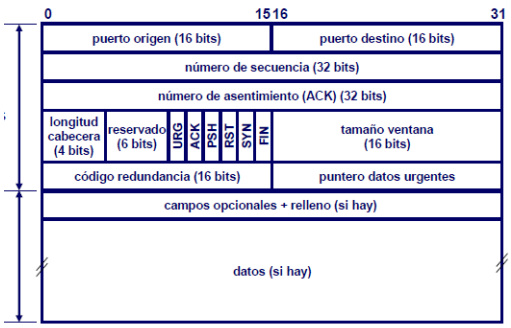
TCP Cheat Sheet – (act. 15/10/2013)

Cabeceras



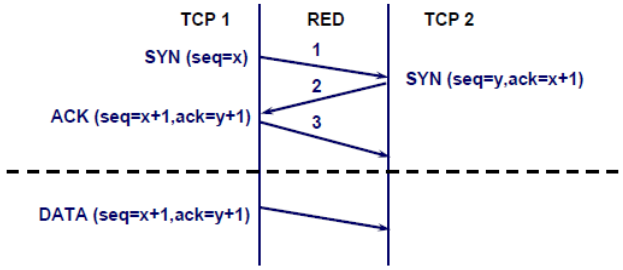
Cabecera variable min20 bytes máximo 60 bytes

Flags:

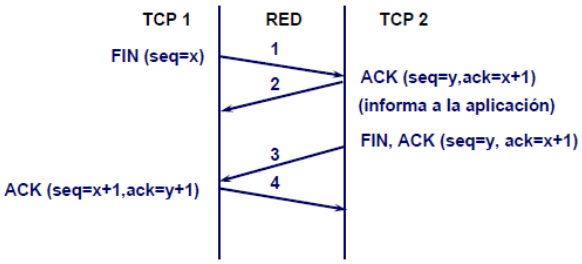
URG: indica que el campo Puntero Datos Urgentes tiene validez.
ACK: indica que el campo Número de Asentimiento tiene validez.
PSH: indica que el segmento requiere envío y entrega inmediata.
RST: indica aborto de la conexión.
SYN: sincroniza números de secuencia en el establecimiento de conexión. Confirmación con ACK
FIN: indica liberación de conexión. Confirmación con ACK.

MSS – MTU: *tamaño más grande de datos (en bytes), que un dispositivo de comunicaciones puede recibir en un único segmento, sin fragmentar. Óptimo: la suma del número de bytes del segmento de datos y la cabeceras debe ser menor que el número de bytes de la unidad máxima de transferencia (MTU) de la red.*

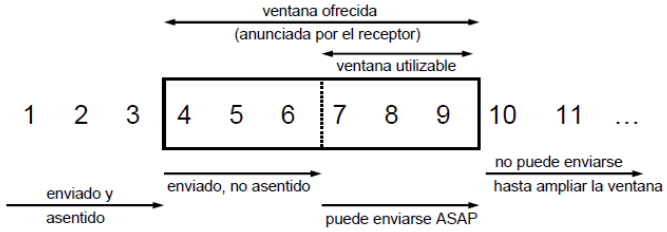
Establecimiento



Cierre



Control de flujo (Ventana deslizante)



Ventana anunciada (cabecera TCP) ***WIN***

Temporizadores en TCP

– Temporizador de retransmisión: Permite retransmitir datos si no se recibe un asentimiento del receptor.
– Temporizador persistente: Permite el flujo de información del tamaño de la ventana aunque el otro extremo haya cerrado la ventana anunciada.
- Temporizador de "keepalive": Permite detectar cuando el otro extremo de la conexión se cae o se reinicializa.
– Temporizador "2MSL": • Permite esperar un tiempo para garantizar que se cierra correctamente la conexión (estado TIME_WAIT)

Control congestión

Detectar la congestión:

- *Asentimiento (ACK) duplicado*
- *Vencimiento de temporizador de retransmisión*

Ventana efectiva:

$$Vef = min(WIN, cwnd)$$

Fast retransmit

Se aplica cuando hay 3 o más asentimientos duplicados
Motivo: si TCP recibe un segmento fuera de secuencia debe generar un asentimiento sin retardos (duplicado)
Algoritmo: 1)Se envía inmediatamente (sin esperar el timeout) el segmento que parece que se ha perdido
2) actualiza $ssthresh = max(2seg, \frac{Vef}{2})$
3)entra en slow start

