Agenda

- 3.1 Servicios de la capa de transporte
- 3.2 Multiplexación y demultiplexación
- 3.3 Transporte no orientado a conexión: UDP
- 3.4 Principios de la transferencia de datos confiable
- 3.5 Transporte orientado a conexión: TCP
  - estructura del segmento
  - transferencia de datos confiable
  - control de flujo
  - gestión de la conexión
- 3.6 Principios del control de congestión
- 3.7 Control de congestión de TCP

TCP: un poco de historia…

- *Transmission Control Protocol*
- Vinton Cerf y Robert Kahn
  - mayo del 74
    - *A Protocol for Packet Network Intercommunication*
    - Antes del PC, de las estaciones de trabajo, de Ethernet por todos lados,…
  - septiembre del 81
    - RFC **793**: Especificación de TCP
  - octubre del 89
    - RFC **1122**: Requerimientos para los *hosts* de Internet
  - mayo del 92
    - RFC **1323**: Extensiones a TCP
  - octubre del 96
    - RFC 2018: Selective ACK

TCP: un poco de historia (2)…

- diciembre del 98
  - RFC 2460: TCP con IPv6
- abril del 99
  - RFC **2581**: Control de congestión de TCP
- junio del 00
  - RFC 2873: Interacción de TCP y IP ("bits de precedencia")
- noviembre del 2000
  - RFC **2988**: *timer* de retransmisión
- septiembre del 06
  - RFC 4614: hoja de ruta de RFCs relacionadas con TCP

TCP: Aspectos generales

- punto a punto:
  - un *sender*, un *receiver*
  - entre *end systems*
- flujo de bytes en orden y confiable:
  - no alineado a bordes de mensajes
- *pipelined*:
  - El control de congestión y el control de flujo de TCP definen el tamaño de ventana
- *Buffers* para enviar y recibir

- servicio de datos *full duplex*
  - flujo de datos en ambas direcciones en la misma (única) conexión
  - MSS: *maximum segment size*
- orientado a conexión:
  - *handshaking* (intercambio de mensajes de control) inicializa el estado del *sender* y del *receiver* previo al intercambio de datos
- flujo controlado:
  - el *sender* no abruma al *receiver*

TCP: estructura del segmento

TCP: Nros. de secuencia y ACKs

Nros. de secuencia:
- *byte stream* número del primer byte de datos en el segmento

ACKs:
- Nro. de sec. del próximo byte esperado desde el otro lado
- ACK acumulativo

P: ¿Cómo maneja el receptor la llegada de segmentos fuera de orden?
- A: La especificación de TCP nada dice. Queda librado a la implementación de turno.

TCP: *Timeout* y *Round Trip Time*

P: ¿Cómo definir el valor del *timeout* de TCP?

- mayor al RTT
  - pero el RTT cambia
- demasiado corto: *timeout* prematuro
  - retransmisiones innecesarias
- demasiado largo: lenta reacción a pérdida de un segmento

P: ¿Cómo estimar el RTT?

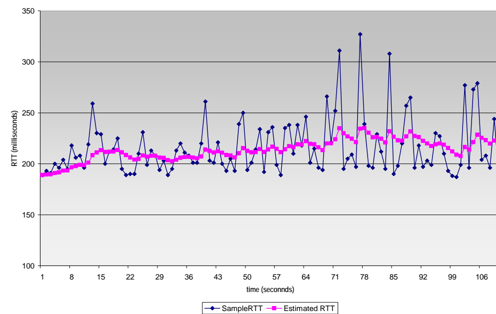
- `MuestraRTT`: tiempo medido desde la transmisión de un segmento hasta la recepción de su ACK
  - ignorando retransmisiones
- `MuestraRTT` varía, buscamos estimar un RTT "suavizado"
  - Promedio de varias medidas recientes, no solamente el valor actual de `MuestraRTT`

TCP: *Timeout* y *Round Trip Time*

`EstimaciónRTT = (1- α)* EstimaciónRTT + α * MuestraRTT`

- La influencia de las muestras anteriores decrece exponencialmente
- Valor típico: $\alpha = 0.125$
- *"Congestion avoidance y control" – Van Jacobson & Michael Karels – November, 1988*

Ejemplo de la **estimación** del RTT



TCP: *Timeout* y *Round Trip Time*

Estableciendo el *timeout*

- `EstimaciónRTT` más "un margen de seguridad"
  - Variación amplia en `EstimaciónRTT` -> mayor margen de seguridad
- primero contempla cuánto la `MuestraRTT` se aparta de la `EstimaciónRTT`:

```
DesvRTT = (1-β)* DesvRTT +
           β * |MuestraRTT-EstimaciónRTT|

(típicamente, β = 0.25)
```

