Contenido 10: Capa Transporte

Lesly Reyes
lareyes2@uc.cl

Álvaro Quintana
alvaroq@uc.cl

Departamento de Ciencias de la Computación
Pontificia Universidad Católica Chile

October 16, 2024

Contenido

Capa Transporte
- Protocolos de Transporte
- TCP UDP
- Multiplexión
- Demultiplexión
- UDP
- Medios no confiables
- Transport Control Protocol TCP
- TCP Handshake

Material Adicional

Protocolos de Transporte

Objetivo: Conseguir que un mensaje llegue desde emisor a receptor

- **Emisor:** Recibe un mensaje de capa de aplicación, lo divide en segmentos, y solicita a la capa de red que envíe los segmentos al receptor. Routers no examinan headers de transporte.
- **Receptor:** Recibe segmentos, los ensambla para formar el mensaje, y lo entrega a la capa de aplicación.

Dos protocolos

TCP-UDP

- **TCP:** Entrega confiable y ordenada. Control de flujo. Establecimiento de conexión.
- **UDP:** Entrega no confiable y desordenada. Más rápida.

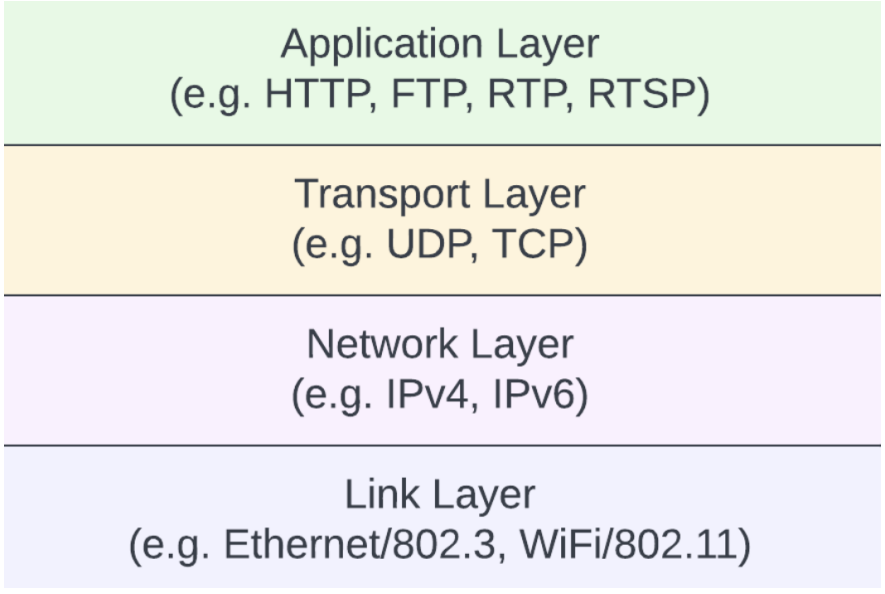
TCP



UDP



Protocolos y Capa donde se encuentran



Protocolos de Transporte

Segmentos

Mensaje se divide en segmentos $\rightarrow$ Segmentos se transmiten en paquetes IP.
¿Cómo identificar mensajes a distintos procesos?

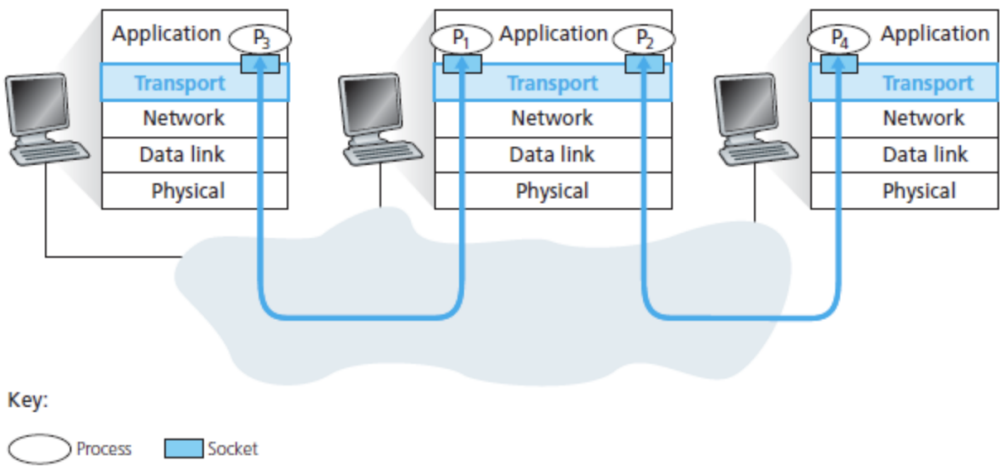
- **Puerto** permite identificar procesos dentro del nodo emisor o del nodo receptor.
- **Puerto** permite multiplexar mensajes

Protocolos de Transporte

Multiplexión

Proceso envían mensajes a través de sockets del Sistema Operativo

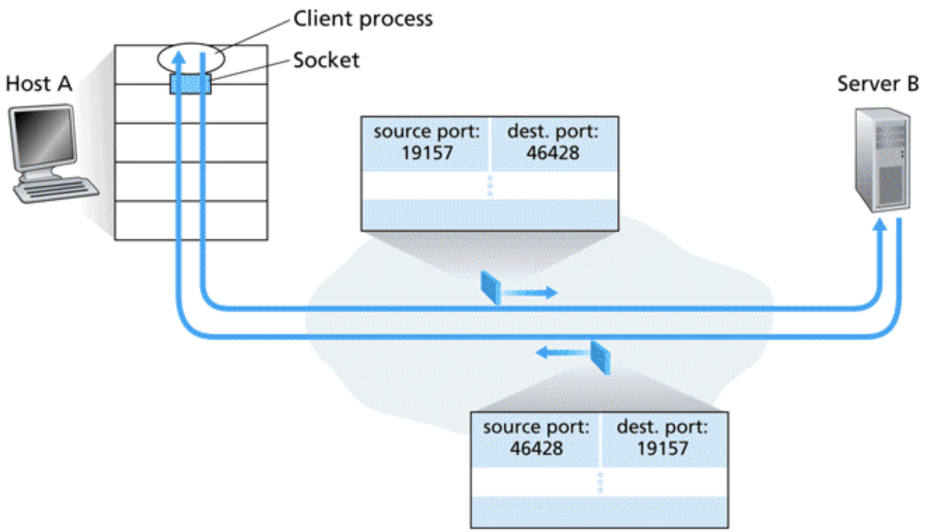
- **Cada socket** se crea con un número de puerto específico
- **Cada host** puede tener múltiples sockets abiertos



Sin Conexión

Segmento UDP identificado por: ⟨IP destino, puerto destino⟩

- Receptor observa el puerto de destino
- Pasa el segmento al proceso asociado al puerto de destino



