T6
NIVELL TRANSPORT

Xarxes de Computadors i Aplicacions

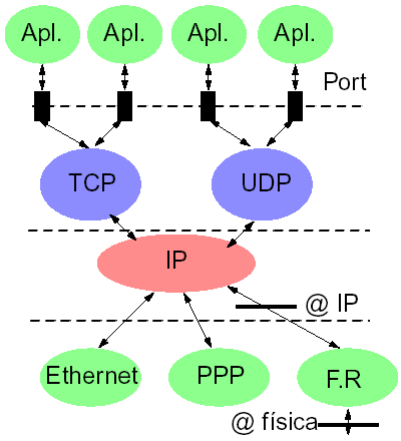
PAU ARTIGAS, DAVID CARRERA i JORDI TORRES
Departament d'Arquitectura de Computadors
UPC, setembre - 2009

Contenido

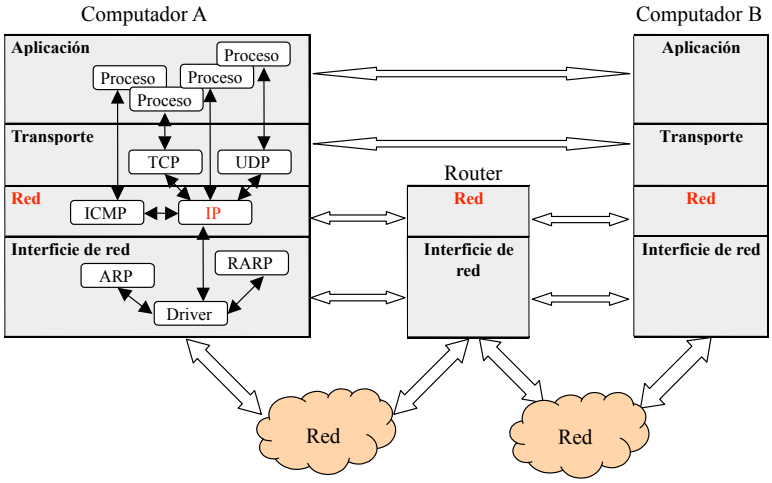
transparències basades en el material docent dels professors José M. Barceló i Jordi Torres de l'assignatura STD del pla 91 de FIB.

1. Introducció: nivel 4 - transport

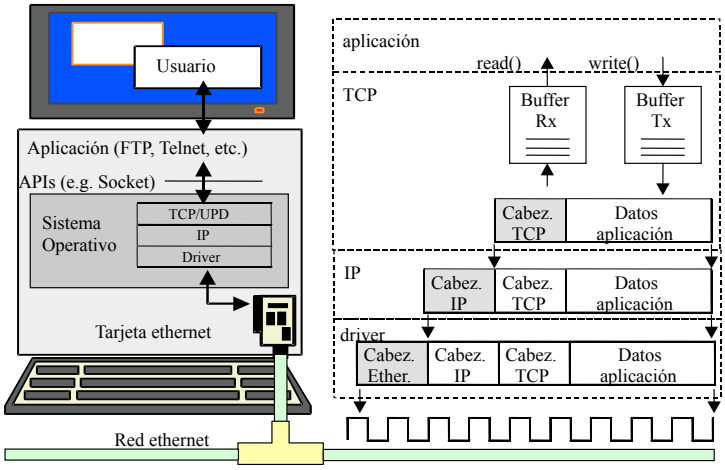
- Entre IP y las aplicaciones tenemos dos posibles protocolos:
  - UDP (No orientado a la conexión)
  - TCP (Orientado a la conexión)



1. Introducció: Arquitectura TCP/IP



1. Introducció: Arquitectura TCP/IP

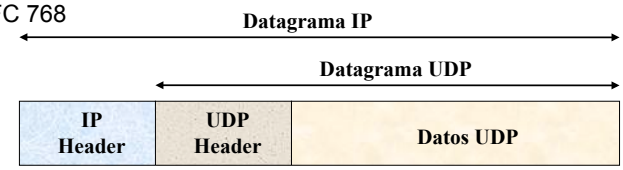


Contenido T5

1. Introducción
2. UDP (User Datagrama Protocol)
3. TCP (Transmission Control Protocol)
4. Cabecera TCP
5. Establecimiento (3wHS) y cierre de la conexión TCP
6. Grafo de estados TCP
7. Control de flujo en TCP
8. Control de errores en TCP
9. Tipos de aplicaciones que usan TCP
10. Control de congestión en TCP
11. Sockets

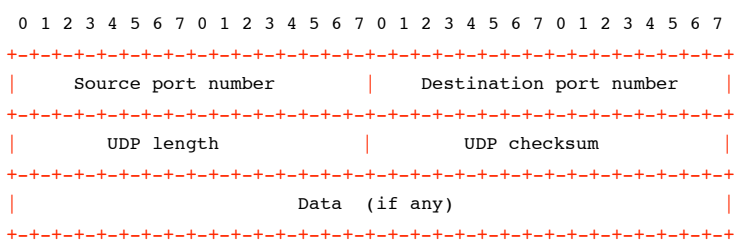
2. UDP (User Datagram Protocol)

- Protocolo de transporte no-orientado a la conexión, cuya unidad de encapsulamiento es el datagrama UDP
- Ideal para comunicaciones en tiempo real
- Cada escritura por parte de la aplicación provoca la creación de un Datagrama UDP
- Cada datagrama UDP creado provoca la creación de un datagrama IP en el nivel 3 (lo veremos en el siguiente tema)
- Si se pierde el datagrama IP o UDP es problema de la aplicación remota incorporar mecanismos de retransmisión
- RFC 768



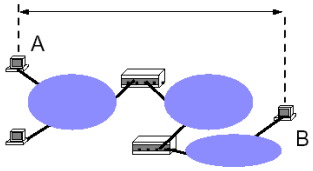
2.UDP (User Datagram Protocol)

- Datagrama UDP (8 bytes de cabecera)
  - **Puertos:** identifican a la aplicación origen y destino
  - **UDP length**: longitud total del datagrama UDP (campo redundante ya que IP lleva la longitud también)
  - **UDP checksum**: detector de errores que aplica a TODO el datagrama (checksum IP sólo cubría la cabecera IP)



3.TCP (Transmission Control Protocol)

- Es un protocolo extremo a extremo



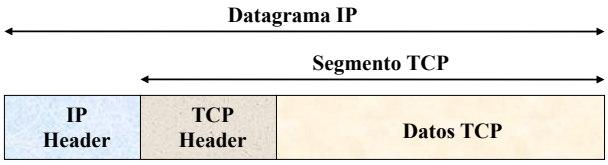
- Utilizando TCP aseguramos que
  - La información llega de forma correcta.
    (es decir, la información incorrecta es reenviada)
  - Y en el orden correcto.
  - Además
    - permite al consumidor no ser inundado por la información del productor.
    - y protege a la red de congestión.

3.TCP (Transmission Control Protocol)

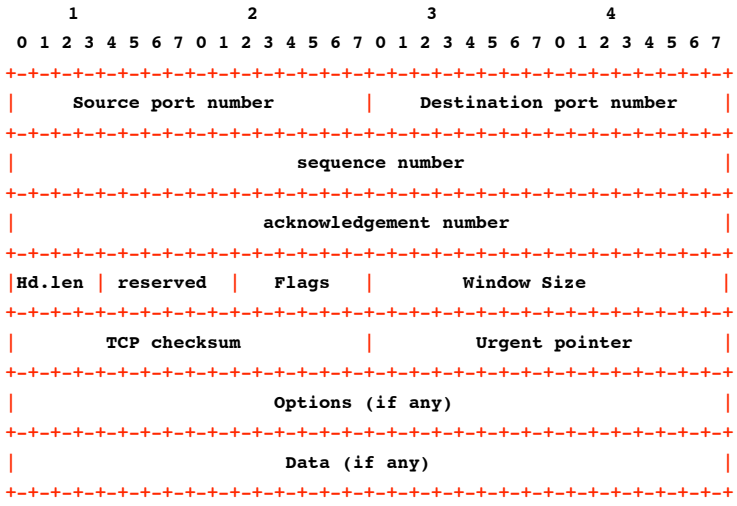
- Protocolo orientado a la conexión:
  - estableciemiento de la conexión,
  - envío de datos
  - y cierre de la conexión
- Tal vez el protocolo MAS complejo e importante de la pila de protocolos.
- **Unidad de datos es el "segmento TCP"**
- Proporciona fiabilidad mediante
  - el control de flujo (**ventana advertida**)
  - control de errores (**ARQ**)
  - y control de la congestión (**ventana de congestión**)

3.TCP (Transmission Control Protocol)

- Los segmentos pueden llegar fuera de orden (debajo hay IP que es no orientado a la conexión, o sea, datagrama),
- por tanto, TCP debe reordenar los segmentos antes de pasarlos a la aplicación



4. Cabecera TCP



4. Cabecera TCP

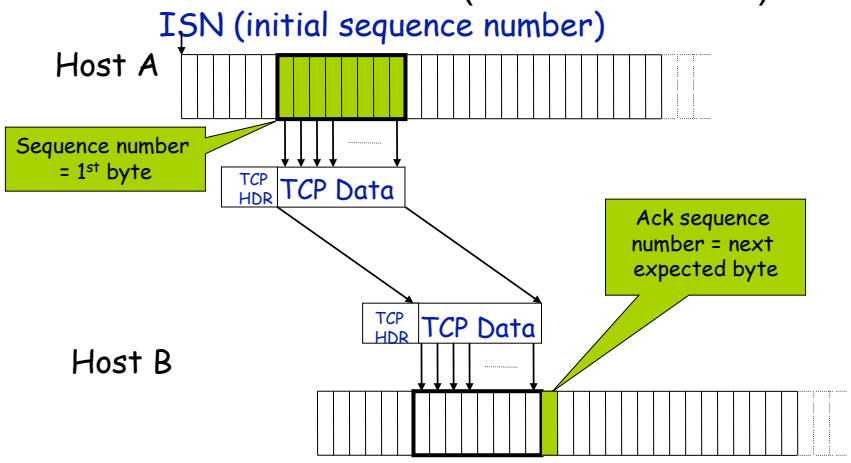
– **Puertos**: identifican las aplicaciones

– **Sequence number**: identifica el primer byte dentro de ese segmento de la secuencia de bytes enviados hasta ese momento.