Fast recovery

Evita entrar en slow start después de un fast retransmit
Motivo: si se reciben asentimientos es que fluyen los datos

Control congestion (slow start+congestion avoidance)

$$mecanismo = \begin{cases} Slow\ Start, & cwnd \leq ssthresh \\ Congestion\ Avoid., & cwnd > ssthresh \end{cases}$$

1)Inicio del intercambio de datos

$$cwnd = 1\ seg\ ;\ \ ssthresh = 65535$$

2)El emisor transmite Vef

…

n+1) Si hay congestión

n+1.a) Congestión por asentimiento duplicado

$$cwnd_{n+1} = cwnd_n\ ;\ \ ssthresh = max(2seg, \frac{Vef}{2})$$

n+1.b)Congestión por timeout

$$cwnd_{n+1} = 1\ seg\ ;\ \ ssthresh = max(2seg, \frac{Vef}{2})$$

…

n+1) Recepción de asentimiento (ACK)

n+1.a) slow start $cwnd \leq ssthresh$

$$cwnd_{n+1} = cwnd_n + 1\ seg$$

n+1.b)Congestion avoidance $cwnd > ssthresh$

$$cwnd_{n+1} = cwnd_n + 1/cwnd_n\ seg$$

Fast recovery – fast restransmit (implementación conjunta)

Cuando se recibe el tercer ACK duplicado:

1) - Actualiza $sthresh = max(2seg, \frac{Vef}{2})$

-Retransmite el segmento perdido

$-cwnd_{n+1} = ssthresh + 3\ seg$

2-a)Cada vez que llega ACK duplicado

$-cwnd_{n+1} = cwnd_n + 1\ seg$

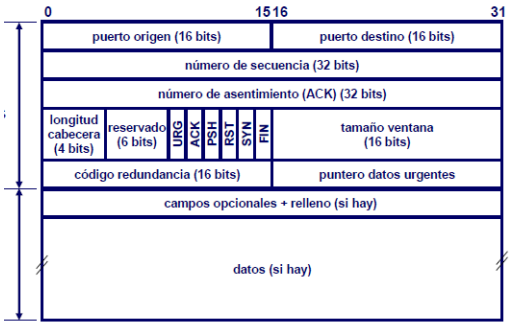
-transmite el segmento si se puede

2-b)Cuando llega ACK de nuevos datos

$-cwnd_{n+1} = ssthresh$

TCP LAB Cheat Sheet

Cabeceras



Cabecera variable min20 bytes máximo 60 bytes

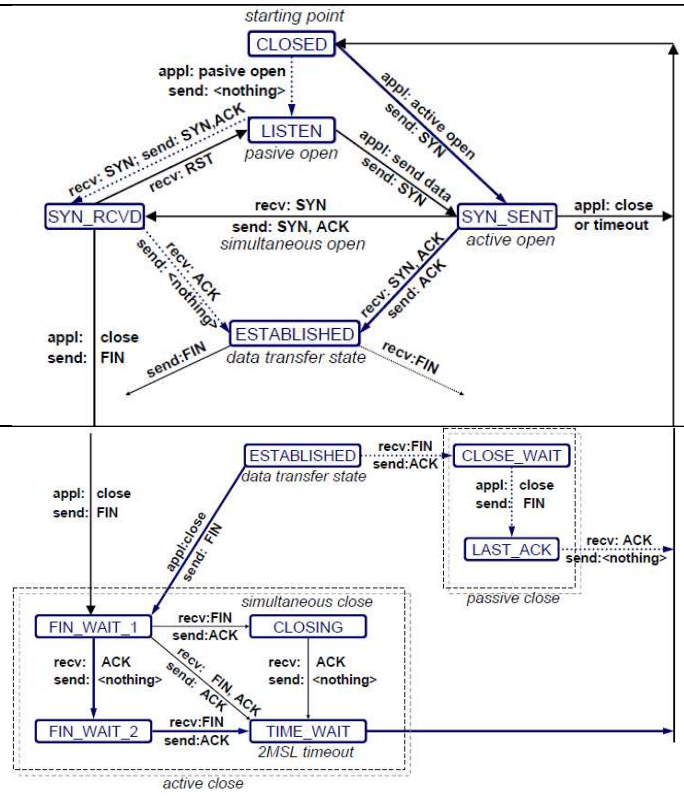
Flags:

URG: indica que el campo Puntero Datos Urgentes tiene validez.
ACK: indica que el campo Número de Asentimiento tiene validez.
PSH: indica que el segmento requiere envío y entrega inmediata.
RST: indica aborto de la conexión.
SYN: sincroniza números de secuencia en el establecimiento de conexión. Confirmación con ACK
FIN: indica liberación de conexión. Confirmación con ACK.