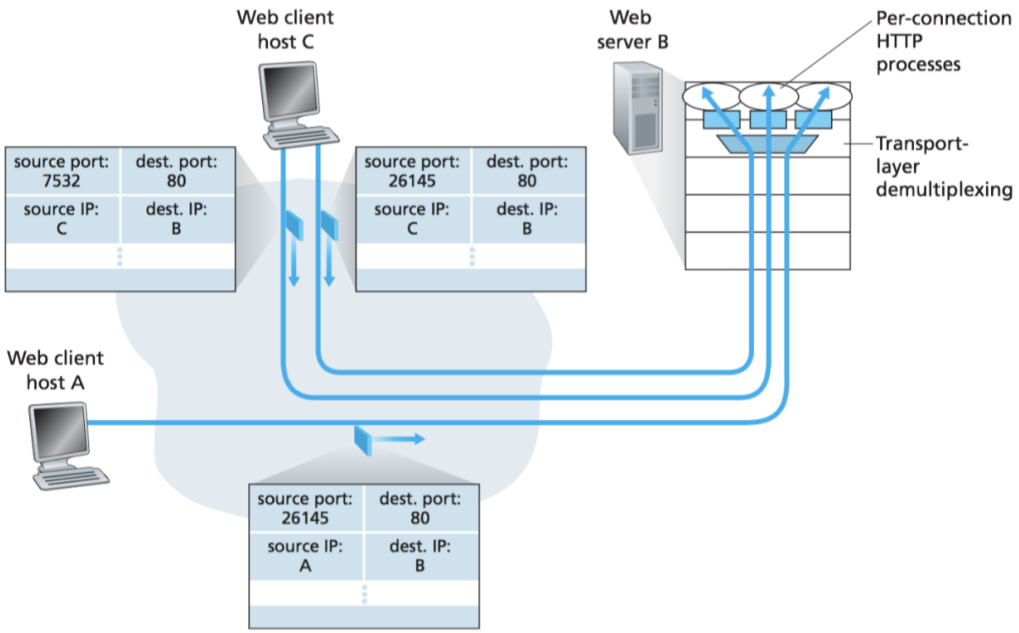
Protocolos de Transporte

Demultiplexión CON conexión

Con Conexión

Segmento TCP identificado por: ⟨IP origen, puerto origen, IP destino, puerto destino⟩

- Servidores pueden gestionar múltiples conexiones TCP
- Se pueden asociar distintos sockets a distintos clientes



UDP

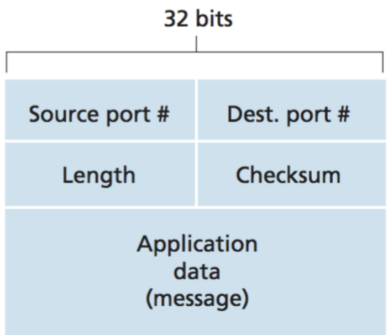
User Datagram Protocol

- Servicio de "mejor esfuerzo" (best-effort).
- Segmentos pueden perderse.
- Segmentos pueden llegar en distinto orden.

Servicio no orientado a conexión (connection-less)

- No se establece conexión previa
- No mantiene estado de conexión
- Cada segmento se gestiona de manera independiente

Segmento UDP solo tiene 8 byte de overhead



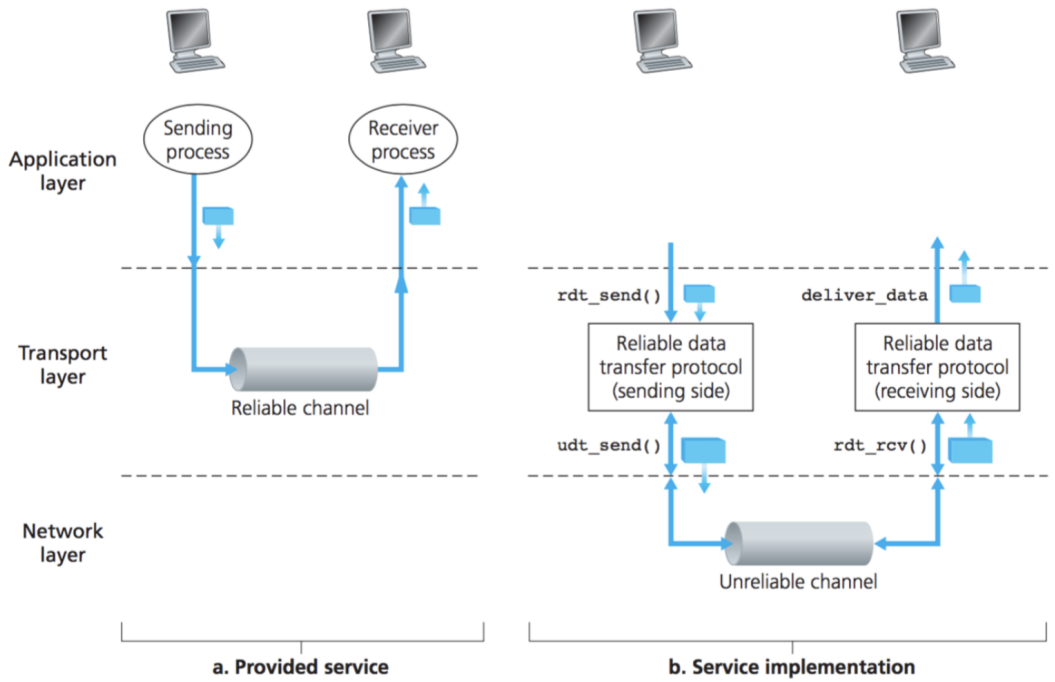
¿Por qué mejor UDP?

- Aplicación puede controlar directamente el envío
- No hay demora en establecer conexión
  - Menor overhead de tiempo
- No se reservan recursos
- Menor overhead de espacio: 8 byte
  - TCP tiene 20 byte

Desafío

¿Cómo podemos transferir datos de forma confiable?

El problema es que no todos los medios son confiables

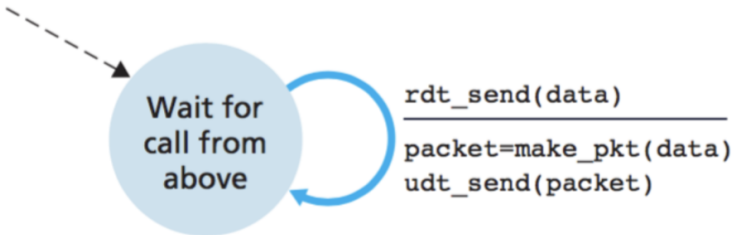


Tipos de comunicación

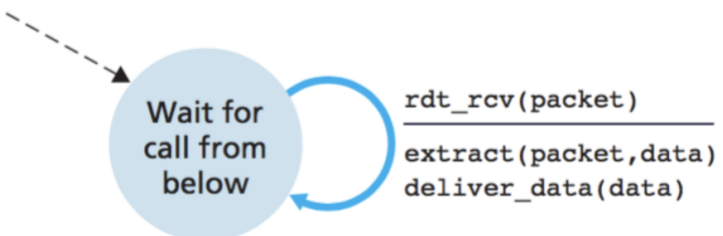
Versión 1.0. Canal de transmisión confiable