- ISN (Initial Seq. Number): primer número de secuencia escogida por el protocolo TCP
- Seq. Number será un número a partir de ISN. Por tanto para saber cuantos bytes llevamos enviados hay que hacer “Seq. Number – ISN”

4. Cabecera TCP

- Números de secuencia (finestra lliscant)



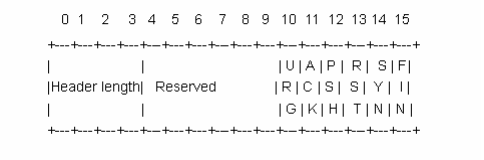
4. Cabecera TCP

– **Ack Number**: contiene el próximo número de seq. que el transmisor del ACK espera recibir
  - Por tanto es el “Seq. Number+1” del último byte recibido correctamente
  - Cuidado !!! TCP es FULL-DUPLEX: cada extremo mantiene un Seq. Number y un Ack Number

– **Header length**: longitud total de la cabecera (opciones variable) en múltiplos de 4 bytes

4. Cabecera TCP

– **Flags**: hay 6 flags (bits) en la cabecera
  - URG: Urgent Pointer field valido
  - ACK: Ack Number es valido
  - PSH: el receptor debe pasar los datos a la aplicación tan rápido como sea posible (por ahora no hay más datos)
  - RST: “Reset” la conexión
  - SYN: Sincronización de los números de secuencia al iniciar la conexión
  - FIN: termina la conexión



4. Cabecera TCP

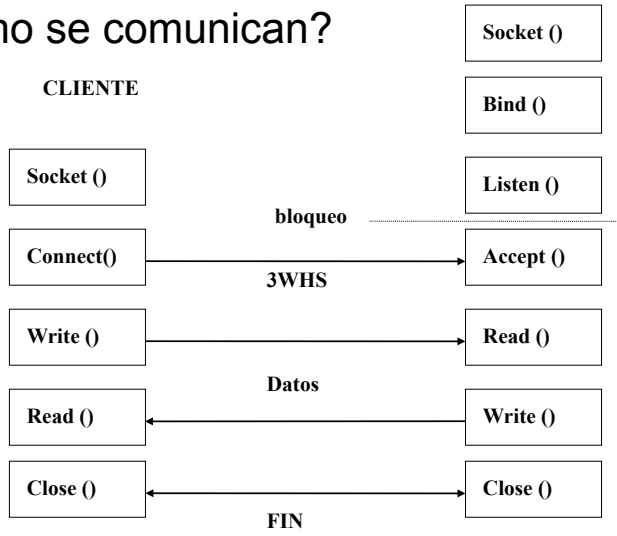
- **Window Size**: tamaño de la ventana advertida por el receptor al transmisor (Sliding Window). Máxima ventana = 65535 bytes
- **Checksum**: de todo el segmento TCP (igual que UDP)
- **Urgent Pointer**: puntero al Seq. Number que indica la parte de datos urgentes dentro del campo de datos
- **Options**: opción de anunciar el MSS (Maximum Segment Size)

Contenido T5

1. Introducción
2. UDP (User Datagrama Protocol)
3. TCP (Transmission Control Protocol)
4. Cabecera TCP
5. Establecimiento (3wHS) y cierre de la conexión TCP
6. Grafo de estados TCP
7. Control de flujo en TCP
8. Control de errores en TCP
9. Tipos de aplicaciones que usan TCP
10. Control de congestión en TCP
11. Sockets

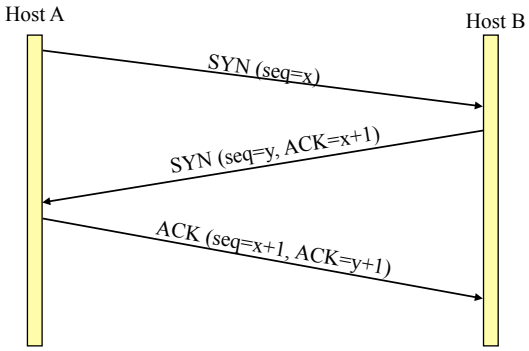
5. Programa cliente-servidor (repaso)

- ¿como se comunican?



5. Establecimiento de la conexión TCP

– Usa el 3-Way Handshake Algorithm:

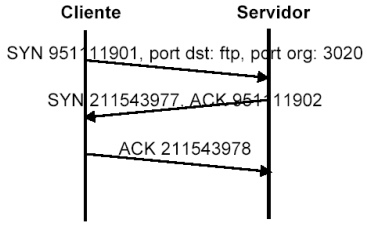


5. Establecimiento de la conexión TCP

- Usa el 3-Way Handshake Algorithm:
  - El cliente envía un **segmento SYN** especificando el puerto destino del servidor, su puerto origen (escogido por el Kernel) y el ISN (Initial Seq Number) escogido al azar por el Kernel
  - El receptor (servidor) devuelve un **segmento SYN+ACK** reconociendo el segmento SYN e indicando su ISN
  - El cliente responde con un **segmento ACK** reconociendo el SYN+ACK
- Una vez establecido la conexión se pasa a la fase de envio de datos
- Es posible negociar ("**indicar**") opciones (e.g.; indicar el MSS)
  - **MSS (Maximum Segment Size):** tamaño máximo de un segmento TCP.
    - Viene fijado por el Kernel: default = 536 bytes para un datagrama IP de 576 bytes (histórico X.25)
    - Sino, fijado por el MTU (Maximum Transfer Unit) menos cabeceras

Ejemplo de conexión TCP

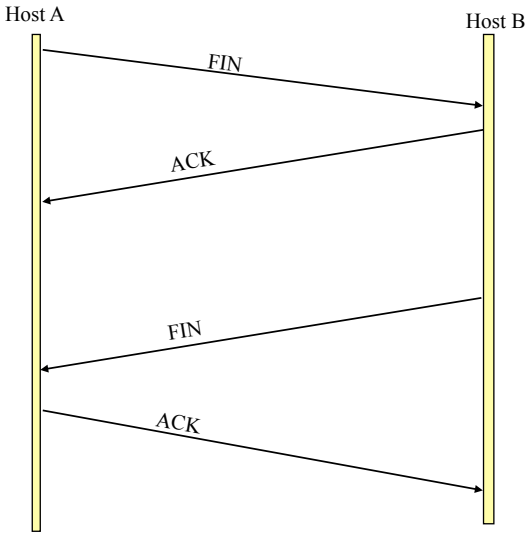
- Ejemplo de establecimiento de la conexión TCP: (recordar que usa el 3-Way Handshake Algorithm):
  - representación "time line":



  - TCP output simplificado:

```
11:27:13.771041        147.83.35.18.3020 > 147.83.32.14.ftp: S
                              951111901:951111901(0)
11:27:13.771491        147.83.32.14.ftp > 147.83.35.18.3020 : S
                              211543977:211543977(0) ack 951111902
11:27:13.771517        147.83.35.18.3020 > 147.83.32.14.ftp: .
                              ack 211543978
```

5. Cierre de la conexión TCP

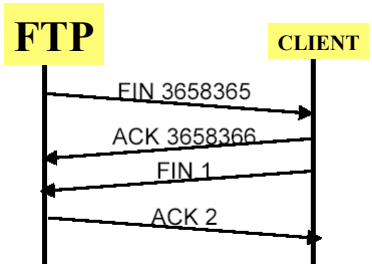


5. Cierre de la conexión TCP

- El cierre de la conexión puede ser debido a varias causas:
  - El cliente o el servidor cierran la conexión (e.g.; LLS close())
  - Por alguna razón se envía un reset de la conexión (flag activo RST)
  - Cierre debido a una interrupción, e.g.; ^D o un ^C, etc, ...
- El cierre normal es debido a un close del cliente lo que provoca el envío de 4 segmentos TCP
- Como la conexión TCP es FDX cada dirección debe cerrar la conexión (envío de segmento FIN y de su correspondiente ACK)
- Es posible que un extremo cierre su lado de la conexión y el otro no. En ese caso, el extremo que no ha cerrado puede enviar datos y el otro extremo los reconocerá (ACKs) aunque haya cerrado su conexión

