La capa de transport de dades

Xavier Vilajosana Guillén
René Serral i Gracià
Eduard Lara Ochoa
Miquel Font Rosselló

PID_00171175

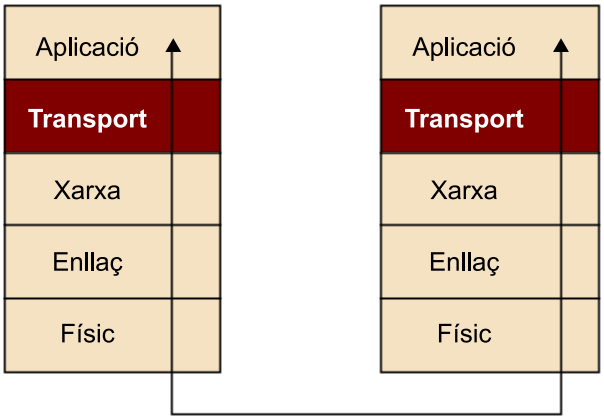
Universitat Oberta de Catalunya
www.uoc.edu

Índex

Introducció

La capa de transport s'encarrega de proveir la comunicació d'extrem a extrem entre processos d'aplicacions ubicades en diferents equips finals. Des del punt de vista de l'aplicació, és com si els equips finals (*hosts*) estiguessin directament connectats, però en realitat podrien estar en llocs oposats del planeta, connectats mitjançant múltiples encaminadors (*routers*) i tipus d'enllaços diferents. La capa de transport ofereix les funcionalitats bàsiques per a assolir aquest nivell de comunicació lògica, sense que les aplicacions s'hagin de preocupar de la infraestructura de xarxa subjacent.

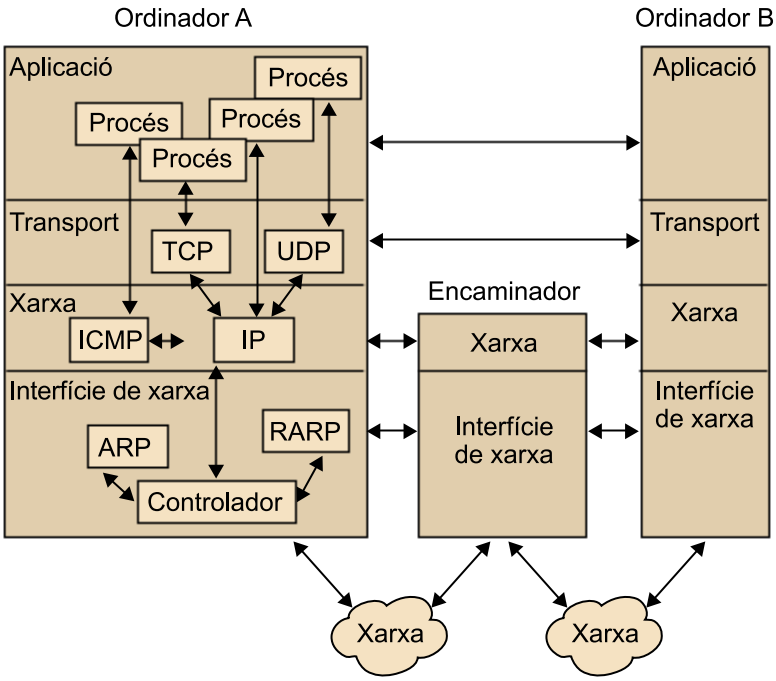
Figura 1. Missatges d'extrem a extrem



Els protocols de la capa de transport s'implementen en els dispositius extrems de la xarxa i no pas en els encaminadors ni dispositius intermedis. La informació transmesa per les aplicacions és convertida a paquets de la capa de transport, coneguts com a **segments de la capa de transport**. Els segments de la capa de transport es construeixen dividint la informació que vol transmetre l'aplicació en segments d'una mida determinada, i afegint-hi una capçalera. Cada segment és enviat a la capa de xarxa, on seran inclosos en els paquets del nivell de xarxa, coneguts com a **datagrames**. A partir de la capa de xarxa, els datagrames són enviats als destinataris passant per diferents dispositius que només examinaran la informació corresponent a la capa de xarxa. Només el receptors en rebre el datagrama a la capa de xarxa extrauran el segment de transport i el passaran a la capa de transport del receptor. La capa de transport processarà el segment i el passarà a l'aplicació corresponent.

En aquest mòdul estudiarem amb detall els protocols i funcionalitats de la capa de transport de dades, fent èmfasi en els protocols de transport de dades d'Internet, l'UDP i el TCP, que ofereixen serveis diferents a les aplicacions que els invoquen.

Figura 2. La jerarquia TCP/IP



1. Serveis oferts per la capa de transport

La capa de transport ofereix les funcionalitats següents:

- Garanteix la transmissió sense errors, d'extrem a extrem, independentment del tipus de xarxa.
- Controla la transmissió d'extrem a extrem.
- És responsable del control de flux de les dades i el control de congestió de la xarxa.
- És responsable d'establir, mantenir i finalitzar les connexions entre dos equips finals o un equip final i un servidor en una xarxa.
- Assegura que les dades arribin sense pèrdues, sense errors i sense ser duplicades.
- Ordena els paquets que arriben.
- S'encarrega de fragmentar els missatges i recompondre'ls en la destinació quan és necessari.
- Permet la multiplexació de diverses connexions de transport sobre una mateixa connexió de xarxa.

Serveis de la capa de transport

Malgrat que no tots els protocols de la capa de transport ofereixen tots aquests serveis. L'UDP, com veurem més endavant, no garanteix la transmissió sense errors d'extrem a extrem, entre d'altres.

2. Relació entre la capa de transport i la capa de xarxa

La capa de transport s'ubica just per sobre de la capa de xarxa en la pila de protocols.

Mentre que la capa de transport s'encarrega de proveir de comunicació lògica entre processos que s'executen en equips finals diferents, la capa de xarxa proveeix de comunicació lògica entre equips finals.

Aquesta diferència és substancial, ja que la capa de transport ha de permetre que múltiples processos es comuniquin de manera lògica fent ús d'una única connexió lògica proveïda per la capa de xarxa. Aquest concepte s'anomena **multiplexació** i **desmultiplexació** de la capa de transport.

A més, la capa de xarxa no dóna garanties de lliurament de la informació, no garanteix el lliurament dels segments i tampoc no garanteix la integritat de la informació continguda en el segment. Per aquesta raó la capa de xarxa és **no confiable**. La missió, doncs, de la capa de transport, és proveir de transmissió de dades fiable i permetre la comunicació procés a procés en una xarxa creada comunicant equip final amb equip final.

Per a introduir els protocols del nivell de transport, ens centrem en la jerarquia de protocols utilitzats a Internet, que defineix dos protocols de transport: l'UDP i el TCP.

El protocol UDP és no orientat a la connexió, cosa que vol dir que no implementa fases d'establiment de la connexió, enviament de dades i acabament de la connexió, mentre que el TCP és orientat a la connexió.

En el cas de la jerarquia TCP/IP[1], es defineixen dues adreces que relacionen el nivell de transport amb els nivells superior i inferior:

(1) *TCP/IP* és la sigla de *transmission control protocol / Internet protocol.*

1) L'adreça IP és l'adreça que identifica un subsistema dins una xarxa.
2) El port identifica l'aplicació que requereix la comunicació.