Se trabaja con dos supuestos

- Sender: Al recibir solicitud de envío, crea un paquete y envía
- Receiver: Al recibir paquete, extrae mensaje y lo pasa a aplicación



a. rdt1.0: sending side



b. rdt1.0: receiving side

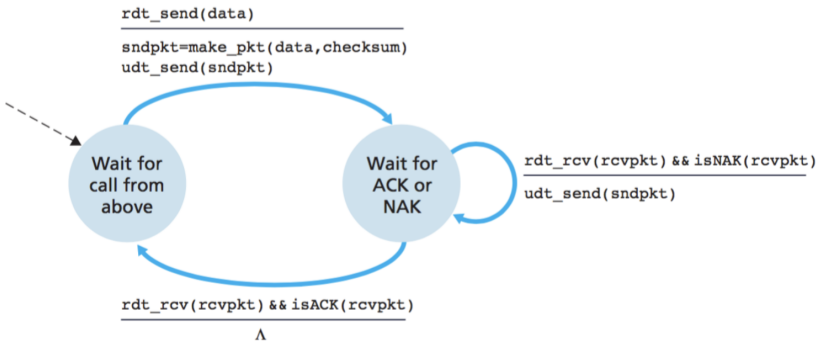
Problema: Los paquetes pueden llegar con errores

Tipos de comunicación

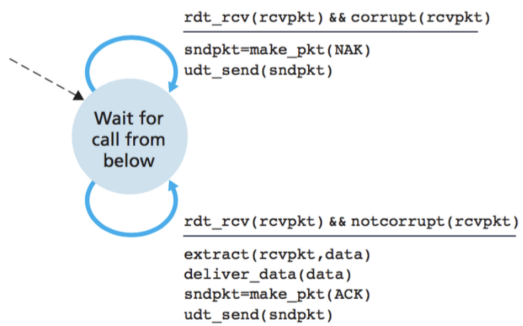
Versión 2.0. Protocolo ARQ (Automatic Repeat reQuest)

Protocolo ARQ (Automatic Repeat reQuest)

- Detección de errores. Envía checksums.(Sol a que los paquetes lleguen con errores)
- Paquetes de feedback. ACK (acknowledgment) y NAK (negative ACK)
- Retransmisiones.



a. rdt2.0: sending side



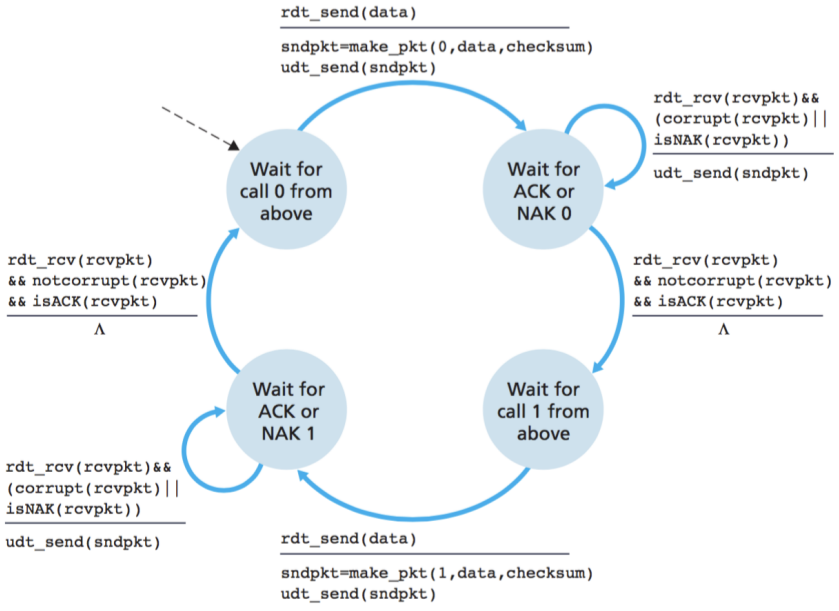
b. rdt2.0: receiving side

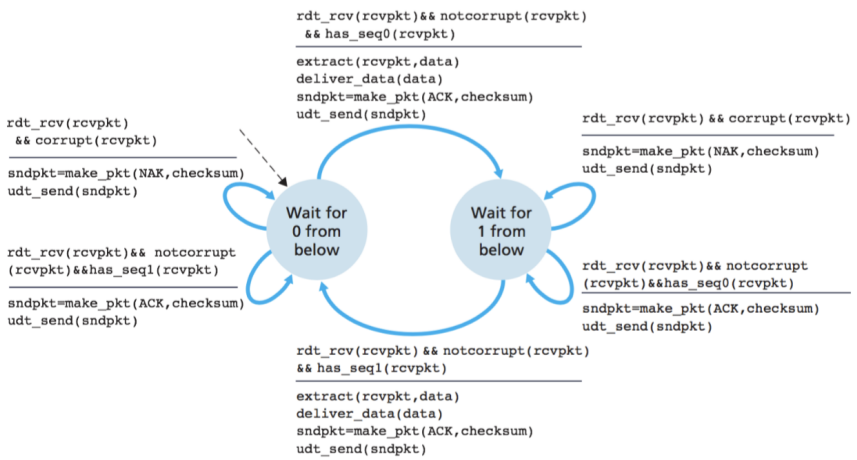
¿Retransmitir al recibir un ACK o NAK corrupto? Pueden llegar duplicados.

Tipos de comunicación

Versión 2.1. Agregar un sequence number. Por ahora, 1-bit.

Agregar un sequence number. Por ahora, 1-bit.



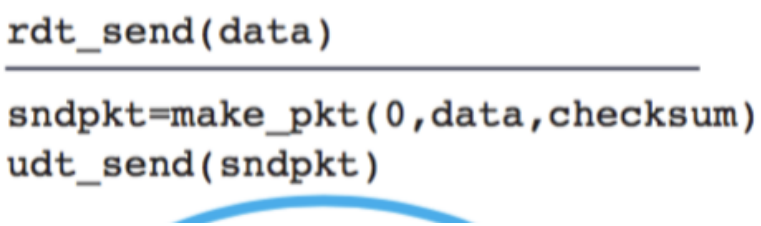


Tipos de comunicación

Versión 2.1. Agregar un sequence number. Por ahora, 1-bit.

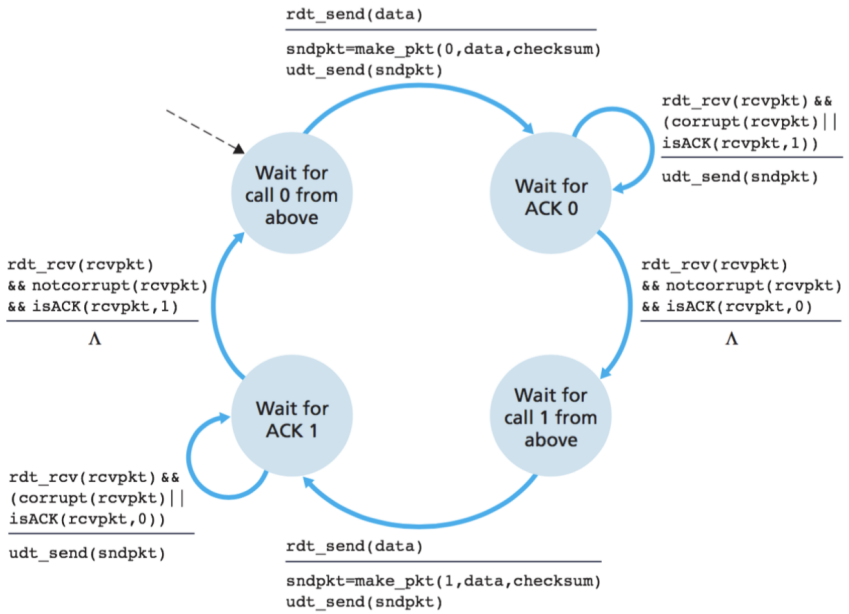
Agregar un sequence number. Por ahora, 1-bit.