Ejemplo de conexión TCP

- Cierre de la conexión TCP




```
11:27:19.397349        147.83.32.14.ftp > 147.83.35.18.3020 :
                       F 3658365:3658365(0)

11:27:19.397370        147.83.35.18.3020 > 147.83.32.14.ftp:
                       . 1:1(0) ack 3658366

11:27:19.397453        147.83.35.18.3020 > 147.83.32.14.ftp:
                       F 1:1(0)

11:27:19.398437        147.83.32.14.ftp > 147.83.35.18.3020 :
                       . 3658366:3658366(0) ack 2
```

Cabecera TCP

paquete:

4500 05dc e597 0000 4006 2687 9353 2350
9353 1f07 **0017 82c0 99a5 68e3 4693 e23b**
5018 3fe0 338c 0000 203e ...

cabecera TCP:

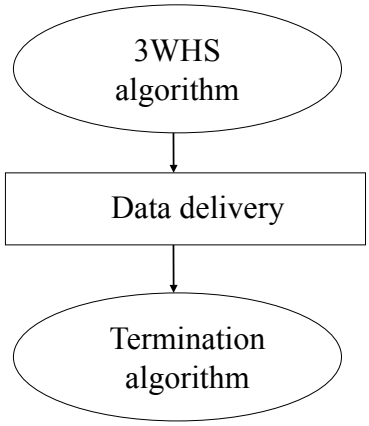
```
0000 0000 0001 0111 1000 0010 1100 0000
1001 1001 1010 0101 0110 1000 1110 0011
0100 0110 1001 0011 1110 0010 0011 1011
0101 0000 0001 1000 0011 1111 1110 0000
0011 0011 1000 1100 0000 0000 0000 0000
```

- port origen?
- port dest.?
- num. sec.?
- num. ack?
- hlen?

- urg?; ack?; psh?; srt?; syn?; fin?
- tam. vent?
- checksum?
- punt. urg?

6. Grafo de estados en una conexión TCP

– En total 11 estados distintos: CLOSE, ESTABLISHED, LISTEN, LAST_ACK, ...



6. Grafo de estados en una conexión TCP

- Estados posibles en una conexión TCP

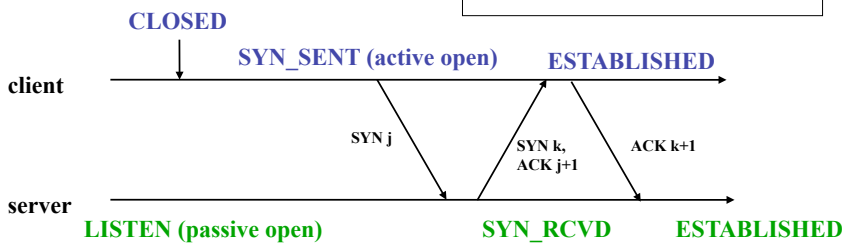
Active open	**CLOSED**	The socket is not being used.
	SYN_SENT	Actively trying to establish connection.
Passive open	**LISTEN**	Listening for incoming connections.
	SYN_RECEIVED	Initial synchronization of the connection under way.
	ESTABLISHED	Connection has been established.
Active close	**FIN_WAIT_1**	Socket closed; shutting down connection.
	CLOSING	Closed, then remote shutdown; awaiting acknowledgment.
	FIN_WAIT_2	Socket closed; waiting for shutdown from remote.
	TIME_WAIT	Wait after close for remote shutdown retransmission.
Passive close	**CLOSE_WAIT**	Remote shutdown; waiting for the socket to close.
	LAST_ACK	Remote shutdown, then closed; awaiting acknowledgment.

6. Grafo de estados en una conexión TCP

- Estados en el establecimiento de la conexión TCP:

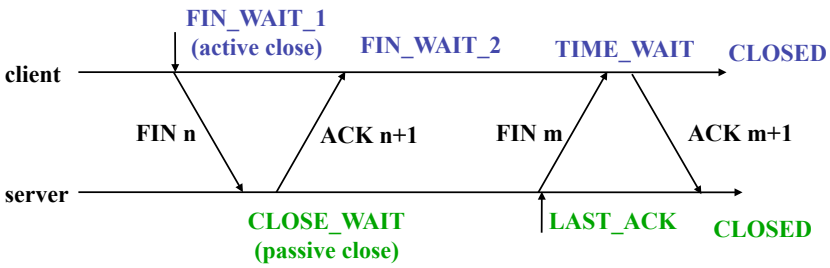
"Time Line horizontal"

AZUL: estados que pasa el cliente
VERDE: estados que pasa el servidor



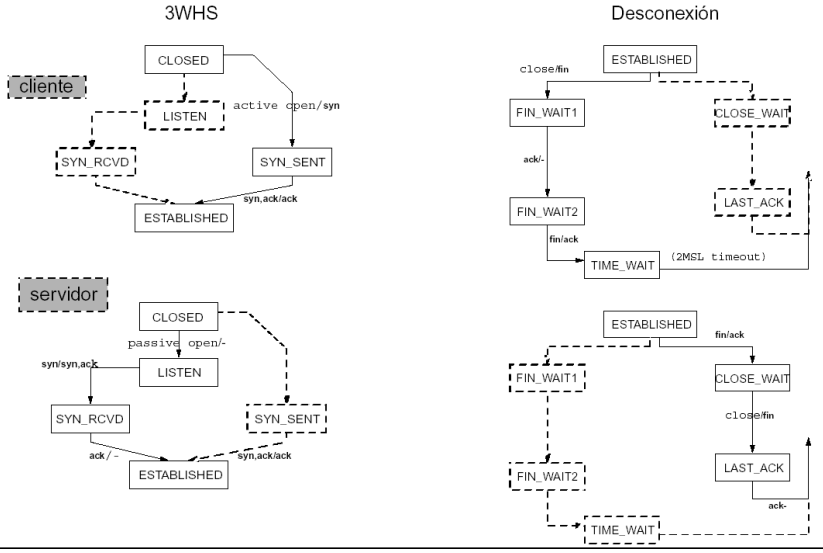
6. Grafo de estados en una conexión TCP

- Estados en el cierre de la conexión TCP:



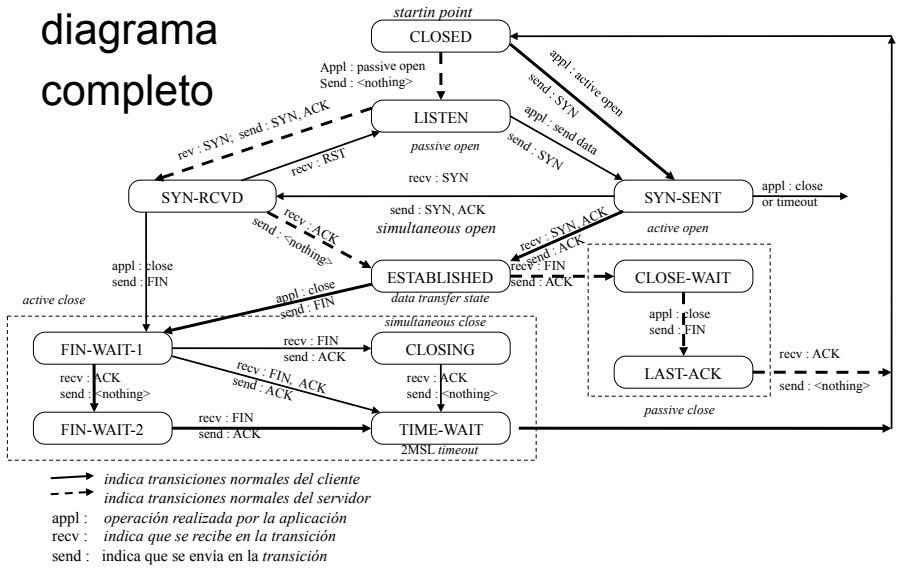
6. Grafo de estados en una conexión TCP

- ya hemos visto que los estados en cada extremo de la conexión són diferentes:



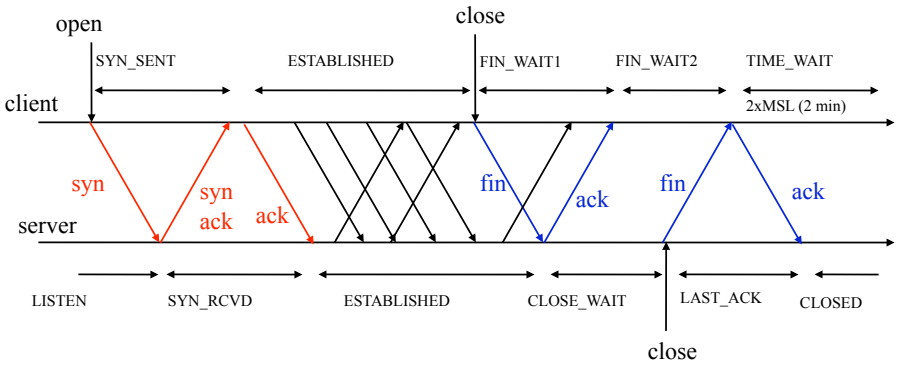
6. Grafo de estados en una conexión TCP

diagrama completo



6. Grafo de estados en una conexión TCP

- ejemplo conexión completa:



En el mismo escenario, ejecutamos el comando netstat -a en el servidor S y obtenemos la siguiente salida:

Local Adress	Remote Adress	State
*.111	*.*	LISTEN
*.21	*.*	LISTEN
147.83.30.10.3045	147.83.30.33.60	ESTABLISHED
147.83.30.10.111	147.83.30.20.3200	SYN_RCVD
127.0.0.1.111	127.0.0.1.3200	ESTABLISHED
147.83.30.10.21	147.83.30.20.2000	FIN_WAIT1

a) ¿Cuántos servidores tenemos activos en S?

c) ¿Por qué en la columna “Local Address” encontramos dos direcciones IP distintas?

a) ¿Qué secuencia de intercambio de segmentos ha ocurrido probablemente en la conexión en estado SYN_RCVD?

b) ¿Y en la conexión en estado FIN_WAIT1?

c) ¿Qué esperas que suceda en los próximos segundos en las dos anteriores onexiones?

