de un tiempo, intenta de nuevo enviando un nuevo mensaje SYN, realizando una conexión exitosa, transfiere datos y cierra la conexión.
- Luego, el primer SYN llega retrasado, y el receptor podría iniciar una conexión ya descartada por el transmisor.

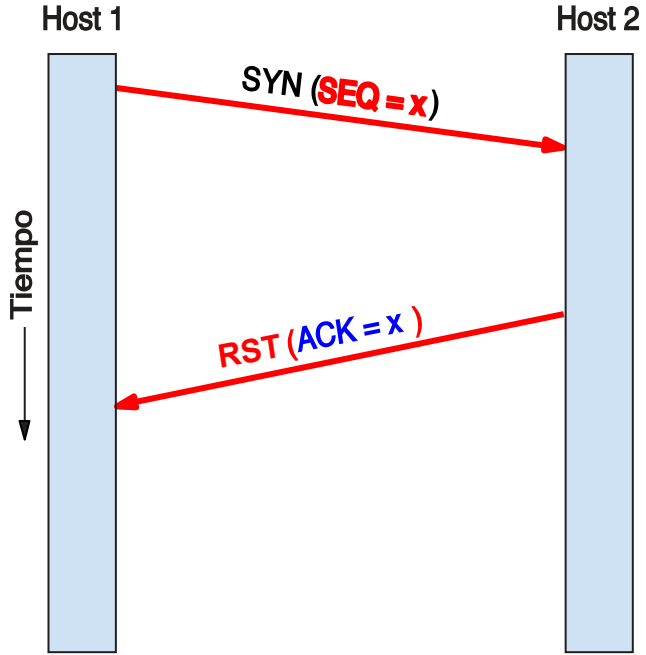


Inicio de una conexión: Acuerdo de 3 vías - situaciones no normales

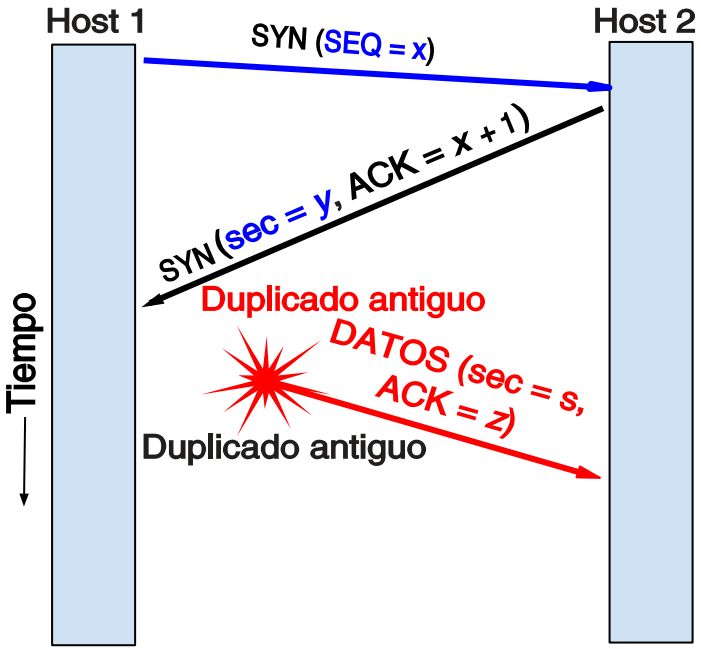
Problemas con los SYN viejos



Inicio de una conexión: Acuerdo de 3 vías - situaciones no normales



Un extremo ejecuta un Connect, pero el otro extremo no ha ejecutado un Acept

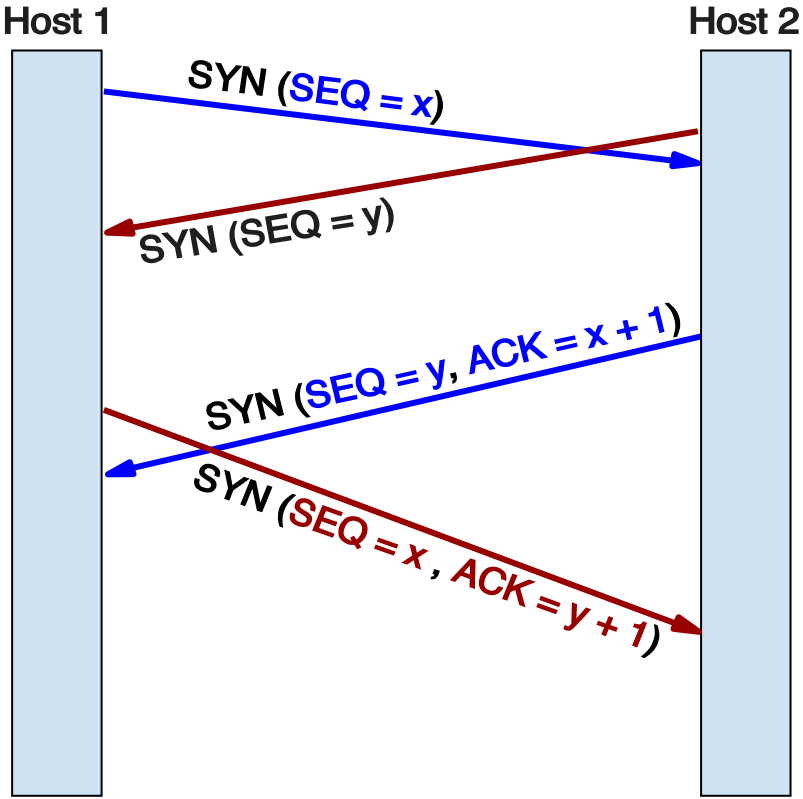


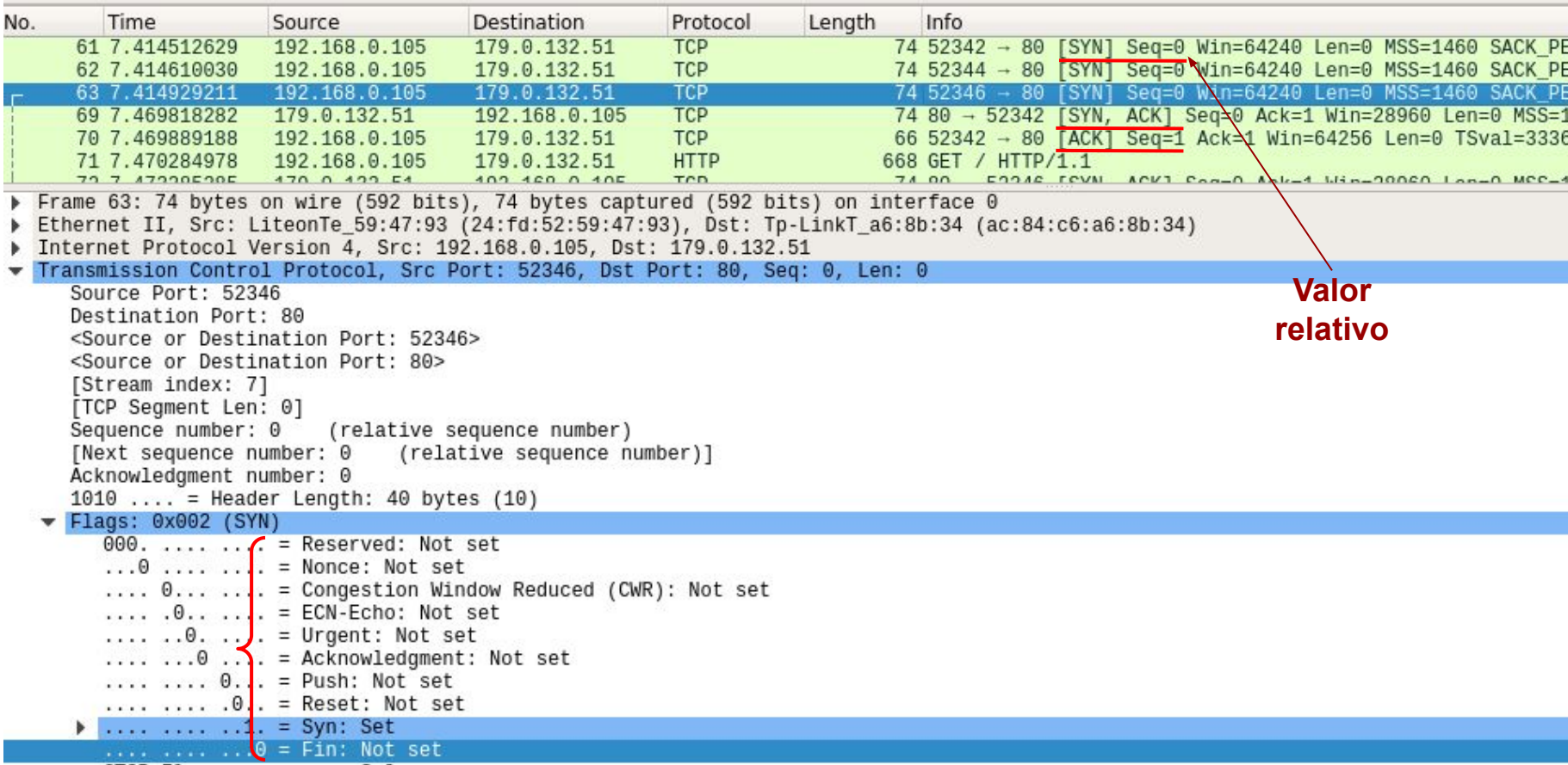
Llegan datos de una conexión anterior o antiguos. Serán rechazados por no seguir la secuencia

Figura obtenida de: A. Tanenbaum, D. Wetherall, "Redes de computadoras", Quinta edición (2012), pag. 443

Protocolo TCP - Establecimiento de una conexión

Ambos extremos intentan comenzar una conexión al mismo tiempo. Se establece una conexión con números de secuencia correcto y sin ambigüedades

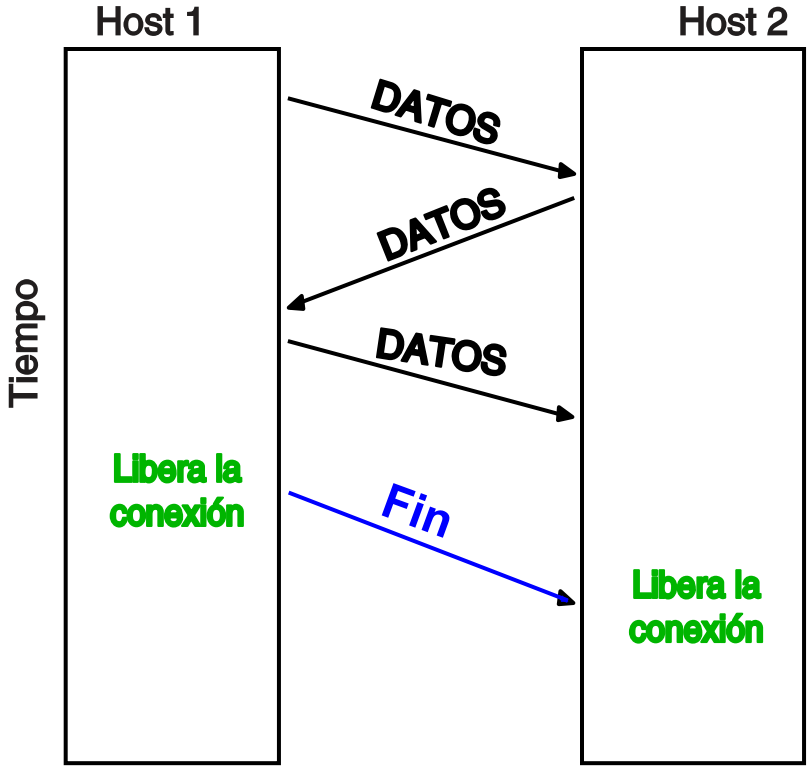



No. Time Source Destination Protocol Length Info
61 7.414512629 192.168.0.105 179.0.132.51 TCP 74 52342 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PE
62 7.414610030 192.168.0.105 179.0.132.51 TCP 74 52344 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PE
63 7.414929211 192.168.0.105 179.0.132.51 TCP 74 52346 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PE
69 7.469818282 179.0.132.51 192.168.0.105 TCP 74 80 → 52342 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1
70 7.469889188 192.168.0.105 179.0.132.51 TCP 66 52342 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3336
71 7.470284978 192.168.0.105 179.0.132.51 HTTP 668 GET / HTTP/1.1
Frame 63: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
Ethernet II, Src: LiteonTe_59:47:93 (24:fd:52:59:47:93), Dst: Tp-LinkT_a6:8b:34 (ac:84:c6:a6:8b:34)
Internet Protocol Version 4, Src: 192.168.0.105, Dst: 179.0.132.51
Transmission Control Protocol, Src Port: 52346, Dst Port: 80, Seq: 0, Len: 0
Source Port: 52346
Destination Port: 80
<Source or Destination Port: 52346>
<Source or Destination Port: 80>
[Stream index: 7]
[TCP Segment Len: 0]
Sequence number: 0 (relative sequence number)
[Next sequence number: 0 (relative sequence number)]
Acknowledgment number: 0
1010 .... = Header Length: 40 bytes (10)
Flags: 0x002 (SYN)
000. .... = Reserved: Not set
...0 .... = Nonce: Not set
.... 0... .... = Congestion Window Reduced (CWR): Not set
.... .0.. .... = ECN-Echo: Not set
.... ..0. .... = Urgent: Not set
....0 .... = Acknowledgment: Not set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
....1. = Syn: Set
....0 = Fin: Not set
Valor relativo

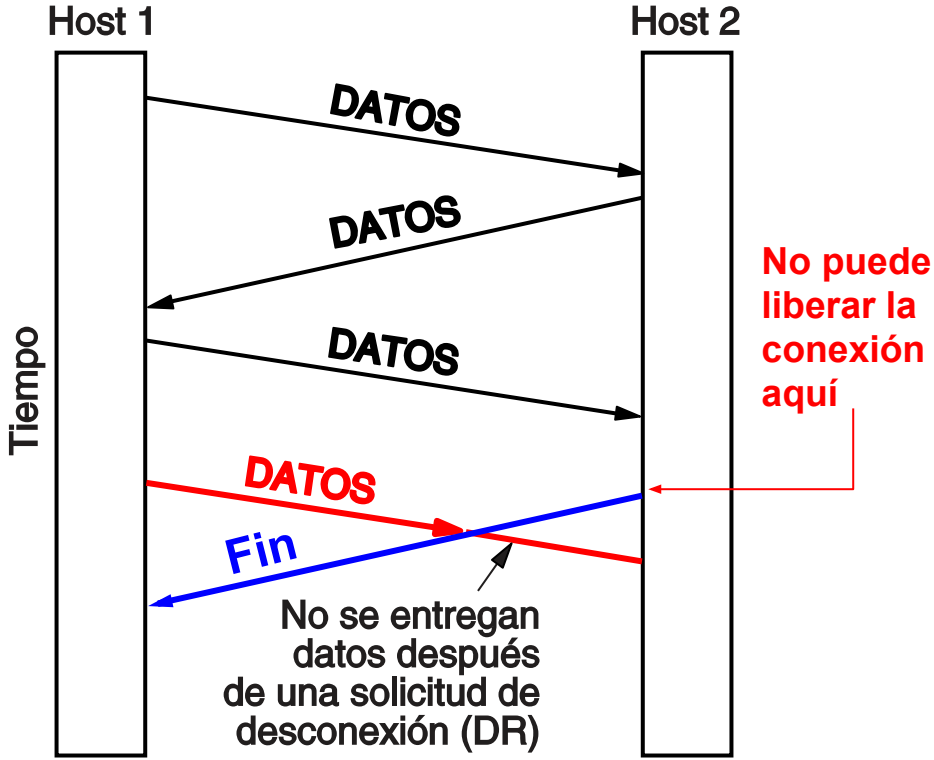
Mecanismos de liberación de una conexión

- **Asimétrica**: La conexión se interrumpe si **una de las partes** libera la conexión.
  - Natural en el sistema telefónico.
  - **En redes de datos, funciona cuando cada host conoce con exactitud la cantidad de datos a enviar y recibir (servidores de datos)**. **Poco usual**.
  - **En una red, puede provocar pérdida de datos.**
- **Simétrica**: **Ambos** deben liberar la conexión.
  - Usado en TCP.
    - El proceso ejecuta una primitiva **CLOSE** y TCP envía un segmento **FIN**.
  - Trata la conexión como dos conexiones simplex.
  - **Ambos deben asegurarse que el otro liberó o liberará la su extremo de la conexión antes de liberar el propio.**
  - **Problema: conexiones semiabiertas si se pierde un FIN o ACK.**

Liberación asimétrica



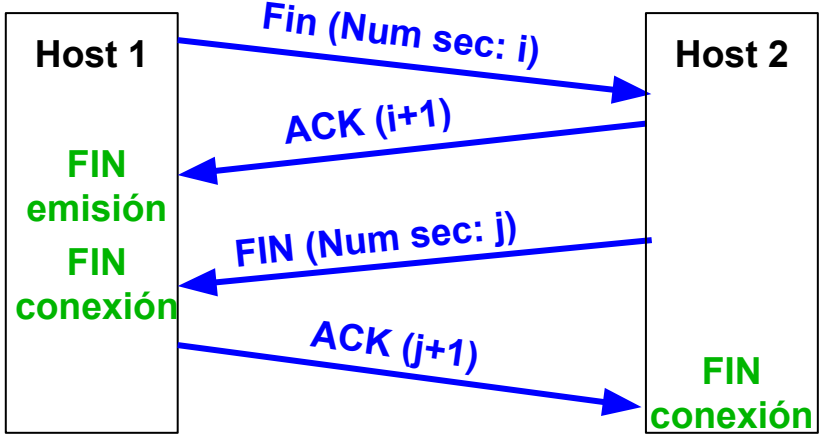
La liberación asimétrica no sirve si los host **no conocen la cantidad** de datos a enviar.