Per a identificar les diferents aplicacions, els protocols TCP[2] i UDP[3] marquen cada paquet (o unitat d'informació) amb un identificador de 16 bits anomenat **port**.

(2) TCP és la sigla de *transmission control protocol.*

(3) UDP és la sigla d'*user datagram protocol.*

- **Ports coneguts**: són regulats per la IANA[4]. Ocupen el rang inferior a 1024 i són utilitzats para accedir a serveis oferts por servidors.

(4) *IANA* és la sigla de la *Internet Assigned Numbers Authority.*

- **Ports efímers**: són assignats de manera dinàmica pels clients dins d'un rang específic per sobre de 1023. Identifiquen el procés del client només mentre dura la connexió. També, però, hi ha alguns ports efímers que són coneguts, ja que són usats per aplicacions específiques. Alguns exemples són el 8080, usat per alguns servidors d'aplicacions, o el 6667, usat per una xarxa parell a parell[5]. Per altra banda hi ha aplicacions que fan servir ports de manera dinàmica, ja que no en tenen cap d'específic.

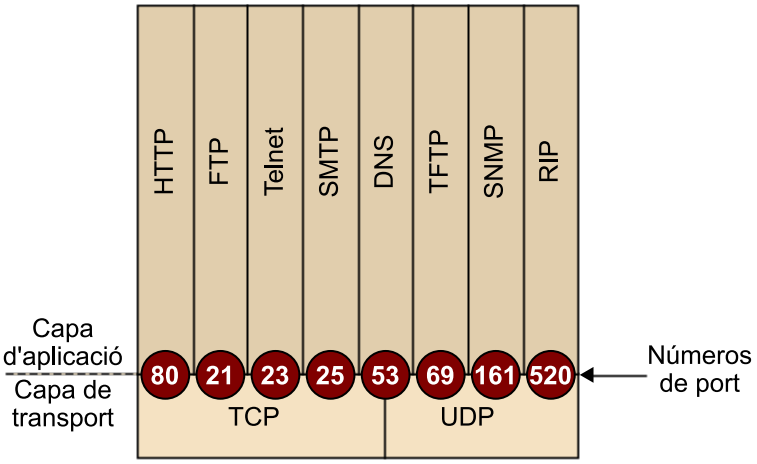
[5] En anglès, *peer to peer*.

0-255 (IANA)	Aplicacions públiques
255-1023 (IANA)	Assignats a empreses amb aplicacions comercials
>1023	No estan registrats (efímers)

Figura 3. Ports d'aplicacions públiques establerts per la IANA

Decimal	Paraula clau	Descripció
0		Reservat
1-4		No assignat
5	RJE	Entrada remota de tasques
7	ECHO	Eco
9	DISCARD	Descartar
11	USERS	Usuaris actius
13	DAYTIME	De dia
15	NETSTAT	Qui està connectat o NETSTAT
17	QUOTE	Citació del dia
19	CHARGEN	Generador de caràcters
20	FTP-DATA	Protocol de transferència d'arxius (dades)
21	FTP	Protocol de transferència d'arxius
23	TELNET	Connexió de terminal
25	SMTP	Protocol SMTP (*simple mail transfer protocol*)
37	TIME	Hora
39	RLP	Protocol d'ubicació de recursos
42	NAMESERVER	Servidor de nom d'amfitrió
43	NICNAME	Qui és
53	DOMAIN	Servidor de denominació
67	BOOTPS	Servidor de protocol d'arrencada
68	BOOTPC	Client del protocol d'arrencada
69	TFTP	Protocol trivial de transferència d'arxius
75		Qualsevol servei privat de connexió telefònica
77		Qualsevol servei RJE privat
79	FINGER	Finger
80	HTTP	Protocol de transferència d'hipertext
95	SUPDUP	Protocol SUPDUP
101	HOSTNAME	Servidor de nom d'amfitrió NIC
102	ISO-TSAP	ISO-TSAP
113	AUTH	Servei d'autenticació
117	UUCP-PATH	Servei de ruta UUCP
123	NTP	Protocol de temps de xarxa
137	NetBIOS	Servei de noms
139	NetBIOS	Servei de datagrames
143	IMAP	*Interim mail access protocol*
150	NetBIOS	Servei de sessió
156	SQL	Servidor SQL
161	SNMP	*Simple network management protocol*
179	BGP	*Border gateway protocol*
190	GACP	*Gateway access control protocol*
194	IRC	*Internet relay chat*
197	DLS	Servei de localització de directoris
224-241		No assignat
242-255		No assignat

Figura 4. Ports usats per alguns dels protocols de nivell d'aplicació



Reflexió

Coneixes aplicacions comercials o de programari lliure que facin servir un port determinat? Quins ports fan servir l'Skype, l'Emule, el BitTorrent i l'MSN?

3. Transport no orientat a la connexió: UDP

L'UDP és un protocol no orientat a la connexió, de manera que no proporciona cap tipus de control d'errors ni de flux, tot i que utilitza mecanismes de detecció d'errors. En cas de detectar un error, l'UDP no lliura el datagrama a l'aplicació, sinó que el descarta.

Cal recordar que per sota, el protocol UDP està fent servir IP, que també és un protocol no orientat a la connexió. L'UDP el que ofereix per sobre d'IP és la possibilitat de multiplexar connexions de diferents processos sobre una mateixa connexió de xarxa, per mitjà dels ports.

La simplicitat del protocol UDP fa que sigui ideal per a aplicacions que requereixen poc retard en l'enviament de la informació (per exemple aplicacions en temps real o el DNS[6]). L'UDP també és ideal per als sistemes que no poden implementar un sistema tan complex com el TCP.

(6)*DNS* és la sigla de *domain name server*.

L'UDP és útil en aplicacions en què la pèrdua de paquets no sigui un factor crític, com telefonia o videoconferència, monitoratge de senyals, etc. Aquestes aplicacions no toleren retards gaire variables, o sigui, si un datagrama arriba més tard de l'instant que hauria de ser rebut, llavors es descarta perquè és inservible per a l'aplicació. Això es tradueix en disrupcions en el so o la imatge, que si bé no són desitjables no impedeixen la comunicació. La taula següent mostra algunes de les aplicacions més comuns que fan ús d'UDP.

Aplicacions	Ports
TFTP	69
SNMP	161
DHCP-BOOTP	67/68
DNS	53
RTP	Ports parells dinàmics començant pel 1234

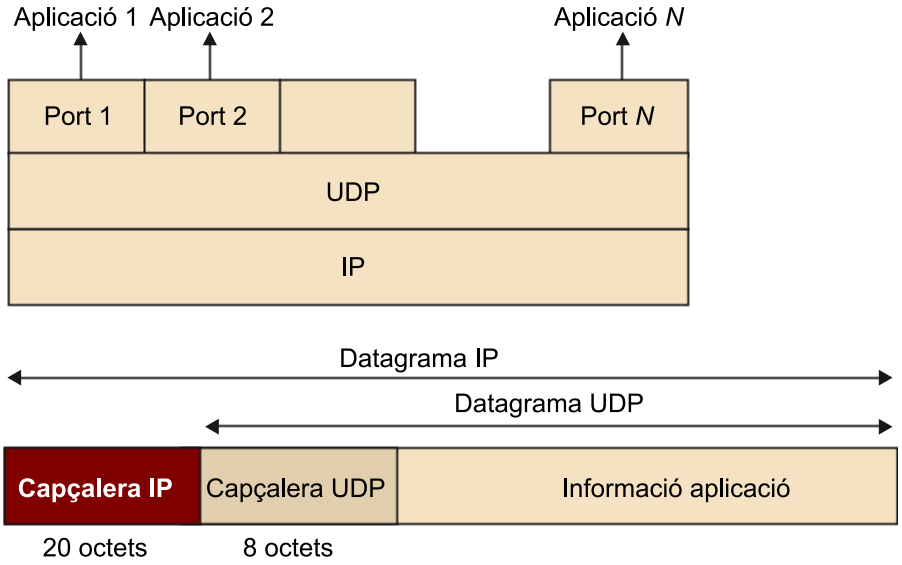
3.1. Encapçalament UDP

La unitat d'encapsulament d'UDP és el datagrama UDP.

