Lab 6. TCP

1. Objectius de la pràctica

Aquesta pràctica té l'objectiu d'estudiar el comportament del protocol TCP i aprendre el funcionament de la comanda `tcpdump`, i especialment saber interpretar el bolcat d'aquesta comanda.

2. Introducció a TCP

TCP és el protocol de nivell de transport que es fa servir en Internet per a la transmissió fiable d'informació. TCP és un protocol extrem a extrem, ARQ (*Automatic Repeat reQuest*), orientat a la connexió, amb els següents objectius: (i) recuperació d'errors, per tenir una transmissió fiable; (ii) control de flux, per adaptar la velocitat entre els dos nodes que es comuniquen; i (iii) control de congestió, per adaptar la velocitat a la xarxa (i evitar així que es col·lapsi).

TCP és un protocol bidireccional, i per a cada direcció es comporta com mostra la Figura 20. Així com l'aplicació escriu la informació que ha d'enviar en el primari, TCP la guarda en un *buffer* de transmissió. Quan el *buffer* està ple, el SO bloqueja l'aplicació fins que torna a haver-hi espai. TCP va agafant aquesta informació i l'envia encapsulada dintre dels segments. A mesura que els segments arriben al secundari, la informació es guarda en un *buffer* de recepció perquè l'aplicació del secundari la vagi llegint. L'objectiu del control de flux és evitar que el *buffer* de recepció s'ompli més aviat del que es llegeix per l'aplicació del secundari (evitant així pèrdues en el receptor). Si la xarxa està congestionada les pèrdues es produiran el *buffer* d'algun dels routers del camí, i s'anomenen pèrdues per congestió.

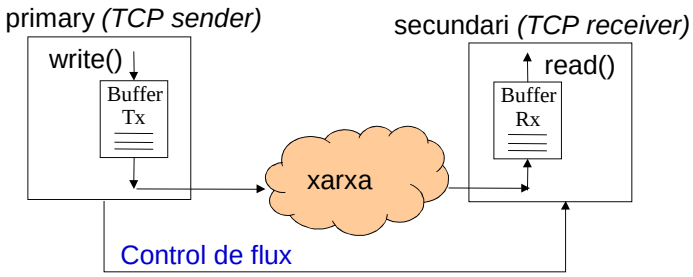


Figura 20: Nivell TCP.

La Figura 21 mostra la capçalera d'un segment TCP. A més del port font i destinació els camps més importants son els següents:

- *sequence number* (número de seqüència).
- *acknowledgement* (o simplement *ack*, confirmació).
- *Length*: mida de la capçalera en words de 32 bits.
- *flags:* U (*urgent*): es fa servir el camp *urgent pointer*. A (*ack*): es fa servir el camp d'ack; P (*push*). passar la informació el més aviat possible a l'aplicació. R (reset): avortar la connexió. S (syn): inici de la connexió. F (fin) terminació de la connexió.
- *Advertized window* (finestra advertida): es fa servir pel control de flux.
- *Options*: Les més importants son: (i) mss (maximum segment size), suggereix la mida del camp d'informació a la màquina remota (típicament la MTU de la xarxa – 40). (ii) timestamp: per a mesurar el retard d'anada i tornada (*round trip time*, RTT) i (iii) sack (*selective acks*): per a donar informació sobre els paquets perduts per a poder fer retransmissió selectiva.

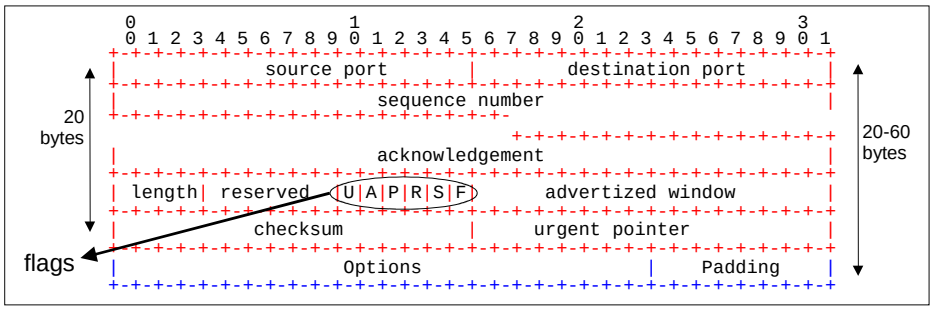


Figura 21: Capçalera TCP.