Local Adress	Remote Adress	State
*.111	*.*	LISTEN
*.21	*.*	LISTEN
147.83.30.10.3045	147.83.30.33.60	ESTABLISHED
147.83.30.10.111	147.83.30.20.3200	SYN_RCVD
127.0.0.1.111	127.0.0.1.3200	ESTABLISHED
147.83.30.10.21	147.83.30.20.2000	FIN_WAIT1

- Recordemos que TCP proporciona fiabilidad mediante

  I. el control de flujo (**ventana advertida**)

  II. control de errores (**ARQ**)

  III. y control de la congestión (**ventana de congestión**)

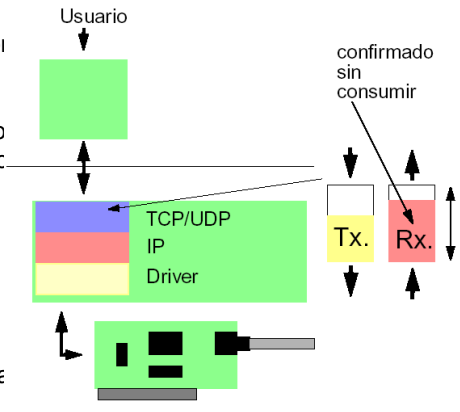
7. Control de flujo

"vigilar que el emisor no sature al receptor"

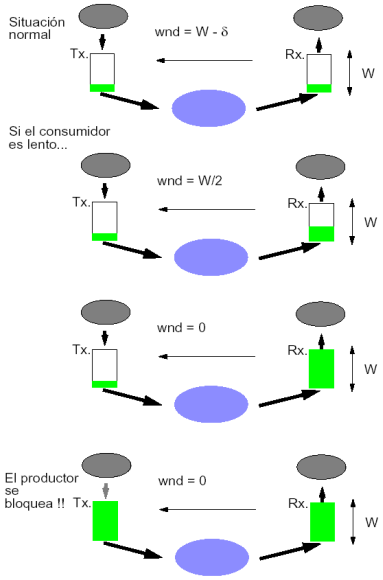
- Objetivo: el emisor no desborde el buffer del receptor por transmitir demasiado
  - el receptor tiene un buffer de recepción
  - la aplicación del receptor puede leer lento.
- Funcionamiento:
  - El receptor anuncia el espacio disponible en el buffer al emisor (con el campo **window size**)
  - El emisor limita los envios

7. Control de flujo

- El transmisor guarda en un buffer los bytes enviados pendientes de confirmación (por si se da el caso de que debe retransmitirlos) así comolos bytes que no se han transmitido (por ejemplo, por haber agotadc la ventana)
- El receptor guarda en un buffer los segmentos recibidos pendientes de ser consumidos por la aplicación. El tamaño de este buffer determina la ventana anunciada por el receptor



7. Control de flujo

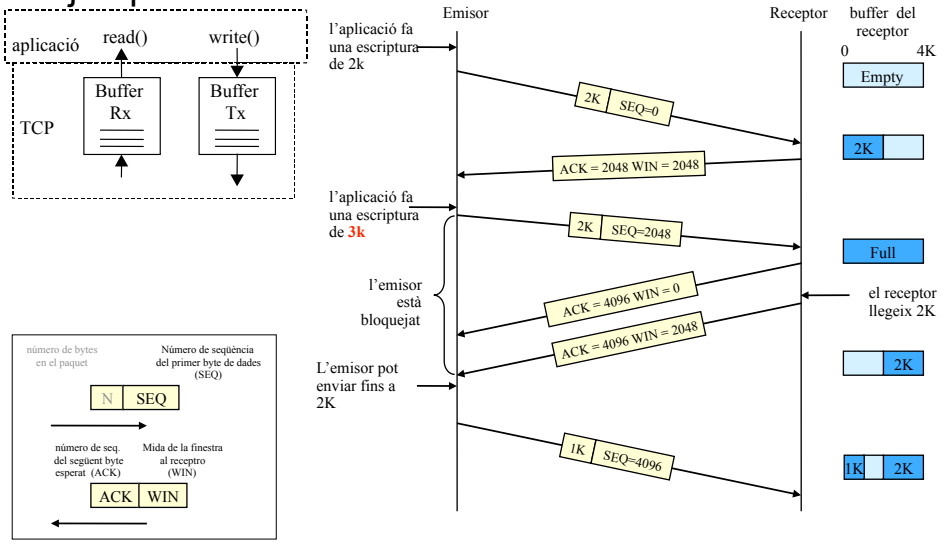


Casos que se pueden dar:

- Si el receptor es más rápido que el emisor, en este caso estara esperando contiamente datos y al ventana podría llegar a ocupar todo el buffer.
- El receptor es igual de rápido que el emisor, en este caso se van transmitiendo datos y el bufer del receptor tiene datos no consumidos y datos que se han recibido del emisor.
- El receptor es más lento que el emisor, en este caso la ventana advertida se ira reduciendo hasta que bloquee el emisor ya que no podrá enviar más datos.

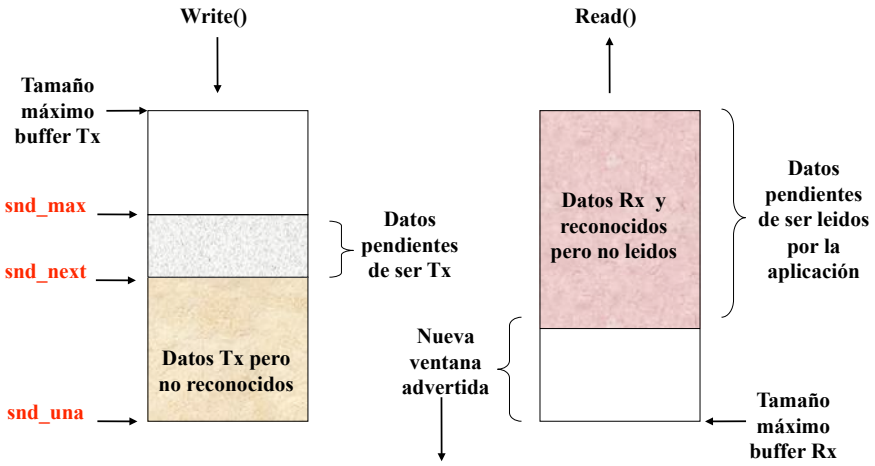
7. Control de flujo

Ejemplo:



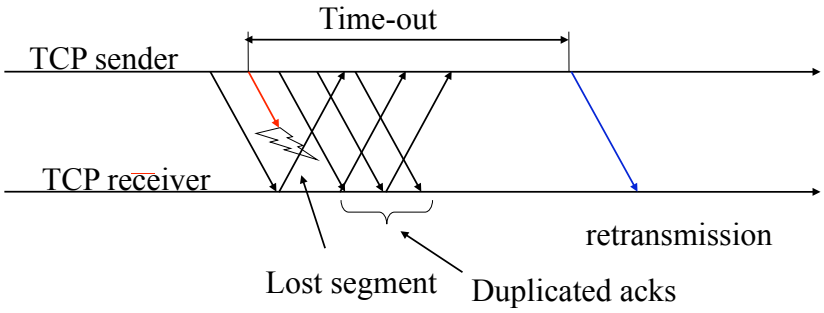
7. Control de flujo

- ¿Tamaño de la nueva ventana advertida?



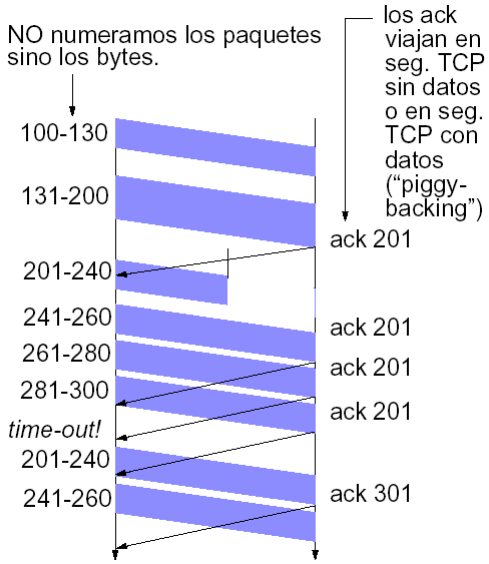
8. Control de errores

- ¿Qué pasa cuando se pierde un paquete?
- TCP consigue transmisión fiable de información mediante el uso de un protocolo de control de errores.