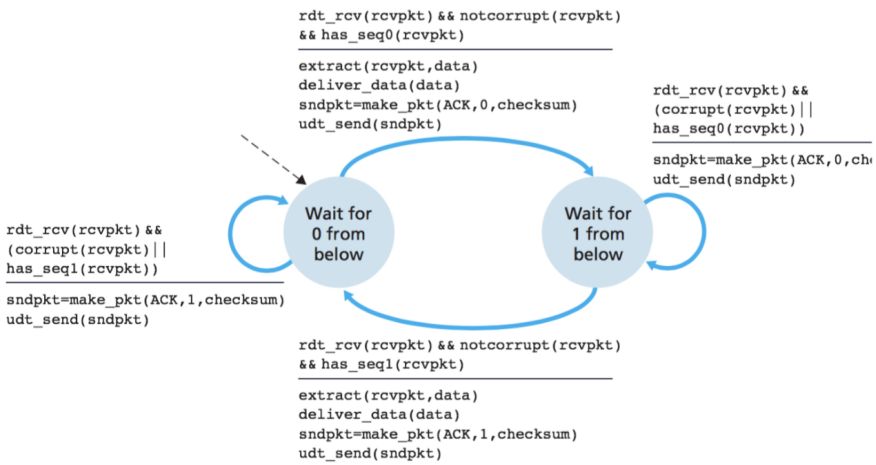
- make_pkt(0,data,checksum)



¿Cuál es el problema de mandar el ack?

Versión 2.2. Podemos eliminar los NAK. Cada ACK adjunta el sequence number





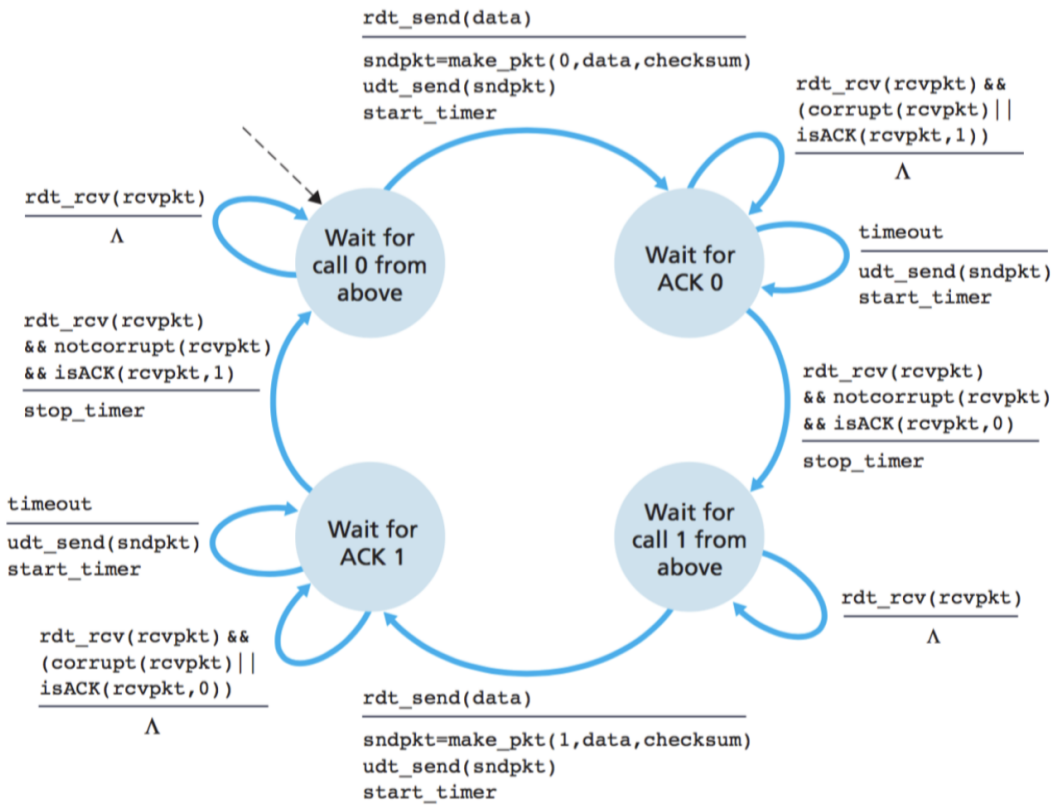
¿Y si se pierden paquetes?

Tipos de comunicación

Versión 3.0 Timeout para retransmisión en el sender

Versión 3.0. Timeout para retransmisión en el sender

- Se implementa en el sender



¿Y si se pierden paquetes?

Tipos de comunicación

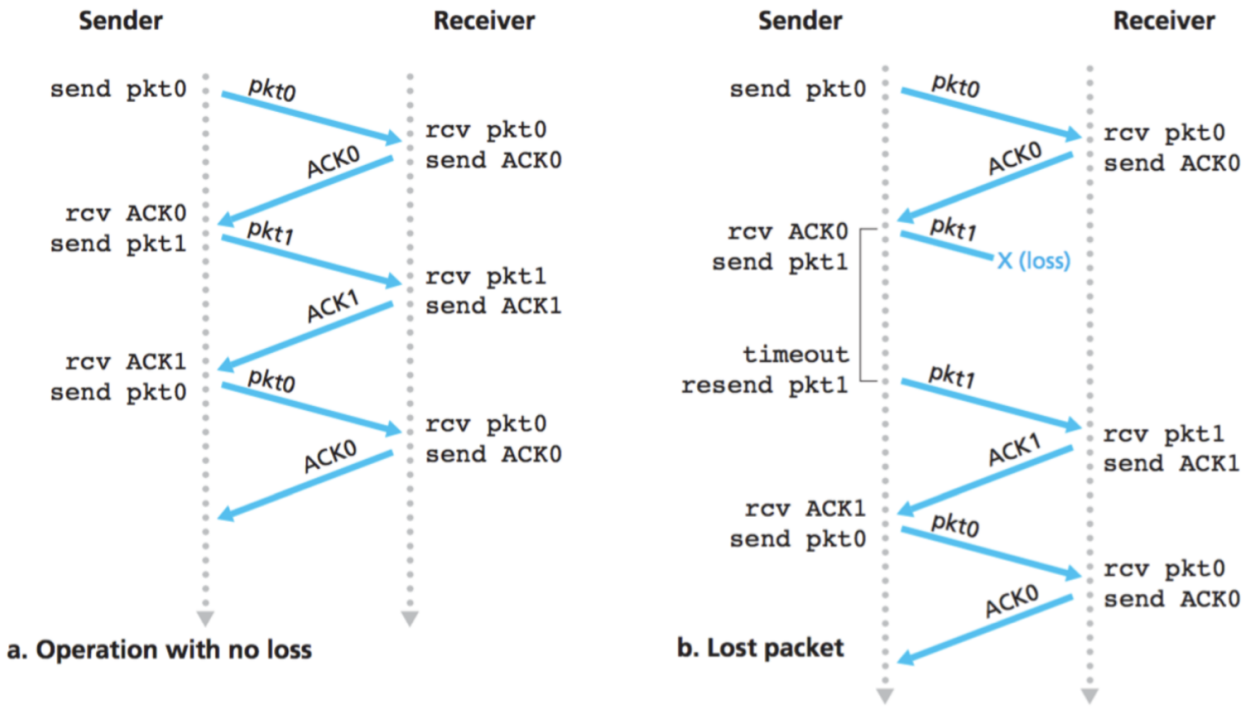
Resumen de las versiones

- Versión 1. Tiene los supuestos, pero los paquetes pueden llegar con errores
- Versión 2. Implementa los checksums, pero pueden haber paquetes de feedback duplicados(ACK O NAK).
- Versión 2.1. Agrega un bit que es el sequence number.
- Versión 2.2. Se pueden eliminar los NAK.
- Versión 3.0. Funciona pero el desempeño es malo