- Cada escriptura per part de l'aplicació provoca la creació d'un datagrama UDP. No hi ha segmentació.

- Cada datagrama UDP creat és encapsulat dintre d'un datagrama IP (nivell 3) per a transmetre'l.

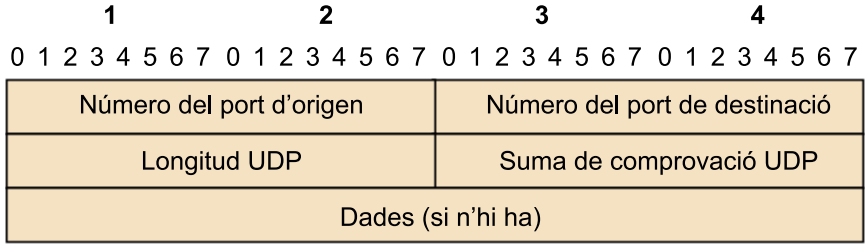
Figura 5



3.2. Capçalera UDP

La capçalera del datagrama UDP està formada per 8 octets. Té quatre camps:

Figura 6



- **Ports** (**origen** i **destinació**): permeten identificar els processos que es comuniquen.

- **Longitud total del datagrama UDP** (*payload* UDP + 8): és un camp redundant, ja que IP duu la longitud també. Totes dues capçaleres especifiquen una mida màxima per a un datagrama de 65.335 octets (16 bits de longitud de datagrama):
  - Longitud màxima d'un datagrama UDP: 65.335 – 20 octets = 65.315 octets
  - Longitud mínima d'un datagrama UDP: 8 octets
  - Longitud de dades d'un datagrama UDP: 65.315 – 8 octets = 65.307 octets
  - La mida d'un datagrama UDP depèn de l'SO. Gairebé totes les API limiten la longitud dels datagrames UDP (el mateix per a TCP) a la màxima longitud que les memòries intermèdies[7] de lectura i escriptura defini-

[7] En anglès, *buffer*.

des per l'SO (anomenades en el sistema *read*() i *write*()). Se solen usar 8.192 octets com a grandària màxima del datagrama UDP (FreeBSD).

- Suma de comprovació UDP (*UDP checksum*): detector d'errors, l'objectiu del qual consisteix a detectar que ningú no ha modificat el datagrama UDP i que arriba a la seva destinació correctament. Si la suma de comprovació[8] UDP és errònia, es descarta el datagrama UDP i no es genera cap tipus de missatge cap a l'origen. La suma de comprovació s'aplica conjuntament a una pseudocapçalera més la capçalera UDP més el camp de dades. Aquesta pseudocapçalera té tres camps de la capçalera del paquet IP que s'envia. Cal esmentar que la suma de comprovació del datagrama IP només cobreix la capçalera IP. La suma de comprovació es calcula fent el complement a 1 de la suma de totes les paraules de 16 bits que conformen el datagrama UDP.

[8]En anglès, *checksum*.

Exemple

Suposem que tenim el datagrama (descompost en tres paraules de 16 bits):

```
0110011001100000
0101010101010101
1000111100001100
```

La suma de les dues primeres paraules de 16 bits és:

```
0110011001100000
0101010101010101
_________________
1011101110110101
```

Afegint la tercera paraula obtenim:

```
1011101110110101
1000111100001100
_________________
0100101011000010
```

Ara fem el complement a 1 (canviant 0 per 1) i obtenim:

```
1011010100111101
```

Aquesta darrera paraula s'afegeix també al datagrama UDP. En la recepció es fa la suma de totes les paraules de 16 bits, que ha de donar:

```
1111111111111111
```

4. Principis de transferència fiable de dades

En aquesta secció es consideraran els principis de transferència fiable de dades en un context general. Aquests principis són genèrics i poden ser aplicats a qualsevol protocol de nivell de transport que busqui garantir la transferència fiable de les dades.

Vegeu també

Vegeu com el protocol TCP explota els principis de transferència de dades amb més detall en l'apartat 6 d'aquest mòdul didàctic.

Un canal fiable de comunicació és aquell que assegura que la informació transmesa arriba d'extrem a extrem sense errors, sense pèrdues, i que la informació arriba en el mateix ordre en què ha estat enviada. Com veurem, adreçar aquest problema no és trivial i requereix un seguit de mecanismes que permetin, donat un canal no fiable, controlar el flux de la informació i controlar també els errors que es produeixen en el canal.

Per tal d'assegurar la transferència fiable de dades, tal com s'ha dit, cal controlar els errors que es produeixen en la transmissió.

Quan es rep un datagrama amb errors, es poden adoptar una de les solucions següents:

- Descartar la trama: és útil en aplicacions que toleren un cert grau d'error en la informació rebuda.
- Intentar corregir els errors amb el codi correcte corresponent (s'evita un retard però els codis per a corregir errors requereixen molta redundància de la informació).
- Sol·licitar la retransmissió: es produeix un retard.

La majoria de protocols opten per sol·licitar la retransmissió del datagrama. Els protocols de demanda de repetició automàtica ARQ[9] recullen un seguit de mecanismes que permeten assegurar la transmissió fiable de la informació, que aquesta arribi sense errors, sense duplicats i mantenint l'ordre d'enviament.

(9)*ARQ* és la sigla d'*automatic repeat request*.

4.1. Protocols ARQ: control d'errors

Els protocols ARQ s'encarreguen de retransmetre de manera automàtica la informació que no arriba o arriba amb errors al destinatari. El funcionament es basa en l'enviament de missatges de confirmació[10] feta pel destinatari d'aquells datagrames que han arribat. (En alguna versió només s'avisa d'aquells que no han arribat, com veurem més endavant.)

(10)En anglès, *acknowledgements*, abreujat *ack*.

Hi ha dues tècniques ARQ:

Transmissions orientades a caràcter			Transmissions orientades a bit	
Idle RQ	Stop & Wait	Ret. Implícita	**Retransmissió contínua**[11]	Go-Back-N
		Ret. Explícita		Retransmissió selectiva

(11)En anglès, *continuous RQ*.

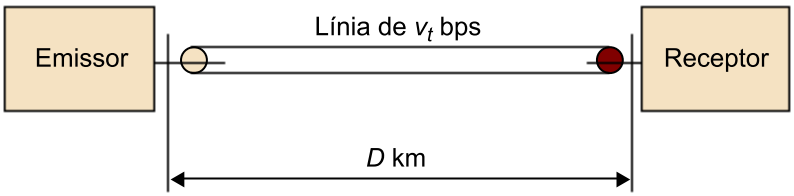
El funcionament bàsic d'un protocol ARQ és el següent:

- L'emissor envia datagrames d'informació, que es van desant en una memòria intermèdia de transmissió.
- Quan la memòria intermèdia de transmissió està plena, l'emissor bloqueja l'escriptura al nivell superior fins que rebi confirmacions de datagrames rebuts pel receptor.
- A mesura que les confirmacions arriben a l'emissor, aquest esborra la informació confirmada de la memòria intermèdia per tal que el nivell de xarxa superior hi pugui tornar a escriure.
- En cas d'error, l'emissor sempre pot tornar a enviar la informació no confirmada, ja que la manté en la memòria intermèdia.