2.1. Establiment i terminació d'una connexió

La Figura 22 mostra les fases d'establiment (*three way handshake*) i terminació d'una connexió TCP. L'extrem que envia el primer segment és per definició el client. Aquest segment no porta dades, i només té activat el *flag* de syn. El servidor

contesta amb un segment amb el *flag* de syn i ack activats, confirmant l'anterior. Quan el client envïi l'ack la connexió quedarà establerta (established). La terminació es produeix després d'enviar-se segments amb el *flag* de fin activat i els seus respectius acks.

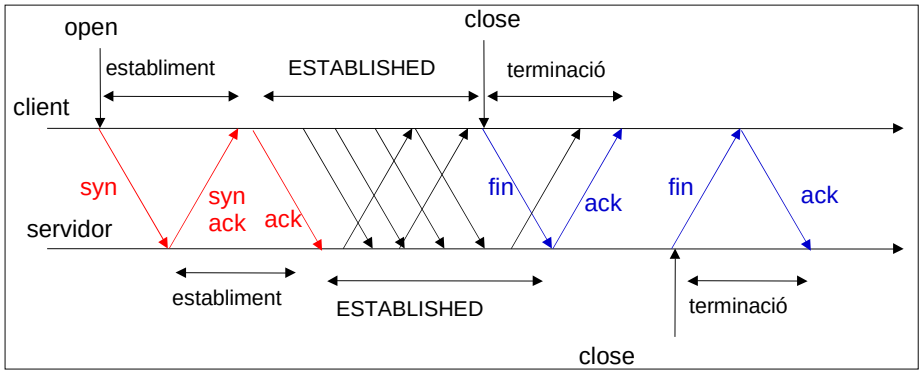


Figura 22: Establiment i terminació d'una connexió TCP.

2.2. Números de seqüència

En TCP el número de seqüència identifica el primer byte de dades que porta el segment. El primer segment porta l'*initial sequence number*, que és un número aleatori de 32 bits. A partir d'aquest valor, el número de seqüència s'incrementa amb el nombre de bytes que porta el segment (veure la Figura 23). La confirmació (ack) identifica el pròxim byte que espera rebre el secundari (i confirma tots els anteriors).

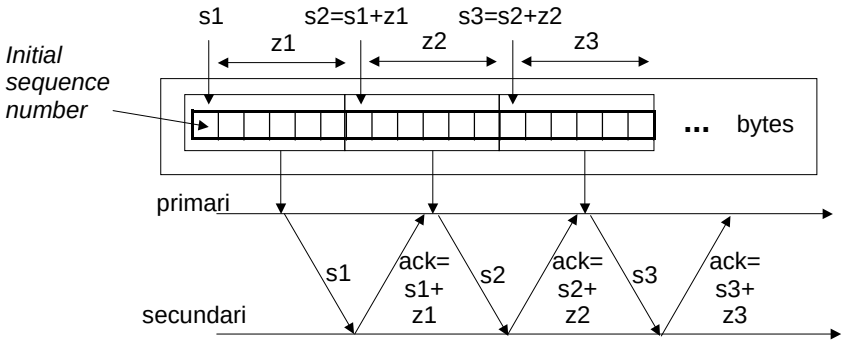
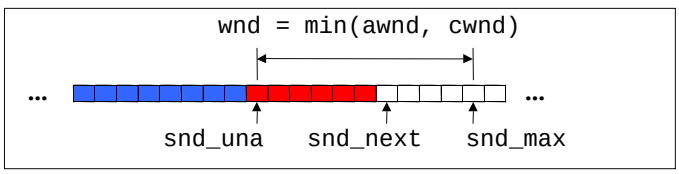


Figura 23: Evolució dels números de seqüència de TCP.