Tipos de comunicación

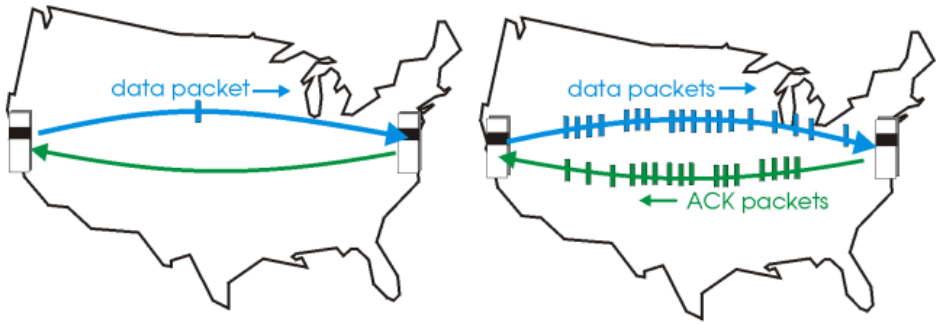
Funcionamiento sin perdida/con perdida

Funcionamiento sin perdida de paquetes a lado izquierdo, con perdida lado derecho



Protocolos con Pipeline

Protocolos mandan de un paquete a la vez (stop and wait). Se envían varios paquetes simultáneos en modo pipeline

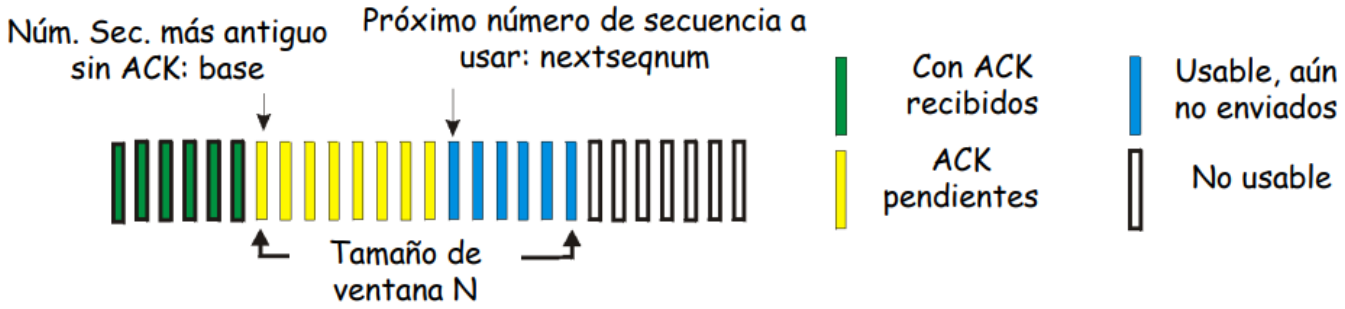


(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

Se necesitan más características.

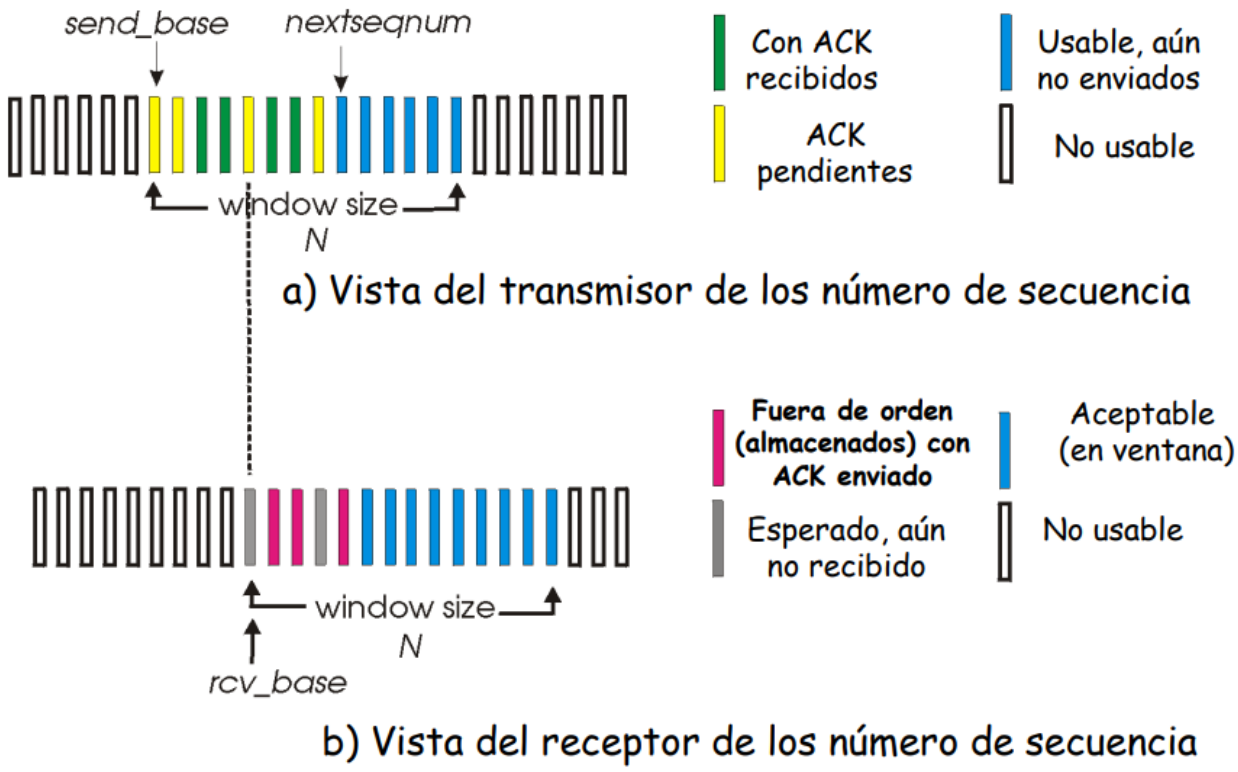
- Números de secuencia deben ser incrementales
- Tanto emisor como receptor necesitan buffers
- Dos enfoques para manejar paquetes: Go-Back-N, Selective Repeat

Go-Back-N



- Emisor puede mantener hasta N paquetes sin ACK
- Receptor envía ACK por el último paquete recibido correctamente
- Emisor usa timer para paquete más antiguo sin ACK
- Ejemplo con N=4

Selective Repeat (repetición selectiva)

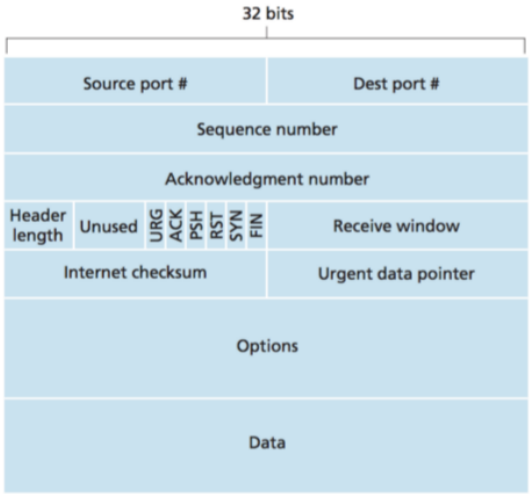


- Receptor envía acuse de recibo individuales de todos los paquetes recibidos
- Transmisor sólo re-envía los paquetes sin ACK recibido. Usa un timer para cada paquete sin ACK. Si el timer expira reenvia los paquetes sin ACK.