Terminologia

En algunes referències, l'emissor és conegut també com a *primari*. El receptor s'anomena en molts casos *secundari*.

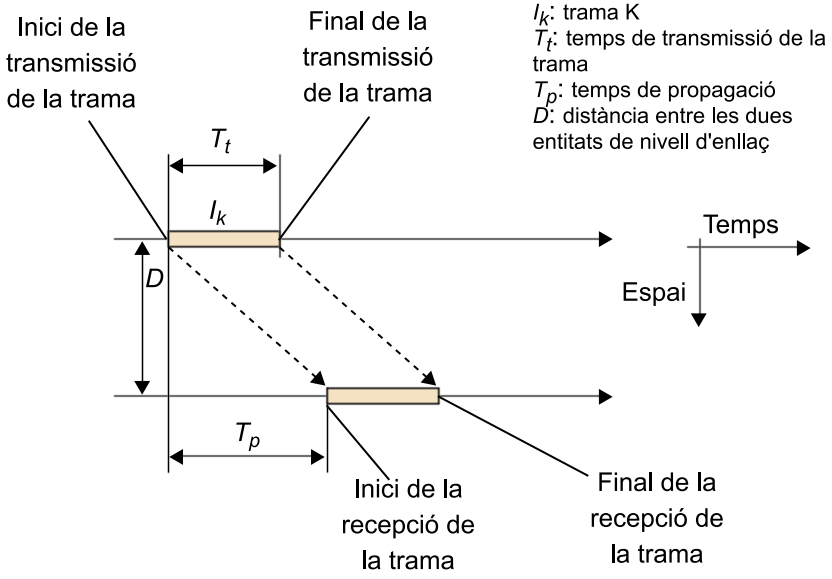
Figura 7



4.2. Diagrames de temps

Els diagrames de temps que farem servir en aquesta assignatura són una representació espaciotemporal de la transmissió de datagrames entre les dues entitats de nivell de transport. Aquest tipus de diagrames també es fa servir per a l'estudi de la comunicació entre entitats en molts altres contextos, com el nivell d'enllaç. Com a exemple d'aquest tipus de diagrames, la figura 8 mostra la transmissió d'una trama anomenada I_K. La interpretació de l'índex K és la següent: si enumerem la seqüència de datagrames transmesos amb els números (1, 2, 3, ...), I_K representa el datagrama K de la seqüència transmesa.

Figura 8. Diagrama de temps de la transmissió d'una trama

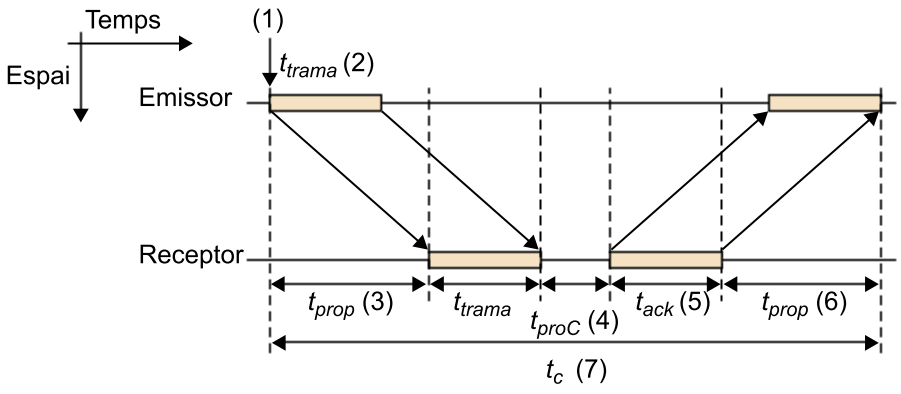


En la figura 8 hi ha dos eixos temporals; sobre cada un es representen els esdeveniments que tenen lloc en cada una de les estacions (emissor i receptor): en l'eix superior, els que es duen a terme en l'estació que transmet el datagrama, i en l'inferior, els que es duen a terme en l'estació que el rep.

4.3. Càlculs sobre els diagrames de temps

La figura 9 ens exemplifica la comunicació entre un emissor i un receptor. Els nombres entre parèntesis ens indiquen punts on prenem mesures temporals.

Figura 9



1) El nivell superior escriu la informació que s'ha de transmetre. L'emissor acobla el datagrama d'informació I_K de L_t bits amb aquesta informació, la desa en la memòria intermèdia de transmissió i la passa al nivell inferior per a transmetre-la. Generalment suposarem un temps de processament de 0 s:

$$t_{processament\ transmissió} = 0$$

2) t_{Trama} és el temps que es triga a posar el datagrama d'informació en la línia de transmissió. Si v_t és la velocitat de transmissió de l'enllaç:

$$t_{Trama} = \frac{L_{Trama}\ (\text{bits})}{v_t\ (\text{bps})}$$

3) t_{prop} és el temps de propagació de cada bit pel medi. Si v_{prop} és la velocitat de propagació del medi:

$t_{prop} = \frac{D\ (\text{m})}{v_{prop}\ (\text{m/s})}$, en què $v_{prop} = 3 \cdot 10^8$ m/s en el buit o $2 \cdot 10^8$ m/s en un conductor

4) t_{proc} és el temps de processament del datagrama rebut. Quan arriba l'últim bit de I_K al receptor, el nivell superior llegeix la informació rebuda en un temps que suposarem que és igual a 0.

$$t_{processament\ recepció} = 0$$

5) t_{ack} és el temps que triga el receptor a posar el datagrama de confirmació *ack* a la línia de transmissió. Normalment $L_{ack} < L_{Trama}$; per tant $t_{ack} < t_{Trama}$.

$$t_{ack} = \frac{L_{ack}\ (\text{bits})}{v_t\ (\text{bps})}$$

6) Els datagrames de confirmació triguen t_{prop} a arribar a l'emissor. Quan l'emissor el rep, esborra I_K de la memòria intermèdia de transmissió, i repeteix el procés per al datagrama següent ($I_K + 1$).

7) T_c és el temps invertit en la transmissió d'un datagrama:

$$T_c = t_{Trama} + t_{ack} + t_{\text{proc}} + 2t_{\text{prop}} \approx t_{Trama} + t_{ack} + 2t_{prop}$$

Confirmació negativa

En els diagrames de temps indicarem les confirmacions negatives amb el terme *nak* (*no acknowledgement*).

4.4. Protocol Idle RQ: Stop & Wait

El protocol Stop & Wait, que podem traduir com 'para i espera', és el protocol ARQ més senzill. El principi de funcionament del protocol Stop & Wait és no transmetre un datagrama nou fins que no es té la certesa de la recepció correcta del datagrama anterior.