Estados de TCP

CLOSED *No existe conexión activa ni pendiente*
LISTEN *El servidor está esperando por una conexión*
SYN_RCVD *Una petición de conexión ha llegado, espera ACK*
SYN_SENT *El cliente ha comenzado a abrir una conexión*
ESTABLISHED *Estado de transferencia de datos*
FIN_WAIT_1 *El cliente ha dicho que ha acabado*
FIN_WAIT_2 *El servidor ha aceptado liberación conexión*
TIME_WAIT *Espera por paquetes pendientes ("*2MSL wait state*")*
CLOSING *Ambos extremos han intentado cerrar la conexión simultáneamente.*
CLOSE_WAIT *Servidor ha iniciado liberación conexión*
LAST_ACK *Espera por ACK del FIN*

Diagrama



Interfaces en las máquinas linux

Loopback: se suele utilizar cuando una transmisión de datos tiene como destino el propio host. La que se utiliza de forma mayoritaria es la '127.0.0.1'.
Interfaces físicos Ethernet: Numerados de eth0-ethn. Estos dispositivos envían tráfico fuera de la máquina.

Herramientas *Conocer los sockets pasivos que escuchan*

(-l listening; -t tcp; -n numeric; -4 TCP/IPv4)
`netstat -ltn4`

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN
...
```

Herramientas *Conocer el estado de las conexiones*

`netstat -tn (puede usarse -4)`

```
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 163.117.141.200:39325   163.117.141.200:636     ESTABLISHED
tcp        0      0 163.117.141.197:56932   163.117.141.197:8007    ESTABLISHED
tcp        0      0 127.0.0.1:8100          127.0.0.1:53883         ESTABLISHED
```

Tcpdump *(notación)*