TCP

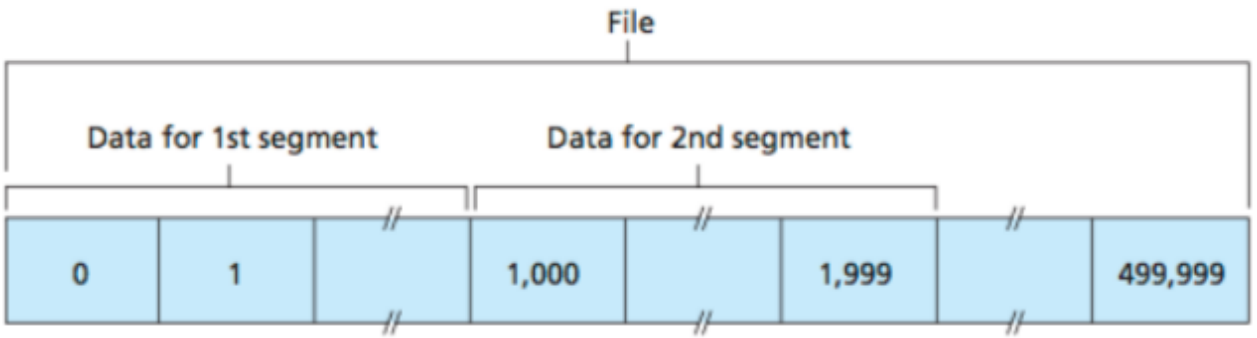
- Protocolo de transmisión fiable
- Transmisión encadenanada chained
- Control de flujo a través del tamaño de la ventana
- Buffer en emisor y en receptor
- Orientado a conexión

  Protocolo de establecimiento de conexión (handshake) antes de enviar paquetes de datos



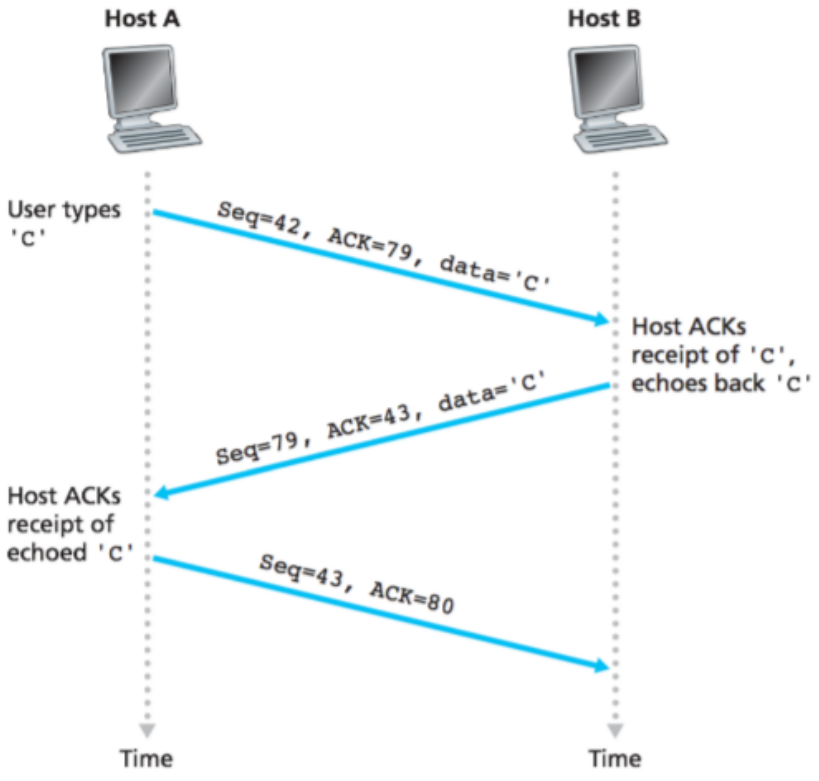
TCP

- Tamaño máximo de segmento: MSS, Maximum Segment Size
- Números de secuencia además cuentan bytes en la secuencia
- ACK contienen el sequence number del siguiente byte que se espera. Cumulative ACKs.
- Mensaje de respuesta funciona también como ACK. Piggybacked ACK



TCP

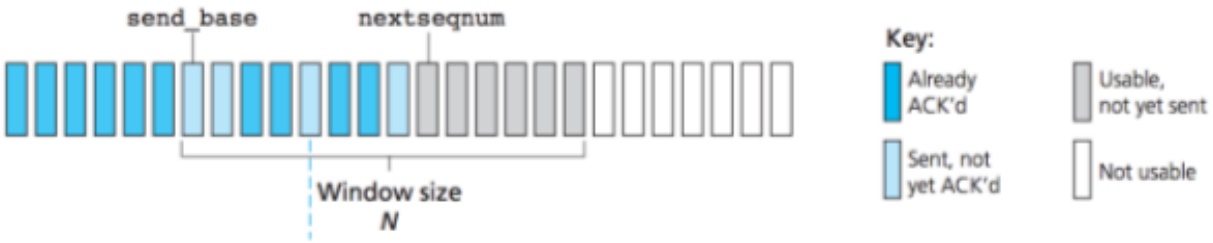
Algoritmo de transmisión fiable de TCP



Ejemplo: TELNET

TCP

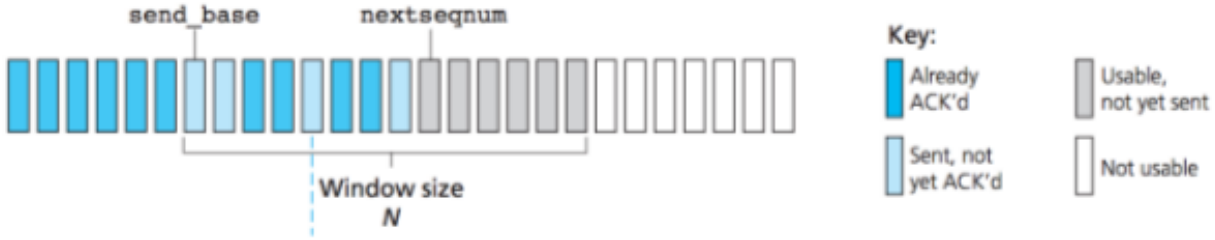
Algoritmo de transmisión fiable de TCP



- SendBase es la base de la ventana
- NextSeqNum se actualiza al crear un segmento nuevo para enviar
- Al expirar el timer se retransmite el segmento con menor sequence number.
    - El paquete enviado, o el ACK pueden haberse perdido