2.3. Mecanisme de finestra

TCP té un mecanisme de finestra variable que ve donada per: wnd = min(awnd, cwnd), on awnd és la finestra advertida pel node remot (control de flux) i cwnd és la finestra de congestió (veure la Figura 24). La finestra advertida s'inicia cada vegada que s'envia un segment al nombre de bytes lliures de la cua de recepció . D'aquesta manera el primari no enviarà mai més bytes dels que pot emmagatzemar el secundari. La finestra de congestió (cwnd) té l'objectiu d'adaptar-se a l'estat de congestió de la xarxa. El seu valor es calcula a partir d'un conjunt d'algorismes. A continuació s'expliquen els més importants.




```
Llegenda:
    segment enviat i confirmat
    segment enviat però no confirmat
    segment encara no enviat
  awnd Finestra advertida
  cwnd Finestra de congestió
snd_una  primer segment no confirmat
snd_next pròxim segment a enviar
snd_max  últim segment que es pot enviar
         (sino es confirmen noves dades)
```

Figura 24: Mecanisme de finestra de TCP.

2.4. Finestra de congestió

Típicament, quan vàries connexions es reparteixen un enllaç d'Internet, el mecanisme de control de congestió de TCP és el responsable d'aconseguir que cada una es quedi amb una part de la velocitat de transmissió de l'enllaç. Si les connexions transmeten "massa", aleshores hi ha pèrdues i les connexions han de transmetre "menys" per adaptar-se a la velocitat efectiva que poden aconseguir de l'enllaç. En aquesta situació, la quantitat d'informació que poden transmetre les

connexions ve donada per la mida de la finestra de congestió (cwnd). Així doncs, transmetre més o menys és equivalent a augmentar/disminuir la mida de cwnd.

TCP fa servir dos algorismes bàsics per a calcular la cwnd: el *slow start* (SS) i el *congestion avoidance* (CA). L'objectiu de l'SS és incrementar cwnd el més aviat possible a un valor on no es produeixin pèrdues per congestió. A partir d'aquest punt, cwnd es calcula amb l'algorisme CA. L'objectiu de CA és incrementar lentament cwnd per poder aprofitar més velocitat de transmissió que pugui quedar disponible. El canvi de SS a CA es produeix quan cwnd assoleix un llindar (*threshold*) mantingut en la variable *slow-start threshold, ssthresh*. La Figura 25 mostra els algorisme SS i CA. Fixeu-vos que mss és la mida d'un segment. Per tant, quan cwnd s'augmenta amb mss (com fa SS), es pot enviar un segment més sense confirmar. Quan cwnd s'incrementa amb mss/cwnd (com fa CA), s'hauran de rebre cwnd segments perquè cwnd s'incrementi amb mss.

Inicialització:

```
cwnd = mss ;
ssthresh = ∞ ;
```

Quan es rep un ack que confirma noves dades:

```
if(cwnd < ssthresh)  /* Slow Start */
      cwnd = cwnd + mss ;
else              /* Congestion Avoidance */
      cwnd = cwnd + mss*mss/cwnd ;
```

Quan s'excedeix el temps màxim d'espera de la confirmació d'un segment (*time-out*):

```
Retransmet el segment snd_una    ;
cwnd = mss ;
ssthresh = max(2, min(awnd, cwnd) / 2) ;
```

Figura 25: Algorismes de *Slow Start* i *Cogestion Avoidance*.

La Figura 26 mostra l'evolució típica de cwnd. Quan s'inicia la connexió, TCP comença amb SS i la cwnd s'incrementa ràpidament fins a la finestra advertida (awnd). Si la transmissió és dintre d'una mateixa LAN típicament no hi ha pèrdues i la finestra de TCP es mantindrà constant i igual a awnd quan cwnd arribi al seu valor. Si hi ha pèrdues (perquè la connexió travessa un enllaç congestionat, aleshores es produiran *time-outs* dels segments que no es confirmen i es retransmetran, reduint cada vegada ssthresh al valor que tenia la finestra en el moment del *time-out*. En aquest cas l'evolució de cwnd segueix una forma de dent de serra com el de la figura.