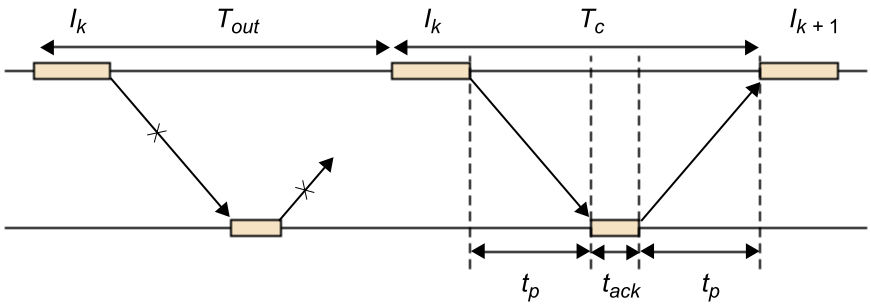
Per a aconseguir-ho es defineixen dos tipus de datagrames:

1) Els **datagrames d'informació**, que porten la informació que s'intercanvien les entitats del nivell superior.

2) Els **datagrames de confirmació** (*ack*), que formen part del protocol i que no porten informació dels nivells superiors.

La figura 10 mostra el funcionament d'aquest protocol. En la part dreta d'aquesta figura l'emissor envia datagrames d'informació al receptor, i el receptor les confirma:

Figura 10



Fixeu-vos que tant el receptor com l'emissor inverteixen un cert temps en el processament dels datagrames rebuts. Això implica que, tot i que el temps de propagació és constant, a causa del temps de processament dels datagrames en el receptor, el temps d'espera de les confirmacions és variable.

Per simplificar-ho, suposarem generalment que el temps de processament dels datagrames és zero, excepte quan sigui necessari tenir-lo en compte.

La mida dels datagrames de confirmació és normalment molt més petita que la mida dels datagrames d'informació, atès que no porten dades del nivell superior. A més, s'ha de tenir en compte que el codi detector d'errors també s'ha d'aplicar a les confirmacions per tal d'assegurar-ne la recepció correcta.

En cas que hi hagi error, el receptor sol·licita la retransmissió del datagrama d'informació. Això es pot fer mitjançant les retransmissions implícites i les explícites.

Trama perduda

Indicarem que una trama es perd (és a dir, que els errors impedeixen reconèixer-ne la recepció) amb una fletxa que no arriba a l'eix contrari i una creu.

4.5. Protocol Stop & Wait amb retransmissions implícites

En aquest cas, les regles que segueixen l'emissor i el receptor són les següents:

1) El receptor envia confirmacions positives A_K dels datagrames que rep sense errors.

2) L'emissor, després d'enviar un datagrama I_K, espera un temps T_0 (activa un temporitzador[12]) per a rebre la confirmació positiva, A_K.

- Si un cop esgotat el temps T_0 la confirmació no ha arribat, retransmet el datagrama I_K i torna a activar el temporitzador.

[12]En anglès, *time-out*.

- Si rep la confirmació positiva, A_K, desactiva el temporitzador, accepta noves dades del nivell superior, munta un nou datagrama I_{K+1} i repeteix aquest procés.

El valor T_0 del temporitzador s'ha de fixar de manera que l'emissor tingui temps de rebre la confirmació (és a dir, $T_0 > T_c = t_{ack} + 2t_p + t_{Trama}$, en què t_{Trama} és el temps que es triga a posar el datagrama d'informació en la línia de transmissió, t_{ack} el temps de càlcul de l'*ack* i t_p el temps de propagació). Si el valor del temporitzador fos massa petit, es produiria la retransmissió innecessària de les trames.

4.6. Protocol Stop & Wait amb retransmissions explícites

En aquest cas, les regles són les mateixes que en la retransmissió implícita, amb la diferència que el receptor també envia confirmacions negatives si rep una trama amb errors.

En general, la confirmació negativa permet una retransmissió més ràpida de la trama errònia, perquè no cal esperar que s'esgoti el temporitzador. Ara bé, en la transmissió també és necessari emprar un temporitzador, ja que les confirmacions poden arribar amb errors o es poden perdre.

Reflexió

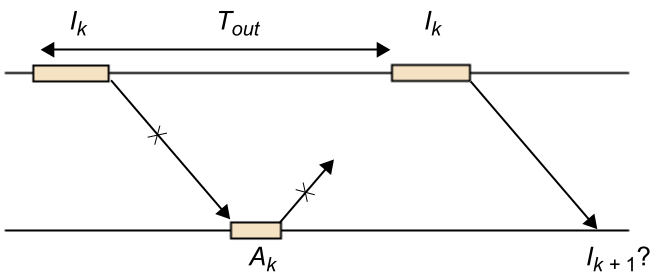
Que passaria si es perdés la confirmació A_K i l'emissor tornés a rebre el datagrama I_K? Com s'adonaria el receptor que la trama I_K està duplicada?

4.7. Necessitat dels números de seqüència

Els protocols ARQ necessiten un número de seqüència per a poder relacionar els datagrames d'informació i les seves confirmacions corresponents. Aquest número de seqüència en la pràctica és un dels camps de la capçalera que afegeix el protocol.

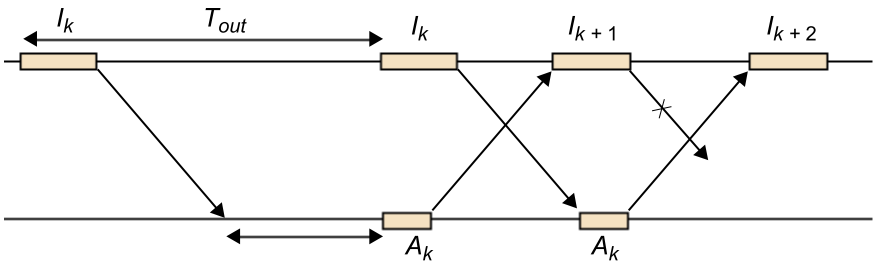
a) Necessitat de numerar els datagrames d'informació. Si es perd la confirmació A_K i l'emissor retransmet el datagrama I_K, el receptor no tindria manera de detectar la recepció duplicada del datagrama I_K, tret que el datagrama porti un número de seqüència.

Figura 11



b) Necessitat de numerar els datagrames de confirmació. Suposem que el temps de processament del receptor és excessiu (és una estació multitasca molt carregada). Expiraria el temporitzador de l'emissor i retransmetria el datagrama I_K abans de rebre la seva confirmació A_K.

Figura 12



La segona confirmació de I_K es podria entendre com una confirmació del paquet I_{K+1}, el qual s'ha perdut i, per tant, mai no es retransmetria.

Si les confirmacions no portessin número de seqüència, l'emissor no detectaria la recepció de les confirmacions duplicades. Podria passar que es perdés un datagrama d'informació.