TCP

Algoritmo de transmisión fiable de TCP



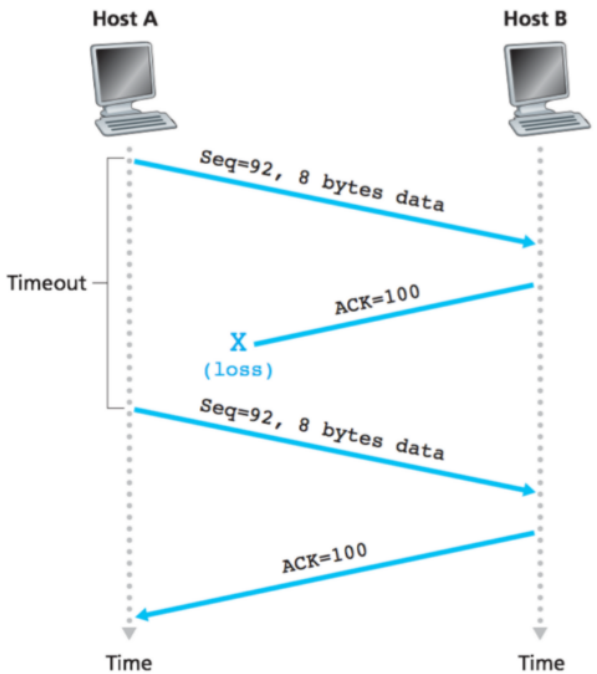
- Si llega un ACK con valor mayor a SendBase, se actualiza SendBase
    - Esto significa que ya se han recibido los paquetes con valor menor a SendBase
    - Puede haberse perdido el ACK de los paquetes anteriores
    - Este es el funcionamiento del cumulative ACK

TCP

Caso: ACK perdido y retransmisión

Si el ACK se pierde, se produce timeout

- A reenvía el paquete más antiguo sin ACK (Seq=92)
- A reinicia el timer
- B recibe Seq=92 duplicado. Lo descarta, pero reenvía ACK

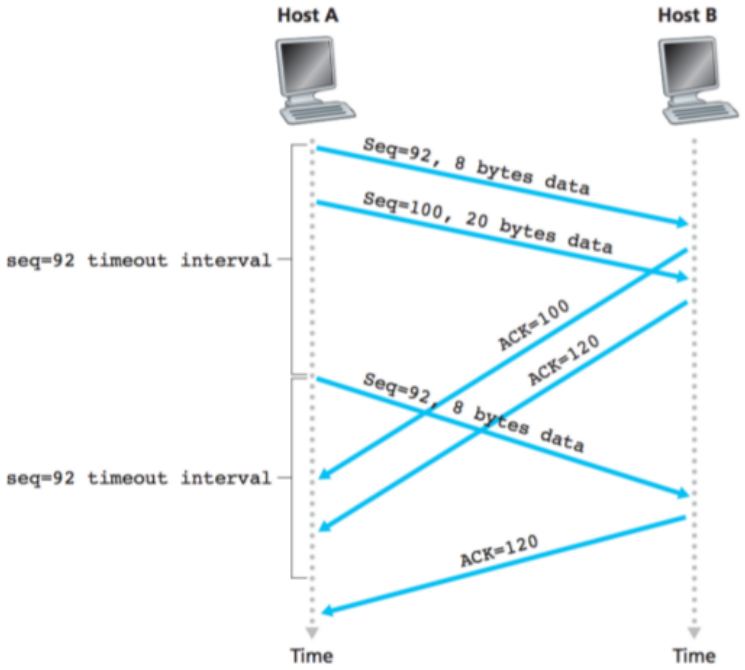


TCP

Caso: Timeout prematuro

Timeout ocurre antes que llegue el ACK

- A reenvía el paquete más antiguo sin ACK (Seq=92), y reinicia timer (Seq=92)
- A recibe ACK y reinicia timer, por Seq=100
- A NO reenvía Seq=100, ya que su ACK llega antes del timeout
- B recibe Seq=92 duplicado. Lo descarta, pero envía (cumulative) ACK

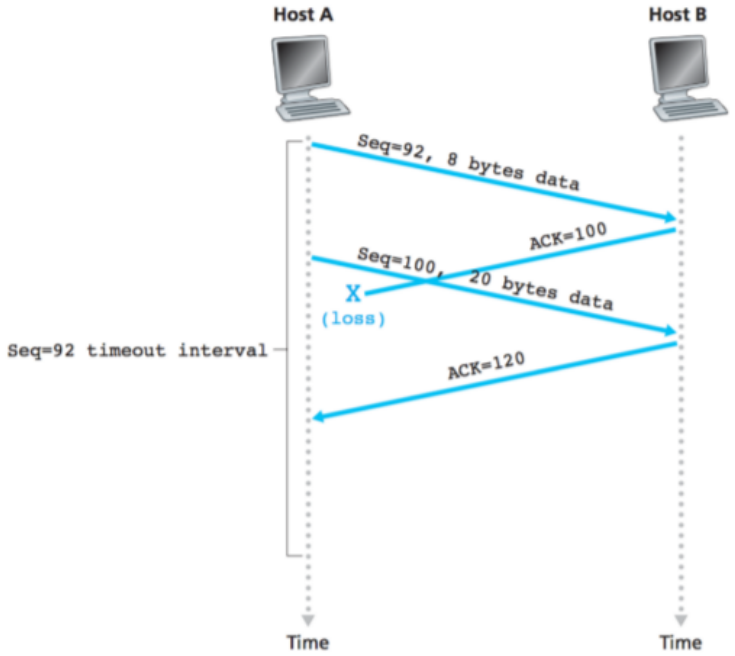


TCP

Caso: Cumulative ACK evita retransmisión

ACK por Seq=92 se pierde

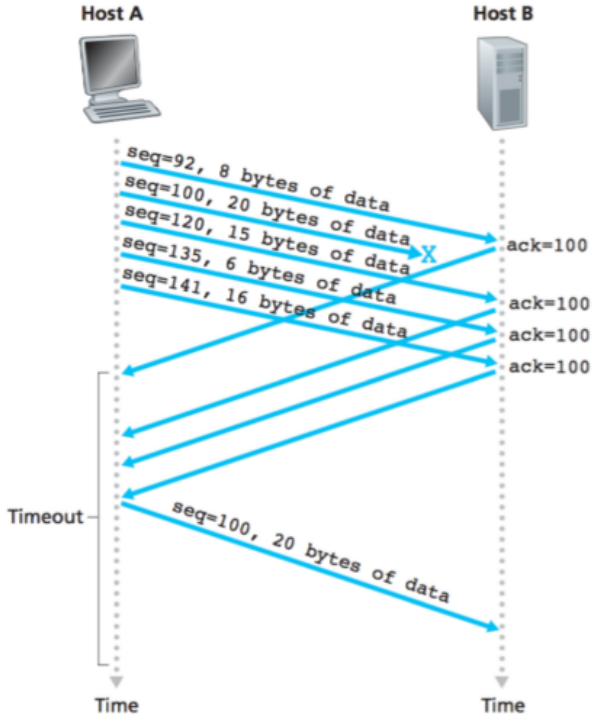
- El ACK=120 (ACK de Seq=100) llega antes del timeout de Seq=92
- Cumulative ACK (ACK=120) indica que se ha recibido el Seq=92 y el Seq=100
- A NO reenvía Seq=92
- A incrementa SendBase