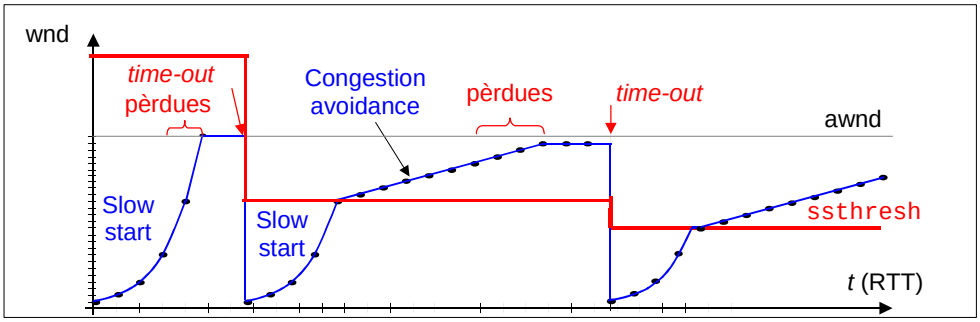


Figura 26: Evolució típica de la finestra de TCP quan hi ha un enllaç congestionat.

3. La comanda tcpdump

La comanda tcpdump permet capturar els paquets que arriben o s'envien des d'una interfície d'acord amb una certa expressió. Els paquets es capturen en el moment en que es passen o es reben pel *driver* de la interfície. Per defecte, tcpdump posa la interfície en mode promiscu, per capturar tots els paquets que hi arriben (vagin dirigits o no a la targeta on es capturen). El format bàsic de la comanda és:

```
tcpdump <opcions> <expressió>
```

Les opcions més comuns son:

- `-i <inteficie>`: captura els paquets de `<interficie>`. Per exemple: `tcpdump –i e0`
- `-n`: Perquè tcpdump no intenti resoldre les adreces als noms.
- `-x`: perquè tcpdump també faci un bolcat en hexadecimal del contingut del paquet.
- `-X`: perquè faci un bolcat en hexadecimal i ASCII del contingut del paquet.
- `-e`: perquè imprimeixi també la capçalera de nivell d'enllaç.
- `-s <n>`: perquè tcpdump capturi fins a <n> bytes de cada paquet (per defecte en captura fins a 64).
- `-c <n>`: captura <n> paquets i acaba

- `-v`: perquè sigui més *verbose* (doni més informació dels paquets capturats). Podem posar `-vv` i `-vvv` perquè doni encara més informació.

Les expressions més comuns son:

- `src|dst host|net|port <i>`: captura els paquets que tenen en el camp font|destinació el host|xarxa|port <i>. Per exemple: `tcpdump src net 10.0.0.0/24`
- `host|net|port <i>`: captura els paquets que tenen en el camp font o destinació el host|xarxa|port <i>. Per exemple: `tcpdump net 10.0.0.0/24`
- `ip|arp|tcp|udp|icmp`: captura paquets d'un d'aquest tipus.

Les expressions admeten els operadors `and`, `or` i `not`. Per exemple:

```
tcpdump –ni e0 icmp and host 10.0.0.1 and not host 10.0.0.2
```

capturarà tots els paquets icmp que tinguin com adreça font o destinació 10.0.0.1 però que no tinguin com a font o destinació l'adreça 10.0.0.2

3.1. Bolcat de tcpdump

Cada vegada que tcpdump captura un paquet, fa un bolcat (*dump*) indicant la informació que tcpdump considera més interessant. La informació que mostra en el bolcat depén del tipus de paquet capturat. La Figura 27 mostra el bolcat d'un segment TCP. Si volem més informació podem demanar a tcpdump que a més del bolcat per defecte ens faci el bolcat del paquet en hexadecimal (opció –x, i asci amb l'opció –X). La Figura 28 n'és un exemple.