8. Control de errores

- ej:



8. Control de errores

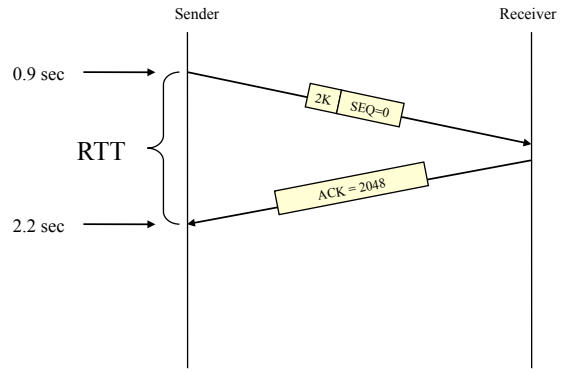
- Si un **segmento se pierde** y el emisor no recibe su ACK, los siguientes segmentos se van enviando y el receptor los reconocera con el ACK del anterior al que no se ha recibido, pero el emisor necesita el ACK del segmento perdido. Cuando tengamos el **timeout** (**en el emisor**) de dicho segmento lo retransmitira generando en el receptor un ACK del último paquete que llego.
- Si un segmento llega al receptor, pero no se recibe su ACK en el emisor, pero si el ACK del siguiente segmento, la recepcion del mismo **implica la recepción de todos los anteriores**.

8. Control de errores

- Retransmisión de los paquetes:
  - Cuando un paquete no recibe su ACK dentro de un periodo de tiempo, TCP asume que se ha perdido y vuelve a retransmitirlo.
  - TCP intenta calcular el “round trip time” (RTT) para un paquete y su ACK.
  - A partir de RTT, TCP puede suponer lo que debe esperar.
  - El cálculo del RTT no forma parte de las especificaciones de TCP!
- ¿Como fijar los valores de time-out?
  - Dependiendo de los retardos que sufren los ACKs, vamos fijando el valor del temporizador de forma adaptativa.

8. Control de errores

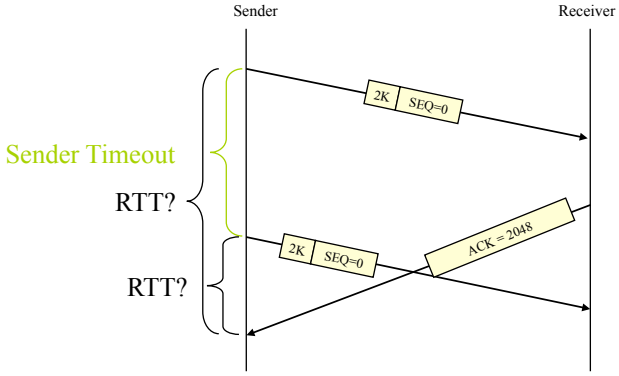
- Ejemplo de cálculo del RTT



RTT = 2.2 sec - 0.9 sec. = 1.3 sec

8. Control de errores

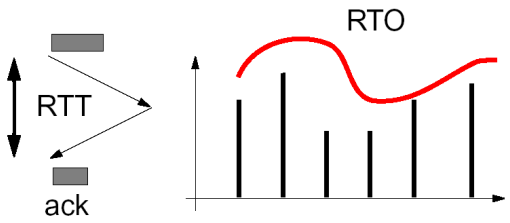
- Problemas con el cálculo del RTT



En este caso no se debe actualizar el RTT basado en información obtenida de paquetes retransmitidos (idea del Karn's Algorithm).

8. Control de errores

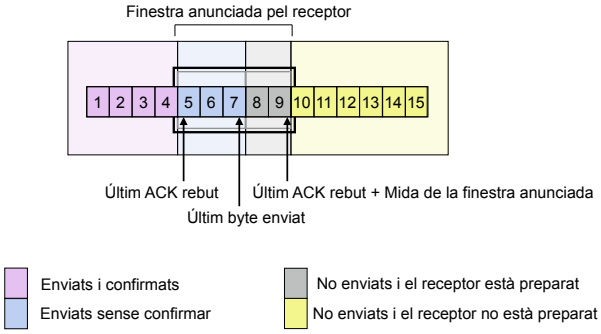
- Otro problema es que el RTT puede fluctuar mucho
  - » recordemos que es conexión extremo-extremo y que en medio puede haber cualquier tipo de red → los tiempos no son fijos.
- ¿Como fijar los valores de time-out?
  - Dependiendo de los retardos que sufren los ACKs, vamos fijando el valor del temporizador **de forma adaptativa**.



RTT: Round Trip-Time
RTO : Retransmission Time-Out

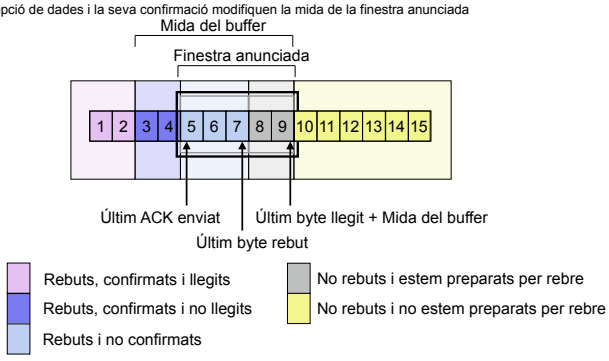
Finestra Lliscant - Emisor

- Tipus
  - Enviats i confirmats
  - Enviats, però no confirmats
  - No enviats i el receptor està preparat
  - No enviats i el receptor no està preparat
- Funcionament
  - Els ACKs (confirmacions) fan "lliscar" la finestra
  - La finestra anunciada pel receptor decideix la mida

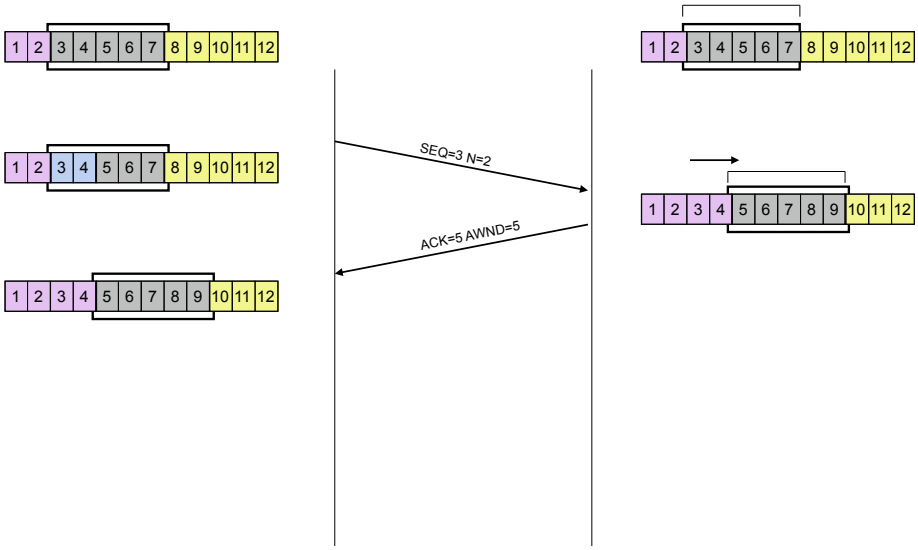


Finestra Lliscant - Receptor

- Tipus
  - Rebuts, confirmats i llegits per l'aplicació
  - Rebuts, confirmats i no llegits per l'aplicació
  - Rebuts i no confirmats
  - No rebuts i estem preparats per rebre
  - No rebuts i no estem preparats per rebre
- Funcionament
  - Les lectures per part de l'aplicació fan "lliscar" la finestra
  - La recepció de dades i la seva confirmació modifiquen la mida de la finestra anunciada