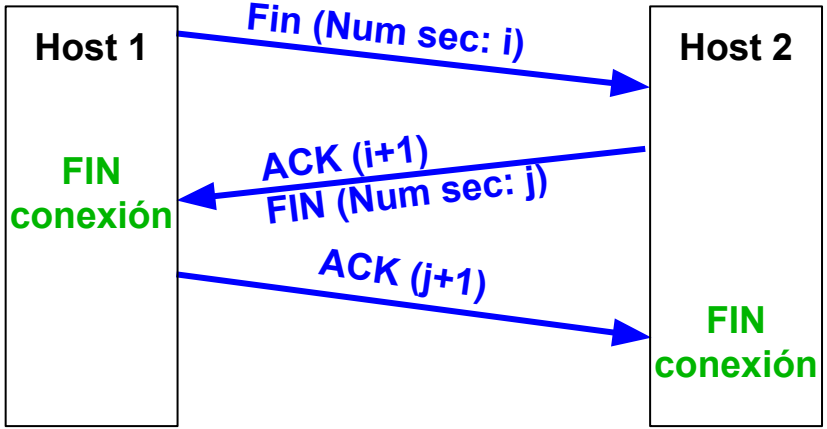
Figuras basadas en: A. Tanenbaum, D. Wetherall, "Redes de computadoras", Quinta edición (2012), pag. 444

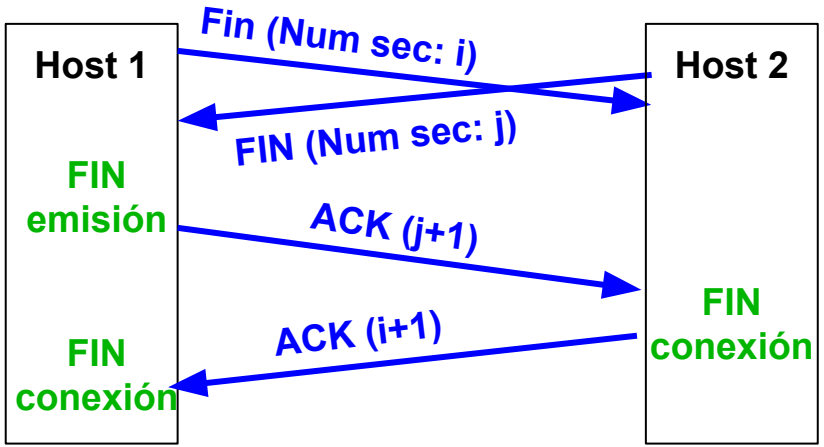
Protocolo TCP - Liberación de una conexión - situación normal

- Una conexión se trata como dos conexiones simplex, y se libera cada conexión simplex por separado.
  - Se envía un segmento de fin de conexión y un ack en cada sentido.
  - Se utiliza un temporizador por si se pierde el ack.
  - Una conexión cerrada en un sentido, puede seguir enviando datos en el otro sentido.



Protocolo TCP - Liberación de una conexión - situación normal





Liberación Simétrica: Necesidad de acuerdo de 3 vías + 3 temporizadores

- Problema: Los mensajes FIN pueden perderse.
  - El host que envía un mensaje FIN debe incluir un temporizador.
  - Si se vence el temporizador, se reenvía el mensaje FIN.
- Problema: Los mensajes ACK o ACK+FIN pueden perderse.
  - El host que envía un ACK debe iniciar un temporizador.
  - Si se vence el temporizador, cierra la conexión.
- El emisor puede enviar paquetes seguidos de un paquete FIN, pero pueden desordenarse.
  - Los números de secuencia resuelven este problema.
- Temporizador FIN < Temporizador ACK.
- Debe haber un límite en la cantidad de mensajes FIN enviados.
- Temporizador de verificación de “estado activo”.

Liberación Simétrica: Necesidad de acuerdo de 3 vías + temporizadores

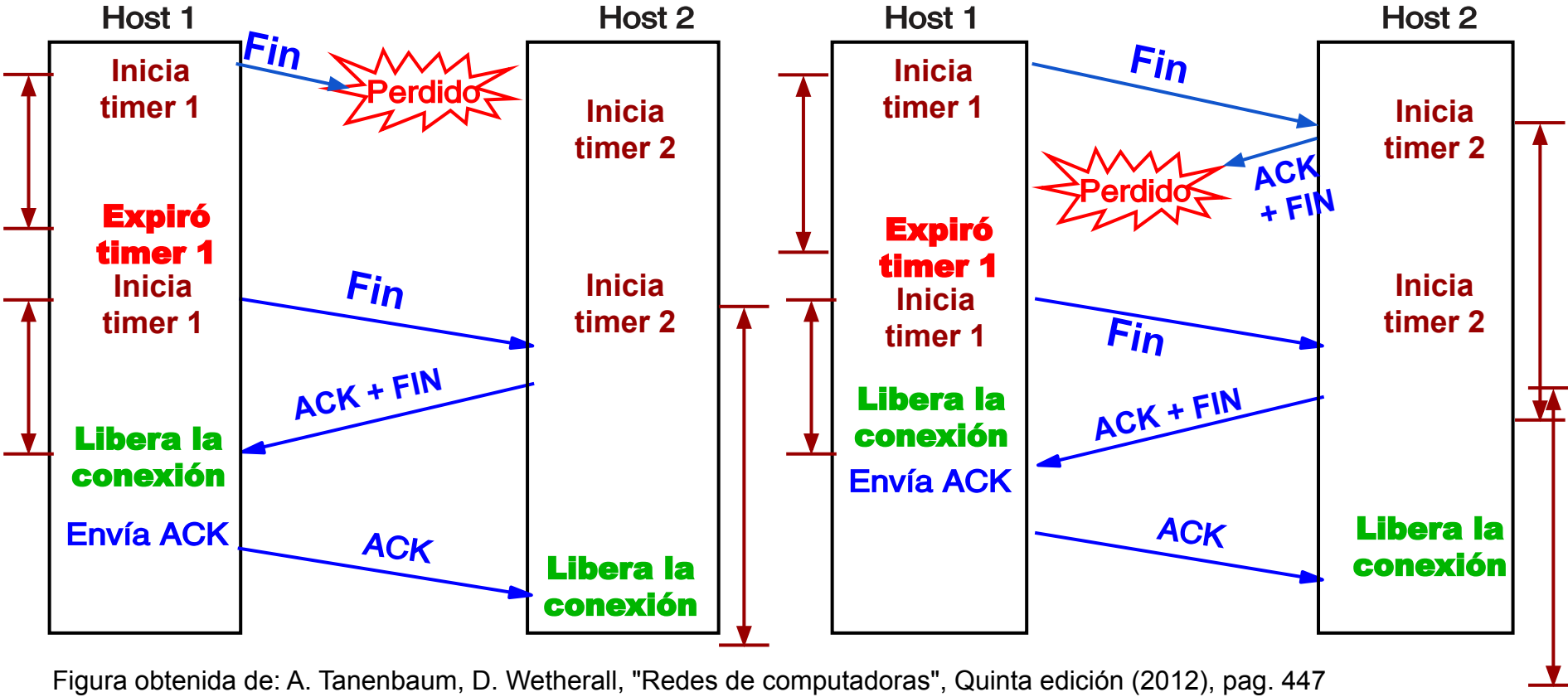
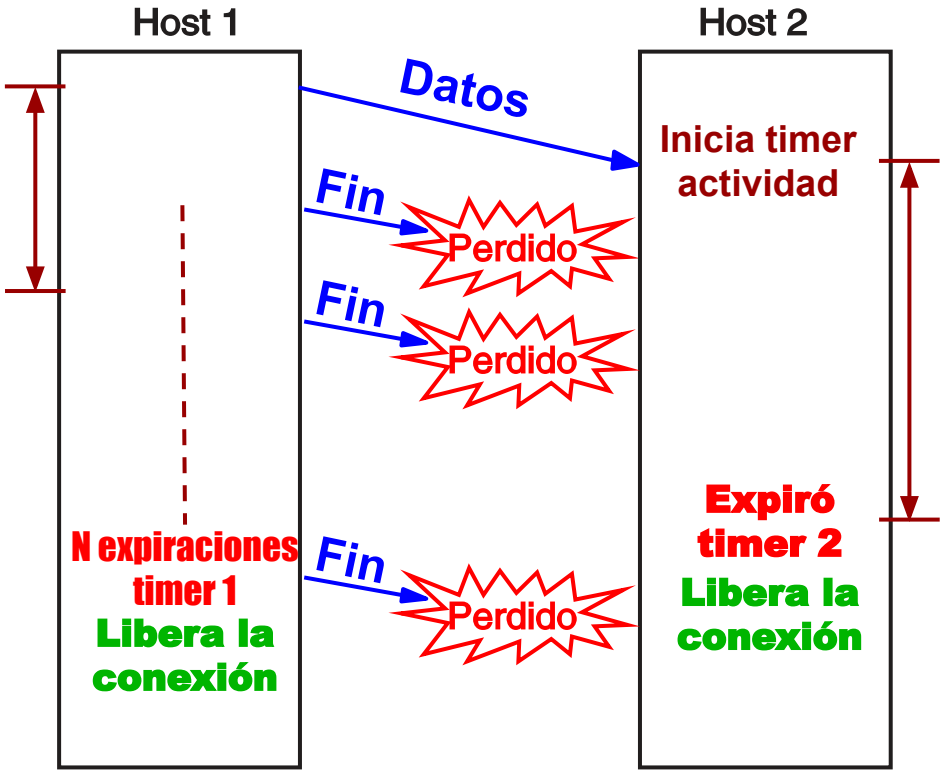


Figura obtenida de: A. Tanenbaum, D. Wetherall, "Redes de computadoras", Quinta edición (2012), pag. 447

Liberación Simétrica: Necesidad de acuerdo de 3 vías más temporizadores



Supervisor de estado de la conexión:

- Cada extremo **verifica frecuentemente si hay actividad** en la conexión.
  - Si no hay actividad durante cierto tiempo, libera la conexión.
- En caso de no tener datos para enviar, los extremos de la conexión envían **periódicamente segmentos ficticios** (avisan que el extremo está activo).