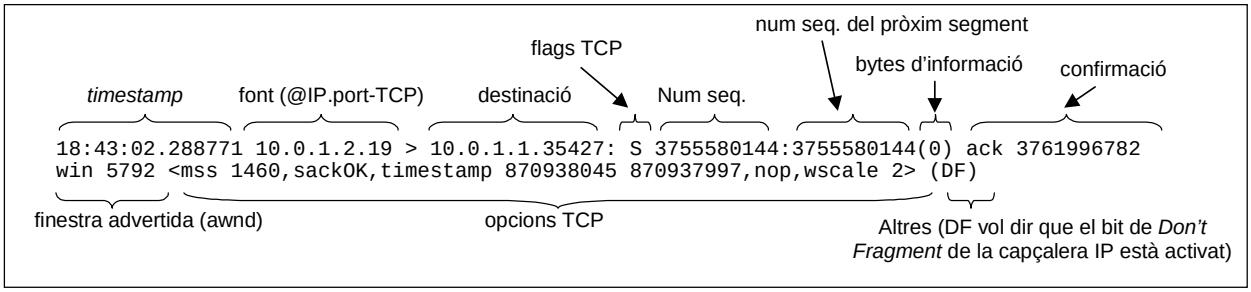


Figura 27: Bolcat d'un segment TCP.

```
xc# tcpdump -Xns 100 -i ppp0
...
18:56:02.628170 10.0.1.1.35434 > 10.0.1.2.21: P 1:17(16) ack 21 win 1460 <nop,nop,timestamp
871718463 871714750> (DF) [tos 0x10]
0x0000   4510 0044 bfe9 4000 3f06 65b8 0a00 0101        E..D..@.?.e.....
0x0010   0a00 0102 8a6a 0015 100a f0d1 105f 6243        .....j....._bC
0x0020   8018 05b4 bf2c 0000 0101 080a 33f5 5e3f        .....,......3.^?
0x0030   33f5 4fbe 5553 4552 2061 6e6f 6e79 6d6f        3.O.USER.anonymo
0x0040   7573 0d0a                                      us..
18:56:02.710769 10.0.1.2.21 > 10.0.1.1.35434: . ack 17 win 1448 <nop,nop,timestamp 871718503
871718463> (DF)
0x0000   4500 0034 2d96 4000 4006 f72b 0a00 0102        E..4-.@.@..+....
0x0010   0a00 0101 0015 8a6a 105f 6243 100a f0e1        .......j._bC....
0x0020   8010 05a8 3874 0000 0101 080a 33f5 5e67        ....8t......3.^g
0x0030   33f5 5e3f                                      3.^?
^C
```

Figura 28: Bolcat de tcpdump en hexadecimal.

Cal destacar el següent:

- El timestamp té el format hora:minuts:segons. Com que els segons es donen amb 6 decimals, tenim una resolució de microsegons (en l'exemple de la Figura 27, el paquet s'ha capturat a les 18:43 i 2 segons, 288 ms, 771 μs). El bolcat no diu si el paquet que s'ha capturat s'ha rebut o transmés. Això ho podem deduir de les adreces. Per exemple, si s'ha capturat en la màquina 10.0.1.2, aleshores el paquet s'ha transmés.

- Per a seguir millor la traça, tcpdump dóna el número de seqüència del paquet i el número de seqüència que portarà el següent paquet. D'aquesta forma, podem veure fàcilment quan es transmeten segments fora d'ordre (que normalment és una indicació de que s'han perdut segments). A més, si tcdpump captura els paquets de syn, normalitza el número de seqüència restant el número de seqüència inicial perquè sigui més fàcil de llegir (tal com mostra la Figura 28). A més, amb aquesta normalització el número de seqüència ens diu directament quants de bytes d'informacío s'han enviat. Si un paquet no porta bytes d'informació, típicament tcpdump no ens mostra els números de seqüència sino només la confirmació (segon paquet de la traça de la Figura 28).