TCP

Retransmisión rápida

- Timeouts pueden tomar mucho tiempo
- Retransmisión rápida envía proactivamente segmentos probablemente perdidos, antes que ocurra un timeout



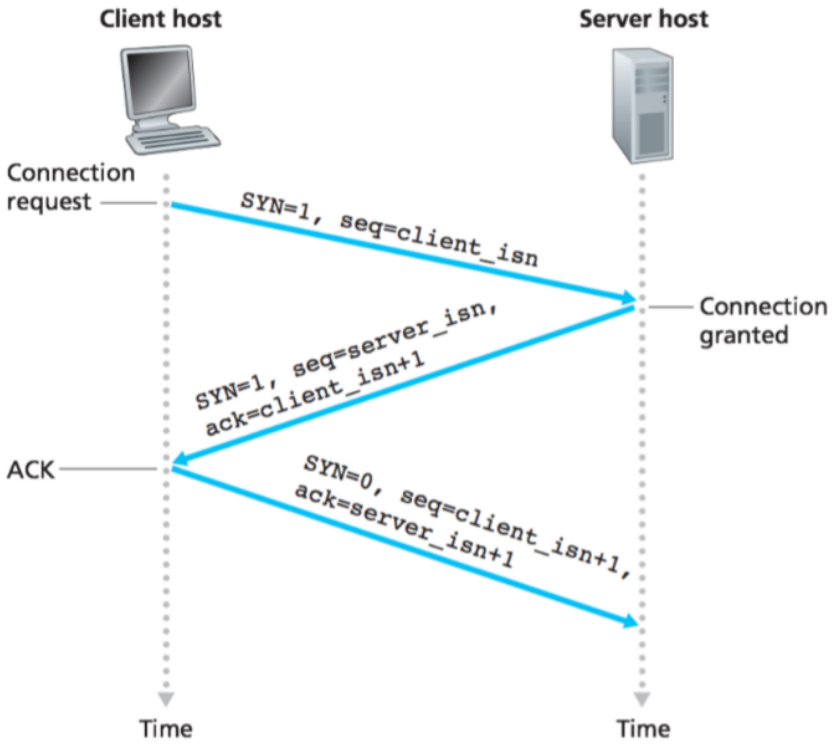
TCP

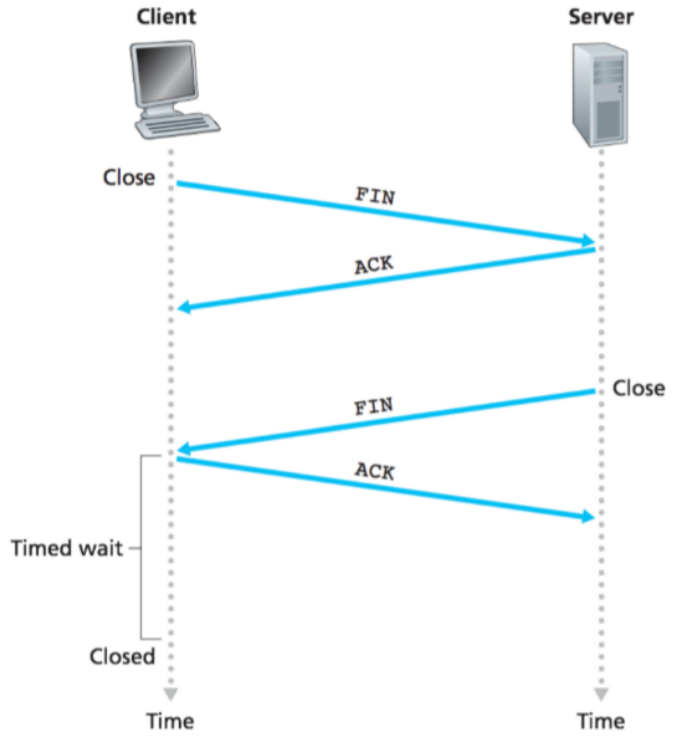
TCP HANDSHAKE PROTOCOL

Establecimiento de conexión TCP

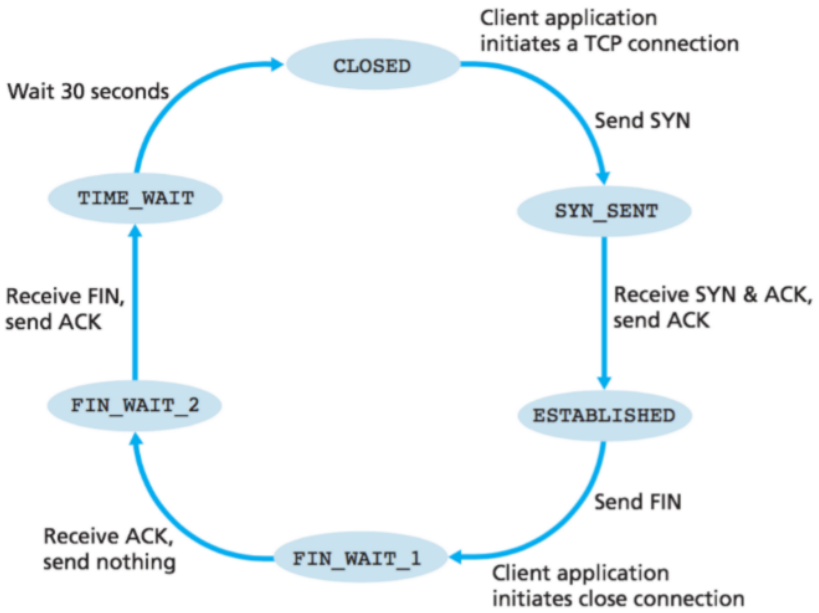
- Paso 1. Cliente envía segmento SYN al servidor
  - Incluye número de secuencia inicial (de cliente)
- Paso 2. Servidor recibe SYN, y responde con SYN ACK
  - Servidor asigna buffer
  - Servidor establece número de secuencia inicial (de servidor)
- Paso 3. Cliente recibe SYN ACK, y responde con ACK
  - Este paquete ya puede contener datos

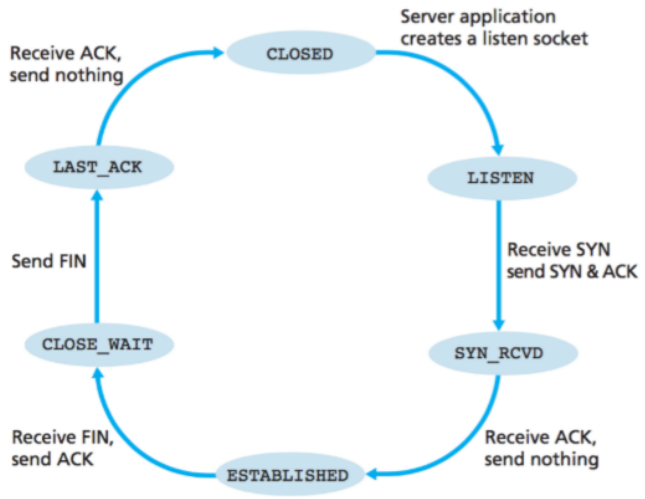
Establecimiento de conexión TCP





Cliente y servidor





Material Adicional

- Video Capa Transporte: Link Video

Próximo contenido a revisar será Capa Red