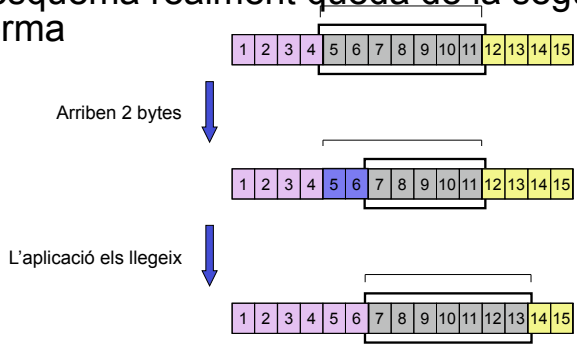


l'aplicació llegeix immediatament

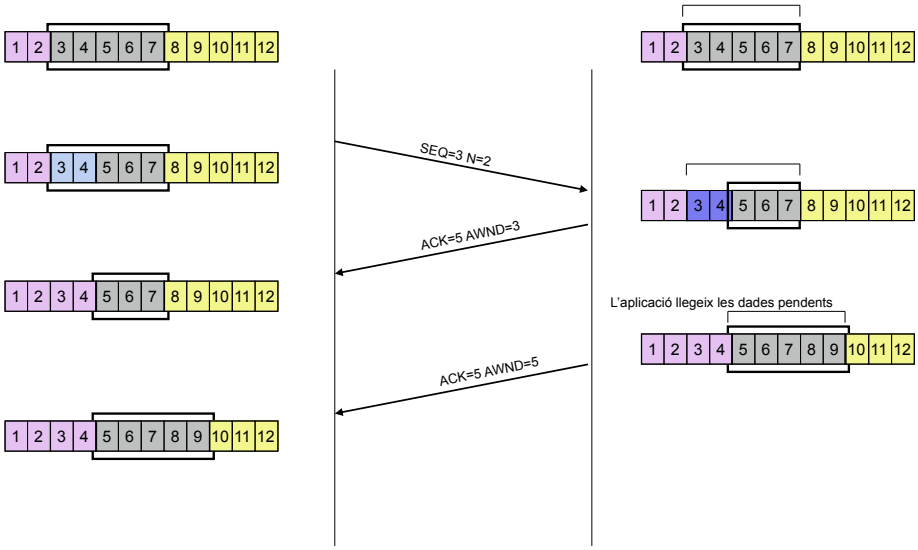


Finestra Lliscant - Receptor

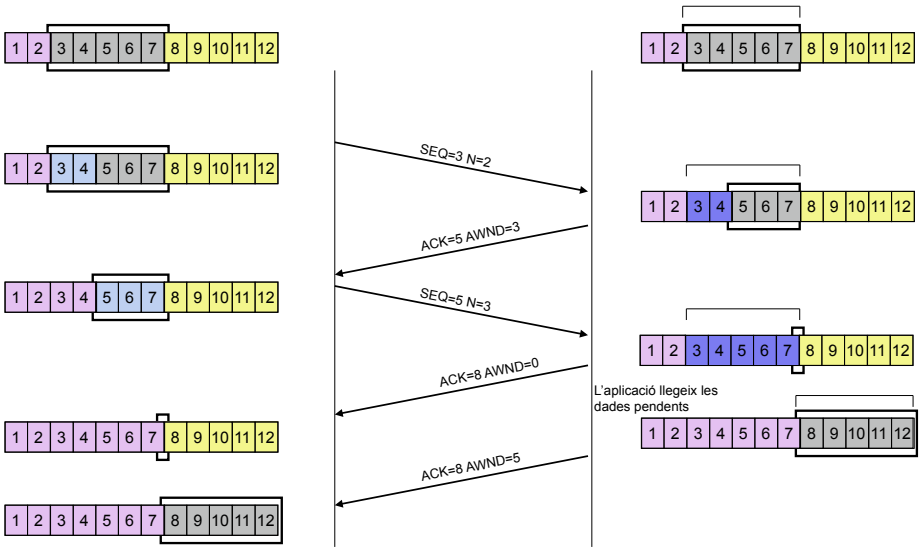
- Comentaris
  - Si l'aplicació llegeix de forma immediata l'esquema realment queda de la següent forma



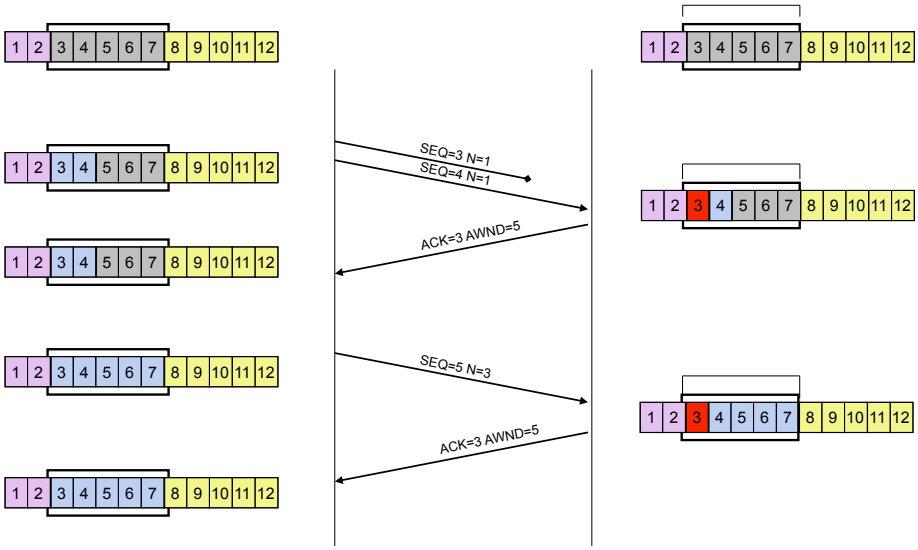
l'aplicació no llegeix - 1



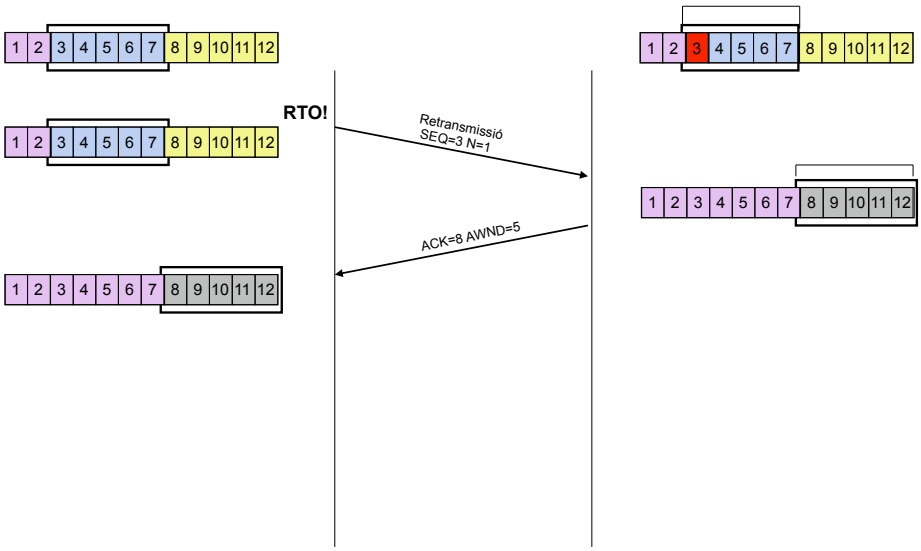
l'aplicació no llegeix - 2



es perd un segment



es perd un segment – continuació



Contenido T5

1. Introducción
2. UDP (User Datagrama Protocol)
3. TCP (Transmission Control Protocol)
4. Cabecera TCP
5. Establecimiento (3wHS) y cierre de la conexión TCP
6. Grafo de estados TCP
7. Control de flujo en TCP
8. Control de errores en TCP
9. Tipos de aplicaciones que usan TCP
10. Control de congestión en TCP
11. Sockets

9.Tipos de aplicaciones que usan TCP

- **Interactive aplications**: aquellas que envían poca cantidad de datos
  - ej: rlogin, telnet
  - Transmiten segmentos de tamaño muy pequeño (90 % de los segmentos tienen menos de 10 bytes de datos)
  - No es importante el control de la congestión
  - Importante los algoritmos de Nagle y los “Delayed ACKs”

9.Tipos de aplicaciones que usan TCP

Named for its creator, John Nagle, the **Nagle algorithm** is used to automatically concatenate a number of small buffer messages; this process (called nagling) **increases the efficiency of a network application system by decreasing the number of packets that must be sent.** Nagle's algorithm, defined in 1984 as Ford Aerospace and Communications Corporation Congestion Control in IP/TCP Internetworks (IETF RFC 896) was originally designed to relieve congestion for a private TCP/IP network operated by Ford, but has since been broadly deployed

9.Tipos de aplicaciones que usan TCP

Delayed ACKs

TCP has a rule like the following: If you send me two packets, I will send you one acknowledgement (ACK). If you send me one packet, I will wait 100 ms but not more than 200 ms before I respond with an ACK. The reason for delay is the chance a second packet might be coming in which case I should wait before I respond with an ACK.

This rule reduces the number of unnecessary ACKs. However, it causes relatively large delays in special cases, particularly for chatty applications that run across LANs. It can also be substantial for applications that run across WANs.

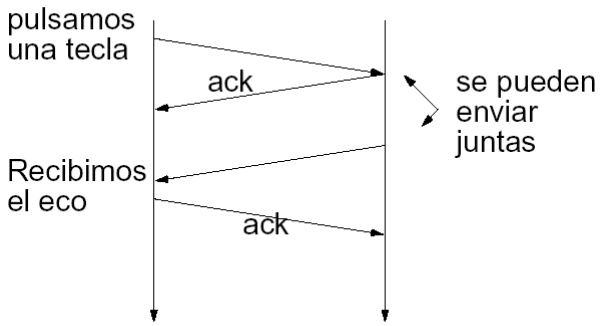
9.Tipos de aplicaciones que usan TCP

– **Bulk Transfer**: aplicaciones que envían gran cantidad de datos
  - ej: FTP
  - Los segmentos llevan mas de 512 bytes de datos
  - Importante los algoritmos de control de la congestión (Slow Start, Congestion Avoidance, Fast Retransmit y Fast Recovery)

! Que en las aplicaciones interactivas no sea importante el control de la congestión, no significa que no lo implementen, sólo que no se ven afectadas por este control de la congestión.