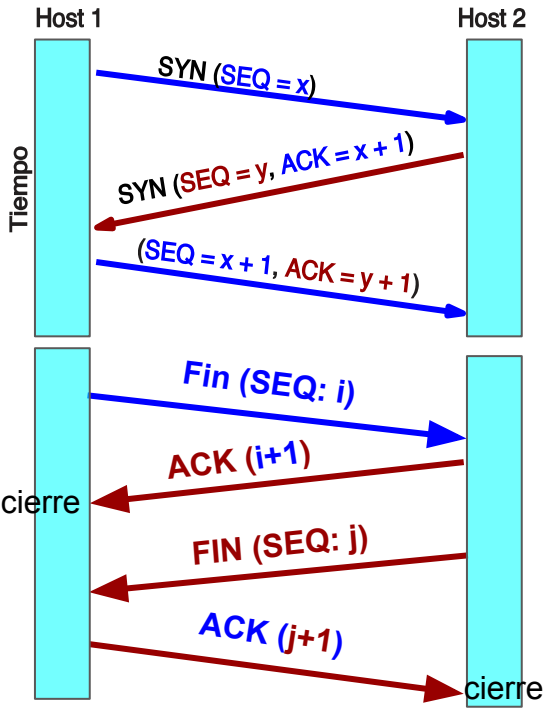
UNCUYO
UNIVERSIDAD NACIONAL DE CUYO

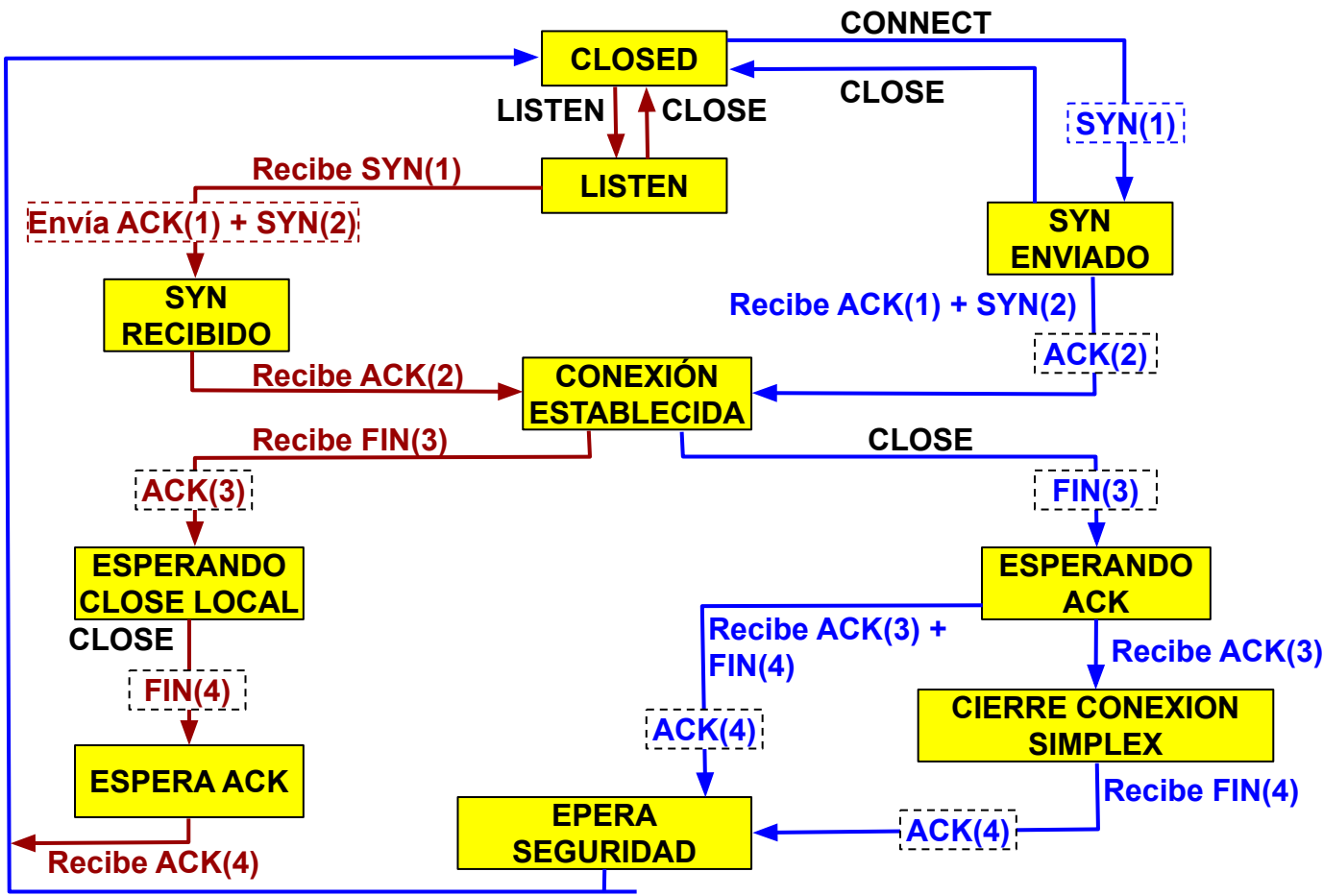
FACULTAD DE INGENIERÍA

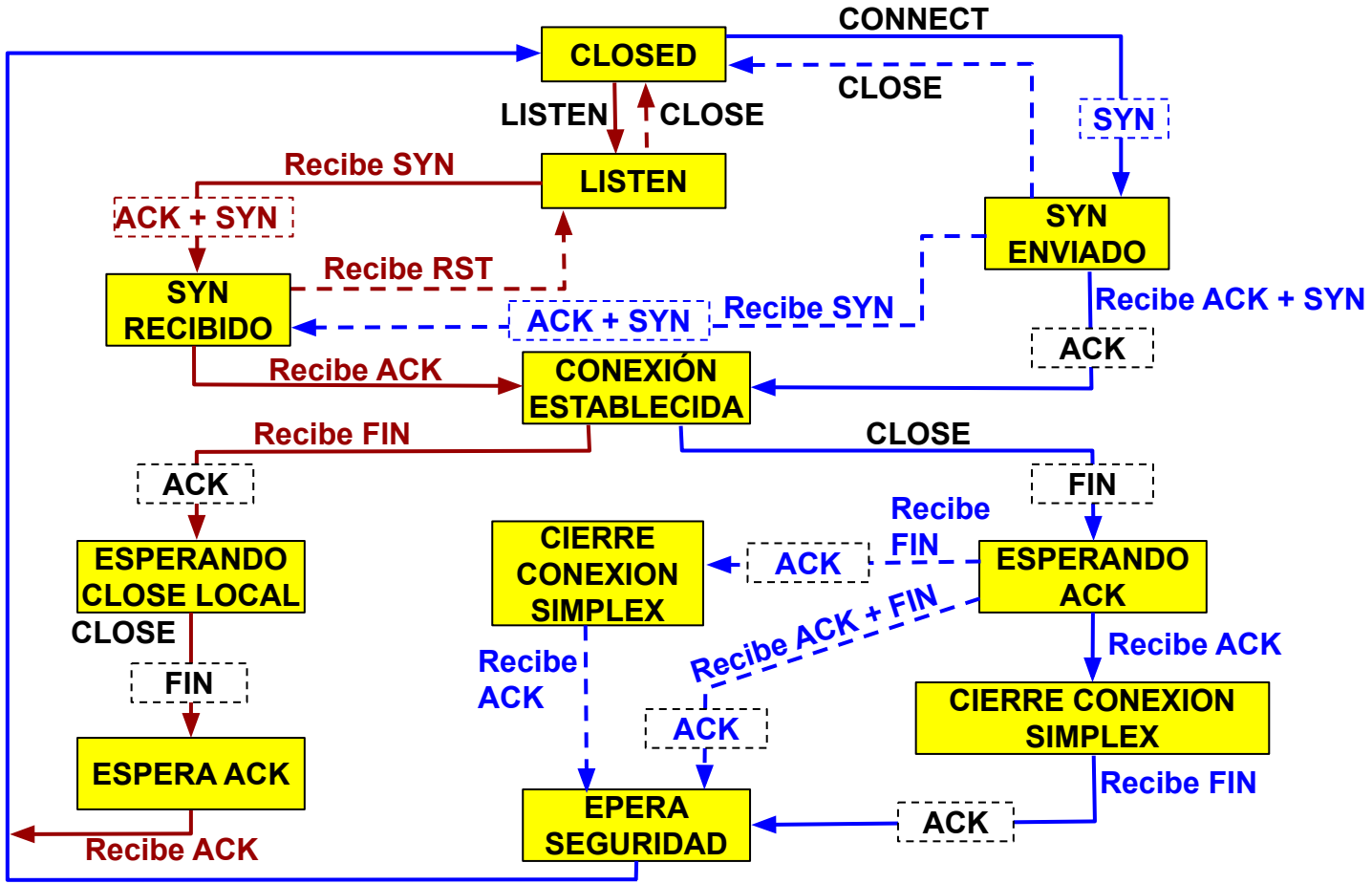
Licenciatura en Ciencias de la Computación

CONNECT: Comando ejecutado por un proceso.

FIN — Mensaje enviado por uno de los extremos





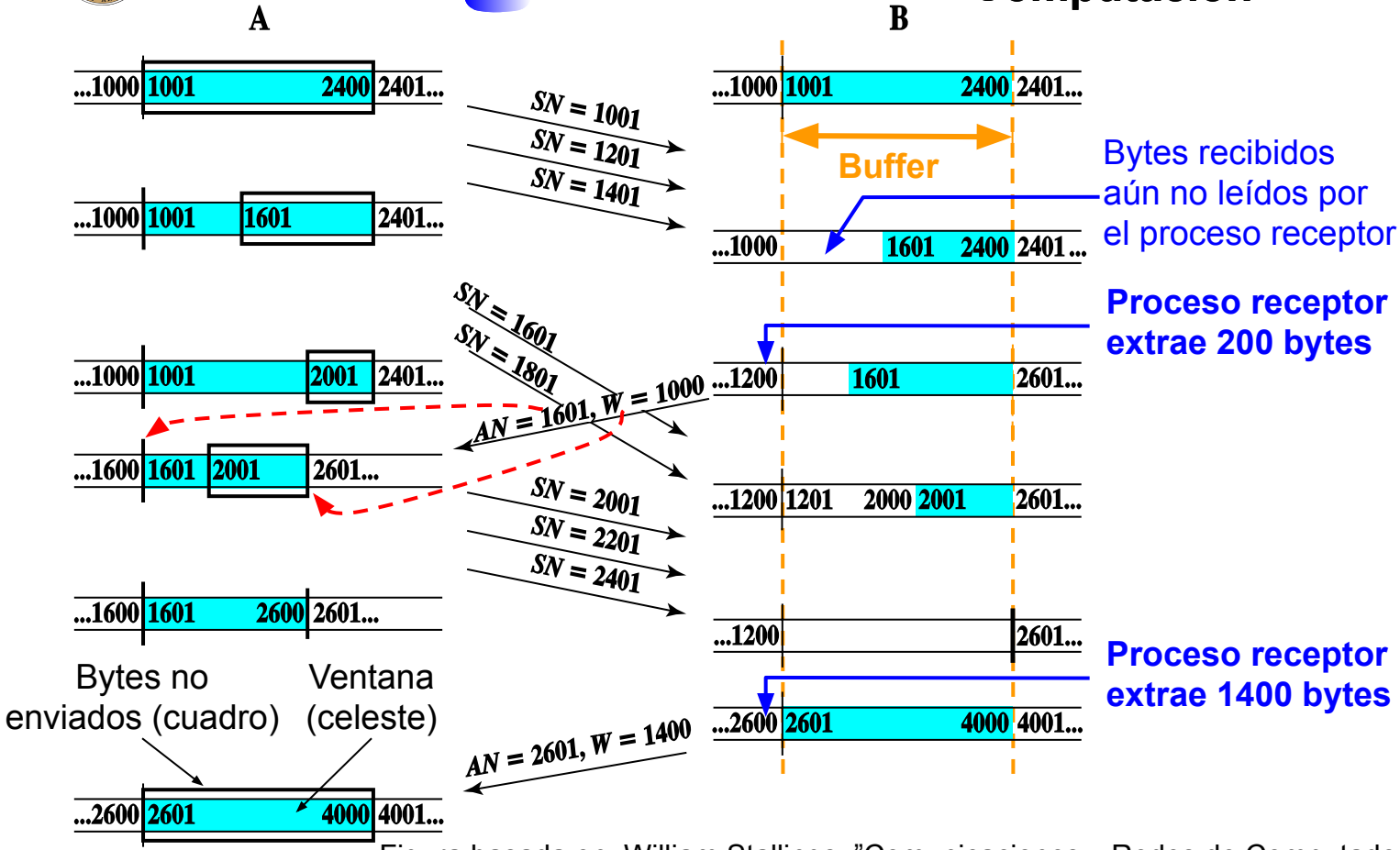

CONNECT
CLOSED
CLOSE
LISTEN
CLOSE
SYN
Recibe SYN
LISTEN
ACK + SYN
SYN ENVIADO
Recibe RST
SYN RECIBIDO
ACK + SYN
Recibe SYN
Recibe ACK + SYN
ACK
Recibe ACK
CONEXIÓN ESTABLECIDA
Recibe FIN
CLOSE
ACK
FIN
Recibe FIN
CIERRE CONEXION SIMPLEX
ACK
ESPERANDO ACK
ESPERANDO CLOSE LOCAL
CLOSE
Recibe ACK
Recibe ACK + FIN
FIN
Recibe ACK
ACK
CIERRE CONEXION SIMPLEX
ESPERA ACK
Recibe FIN
EPERA SEGURIDAD
ACK
Recibe ACK

Control de flujo - Ventanas deslizantes

- Para **control de flujo** se utiliza **ventanas deslizantes** debido a la alta latencia.
  - Buffers emisor y receptor de tamaño variable
  - **El receptor envía mensajes de tamaño de ventana indicando al emisor cuantos bytes está dispuesto a recibir.**
- Ventana del receptor: Números de secuencia que el receptor puede recibir
  - El límite inferior se mueve a medida que llegan segmentos correctos desde el emisor, para **rechazar segmentos repetidos**.
  - El límite superior se mueve a medida que el **proceso receptor extrae datos del buffer de recepción**.
- Ventana del emisor: Números de secuencia que el emisor puede enviar.
  - El límite inferior se mueve a medida que llegan ACK desde el receptor, **para sacar paquetes que ya se recibieron.**
  - El límite superior se mueve a medida que llegan **notificaciones de tamaño de ventaja** desde el receptor Y el emisor tenga datos para enviar.

Protocolo TCP: Ventanas deslizantes

- Cada byte tiene su número de secuencia de 32 bits.
- EL ACK de un segmento puede ir incluido en un segmento de datos o ir aislado.
- El Ack reconoce los bytes recibidos indicando el próximo byte que se espera recibir (si se recibió correctamente hasta el 40021, se indicará 40022).
- Un segmento retransmitido puede incluir un número diferente de bytes.
- Si el tamaño de ventana es cero, el emisor puede enviar dos tipos de segmentos:
  - Segmento urgente.
  - Sonda de ventana: segmento de un byte para que el receptor vuelva a anunciar el tamaño de ventana y el próximo byte (por si llega a perderse una actualización de ventana).



SN: Sequence number (1º byte).
W: Window
AN: ACK number

En el ejemplo:
Tamaño máximo del buffer del receptor: 1400 bytes.
Por simplicidad, suponemos segmentos de tamaño fijo de de 200 bytes, y que el emisor siempre tiene datos para enviar.

Figura basada en: William Stallings, "Comunicaciones y Redes de Computadores", 7ed, 2004, pag. 661

Protocolo TCP: Ventanas deslizantes

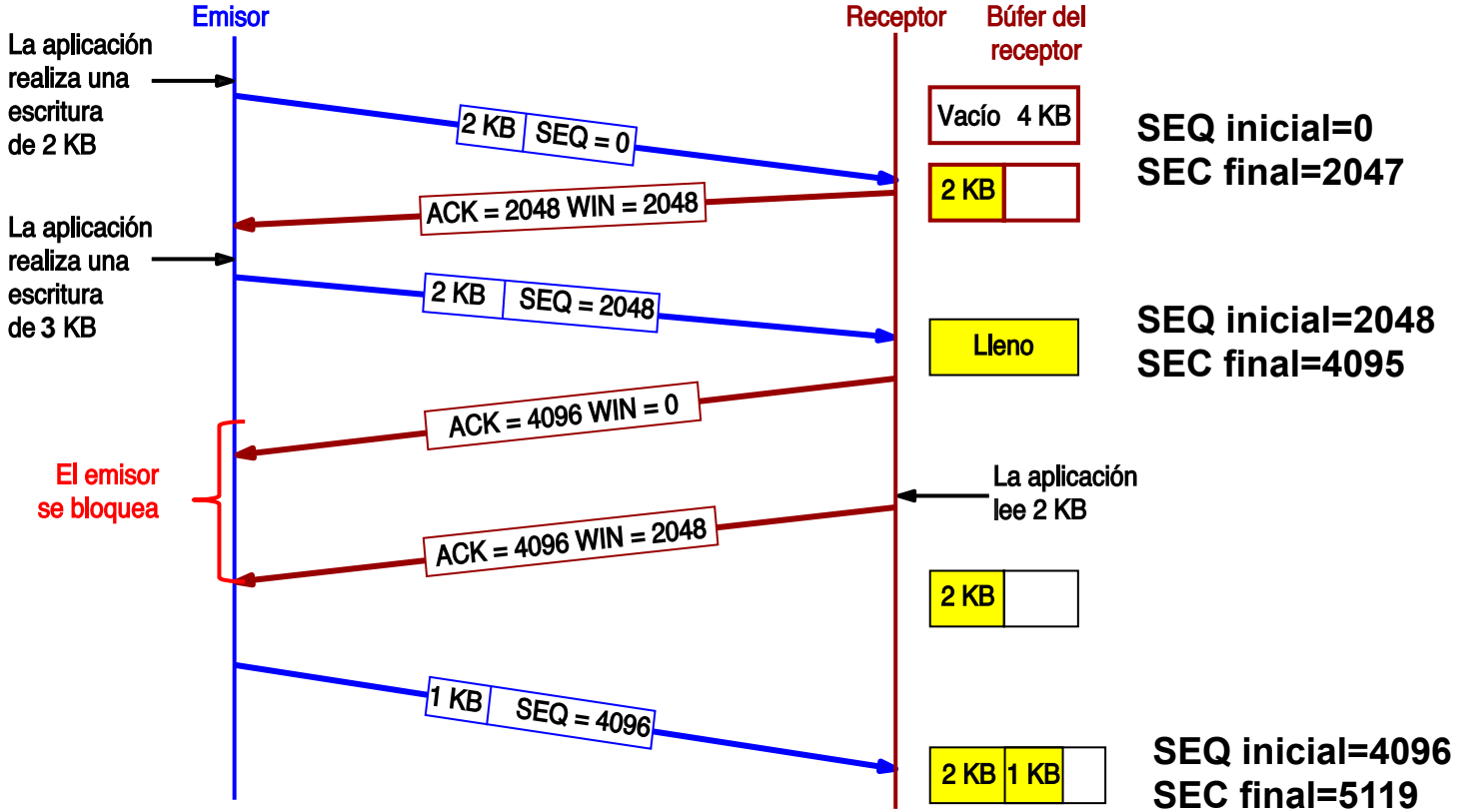
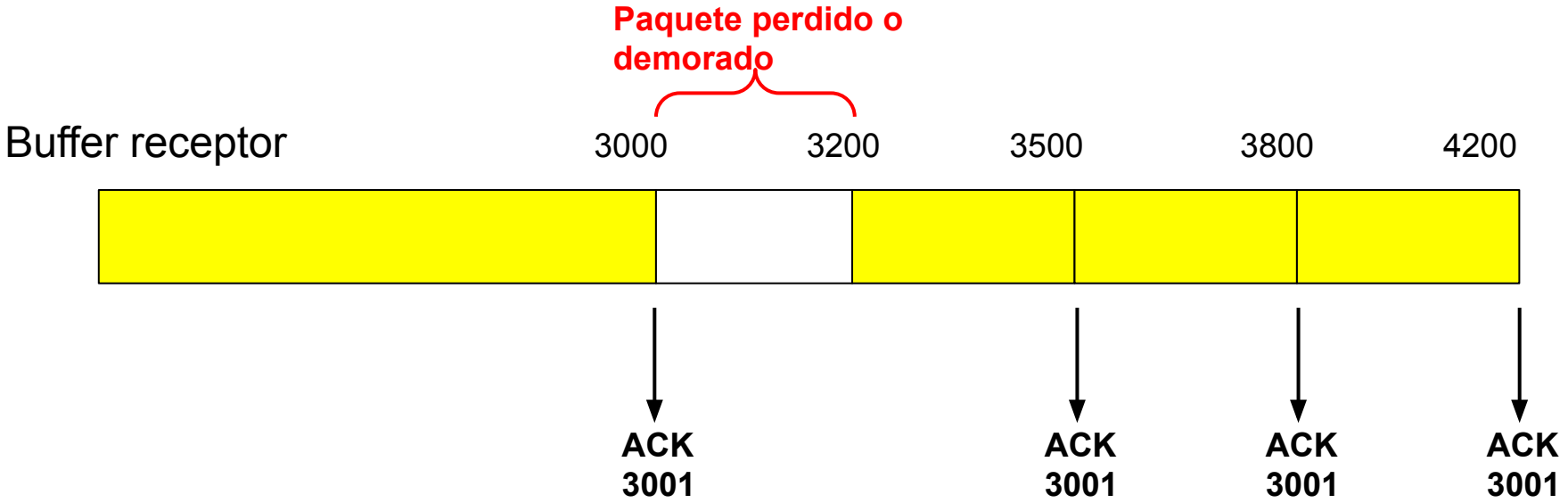


Figura basada en: A. Tanenbaum, D. Wetherall, "Redes de computadoras", Quinta edición (2012), pag. 485

Control de Congestión

- Señales de congestión: Los host necesitan detectar que está ocurriendo una congestión en algún punto de la red o (mejor aún) que está apunto de producirse.
  1. ECN: **Paquetes marcados**. (Recordar notificación congestión IP) o **Mensajes de notificación** enviados por enrutadores IP.
  2. Detectar **ACK repetidos** (cuando hay huecos en el buffer del recepción)
  3. Detectar **Pérdida de paquetes** (expiración RTO).
- El protocolo debe ser **eficiente**: La red debe trabajar a su **máxima capacidad (velocidad) sin producir congestión**.
  - **Equilibrio muy difícil de lograr:**
    - Se debe trabajar lo más cerca posible de la congestión (ya que esta será la máxima capacidad), pero sin producirla.
    - La capacidad máxima es muy variable.
- El protocolo debe ser **equitativo entre diferentes flujos**. Muy difícil (ver siguiente).