Entonces se establece el intervalo de *timeout*:

```
IntervaloTimeout = EstimaciónRTT + 4 * DesvRTT
```


Agenda

- 3.1 Servicios de la capa de transporte
- 3.2 Multiplexación y demultiplexación
- 3.3 Transporte no orientado a conexión: UDP
- 3.4 Principios de la transferencia de datos confiable
- 3.5 Transporte orientado a conexión: TCP
  - estructura del segmento
  - transferencia de datos confiable
  - control de flujo
  - gestión de la conexión
- 3.6 Principios del control de congestión
- 3.7 Control de congestión de TCP

TCP: transferencia de datos confiable

- TCP crea un servicio **rdt** sobre el servicio no confiable de IP
- Segmentos *pipelined*
- ACKs acumulativos
- TCP utiliza un único *timer* de retransmisión
- Las retransmisiones son disparadas por:
  - eventos *timeout*
  - ACKs duplicados
- Inicialmente consideramos un transmisor TCP simplificado:
  - ignora ACKs duplicados
  - ignora control de flujo y control de congestión
  - los datos son menores a MSS

TCP: eventos del emisor

Datos recibidos desde Aplicación

- Crea segmento con número de secuencia
- El número de sec. es el número del primer byte de datos en el segmento
- lanza *timer* si ya no estaba corriendo (pensar al *timer* asociado al segmento más viejo no-ACKed)
- Intervalo de expiración: `IntervaloTimeout`

timeout:

- Retransmite el segmento que causó el *timeout*
- Re-lanza el *timer*

Ack recibido:

- Si reconoce segmentos no reconocidos antes
  - Actualiza la información de segmentos reconocidos
  - lanza el timer si hay segmentos no-ACKed


```
NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum

loop (forever) {
  switch(event)

  event: data received from application above
      create TCP segment with sequence number NextSeqNum
      if (timer currently not running)
          start timer
      pass segment to IP
      NextSeqNum = NextSeqNum + length(data)

  event: timer timeout
      retransmit not-yet-acknowledged segment with
           smallest sequence number
      start timer

  event: ACK received, with ACK field value of y
      if (y > SendBase) {
          SendBase = y
          if (there are currently not-yet-acknowledged segments)
               start timer
          }

} /* end of loop forever */
```

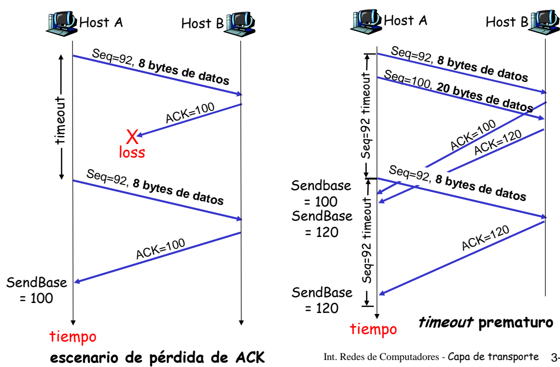
Emisor TCP (simplificado)

Comentario:
- SendBase-1: last cumulatively ack'ed byte

Ejemplo:
- SendBase-1 = 51; y= 83, entonces el receptor busca 83+ ;

y > SendBase, Nuevos datos son ACKed

TCP: escenarios de retransmisión



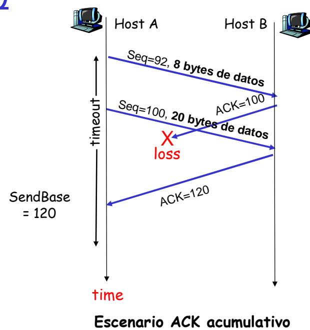
escenario de pérdida de ACK

Host A | Host B

Seq=92, 8 bytes de datos
Seq=100, 20 bytes de datos
ACK=100
ACK=120
Seq=92, 8 bytes de datos
ACK=120
Seq=92 timeout
Sendbase = 100
SendBase = 120
SendBase = 120
tiempo

timeout prematuro

TCP: escenarios de retransmisión (más)



Escenario ACK acumulativo

TCP: generación de ACK

[RFC 1122, RFC 2581]