4.8. Eficiència del protocol Stop & Wait

L'eficiència del protocol Stop & Wait quantifica la pèrdua de capacitat de transmissió d'informació.

$$\text{Velocitat de transmissió} \rightarrow V_t = \frac{1}{\textit{Temps de transmissió d'un bit}}$$

$$\text{Velocitat efectiva } (\textit{throughput}) \rightarrow V_{ef} = \frac{\textit{Bits d'informació}}{\textit{Temps de transmissió}}$$

L'eficiència és la relació entre la velocitat mitjana a la qual es transmeten els bits d'informació i la velocitat màxima a la qual es pot transmetre.

$$\textit{Eficiència} = \frac{V_{ef}}{V_t} = \frac{\textit{Bits d'informació} \times \textit{Temps de transmissió d'un bit}}{\textit{Temps de transmissió}} = \frac{\textit{Durada de la transmissió d'informació}}{\textit{Temps de transmissió}} = \frac{\textit{Bits d'informació}}{\textit{Bits que passen en el temps de transmissió}}$$

L'eficiència es pot veure també com el temps en què es transmet un datagrama d'informació respecte al temps total que es necessita com a mínim per a transmetre'l.

$$E = \frac{t_{Trama}}{T_c} = \frac{t_{Trama}}{t_{Trama} + t_{ack} + 2t_p} \overset{(1)}{\approx} \frac{t_{Trama}}{t_{Trama} + 2t_p} = \frac{1}{1 + 2t_p / t_t} = \frac{1}{1+2a} \quad \text{, en què } a = \frac{t_p}{t_t}$$

Suposem que:

- $t_{Trama} > t_{ack}$

- Els datagrames d'informació són tots iguals, de longitud L bits: $t_{Trama} = L_{Trama} / V_t$.

- Els datagrames de confirmació són de M bits i són tots iguals: $t_{ack} = M_{ack} / V_t$.

$$v_{ef} = v_t \cdot Eficiència = \frac{v_t}{1+2a}$$

Concepte de $a = t_p / t_t$:

$$a = \frac{t_p}{t_t} = \frac{t_p}{L / v_t} = \frac{t_p v_t}{L} = \frac{Nombre\ de\ bits\ que\ caben\ en\ l'enllaç}{Nombre\ de\ bits\ de\ la\ trama}$$

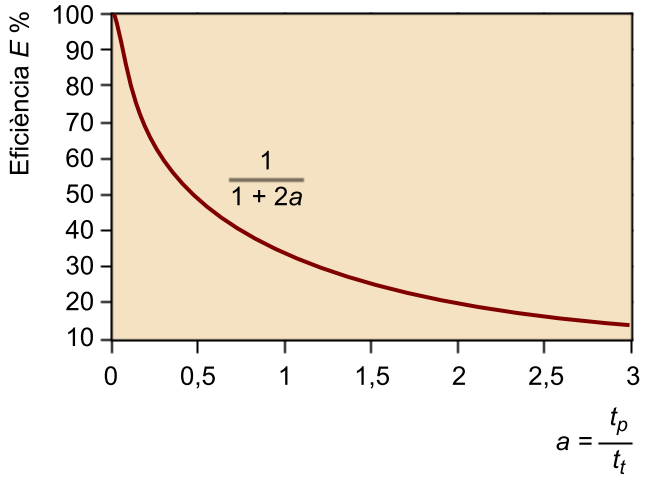
- Si $a > 1$ quan el primer bit del datagrama arriba al receptor, l'emissor ja ha acabat de transmetre la trama.

- Si $a < 1$ quan el primer bit del datagrama arriba al receptor, l'emissor encara no ha acabat de transmetre.

Figura 13

4.8.1. Eficiència de l'Stop & Wait en funció de $a = t_{prop} / t_{Trama}$

Figura 14



Si a és molt petita ($t_p < t_t$), llavors l'eficiència s'aproxima al 100%. Si no l'eficiència decreix ràpidament (quan $t_p = t_t$, $E \approx 30\%$)

El protocol Stop & Wait és força ineficient. Imaginem dos equips finals ubicats a 3.000 km de distància i connectats per una xarxa. Suposem que entre aquest dos equips finals el temps de propagació i retorn és de 30 ms (*RTT*). Suposem

també que la xarxa té una velocitat de transmissió d'1 Gbps (10^9 bits per segon). La mida del datagrama és L = 1.000 octets (8.000 bits) per datagrama, incloent-hi capçalera i dades.

El temps de transmissió d'un datagrama en un enllaç d'1 Gbps és:

$$T_{Trama} = (800 \text{ bits/datagrama}) / (10^9 \text{ bits} \times s^{-1}) = 8 \text{ µs}$$

Si calculem ara el temps de transmissió del datagrama suposant que el temps de l'*ack* és negligible (0 ms):

$$T_c = RTT + T_{ack} + T_{Trama} = 30 \text{ ms} + 8 \text{ µs} + 0 \text{ s} = 30{,}008 \text{ ms}$$

D'aquesta manera podem veure l'eficiència del protocol, és a dir, podem veure que en 30,008 ms només s'està transmetent informació durant 0,008 ms.

$$E_c = T_{Trama} / T_{c =} 0{,}008/30{,}008 \text{ ms} = 0{,}00027$$

Aquest resultat ens diu que s'està enviant informació només 2,7 centèsimes de l'1% del temps.

La velocitat efectiva[13] és:

(13)En anglès, *throughput*.

$$V_{ef} = 8.000 \text{ bits} / 30{,}008 \text{ ms} = 267 \text{ kbps}$$

Tenint en compte que l'amplada de banda accessible era d'1 Gbps es veu clar que l'eficiència de Stop & Wait és molt baixa. La solució a aquest problema és senzilla: l'emissor, en lloc d'enviar un únic datagrama, podrà enviar múltiples datagrames, com veurem en els protocols ARQ continus que es descriuen en el subapartat següent.

Activitat 1

Calculeu l'eficiència màxima que es pot aconseguir en un enllaç punt a punt que fa servir un protocol Stop & Wait en els casos següents:

a) Enllaç entre dues estacions d'una xarxa de dimensions reduïdes amb les dades següents:

- Distància entre les estacions = 1 km
- Velocitat de transmissió de l'enllaç = 10 Mbps
- Velocitat de propagació en l'enllaç = 2 · 108 m/s
- Mida mitjana de les trames d'informació = 5 kb
- Mida de les confirmacions negligible

b) Enllaç via satèl·lit amb les dades següents:

- Temps de propagació entre estacions = 270 ms
- Velocitat de transmissió de l'enllaç = 56 kbps
- Mida mitjana de les trames d'informació = 4 kb
- Mida de les confirmacions negligible

Solució

a) $t_p = D/v_p = 103 / 2 \cdot 108 = 0{,}5 \cdot 10^{-5}$ s
$t_t = L/v_t = 5 \cdot 103 / 10 \cdot 106 = 5 \cdot 10^{-4}$ s
$E\ (\%) = t_t / (t_t + 2t_p) \cdot 100 = 98\%$

b) $t_t = 4 \cdot 103 / 56 \cdot 103 = 71{,}5$ ms, d'on $E = 12\%$.

4.9. Protocols continus RQ

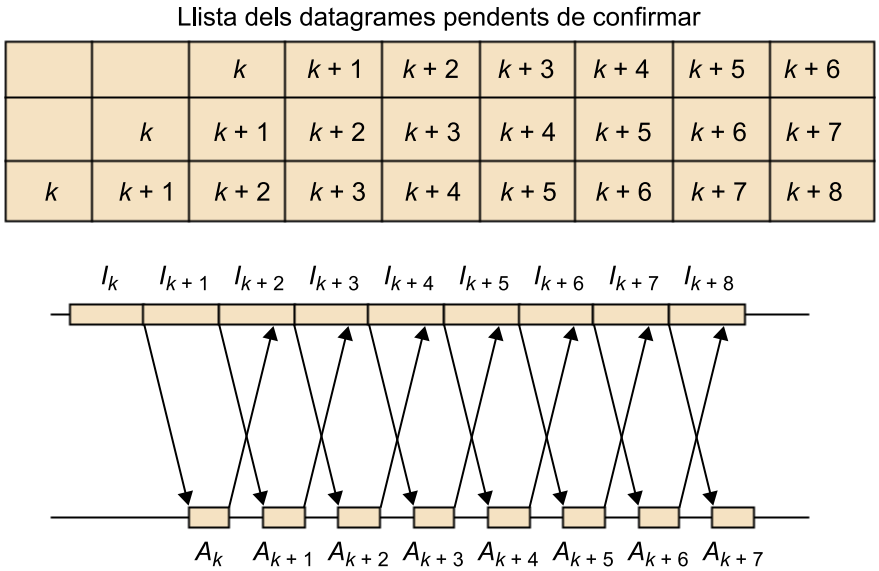
Acabem de veure que el protocol Stop & Wait pot arribar a ser molt ineficient, a causa que l'emissor no pot transmetre durant el temps d'espera de les confirmacions.