Fixeu-vos que per parar la captura de paquets s'ha de premer CONTROL-C. En el bolcat del primer segment de la Figura 28 podem veure que la capçalera IP (la teniu en la Figura 29) té 20 bytes. Això ho podem deduïr perquè el primer byte del bolcat és 45: 4 és la versió i 5 és la mida de la capçalera en words de 32 bits (és a dir, 5 x 4 = 20 bytes). Si contem 10

grups de 4 xifres hexadecimals (els 20 bytes de la capçalera IP) arribem on comença la capçalera TCP. De la capçalera TCP (Figura 21) deduïm que 8a6a és el port font, 0015 és la destinació, 100af0d1 és el número de seqüència, 105f6243 és la confirmació i 8 és la mida de la capçalera (32 bytes). Així doncs, la capçalera TCP porta 12 bytes d'opcions (l'opció *timestamp* més el *padding*).

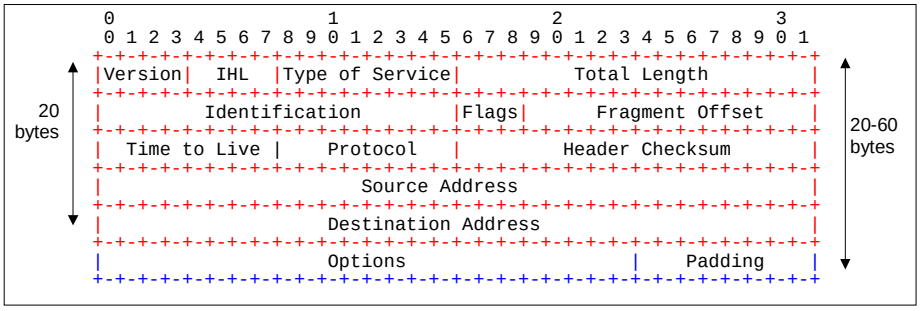


Figura 29: Capçalera IP.

4. Realització de la pràctica

Per a fer la pràctica capturarem segments TCP d'una connexió que hi ha entre un client i un servidor a través d'un router, tal com mostra la Figura 30. Els enllaços són ethernet.

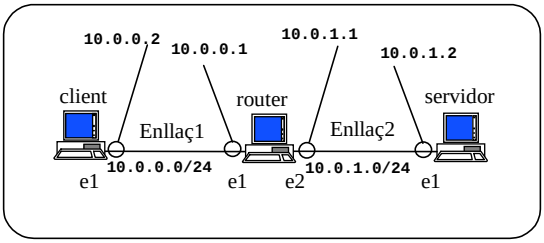


Figura 30: Topologia de la pràctica.

Per poder seguir més fàcilment la captura de les traces de TCP, en la imatge "xarxes" les opcions sack i window-scaling estan desactivades. Altrament es poden desactivar amb les comandes de la Figura 31.

```
servidor# sysctl -w net.ipv4.tcp_sack=0
servidor# sysctl -w net.ipv4.tcp_window_scaling=0
```

Figura 31: Desactivació de l'opció *sack* i *window-scaling* en el servidor.

4.1. Anàlisi dels segments d'una connexió interactiva TCP

Configura la xarxa de la Figura 30. Assegura't fent ping que hi ha connexió entre el client i el servidor.

1. Executa wireshark en el client.
2. Executa telnet 10.0.1.2 en el client, usuari xc, password xc. Observa les comandes capturades amb wireshark al mateix temps que hi ha el diàlec de la connexió telnet. Identifica el *three-way-handshake*. Observa amb els acks que els segments de SYN consumeixen un número de seqüència, i no porten cap byte de dades.
3. Observa en el bolcat de wireshark que la informació no està encriptada. Identifica en el bolcat l'usuari i el password.
4. Executa "netstat –nat" en el client i el servidor. Identifica els sockets que pertanyen a la connexió i l'estat del socket TCP. Identifica quin és el well-kwon port i el port efímer.
5. Executa exit en la connexió telnet i oberva el missatges de FIN en la desconnexió. Comprova amb els acks que els segments de FIN consumeixen un número de seqüència.
6. Després de tancar la connexió telnet observa els estats dels sockets del client i el servidor. Comprova que un dells s'ha tancat, mentre l'altra continua en TIME-WAIT durant uns 2 minuts abans de tancar-se.