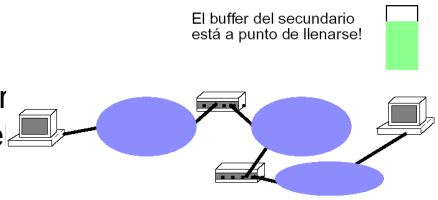
9.Tipos de aplicaciones que usan TCP

- Piggybacking

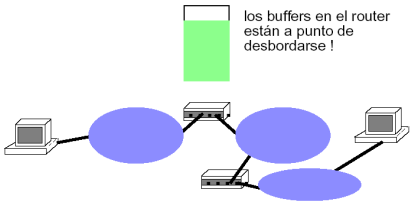


10. Control de congestión

- El algoritmo de ventana permite al receptor ejercer un control de flujo sobre e emisor



- Sin embargo los problemas podrían estar en la red (ej: router)



→ Necesitamos control de congestión

10. Control de congestión

- En las “Bulk Data Transfer”:
  - hace falta un control de la congestión debido a que enviamos muchos datos de golpe y los buffers pueden desbordarse (ya sea de los hosts o de los routers)
  - Mecanismo de control de la congestión:
    - Slow Start,
    - Congestion Avoidance,
    - Fast Retransmit,
    - Fast Recovery

10. Control de congestión

- Implementaciones TCP: muy variadas

  Todas están obligadas a implementar Slow Start y Congestion Avoidance

  - TCP Tahoe (1988): slow start, congestion avoidance, Fast Retransmit
  - TCP Reno (1990): Tahoe + Fast recovery + TCP header prediction
  - TCP Sack: Reno+ Selective ACK
  - Otras:
    - multicasting,
    - routing tables,
    - ....

10. Control de congestión

- Slow Start (SS) y Congestion Avoidance (CA):
  (Funcionan conjuntamente)

  – Queremos "arrancar" la conexión de forma que averigüemos a qué velocidad podemos transmitir los datos.

  – Introducimos una nueva ventana: **`cwnd`** (ventana de congestión,"congestion window").

  - Ventana advertida: Control de flujo ejercido por el receptor.
  - Ventana de congestión: Control de congestión ejercido por el emisor.

  → El valor de la ventana que se aplica és el mínimo de ambos valores

10. Control de congestión

- Slow Start (Funcionamiento):
  - Inyectar segmentos en la red a la velocidad a la que recibimos los ACKs desde el otro extremo
  - Ventana advertida (snd_wnd): ventana advertida por el receptor
    - *Es un control de flujo impuesto por el receptor que es el que fija el valor de la ventana advertida*
    - *S'explicita en un camp dels segments TCP que s'envien*
  - Ventana de congestión (snd_cwnd): ventana que se inicializa a 1
    - *Es un control de flujo impuesto por el transmisor*
    - *És un comptador intern que no es mostra explícitament*
  - Cada vez que se recibe un ACK → snd_cwnd++ (se incrementa en 1).

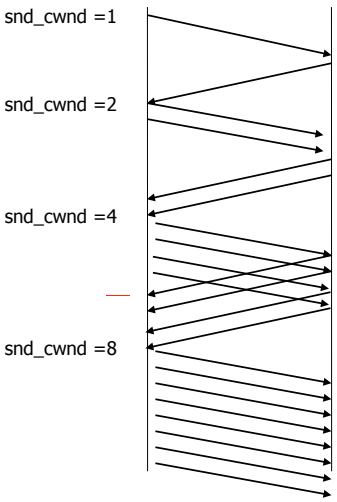
El transmisor transmite:

Ventana de TX = min (snd_wnd, snd_cwnd)

10. Control de congestión

- Slow Start (SS):

Cada vez que recibimos un ack, incrementamos `cwnd` en 1.

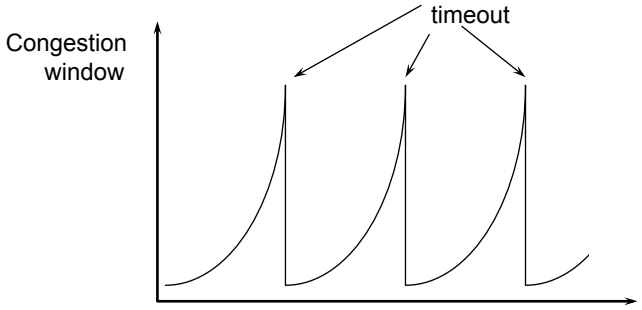


Inicialmente { snd_wnd =10; snd_cwnd =1 }

Min (snd_wnd, snd_cwnd)

10. Control de congestión

- El incremento de segmentos transmitidos es exponencial
- Perdida de paquetes indica congestión (se sabe por timers en el que envia)
- Cuando ocurre un timeout, la ventada de congestión se reduce a 1 (todo empieza de nuevo!)



10. Control de congestión

- TCP Slow Start, solo, es claramente ineficiente.

  la ventana de congestión crece exponencial, pero, cae al tamaño de un segmento

  →esto claramente conlleva un *throughput* pobre

- Solución:

  Establecer un umbral a partir del cual se incrementa linealmente en lugar de exponencialmente para aumentar la eficiencia. → Congestion Avoidance

10. Control de congestión

- Slow Start y Congestion Avoidance:
  (Funcionan conjuntamente)

  – Congestion Avoidance
    - Con SS vamos aumentando la ventana de congestión hasta que llega un momento en que llenamos el buffer de algún router y se producen pérdidas.
    - Con Congestion Avoidance (CA) intentamos mantenernos al valor límite de throughput sin tener pérdidas.

10. Control de congestión

– Congestion Avoidance (CA) (Funcionamiento):
  - Si comenzamos con Slow Start, enviamos segmentos de forma exponencial
  - Si la red se congestiona queremos que la red se recupere (congestion avoidance) sin dejar de transmitir y comenzar de nuevo en un slow start
  - Es un control de flujo impuesto por el transmisor
  - CA hace que cuando se detecta una perdida, en vez de transmitir exponencialmente segmentos, pasemos a transmitir de forma lineal hasta que se recupere la red

10. Control de congestión

- Algoritmo de “Congestion Avoidance”:

  `cwnd` = 1; `ssthresh` = Ventana máxima

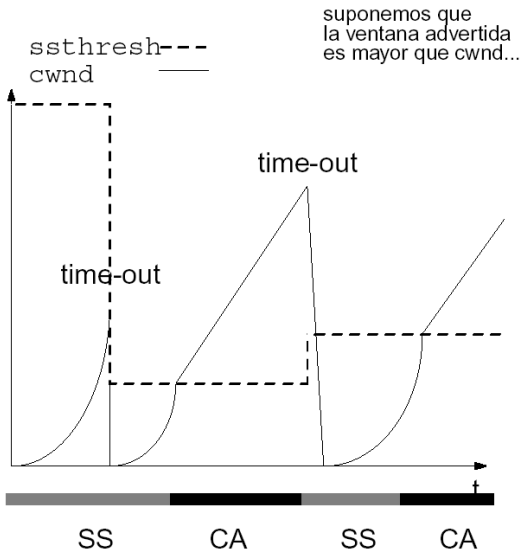
  Con SS vamos incrementando `cwnd` hasta que hay una pérdida (time-out)

  `ssthresh` <- max(2,1/2 min(`cwnd`, `awnd`))
  `cwnd` <- 1 e iniciamos SS

  Cuando `cwnd` >= `ssthresh`, abandonamos la fase de SS e iniciamos la fase de CA.

  Durante la fase de CA, cada vez que recibimos un ack, incrementamos `cwnd` <- `cwnd` + 1/`cwnd` (es decir, `cwnd` crece linealmente en el tiempo)

10. Control de congestión

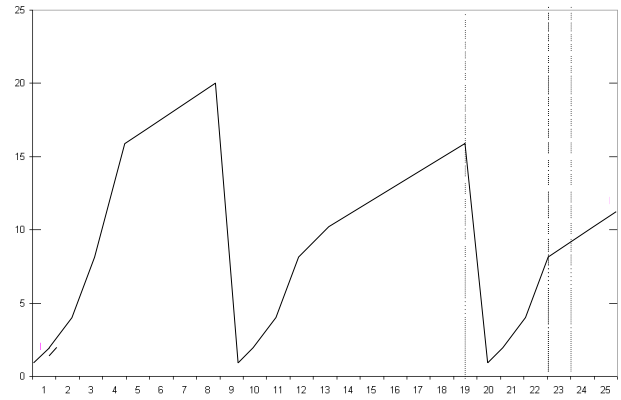


10. Control de congestión

- Congestion Avoidance (CA) (más detallado):
  - Define un nuevo parámetro:
    - Threshold (umbral) (ssthresh) que se inicializa a 65535 bytes
    - Si hay congestión:
      - Debido a una perdida (Tout) o a la recepción de 3 ACKs duplicados → el threshold pasa a valer la mitad de la ventana de transmisión pero no por debajo de 2 segmentos, es decir:
        - » ssthresh=max[2, 1/2 min(snd_wnd, snd_cwnd)]
      - Si además la congestión es por el salto del temporizador, snd_**c**wnd=1
    - Si snd_cwnd < ssthresh → estamos en Slow Start (SS)
      - Cada vez que recibamos un ACK, incrementamos la ventana de congestión de la siguiente manera
        - » Si estamos en SS → snd_cwnd ++
    - Si snd_cwnd ≥ ssthresh → estamos en Congestion Avoidance (CA)
      - Cada vez que recibamos un ACK, incrementamos la ventana de congestión de la siguiente manera
        - » Si estamos en CA → snd_cwnd = snd_cwnd + 1/ snd_cwnd

10. Control de congestió

- Suposeu la següent gràfica de la mida de la finestra de congestió d'una connexió TCP



10. Control de congestió

- Preguntes breus prèvies:
  - Què indiquen les unitats de l'eix de les X? Quina unitat s'està fent servir?
  - Quin és el valor del *Threshold* en el punt 22? i en el 23?