```
Fuente > destino nº secuencia inicial nº secuencia final (nº datos)
```

Herramientas `ssh`

Usar un terminal en otra máquina
maquinaorigen>ssh usuario@ipdestino
Ejemplo:
dds@it011>ssh dds@it012.lab.it.uc3m.es
dds@it012>[todo lo que ejecutemos lo hará en it012]

Herramientas *TCPDump*

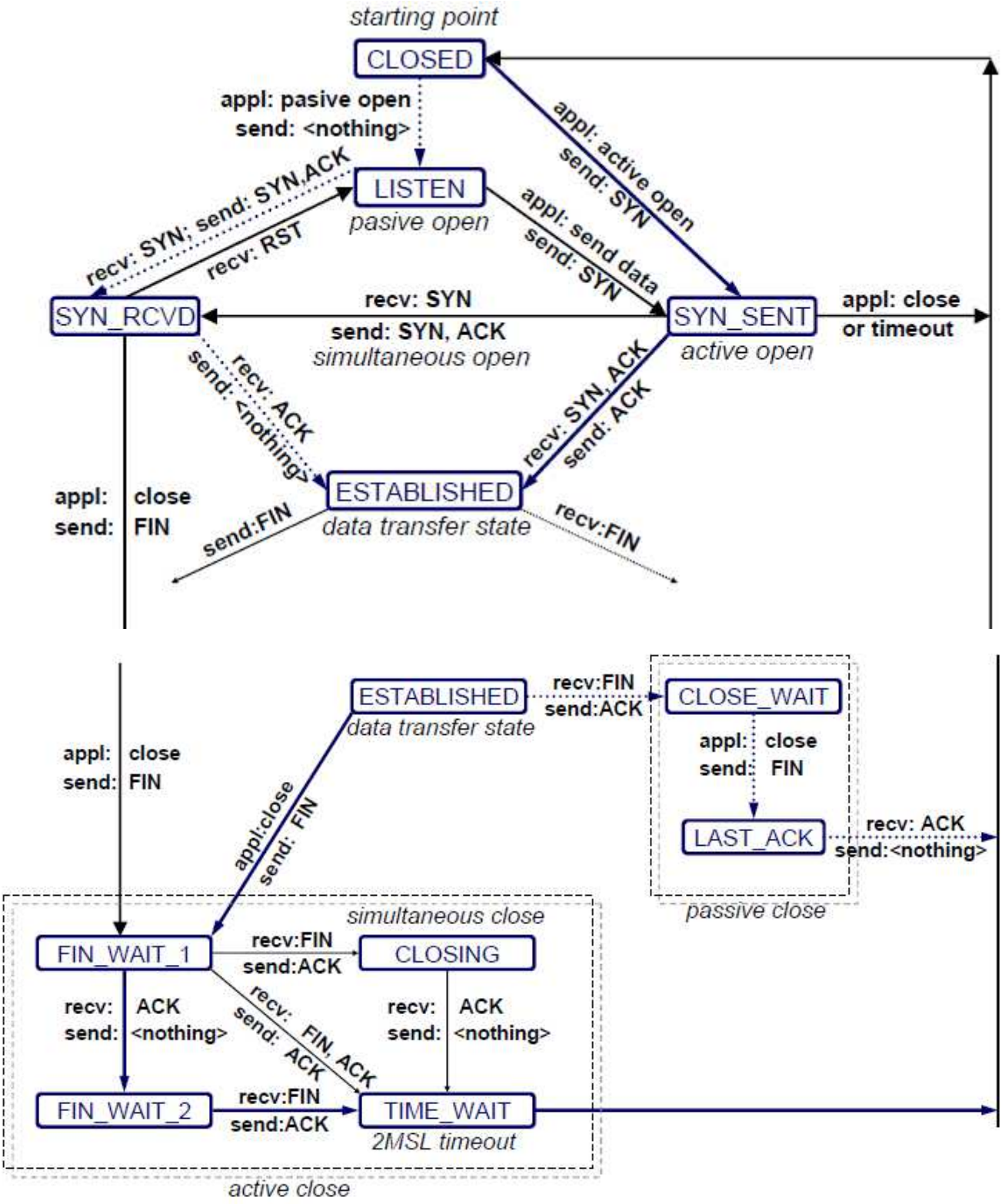
Observar paquetes en interfaz localhost
`sudo tcpdump -i lo port xxxx`
Observar paquetes en interfaz física
`sudo tcpdump -i eth port xxxx`

Sockets cliente

1) Averiguar la dirección IP y puerto del servidor con el que desea comunicarse.
2) Crear un socket: llamada a socket()
Especificar que TCP use cualquier puerto local libre (puerto origen).
Lo hace automáticamente connect()
3) Conectar con el servidor. Esto abre la conexión TCP: llamada a connect().
4) Comunicarse con el servidor. Enviar y recibir por el socket (llamadas a send/recv, write/read, ...).
5) Cerrar el socket: llamada a close().

Sockets servidor

1)Crear un socket: llamada a socket().
2) Asociarlo al puerto local "bien conocido" que ofrece ese servicio: llamada a bind().
3)Poner el socket en modo pasivo, dispuesto a aceptar peticiones de conexión: llamada a listen().
4)Bucle:
Recibir petición de cliente.(accept -> obtenemos socket activo)
4.1)Enviar/recibir por socket (llamadas a send/recv, write/read, ...).
4.2)Cerrar nuevo socket cuando terminemos de atender al cliente (llamada a close()) y volver a punto 4.
5)Cerrar socket pasivo cuando termine el servidor: llamada a close().


starting point
CLOSED
appl: pasive open
send: <nothing>
LISTEN
pasive open
recv: SYN; send: SYN,ACK
recv: RST
appl: active open
send: SYN
appl: send data
send: SYN
SYN_RCVD
recv: SYN
send: SYN, ACK
simultaneous open
SYN_SENT
active open
appl: close
or timeout
recv: ACK
send: <nothing>
recv: SYN, ACK
send: ACK
ESTABLISHED
data transfer state
appl: close
send: FIN
send:FIN
recv:FIN
ESTABLISHED
data transfer state
recv:FIN
send:ACK
CLOSE_WAIT
appl: close
send: FIN
LAST_ACK
recv: ACK
send:<nothing>
passive close
appl: close
send: FIN
appl:close
send: FIN
simultaneous close
FIN_WAIT_1
recv:FIN
send:ACK
CLOSING
recv: ACK
send: <nothing>
recv: FIN, ACK
send: ACK
recv: ACK
send: <nothing>
FIN_WAIT_2
recv:FIN
send:ACK
TIME_WAIT
2MSL timeout
active close