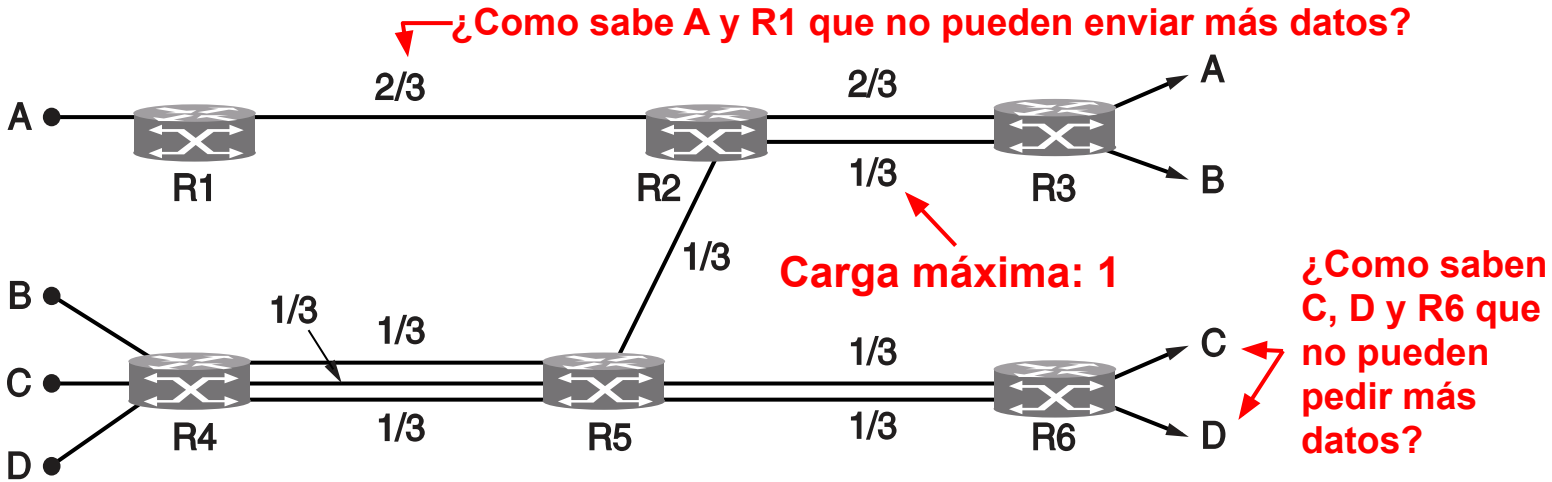
Control de Congestión - ACK repetidos



Si se demoró o perdió un paquete, el receptor enviará ACKs indicando hasta que byte se recibieron correctamente.

Control de Congestión en la capa de transporte

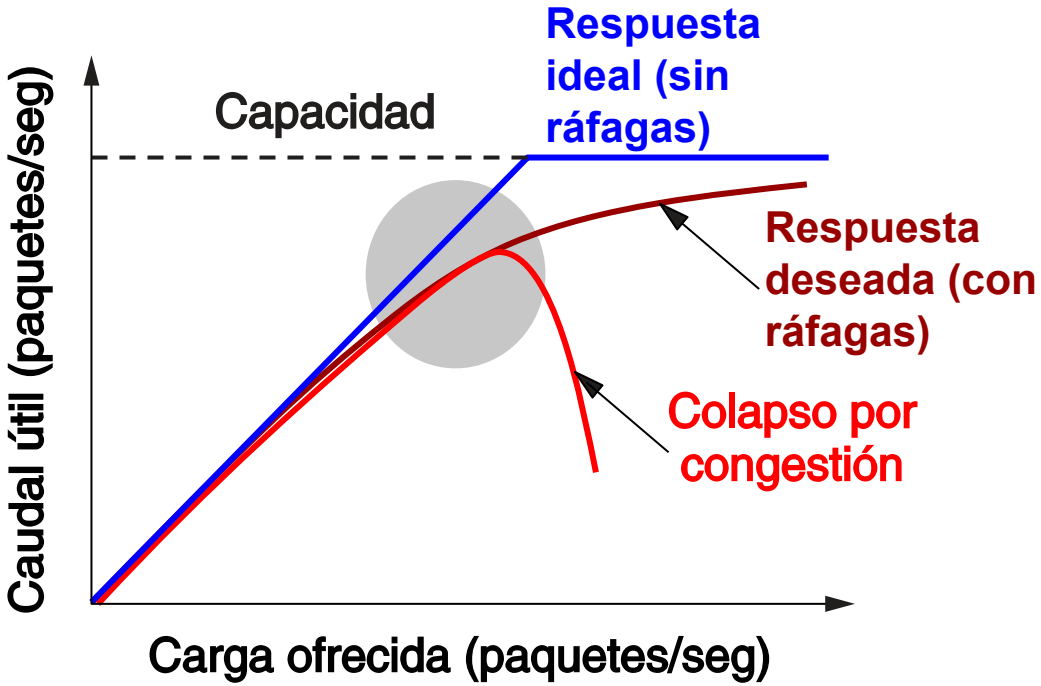
- Dificultad de lograr equidad entre todas las congestiones:



- Convergencia rápida. La cantidad de flujos y necesidades de ancho de banda de cada uno cambian constantemente. El protocolo debe adaptarse rápidamente.

Figura obtenida de: A. Tanenbaum, D. Wetherall, "Redes de computadoras", Quinta edición (2012), pag. 457

Control de Congestión



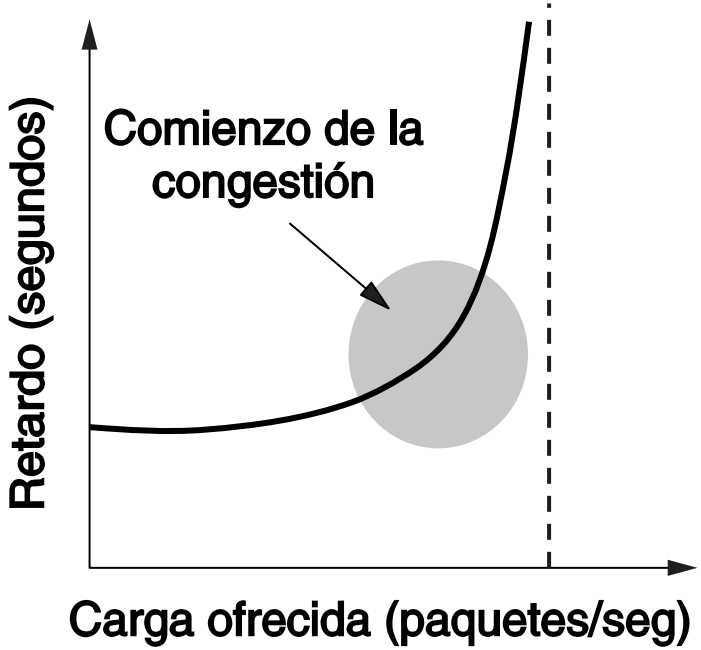


Figura obtenida de: A. Tanenbaum, D. Wetherall, "Redes de computadoras", Quinta edición (2012), pag. 456

Control de Congestión en la capa de transporte: Ajuste dinámico o AIMD

- Cuando se detecta la congestión, se debe disminuir la tasa de envío del emisor. Esto se realiza **disminuyendo la ventana deslizante del emisor**.
- Algoritmo **AIMD** (Additive Increase Multiplicative Decrease):
  - Objetivo: llegar a un punto **eficiente** y **equitativo**.
  - Línea eficiente: los recursos son utilizados al 100%.
  - Línea equidad: ambos flujos comparten por igual los recursos.

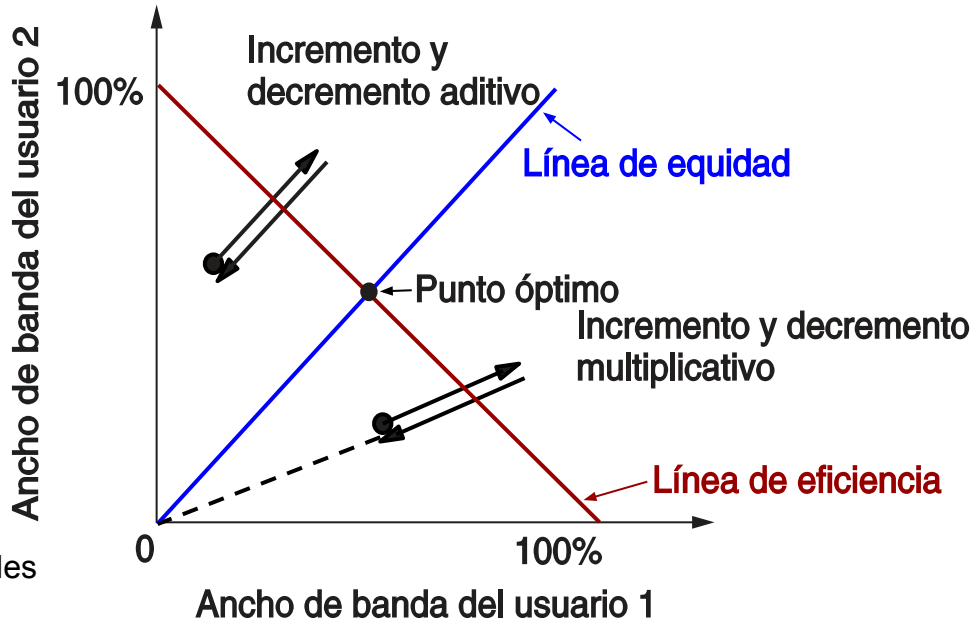


Figura obtenida de: A. Tanenbaum, D. Wetherall, "Redes de computadoras", Quinta edición (2012), pag. 461

Control de Congestión en la capa de transporte: Algoritmo AIMD

- Los emisores aumentan aditivamente sus flujos (ejemplo, sumando 1Mbps)
- Los emisores disminuyen multiplicativamente sus flujos (ejemplo, multiplicando por 0.9)