4.2. Anàlisi dels segments d'una connexió bulk-TCP en una LAN

```
client# tcpdump -ni e1 port chargen
client# telnet 10.0.1.2 chargen
```

Figura 32: Captura d'una connexió al servidor de chargen.

1. Engega tcpdump, i connecta't al port de chargen executant les comandes en dos terminals diferents (Figura 32). El servidor chargen (port 19) envia una seqüència de caràcters ASCII pseudo-aleatòria a la velocitat màxima que permet l'enllaç. Nota: Per a congelar el scroll de pantalla prémer Ctrl-S, per continuar Ctrl-Q.
2. Estima la velocitat de transmissió eficaç de la connexió. Si necessites calculadora, recorda que en la barra d'aplicacions de l'escriptori en tens una. Fixa't que podem estimar la velocitat eficaç a partir dels números de seqüència del primer

i últim segment d'informació de la traça capturada en el client, dividit per l'interval de temps que hi ha entre aquests dos segments.

3. Comprova si hi ha pèrdues. Justifica perquè n'hi ha, o no.
4. Observar que la connexió fa servir l'opció delayed-ack (mentre no hi ha pèrdues el receptor envia un ack cada 2 segments de dades).
5. Observa la relació que hi ha entre els acks rebuts i el número de seqüència dels segments d'informació que s'envien immediatament després de l'ack. Fixa't que en la traça capturada en el servidor la diferència augmenta de cada vegada més, mentre que en la traça capturada en el client la diferència és 0. Perquè és així?
6. Justifica que en la traça capturada en el servidor, la diferència entre el número de seqüència rebut en l'ack i el del segment d'informació que s'envia a continuació, és aproximadament el nombre de bytes d'informació de la connexió que hi ha "en vol". És a dir, bytes enviats pel servidor però que encara no han arribat al client, i per tant, que estan emmagatzemats en el router o que s'han perdut. Fixa't que la mida de la finestra que està fent servir TCP és aquesta diferència més el nombre de bytes d'informació que hi ha en els paquets que envia immediatament després (fins que rep un ack de noves dades). Relaciona l'evolució de la finestra amb el *slow start*.
7. Mira l'evolució de la finestra advertida pel client i el servidor. Perquè penses que la del client varia contínuament i la del servidor no? Mira els últims paquets de la traça capturada en el servidor. Compara la mida de la finestra advertida pel client amb la fa servir el servidor (deduïda de la traça). Quina finestra penses que està limitant la transmissió: l'advertida (awnd) o la de congestió (cwnd)?
8. En aquest experiment es tracta de que el client deixi de llegir el socket perquè s'ompli, i enviï una finestra advertida (awnd) igual a 0. Per parar la connexió amb el servidor i accedir al *prompt* de telnet prémer "Ctrl-AltGr-]" "enter" . Des del *prompt* de telnet es pot sortir amb la comanda quit o continuar al prémer la tecla enter. Connectar el client al servidor de chargen executant "`tcpdump -nli e1 port chargen`" i parar la connexió amb Ctrl-AltGr-]. Observar l'evolució de la finestra advertida que envia el client. Què fa el servidor quan la finestra val 0?
9. Fes servir netstat per veure el nombre de bytes que hi ha en la cua de recepció i transmissió dels sockets en el costat del client i del servidor. Raona les diferències.
10. Amb la connexió de chargen establerta, prova l'execució de tcpdump fent servir expressions (veure la secció 3). Per exemple, les següents comandes capturen els segments que envia o rep el servidor, és a dir, els segments de dades i acks, respectivament.

```
# tcpdump -ni e1 tcp and src 10.0.1.2
# tcpdump -ni e1 tcp and dst 10.0.1.2
```