Evento en el Receptor	Acción de Receptor TCP
Llega segmento en orden, con nro. de sec. esperado. Todos los datos hasta el nro. de sec. esperado ya están ACKed	Retrasar ACK. Espera hasta 500ms por el sig. segmento. Si no hay sig. segmento en ese intervalo, enviar ACK
Llega un segmento en orden, con nro. de sec. esperado. Otro segmento en orden tiene ACK pendiente	Envía inmendiatamente un ACK acumulativo, Reconociendo ambos segmentos en orden
Llega un segmento fuera de orden con nro. de sec. mayor al esperado. *Gap* detectado	Envía inmediatamente ***duplicate ACK***, indicando nro. de sec. del sig. byte esperado (límite inferior del *gap*)
Llega un segmento que parcial o completamente llena el *gap*.	Envío inmediato de ACK, si dicho segmento comienza en el extremo inferior del *gap*

Fast Retransmit

- El período de *timeout* generalmente es largo:
  - Delay largo antes de re-enviar el paquete perdido
- Detectar segmentos perdidos a través de ACKs duplicados.
  - El *sender* frecuentemente envía varios segmentos de manera consecutiva
  - Si el segmento se pierde, podremos recibir varios ACKs duplicados
- Si el *sender* recibe 3 ACKs duplicados para los mismos datos, supone que el segmento de datos posterior al ACKed se perdió:
  - fast retransmit: re-enviar el segmento antes que expire el *timer* de retransmisión

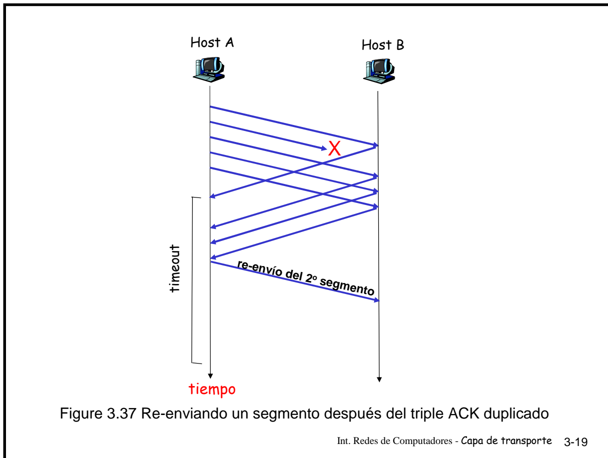
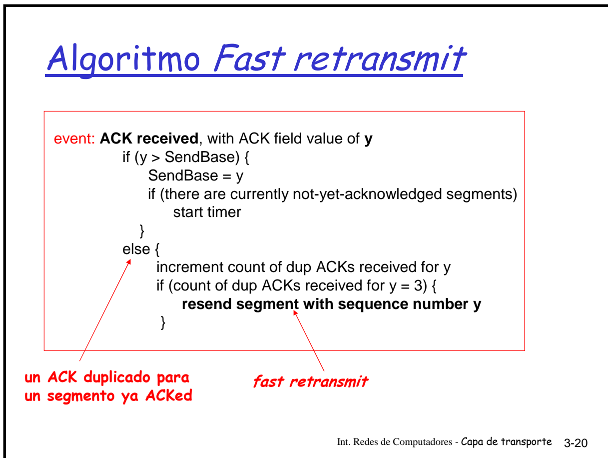
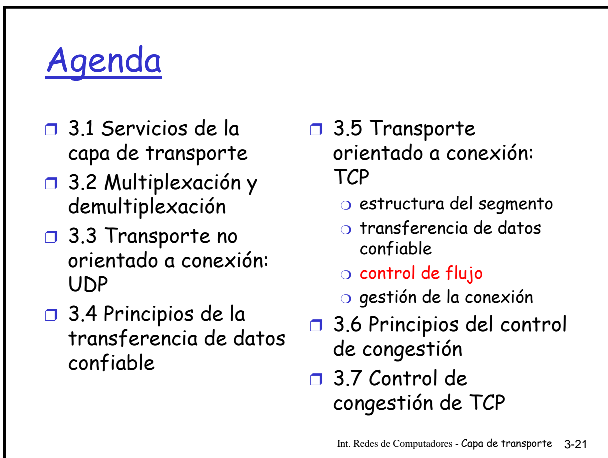


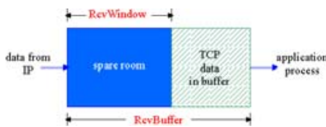
Figure 3.37 Re-enviando un segmento después del triple ACK duplicado





TCP: Control de flujo

- el lado receptor de la conexión TCP tiene un *buffer* de recepción:

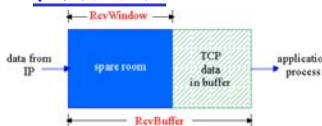


- El proceso de aplicación puede ser lento leyendo del *buffer*

Control de flujo
El emisor no debe desbordar el *buffer* del receptor transmitiendo muy rápido demasiados datos