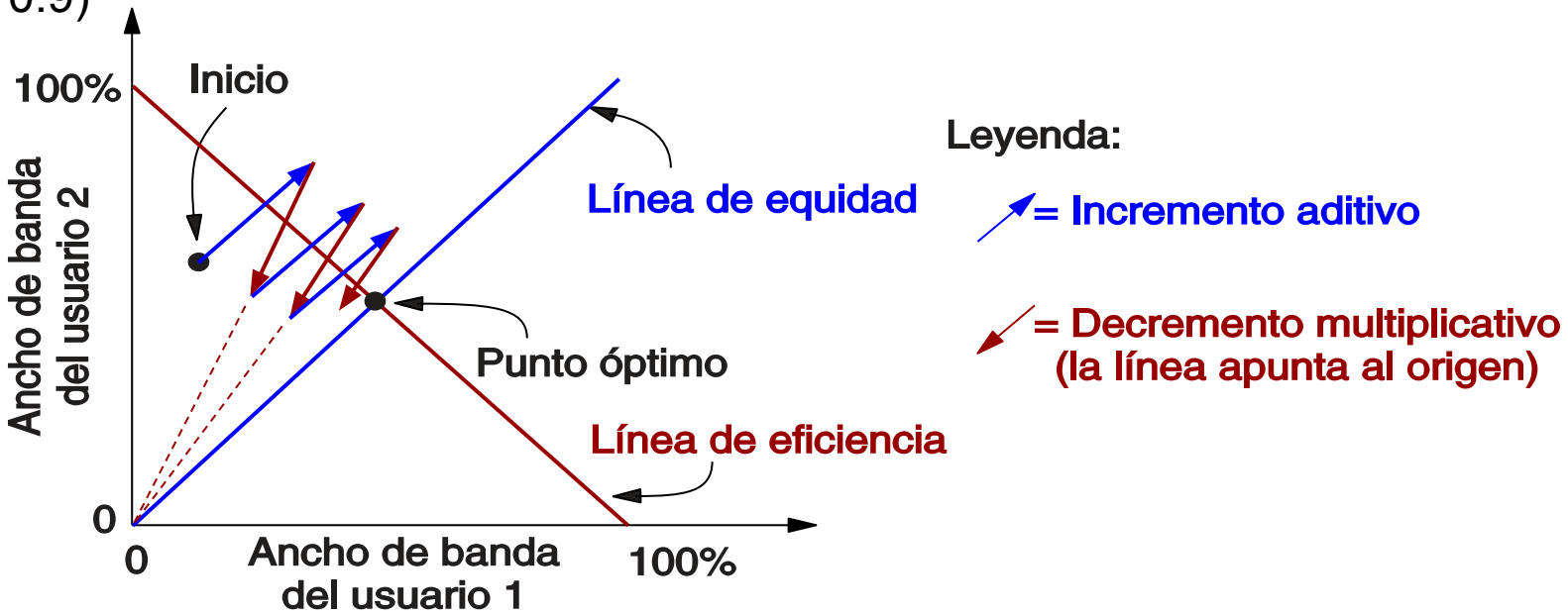
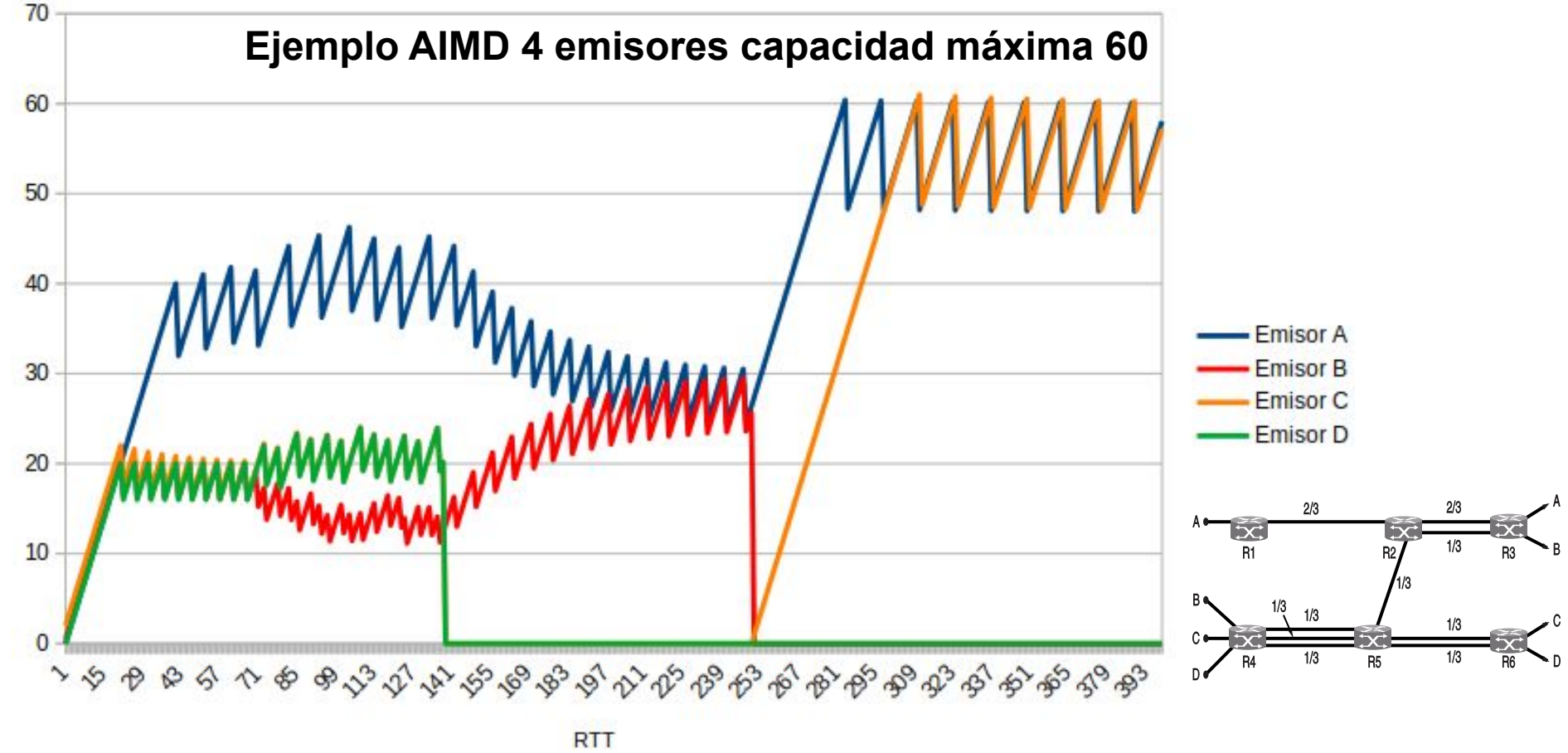
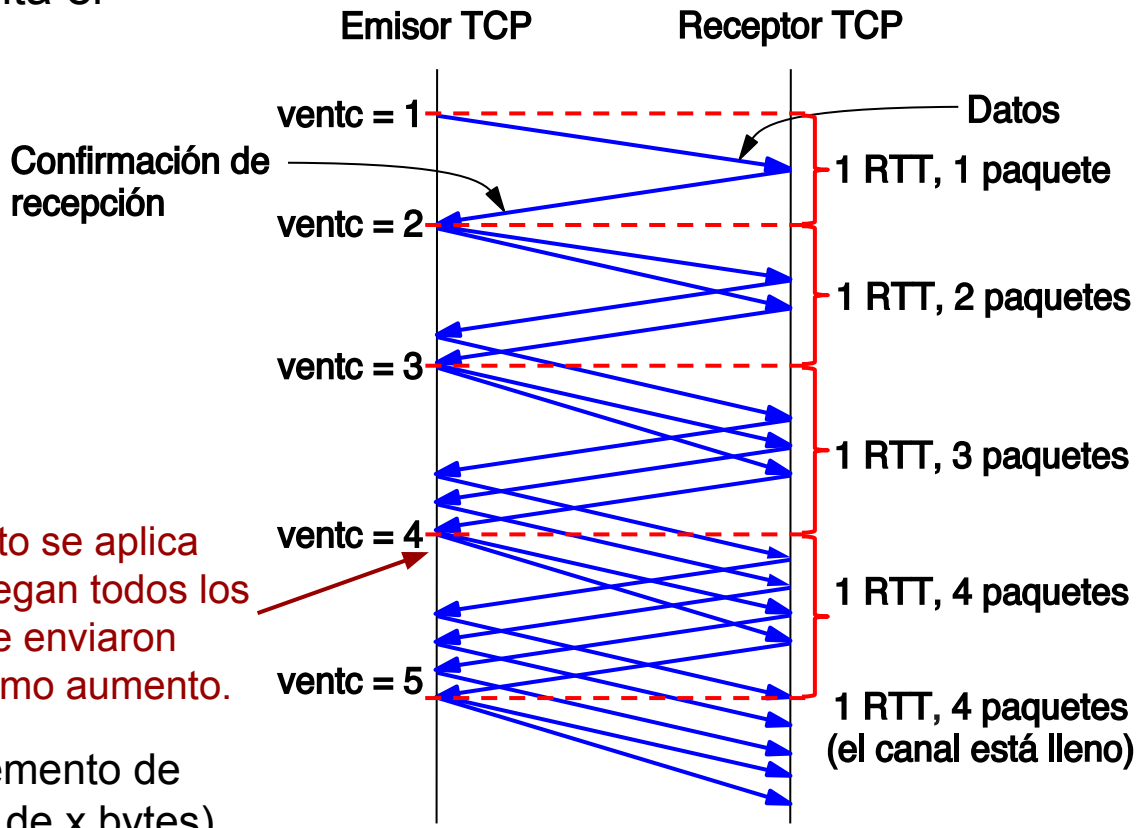


Figura obtenida de: A. Tanenbaum, D. Wetherall, "Redes de computadoras", Quinta edición (2012), pag. 462


Ejemplo AIMD 4 emisores capacidad máxima 60
70
60
50
40
30
20
10
0
1
15
29
43
57
71
85
99
113
127
141
155
169
183
197
211
225
239
253
267
281
295
309
323
337
351
365
379
393
RTT
Emisor A
Emisor B
Emisor C
Emisor D
A
B
C
D
R1
R2
R3
R4
R5
R6
2/3
1/3

¿Cómo se incrementa o decrementa el ancho de banda?

- **Variando el tamaño de la ventana de transmisión.**
- El incremento o decremento se aplica cada RTT (Round-Trip time).



El incremento se aplica luego que llegan todos los bytes que se enviaron desde el último aumento.

(Nota: el gráfico es ilustrativo, el incremento de ventana generalmente será en saltos de x bytes).

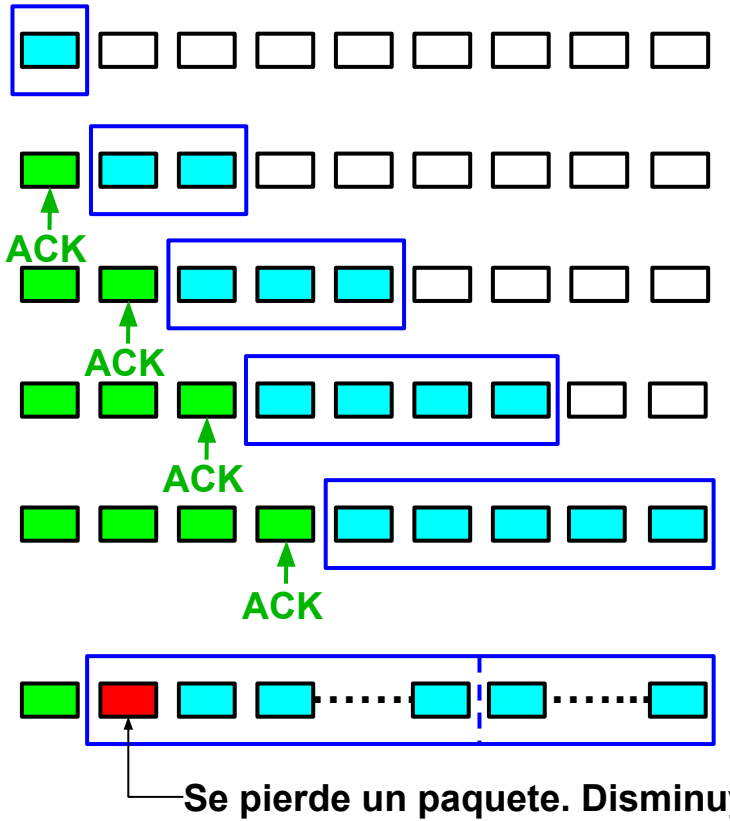
Control de Congestión de TCP - Implementación de AIMD

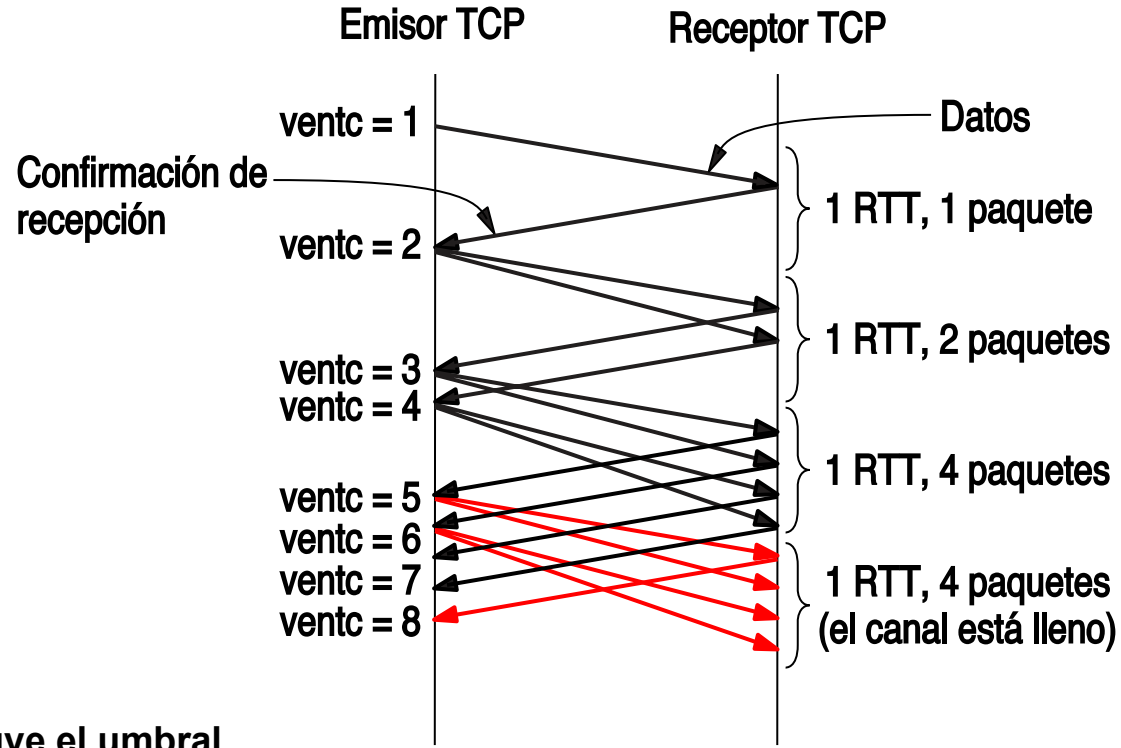
- Se mantiene una **ventana de congestión**, que indica el número de bytes que el emisor puede tener en la red.
- La tasa de datos o ancho de banda es el tamaño de la ventana dividido el tiempo de ida y vuelta (**Velocidad de envío = Tamaño de la ventana/RTT**).
  - Lo que se controla es el **Tamaño de la ventana**.
- Se mantienen **dos ventanas** que se actualizan y rastrean por separado:
  - **Ventana de congestión**.
  - **Ventana de control de flujo**.
  - El número de bytes que puede enviarse es el **menor** informado por ambas ventanas.
- Para usar la tasa de pérdida de paquetes como señal para actuar se requiere que los paquetes descartados por errores se mantengan al mínimo. Por esto es necesario en control de errores en redes inalámbricas en la capa de enlace.

TCP - Control de Congestión - Inicio Lento

- ¿Cual es el **valor inicial** de la ventana de congestión?
  - Si es **muy pequeña**, el **tiempo necesario** aplicando **AIMD** para que llegue a la velocidad óptima **será grande**.
  - Si es **muy grande**, podría **causar congestión severa**.
- Solución empleada en TCP: Al inicio no usar AIMD, usar "**Inicio lento**" (aunque en realidad es rápido).
  - Empieza con una **ventana pequeña**, aumentando exponencialmente su tamaño.
  - Se envía un segmento.
  - Por cada ACK recibido antes de que expire el temporizador de retransmisión, se aumenta la ventana de congestión una cantidad de bytes correspondiente a un segmento.
    - Esto ocasiona que por cada ACK, se transmitan dos segmentos.