4.3. Anàlisi dels segments d'una connexió bulk-TCP amb pèrdues

Per introduir pèrdues afegirem una cua de mida i velocitat fixades per nosaltres a la sortida de l'enllaç1 del router, tal com mostra la Figura 33. Linux permet afegir aquesta cua amb la comanda de la Figura 34 (els paràmetres de la cua es poden modificar canviant `add` per `change` en la mateixa comanda, i la cua es pot eliminar canviant `add` per `del`). Els paquets sortiran d'aquesta cua a una velocitat de 100kbps. Si els paquets arriben a una velocitat major, la cua s'omplirà fins un màxim de 10.000 bytes. Els paquets que arribin quan la cua està plena es descartaran. Fixeu-vos que la cua només afecta a un sentit de l'enllaç:

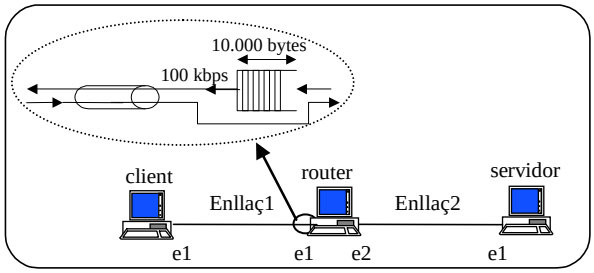


Figura 33: Cua que afegirem a la sortida de l'enllaç e1 del router.

```
router# tc qdisc add dev e1 root tbf burst 5000 rate 100kbit limit 10000
```

Figura 34: Comanda per afegir la cua de la Figura 33.

11. Configura la cua i repeteix les comandes de la Figura 32.
    a) Congela la connexió (Ctrl-S) quan hi ha primers acks duplicats i es produeix la retransmissió del segment perdut. Mira la traça capturada en el client i comprova que efectivament aquest paquet s'havia perdut.
    b) Estima quin és el nombre de bytes d'informació que hi ha "en vol" quan es produeixen pèrdues. Comprova que és major de 10.000 bytes. Comprova que després de la retransmissió la finestra de tcp poc a poc va augmentant fins que torna a haver-hi pèrdues.
12. Estima quina és la velocitat eficaç. Comprova que és aproximadament 100 kbps.

13. Estima que val el RTT en el three-way-handshaking i quan es produeixen pèrdues. Perquè és diferent? Estima quin hauria de ser el RTT degut al retard que introdueix el buffer que hem afegit amb la comanda tc en el router. Comprova que quan es produeixen pèrdues el RTT coincideix aproximadament amb el retard que has calculat.

5. Informe previ

El següent bolcat mostra el timestamp del primer paquet, i els últims 5 paquets d'una captura amb tcpdump. A la vista del bolcat, respon les següents preguntes:

```
1. 18:37:12.234583
2. …
3. 18:38:28.739407 IP 147.83.30.137.22 > 80.102.159.44.1035: P 4672:4801(129) ack 4805119 win 32480
4. 18:38:28.739652 IP 80.102.159.44.1035 > 147.83.30.137.22: P 4805119:4805151(32) ack 4801 win 2092
5. 18:38:28.739729 IP 80.102.159.44.1035 > 147.83.30.137.22: F 4805151:4805151(0) ack 4801 win 2092
6. 18:38:28.851394 IP 147.83.30.137.22 > 80.102.159.44.1035: F 4801:4801(0) ack 4805152 win 32480
7. 18:38:28.851458 IP 80.102.159.44.1035 > 147.83.30.137.22: . ack 4802 win 2092
```

1) Quina és l'adreça IP del client i del servidor?
2) Donar un possible bolcat pel que falta en la primera línia.
3) Quants bytes d'informació (contingut del camp payload) han enviat exactament el client i el servidor?
4) Estimar la velocitat eficaç.
5) Quins son els estats de TCP per els que passa el client i el servidor durant els 5 últims paquets del bolcat?