En els protocols de transmissió contínua es deixa que l'emissor transmeti contínuament trames d'informació mentre s'esperen les confirmacions.

Evidentment, això només és possible si l'enllaç és dúplex, perquè l'emissor i el receptor han de poder ocupar simultàniament el canal. En el cas del protocol Stop & Wait això no és necessari perquè l'emissor i el receptor ocupen el canal alternativament.

En cas d'error, l'emissor es pot veure obligat a retransmetre una trama anterior a l'última transmesa. Així, doncs, en un protocol de transmissió contínua és necessari que l'emissor desi en una llista les trames que s'han enviat i que estan pendents de confirmar, per tal de poder-les retransmetre en cas d'error. Aquesta llista s'anomena **llista de transmissió**.

Figura 15



Quan es produeixen errors aquests s'han de recuperar. Hi ha dues tècniques:

- Go-Back-N.

- Repetició selectiva.

4.10. Protocol Go-Back-N

El protocol Go-Back-N és un protocol ARQ de transmissió contínua en el qual l'emissor manté els datagrames en una memòria intermèdia fins que han estat confirmats. La memòria intermèdia de recepció del receptor només admet datagrames rebuts en seqüència a punt per a ser llegits pel nivell superior.

En cas d'error, el receptor descarta tots els que arriben fora de seqüència, i l'emissor repeteix la transmissió a partir de l'últim datagrama rebut correctament.

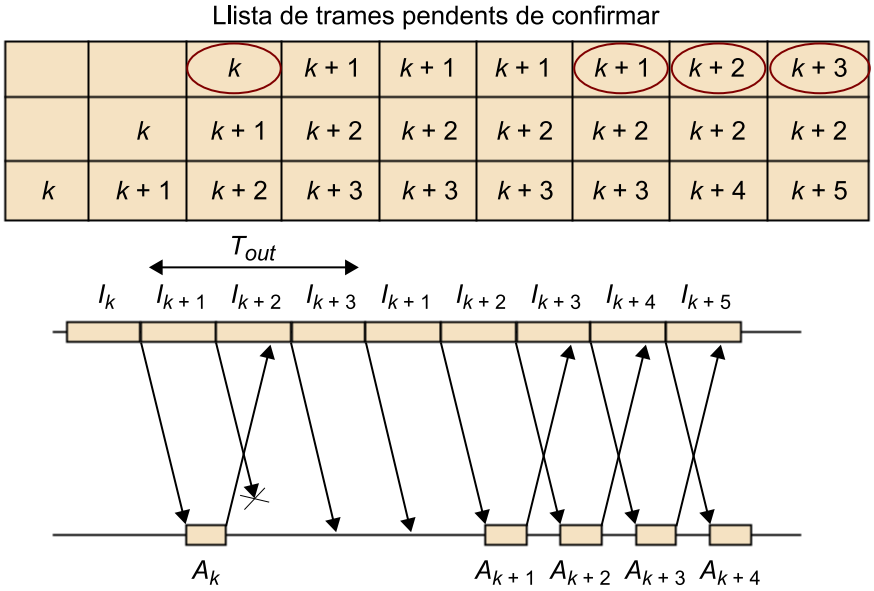
Per tal d'obtenir aquest comportament, la memòria intermèdia de transmissió ha de tenir una capacitat més gran que 1. En canvi, la memòria intermèdia de recepció en el receptor només cal que sigui de mida 1. Per tal d'implementar aquest protocol podem fer servir diferents plantejaments, com per exemple amb temporitzadors o amb confirmacions negatives.

4.10.1. Implementació de Go-Back-N amb temporitzadors

Tal com s'ha dit, l'emissor envia els datagrames i els manté en una memòria intermèdia fins que rep la confirmació. El receptor en rebre els datagrames els confirma si no hi ha errors. En cas d'error o recepció d'un datagrama fora de seqüència, el receptor deixa d'enviar confirmacions fins que rep correctament el datagrama que falta (K). Es descarten tots els datagrames que rep amb un número de seqüència diferent al que s'ha perdut o era erroni (K).

Quan salta el temporitzador de retransmissió d'un datagrama (I_K), l'emissor retransmet el datagrama I_K i continua amb la transmissió dels datagrames següents.

Figura 16

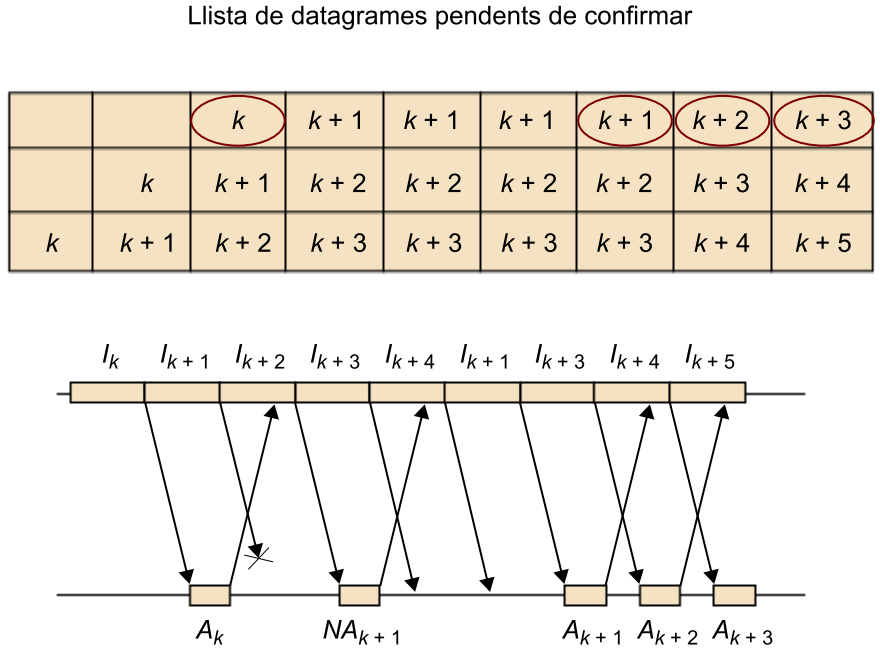


4.10.2. Implementació de Go-Back-N amb confirmacions negatives

En aquest cas, quan el receptor rep un datagrama amb errors o fora de seqüència envia una confirmació negativa en què sol·licita la retransmissió de la trama errònia o la que falta i deixa d'enviar confirmacions fins que rep el datagrama que falta.

Quan l'emissor rep una confirmació negativa, confirma tots aquells datagrames amb número de seqüència anterior al número de seqüència inclòs en la confirmació negativa (aquest procés s'anomena **confirmació acumulativa**). L'emissor seguidament envia de manera seqüencial els datagrames a partir del que no ha estat confirmat.