TCP - Control de Congestión - Inicio Lento

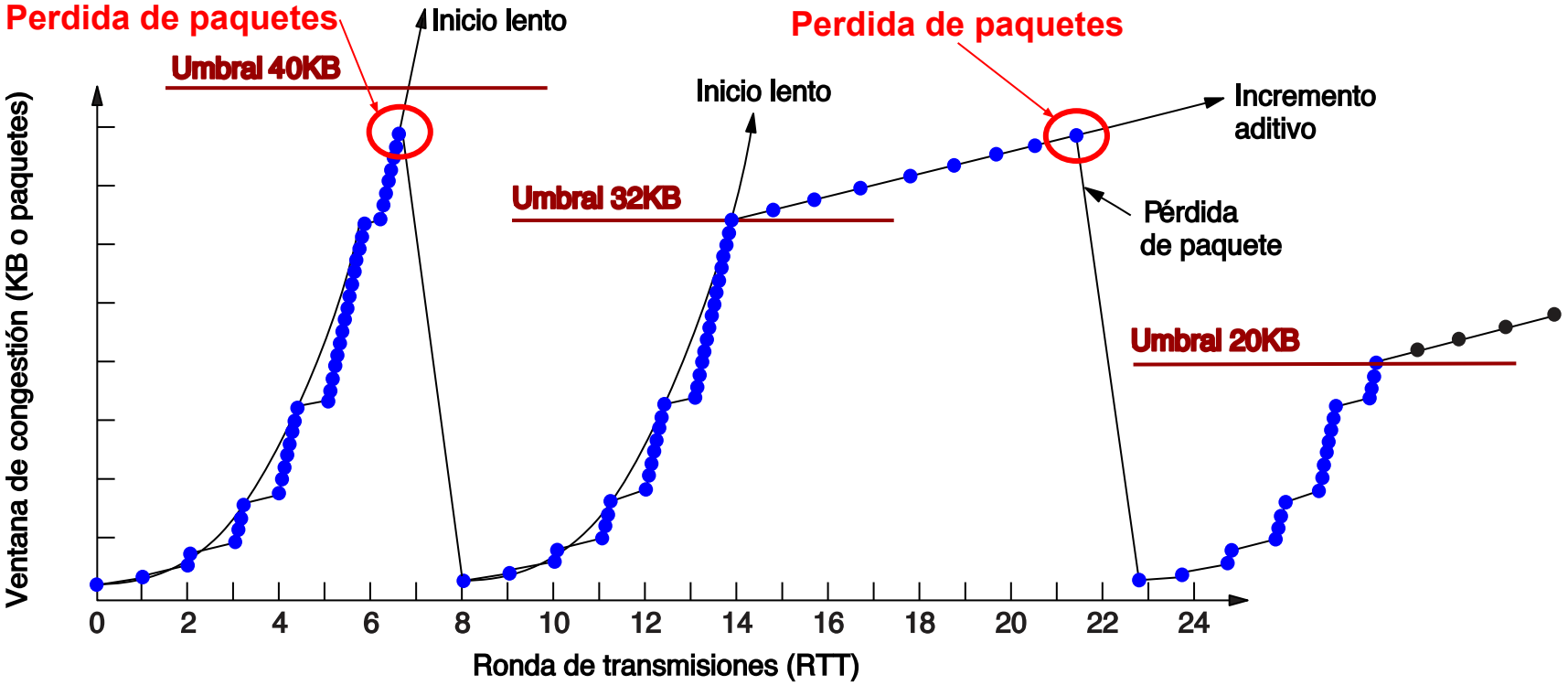




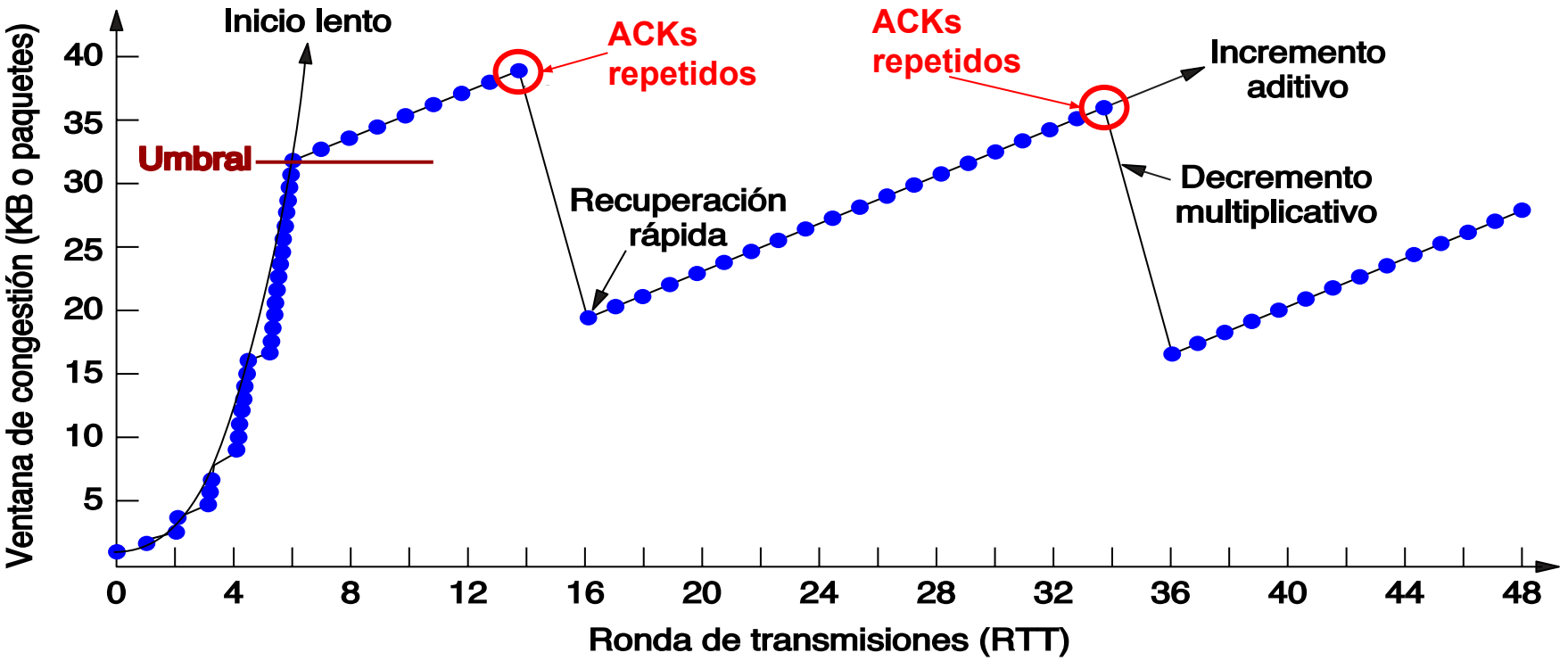
TCP - Control de Congestión

- ¿Cuando se pasa de Inicio lento a AIMD?. Se fija un **umbral**. Se comienza con inicio lento,
  - Si hay congestión antes de llegar al umbral, la ventana de congestión vuelve a 1, el umbral disminuye a la mitad, y se comienza nuevamente.
  - Si se sobrepasa el umbral sin congestión, se pasa a AIMD.
- TCP Reno: Una vez que se conmuta a AIMD, se reconoce el tipo de señal de congestión.
  - Señal de congestión severa (retransmisión de segmentos): el algoritmo vuelve al inicio: inicio lento con ventana de congestión = 1 y umbral menor.
  - Señal de congestión leve (3 ACKs repetidos): Se aplica decremento multiplicativo y continúa con incremento aditivo (AIMD).

TCP - Control de Congestión TCP-Reno



TCP - Control de Congestión TCP-Reno



Protocolo TCP - Ventanas deslizantes - Problemas y soluciones

- Problema: Un ACK con un nuevo valor de W se puede perder. Si previamente se transmitió un mensaje W=0, **se bloquea la comunicación**.
- **Solución:**
  - **Se envían periódicamente notificaciones W.**
  - **El emisor envía periódicamente mensajes de sondeo de W.**

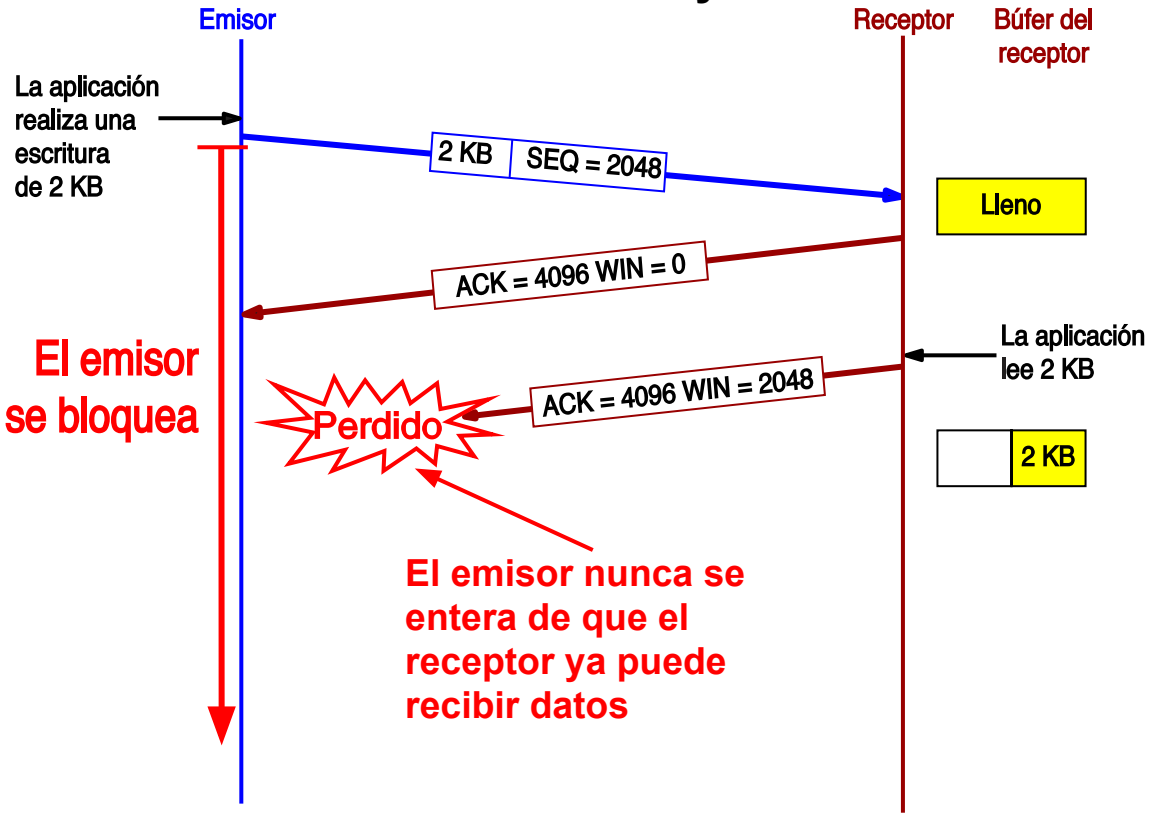
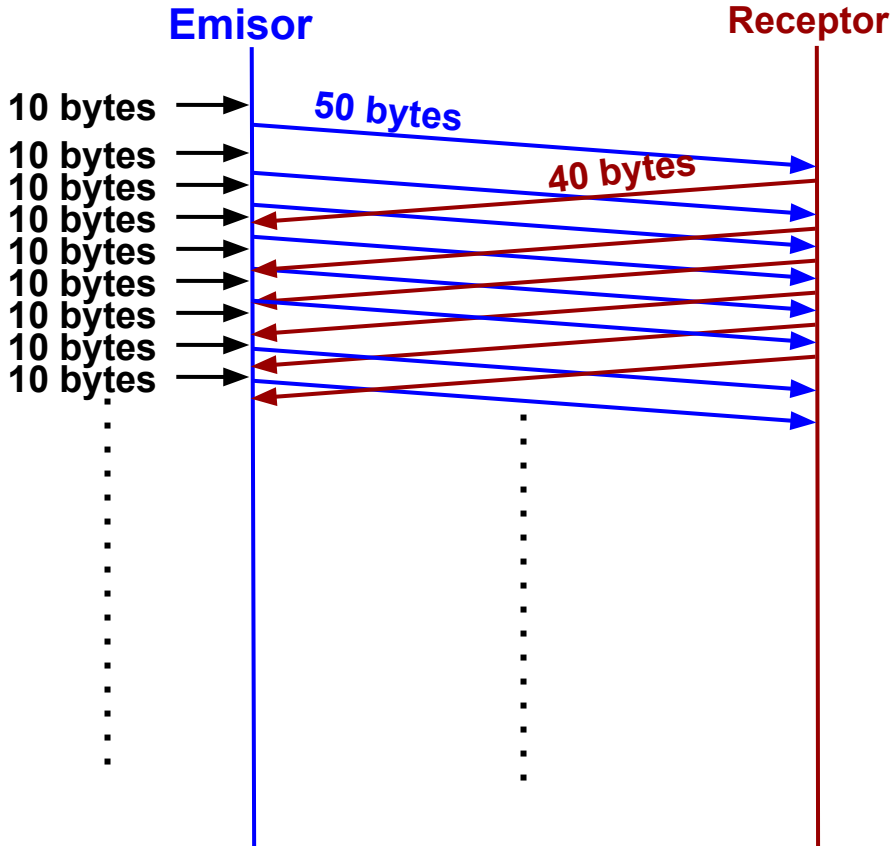


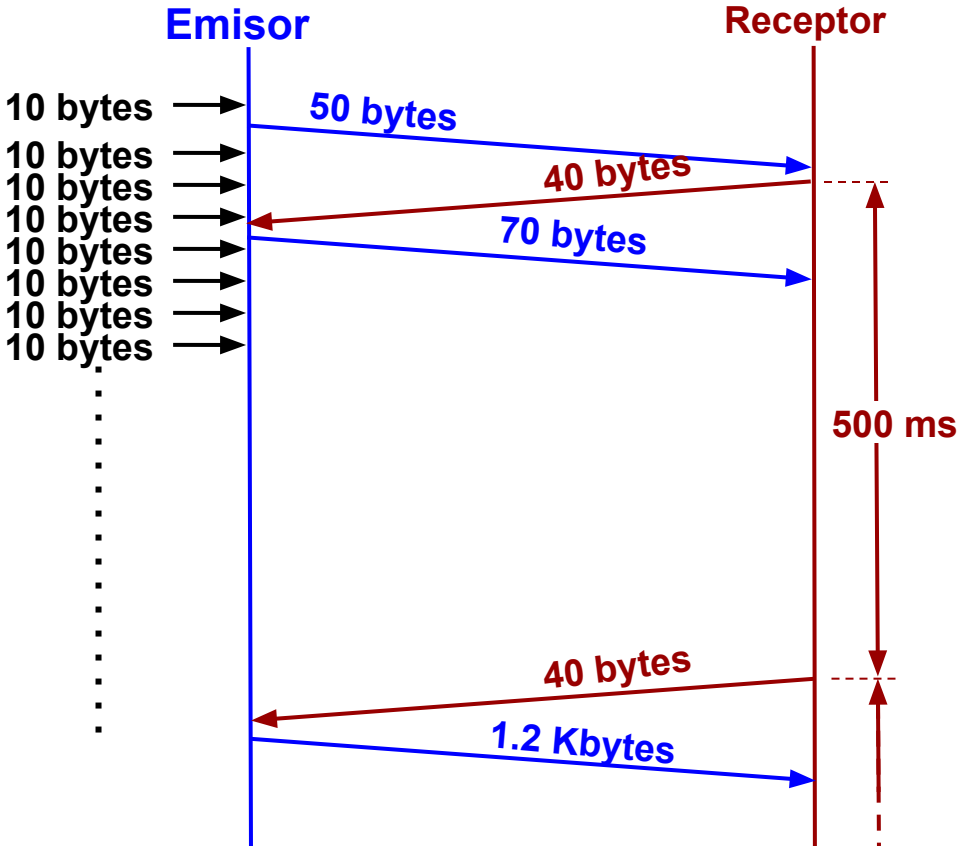
Figura obtenida de: A. Tanenbaum, D. Wetherall, "Redes de computadoras", Quinta edición (2012), pag. 485

Protocolo TCP - Ventanas deslizantes - Problemas y soluciones

- Emisor que **escribe de a un byte o pocos bytes** en el buffer de salida.
  - **Emplea de manera ineficiente el ancho de banda**:
    - Se envíen paquetes con 40 bytes de sobrecarga (encabezado IP + encabezado TCP) y pocos bytes de datos.
    - Se generan gran cantidad de ACKs, con 40 bytes de sobrecarga, para reconocer solo un byte o pocos bytes.
  - Soluciones:
    - **Confirmación de recepción con retardo** (hasta 500 ms). Ahorra paquetes ACK y permite que puedan esperarse datos para anexar ACKs.
    - **Algoritmo de Nagle**: Se envía solo el primer bloque de datos (pudiendo ser un solo byte) y los demás se almacenan en el buffer hasta que llegue el ACK (también ayuda a evitar la congestión).
    - **No útil con aplicaciones que no admiten retardos**: SSH o juegos online (**TCP_NODELAY**).