- Servicio de equiparación de velocidad: adecuar la velocidad de envío con la velocidad de "drenado" del *buffer* por parte de la aplicación del receptor

Control de flujo de TCP: cómo funciona



(Suponga que el receptor TCP descarta segmentos fuera de orden)

- El receptor publica el espacio disponible en su buffer, incluyendo el valor de **RcvWindow** en los segmentos
- El emisor limita los datos no-ACKed a **RcvWindow**
  - garantiza no desbordar el *buffer* del receptor

- Para que no desborde el *buffer*
  ÚltimoByteRcvd - ÚltimoByteRead <= RcvBuffer
- espacio libre en *buffer*

= RcvWindow = RcvBuffer-[ÚltimoByteRcvd - ÚltimoByteRead]

Agenda

- 3.1 Servicios de la capa de transporte
- 3.2 Multiplexación y demultiplexación
- 3.3 Transporte no orientado a conexión: UDP
- 3.4 Principios de la transferencia de datos confiable
- 3.5 Transporte orientado a conexión: TCP
  - estructura del segmento
  - transferencia de datos confiable
  - control de flujo
  - gestión de la conexión
- 3.6 Principios del control de congestión
- 3.7 Control de congestión de TCP

TCP: Gestión de la conexión

Recordar: el emisor y el receptor de TCP establecen una "conexión" antes de intercambiar segmentos de datos

- inicializa variables de TCP:
  - nros. de sec.
  - *buffers*, información de control de flujo (p.e. `RcvWindow`)
- *cliente:* quien inicia la conexión

```
Socket clientSocket = new
Socket("hostname","port
number");
```

- *servidor:* contactado por el cliente

```
Socket connectionSocket =
welcomeSocket.accept();
```

Three way handshake:

Paso 1: el *host* cliente envía un segmento SYN al servidor
- especifica el nro. de sec. inicial
- sin datos

Paso 2: el *host* servidor recibe el SYN, responde con un segmento SYN-ACK sin datos
- el servidor reserva *buffers*
- especifica el nro.de sec. inicial del servidor

Paso 3: el cliente recibe el SYN-ACK, responde con un segmento ACK, que puede tener datos

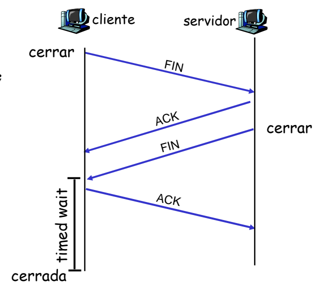
TCP: Gestión de la conexión (cont.)

Cerrando una conexión:

client closes socket: `clientSocket.close();`

Paso 1: el sistema final *cliente* envía un segmento de control FIN al *server*

Paso 2: el *servidor* recibe el FIN y responde con un ACK. Cierra la conexión y envía un FIN.

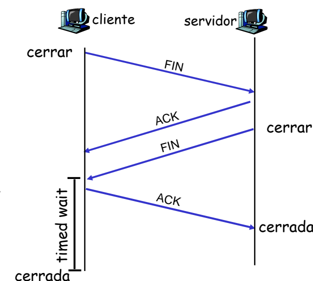


TCP: Gestión de la conexión (cont.)

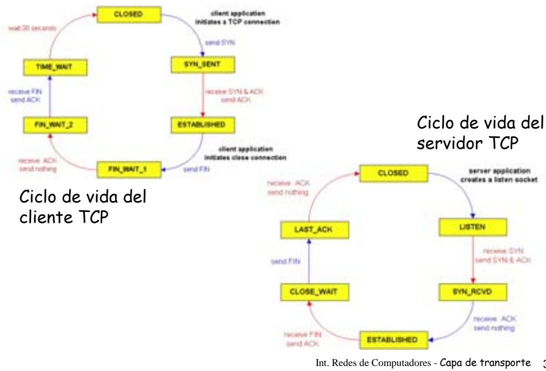
Paso 3: el *cliente recibe el* FIN y responde con un ACK.

Paso 4: el *servidor* recibe el ACK. Cierra la conexión.

Nota: con pequeñas modificaciones, puede manejar FINs simultáneos.



TCP : Gestión de la conexión (cont.)



Algunos comentarios más

- Bandera RST (*Reset*)
- Segmento SYN a un puerto TCP
  - Segmento SYNACK
  - Segmento RST
  - Nada
- Mensaje UDP a puerto que no está escuchando
  - Puedo recibir mensaje ICMP
- *SYN flooding attack*