10. Control de congestió

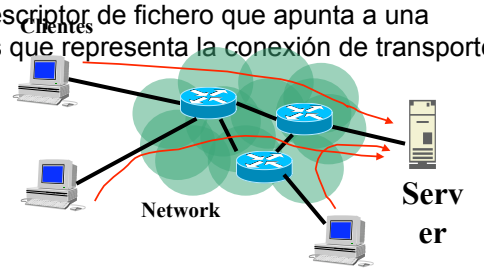
- Suposeu que:
  - La finestra advertica/anunciada és molt gran.
  - Que el receptor i emisor són molt ràpids (suposa infinit). Això vol dir que el temps de enviar tots els segments d'una finestra i rebre els ACKs corresponents, és just un RTT.
  - Suposa que el segment conté 1000 bytes de dades, que el *RTT* és 100 ms (1 ms = 10^{-3} segons).
- Quina és la velocitat (throughput mig) amb la que envia les dades el emisor entre els punts 19 i 25 (inclosos)?

10. Control de congestió

In TCP/IP, **fast retransmit and recovery** (FRR) is a congestion control algorithm that makes it possible to quickly recover lost data packets. Without FRR, the TCP uses a timer that requires a retransmission timeout if a packet is lost. No new or duplicate packets can be sent during the timeout period. With FRR, if a receiver receives a data segment that is out of order, it immediately sends a duplicate acknowledgement to the sender. If the sender receives three duplicate acknowledgements, it assumes that the data segment indicated by the acknowledgements is lost and immediately retransmits the lost segment. With FRR, time is not lost waiting for a timeout in order for retransmission to begin.

11. Sockets (repaso)

- Modelo cliente-servidor
  - Los servidores están esperando peticiones de los clientes
  - Cuando un cliente quiere un servicio (aplicación) debe establecer una conexión con el servidor. A la conexión se le llama en UNIX "socket"
  - Un socket es un descriptor de fichero que apunta a una estructura de datos que representa la conexión de transporte
    - Sockets TCP
    - Sockets UDP

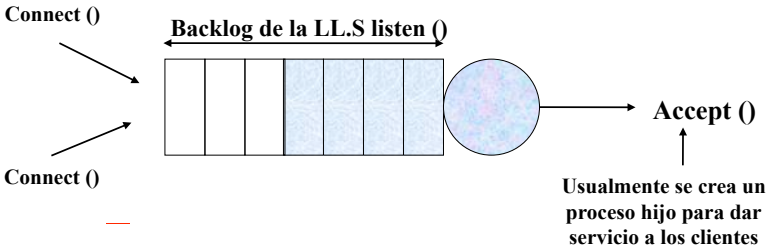


11. Sockets (repaso)

- Puertos TCP/UDP
  - Un puerto identifica la aplicación de red que deseamos utilizar
    - Puertos conocidos ("Well-known ports"): puertos que identifican la aplicación servidor (rango 1-1023), ver /etc/services. Servicios no estándares usan ports > 5000
      - 20/tcp ftp-data
      - 21/tcp ftp
      - 23/tcp telnet
      - 69/udp tftp
      - 80/tcp http
      - .....
    - Puertos efímeros ("ephimeral ports"): puertos usados por los clientes (típicamente entre 1024 y 5000)

11. Sockets

- Relación entre las LL.S y el diagrama de estados TCP

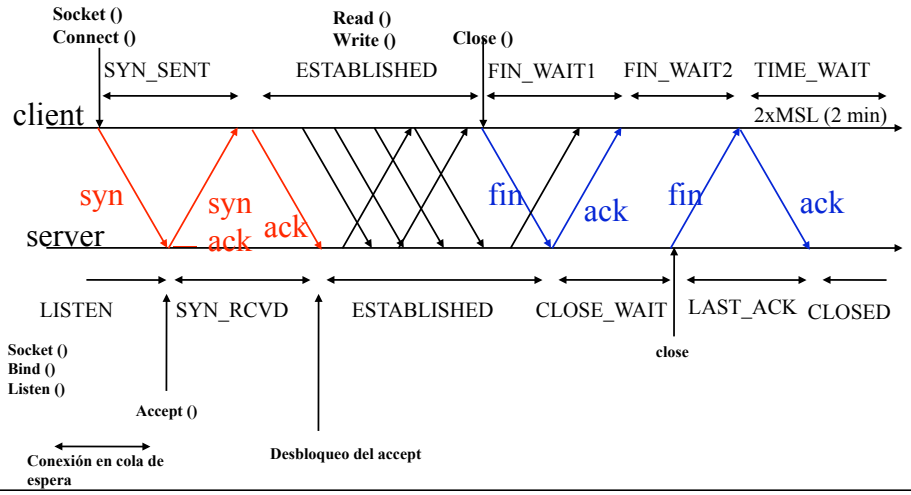


• Cola de conexiones incompletas: conexiones que no han terminado el 3WHS
• Cola de conexiones completas: conexiones que han terminado el 3WHS

Devuelve SYN-ACK, espera un ACK del cliente antes de dar servicio al proximo connect ()

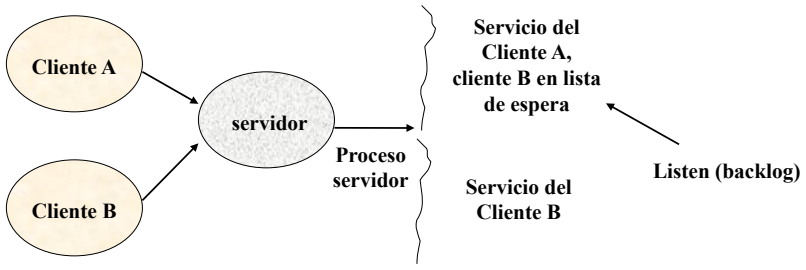
11. Sockets

- Relación entre las syscalls y el diagrama de estados TCP



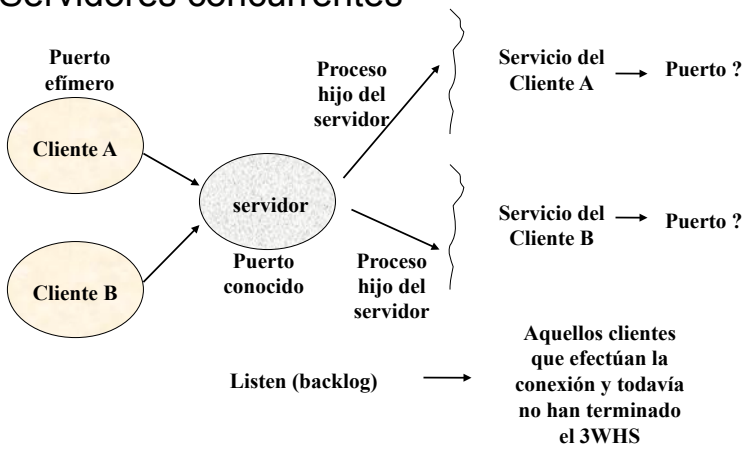
11. Sockets

- Tipos de servidores:
  - Servidores interactivos (iterativos)



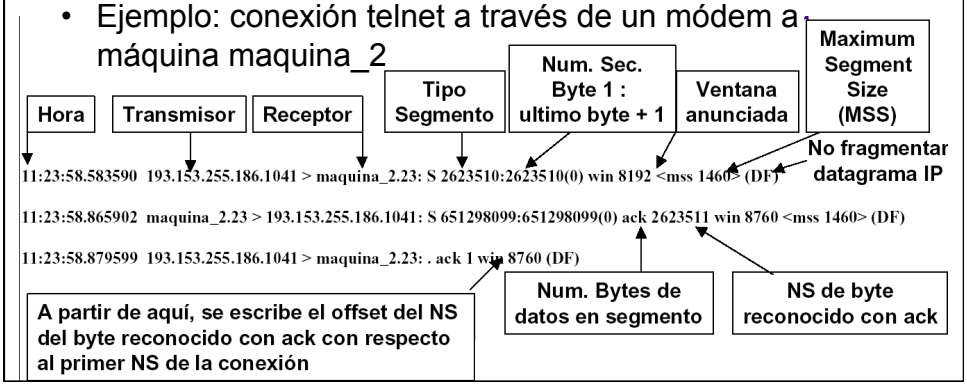
11. Sockets

- Tipos de servidores:
  - Servidores concurrentes



Annex: Using tcpdump

- Programa que coloca a la tarjeta de red en modo promíscuo: todo lo que pasa por la red es recogido y pasado a los módulos IP y TCP. Tcpdump presenta la información al usuario en forma inteligible.
- Ejemplo: conexión telnet a través de un módem a máquina maquina_2



Annex: tcpdump

- **tcpdump -w** filename.trace **-i** interface **-s 100**
  host sender **and host** receiver
- Tracea los paquetes en “interface” y los coloca en “filename.trace”
- Se pueden especificar reglas complejas para el filtrado (aquí sólo indicamos la pareja de hosts).
- **-s 100** especifica cuantos bytes por paquete.
  - 12 bytes for MAC headers
  - 20 bytes IP header
  - 20 bytes TCP header
  - extra room for any options which may appear
- Muchas más opciones disponibles!
- ^C para acabar!