Ventanas deslizantes - Problema del emisor que escribe pocos bytes en el buffer

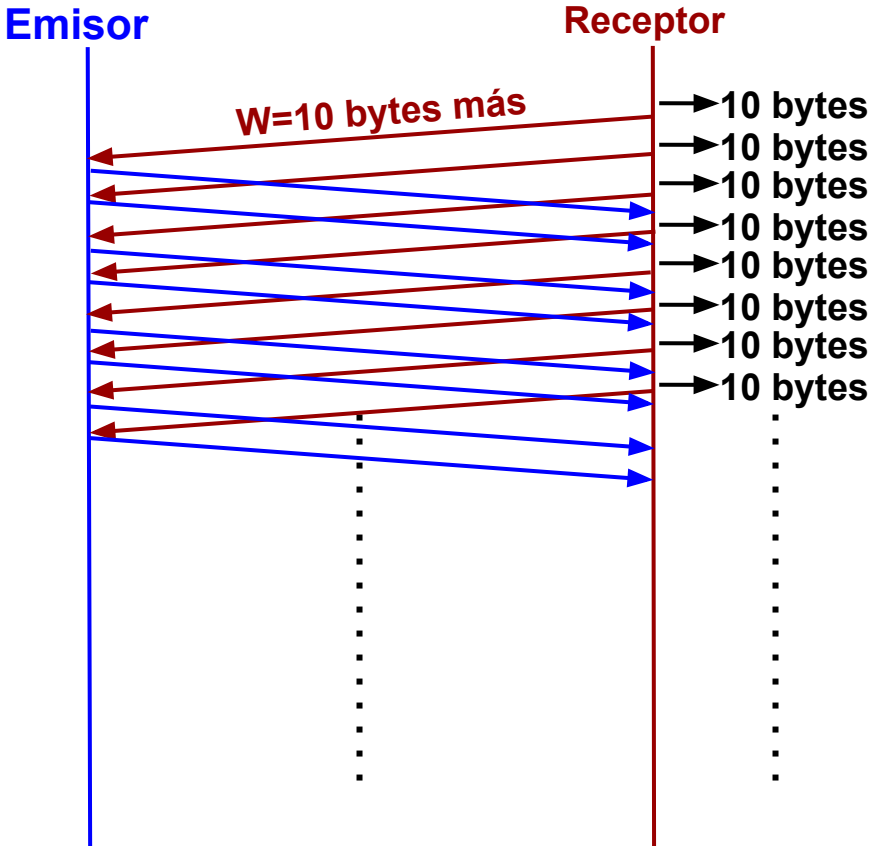


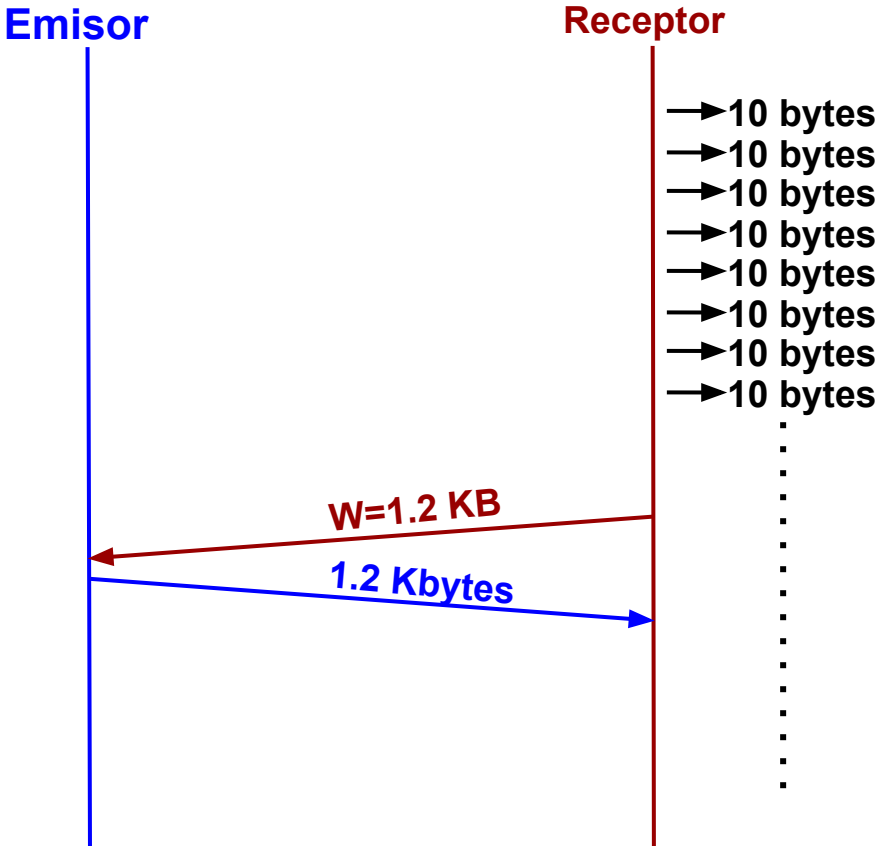


Protocolo TCP - Ventanas deslizantes - Problemas y soluciones

- **Síndrome de la ventana tonta**: Cuando el proceso receptor retira de a un byte o pocos bytes del buffer de recepción.
  - Provoca el envío de actualizaciones de ventana de pequeño tamaño + el emisor envía segmentos de pocos bytes.
  - Solución: **algoritmo de Clark**, se obliga al receptor a esperar a tener una cantidad mínima de bytes libres antes de enviar un anuncio de ventana.
  - Tamaño mínimo, el menor de:
    - Tamaño máximo del segmento que la conexión puede manejar.
    - Mitad del buffer libre.

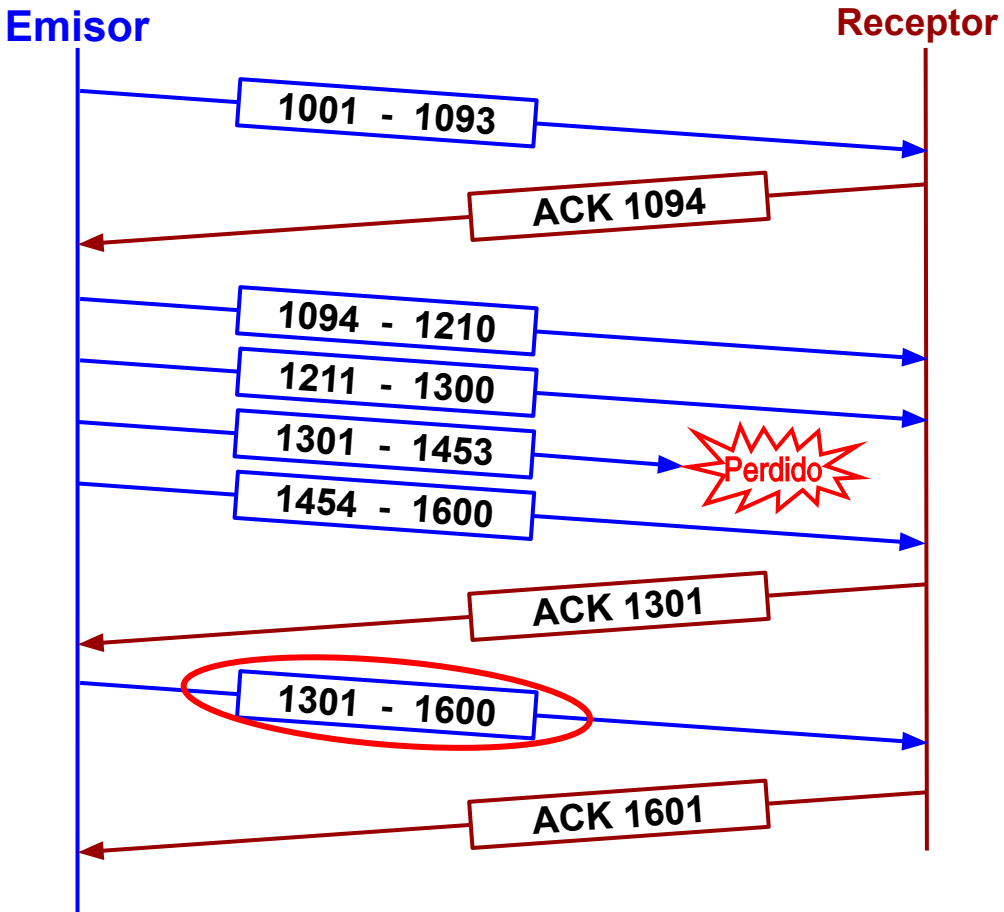
Ventanas deslizantes - Síndrome de la ventana tonta





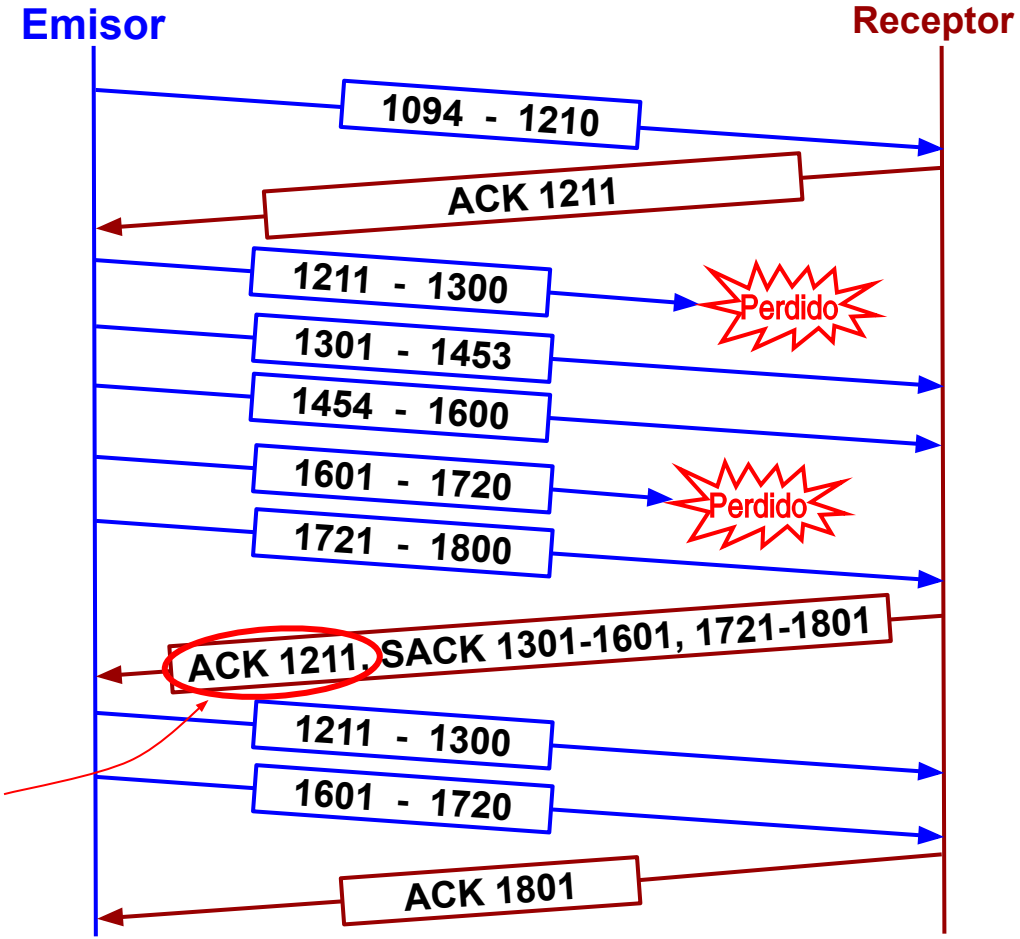
Protocolo TCP - Ventanas deslizantes - Problemas y soluciones

- Problema de las **confirmaciones de recepción acumulativas**: **Pueden originar reenvío de datos que el receptor ya recibió.**
- **Solución: SACK (Selective ACKnowledgements)**



TCP - SACK

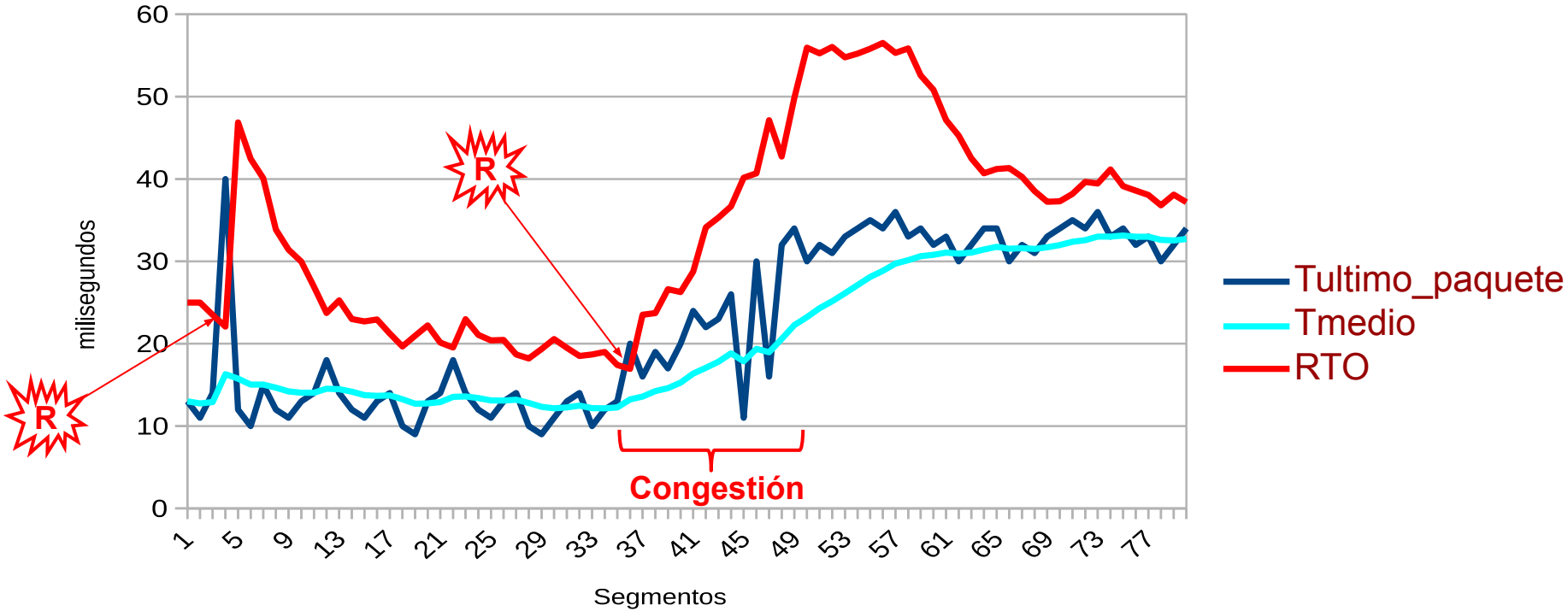
- SACK (Selective ACKnowledgements). Además del ACK, se confirman rangos de bytes posteriores correctamente recibidos.
- RFC 3517 (2003).
- Se permiten SACK hasta 3 rangos de bytes bien recibidos.
- Emisor y receptor deben acordar que utilizarán SACK



Temporizadores TCP (algunos)

- **Temporizador de retransmisión RTO** (Retransmission TimeOut):
  - Tiempo medio de ida y vuelta: $Tmedio = \alpha\ Tmedio + (1 - \alpha)\ Tultimo\_paquete$
    - Tmedio: tiempo de un paquete cuyo ACK se recibe correctamente.
    - α: factor de suavizado (usualmente 7/8).
  - Variación tiempo ida y vuelta:
    $Tvar = \beta\ Tvar + (1 - \beta)\ |Tmedio - Tultimo\_paquete|$
    - β: factor de suavizado (usualmente 3/4).
  - **$RTO = Tmedio + 4 \times Tvar$**
- **Temporizador de persistencia**: Tiempo entre sondeos de ventana enviados por el emisor cuando el último tamaño de ventana fue 0, por si se pierde una nueva actualización de ventana.
- **Temporizador de seguir con vida**: Para indicar al otro extremo que sigue vivo.
- **Tiempo espera de seguridad antes del cierre**: Da tiempo a que paquetes perdidos lleguen una vez cerrada la conexión (ver máquina de estado).

Temporizadores TCP - RTO



Protocolo TCP: Segmento

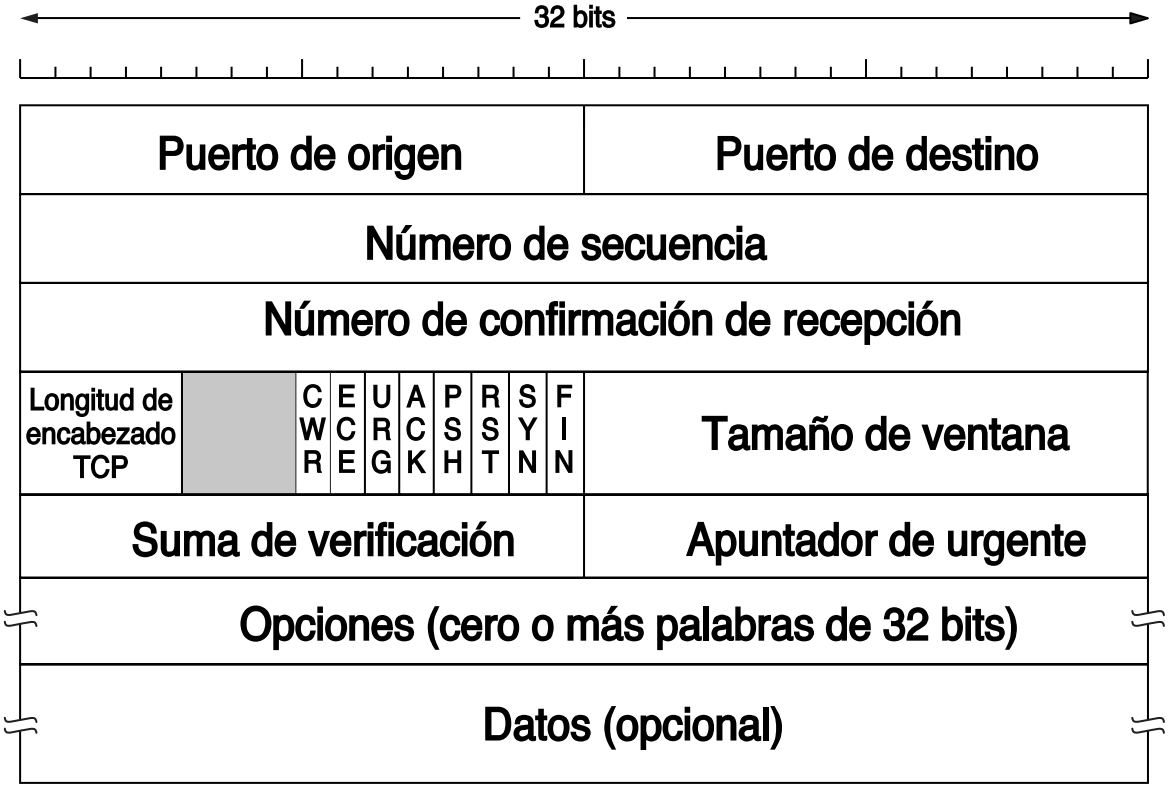


Figura obtenida de: A. Tanenbaum, D. Wetherall, "Redes de computadoras", Quinta edición (2012), pag. 478

Protocolo TCP

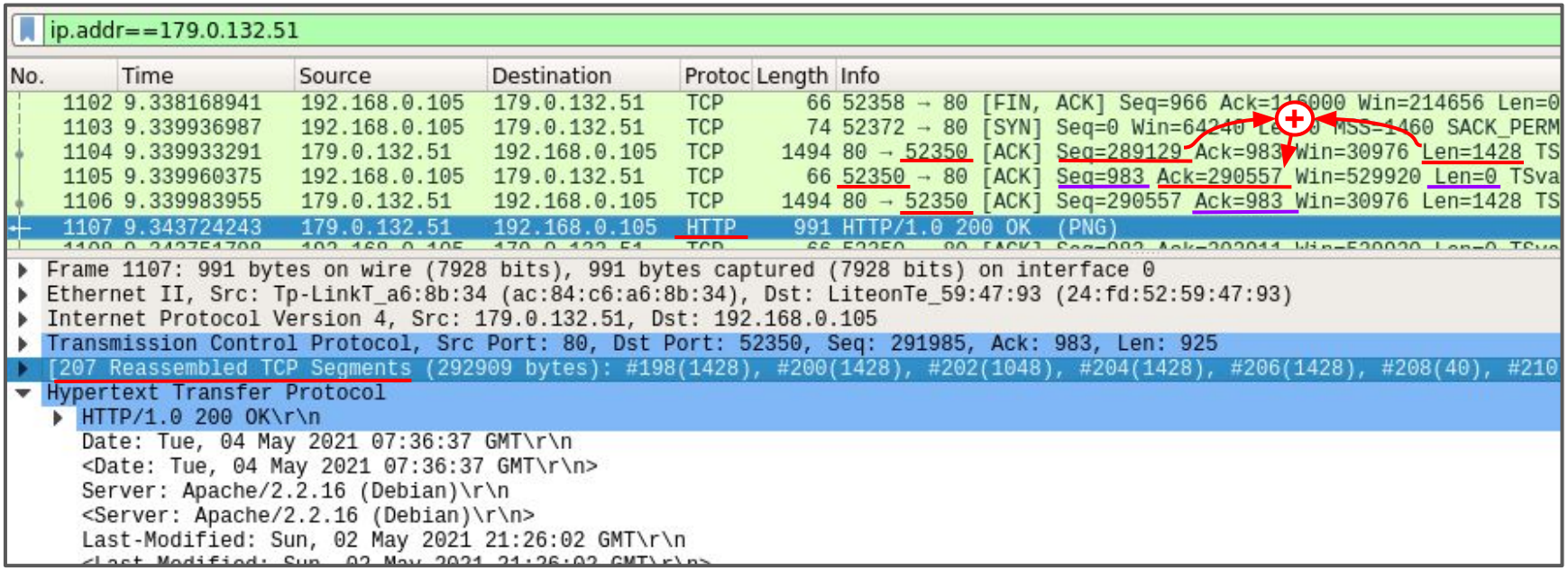
- Número de confirmación de recepción: Confirma el siguiente byte esperado (no el último recibido).
- Longitud del encabezado TCP: Longitud en palabras de 32 bits.
- 4 bits reservados para uso futuro.
- 8 banderas:
  - CWR: El emisor indica que ha reducido su ventana para lidiar con la congestión (es un ACK de la indicación de congestión).
  - ECE: Congestión. El receptor indica que ha recibido un paquete marcado indicando congestión (ver ECN en IP).
  - URG: Datos urgentes (poco usado).
  - ACK: El segmento contiene un ACK (indicado en el campo “Número de confirmación de recepción”)
  - PSH (push): Los datos acumulados en el buffer de transmisión deben ser enviados, y en el receptor, deben entregarse sin demora al proceso receptor (sin almacenarlos en el buffer de recepción).

- 8 banderas (continuación):
  - RST (reset): La conexión debe resetearse (puede haber una confusión en la conexión).
  - SYN: Indica que el segmento es parte de una solicitud de conexión (usado en conjunto con el bit ACK). El segmento puede ser un Connection Request (SYN=1 y ACK=0) o Connection Accepted (SYN=1 y ACK=1).
  - FIN: Se usa para finalizar una conexión.
- Tamaño de ventana: Cantidad de bytes que pueden enviarse.
- Suma de verificación: Incluye encabezados, datos y pseudoencabezado IP.
- Puntero urgente: Puntero desde el byte inicial (número de secuencia) donde terminan los datos urgentes.
- Opciones: Permite agregar opciones (algunas muy utilizadas)
  - Tamaño máximo de segmento que el extremo puede utilizar.
  - Estampa de tiempo: Utilizada para calcular latencias o extender el número de secuencia (para enlaces muy rápidos, 32 bits puede ser muy poco).
  - SACK: Rangos de ACK recibidos.

ip.addr==179.0.132.51

No.	Time	Source	Destination	Protocol	Length	Info
61	7.414512629	192.168.0.105	179.0.132.51	TCP	74	52342 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1
62	7.414610030	192.168.0.105	179.0.132.51	TCP	74	52344 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1
63	7.414929211	192.168.0.105	179.0.132.51	TCP	74	52346 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1
69	7.469818282	179.0.132.51	192.168.0.105	TCP	74	80 → 52342 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1440 S
70	7.469889188	192.168.0.105	179.0.132.51	TCP	66	52342 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=333667032
71	7.470284978	192.168.0.105	179.0.132.51	HTTP	668	GET / HTTP/1.1

```
Sequence number: 0    (relative sequence number)
[Next sequence number: 0    (relative sequence number)]
Acknowledgment number: 0
1010 .... = Header Length: 40 bytes (10)
▼ Flags: 0x002 (SYN)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Nonce: Not set
    .... 0... .... = Congestion Window Reduced (CWR): Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...0 .... = Acknowledgment: Not set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
  ▶ .... .... ..1. = Syn: Set
    .... .... ...0 = Fin: Not set
    [TCP Flags: ·····S·]
Window size value: 64240
[Calculated window size: 64240]
Checksum: 0x6d2a [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0
▼ Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
  ▶ TCP Option - Maximum segment size: 1460 bytes
  ▶ TCP Option - SACK permitted
  ▶ TCP Option - Timestamps: TSval 333666977, TSecr 0
  ▶ TCP Option - No-Operation (NOP)
  ▶ TCP Option - Window scale: 7 (multiply by 128)
▶ [Timestamps]
```



Cuidado: El encabezado TCp no tiene un campo de longitud total o longitud de los datos. El valor de Len lo calcula Wireshark

15	87.022452942	52.0.218.127	192.168.0.105	TLSv1.2	167 Application Data
16	87.022467639	192.168.0.105	52.0.218.127	TCP	66 34288 → 443 [ACK] Seq=1771 Ack=393 Win=501 Len=0 TSval=1117827544 TS…
17	87.022483084	52.0.218.127	192.168.0.105	TLSv1.2	161 Application Data
18	87.022502345	192.168.0.105	52.0.218.127	TCP	66 34288 → 443 [ACK] Seq=1771 Ack=488 Win=501 Len=0 TSval=1117827544 TS…
19	101.007256457	52.0.218.127	192.168.0.105	TCP	66 [TCP Keep-Alive] 443 → 34288 [ACK] Seq=487 Ack=1771 Win=9 Len=0 TSva…
20	101.007290178	192.168.0.105	52.0.218.127	TCP	66 [TCP Keep-Alive ACK] 34288 → 443 [ACK] Seq=1771 Ack=488 Win=501 Len=…
21	121.481088982	52.0.218.127	192.168.0.105	TCP	66 [TCP Keep-Alive] 443 → 34288 [ACK] Seq=487 Ack=1771 Win=9 Len=0 TSva…
22	121.481126578	192.168.0.105	52.0.218.127	TCP	66 [TCP Keep-Alive ACK] 34288 → 443 [ACK] Seq=1771 Ack=488 Win=501 Len=…
23	132.245096359	192.168.0.105	52.0.218.127	TCP	66 [TCP Keep-Alive] 34288 → 443 [ACK] Seq=1770 Ack=488 Win=501 Len=0 TS…
24	132.447402034	52.0.218.127	192.168.0.105	TCP	66 [TCP Keep-Alive ACK] 443 → 34288 [ACK] Seq=488 Ack=1771 Win=9 Len=0 …
25	141.962395473	52.0.218.127	192.168.0.105	TCP	66 [TCP Keep-Alive] 443 → 34288 [ACK] Seq=487 Ack=1771 Win=9 Len=0 TSva…
26	141.962429928	192.168.0.105	52.0.218.127	TCP	66 [TCP Keep-Alive ACK] 34288 → 443 [ACK] Seq=1771 Ack=488 Win=501 Len=…
27	146.802120111	192.168.0.105	52.0.218.127	TLSv1.2	138 Application Data

Interfaz socket de Berkeley

- Es una **interfaz** que permite que un **proceso acceda a la pila de protocolos**.
- El proceso debe crear un **socket**.
  - C, C++: identificador de archivos.
  - Java, C#, Python: Clase.
- El **socket** se identifica mediante una **dirección IP** y un **número de puerto**.
- Fueron liberados por primera vez como parte del sistema operativo BSD[1] (Berkeley Software Distribution).
  - Actualmente utilizado en SO basados en Unix (Por ejemplo, Linux).
  - Windows tiene una API[2] llamada **winsock** similar a socket.

[1] Sistema operativo basado en Unix

[2] Application programming interface

Primitivas de la Interfaz socket

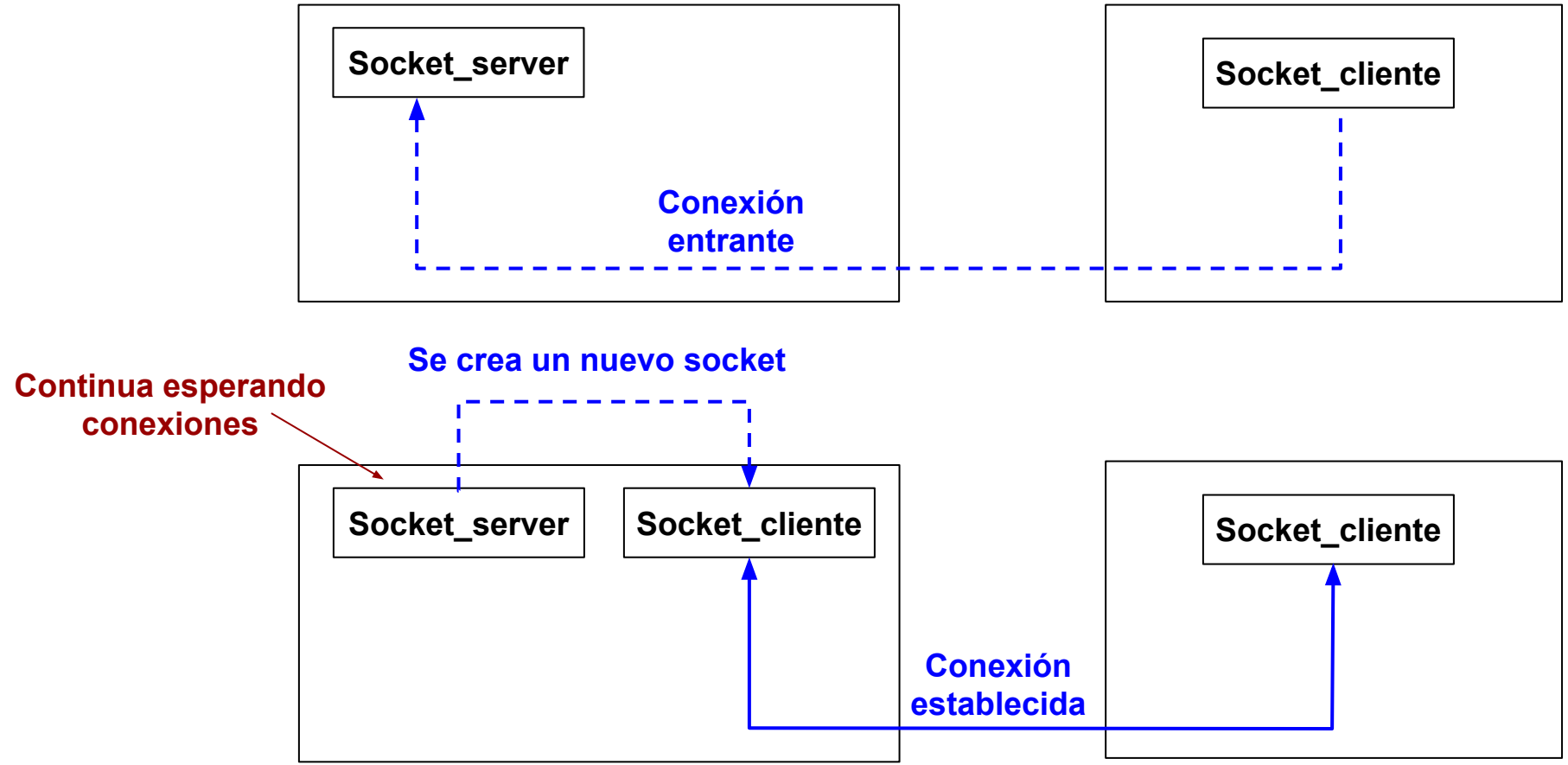
- **Socket**: crea un “punto terminal”, se le asigna espacio de memoria e identificadores en el software de transporte.
  - Parámetros:
    - Formato de direccionamiento.
    - Tipo de servicio.
    - Protocolo.

Primitivas de la Interfaz socket usadas por el proceso que espera conexiones

- **Bind**: Asigna una dirección al socket (una dirección IP y un puerto).
  - Necesaria para sockets TCP que recibirán conexiones o sockets UDP.
  - No necesaria para sockets TCP que comenzarán una conexión.
- **Listen**: Anuncia la disposición para aceptar conexiones (asigna espacio de memoria para almacenar peticiones de conexión). **Solo TCP**.

Primitivas de la Interfaz socket usadas en aplicaciones servidor

- **Accept**: queda en espera de conexiones TCP entrantes. Es una primitiva **bloquante** (bloquea la ejecución del proceso hasta que llega una conexión entrante).
  - Cuando llega una conexión entrante, crea un nuevo socket y devuelve un objeto o descriptor para identificarlo.
    - El nuevo socket está conectado virtualmente al que pidió la conexión.
    - El socket original puede seguir usándose para esperar otras conexiones.
    - Si se va a permitir la conexión de varias aplicaciones clientes, es usual crear nuevos hilos o procesos por cada conexión creada.


Socket_server
Socket_cliente
Conexión entrante
Se crea un nuevo socket
Continua esperando conexiones
Socket_server
Socket_cliente
Socket_cliente
Conexión establecida

Primitivas de la Interfaz socket usadas en aplicaciones cliente

- **Connect**: comienza el procedimiento de conexión. Es una primitiva bloqueante (bloquea al proceso hasta que la conexión es exitosa o fallida).
  - Cuando se desbloquea, se establece la conexión.
- **Send** y **Receive**: enviar y recibir datos a través de una conexión ya establecida. Son primitivas bloqueantes. **Solo TCP**.
- **Send-to** y **Receive-from**: enviar y recibir datos a través de sockets no conectados (**UDP**). Receive-from es bloqueante.
- **Close**: Se libera la conexión (debe liberarse en ambos lados).

Ejemplo socket TCP Servidor en Python

```
socket_server=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
socket_server.bind(('192.168.1.39',1501))
socket_server.listen(5)
(socket_cliente, address) = socket_server.accept()
……
…… enviar y recibir datos……
……..
socket_cliente.close()
```

Ejemplo socket TCP Cliente en Python

```
socket_id=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
socket_id.connect(('192.168.1.39',1500))
……
…… enviar y recibir datos……
……..
socket_server.close()
```