Figura 17



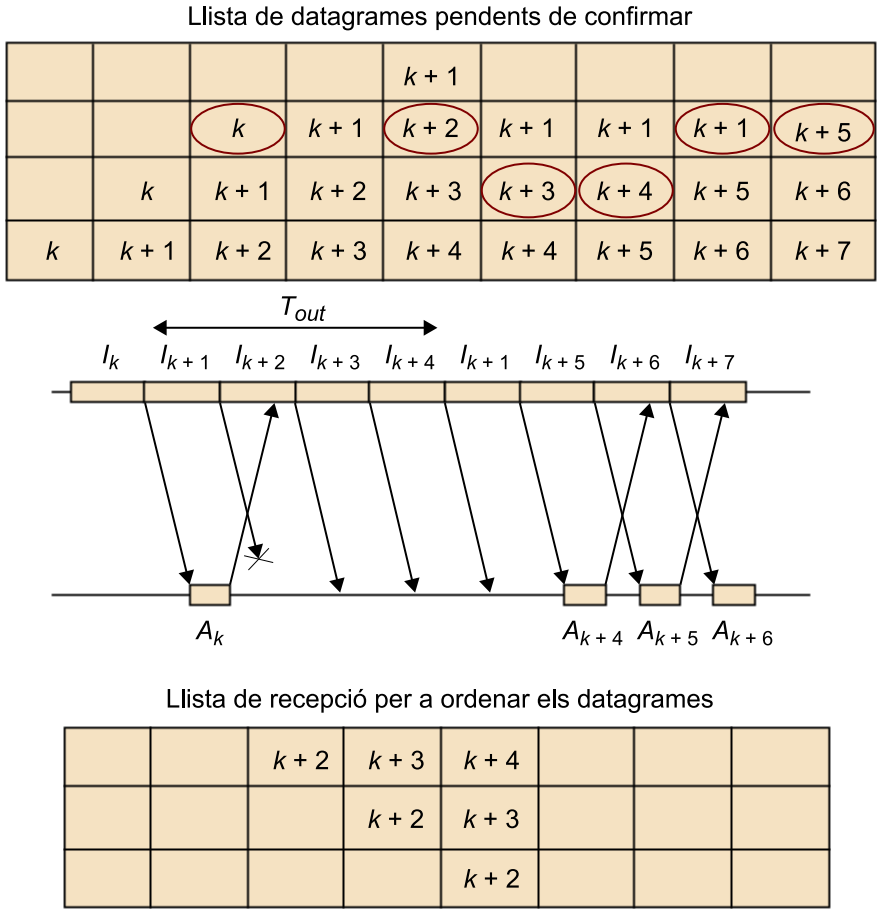
4.11. Protocol de repetició selectiva

El protocol de repetició selectiva és també un protocol ARQ de transmissió contínua. A diferència de Go-Back-N, el receptor emmagatzema els datagrames correctes que han estat rebuts fora de seqüència. La tasca del receptor consisteix a emmagatzemar i ordenar els datagrames que arriben fora de seqüència abans de transmetre'ls al nivell superior. En conseqüència, es necessita una memòria intermèdia de recepció per a mantenir més d'un datagrama. En el protocol de repetició selectiva l'emissor només haurà de retransmetre els datagrames erronis. La implementació d'aquest protocol es detalla a continuació.

4.11.1. Implementació de la repetició selectiva

De la mateixa manera que en Go-Back-N, les confirmacions són acumulatives, és a dir, la confirmació A_K confirma tots els datagrames d'informació amb número de seqüència menor que K. Quan el receptor rep un datagrama d'informació I_K amb errors o fora de seqüència, deixa d'enviar confirmacions fins que rep correctament el datagrama que falta. Mentrestant va desant tots els datagrames que rep amb número de seqüència diferent de K. En el moment que en l'emissor salta el temporitzador de retransmissió d'un datagrama I_K aquest retransmet el datagrama I_K però no retransmet la resta de datagrames que ja ha enviat.

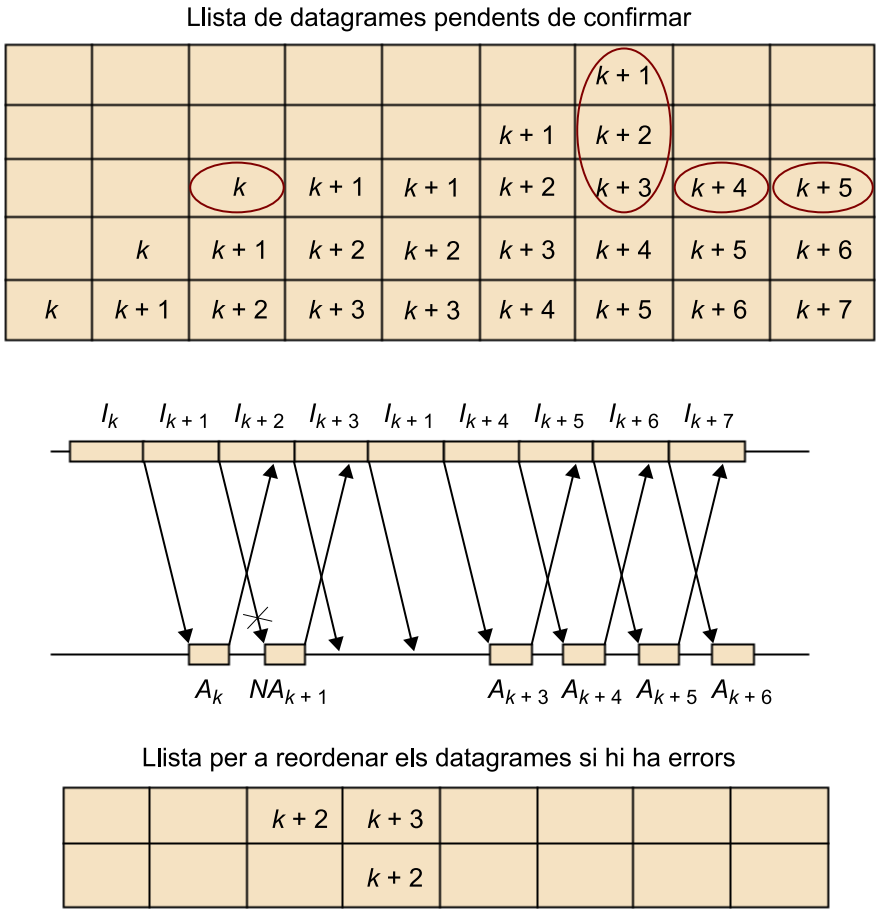
Figura 18



4.11.2. Implementació de la repetició selectiva explícita

En aquesta implementació el receptor envia confirmacions positives quan el datagrama és correcte i confirmacions negatives quan els rep amb errors o fora de seqüència. Les confirmacions són acumulatives, com en el cas anterior, de manera que una confirmació A_K confirma tots els datagrames d'informació amb número de seqüència anterior a K. Mentre el receptor espera la retransmissió d'una trama es deixen d'enviar confirmacions positives o negatives dels datagrames posteriors a K. El receptor activa un temporitzador de retransmissions negatives dels datagrames no rebuts per tal de tornar a enviar la demanda en el cas que no es rebi en un període de temps determinat. Tan bon punt les confirmacions negatives arriben a l'emissor, aquest retransmet les trames demanades.

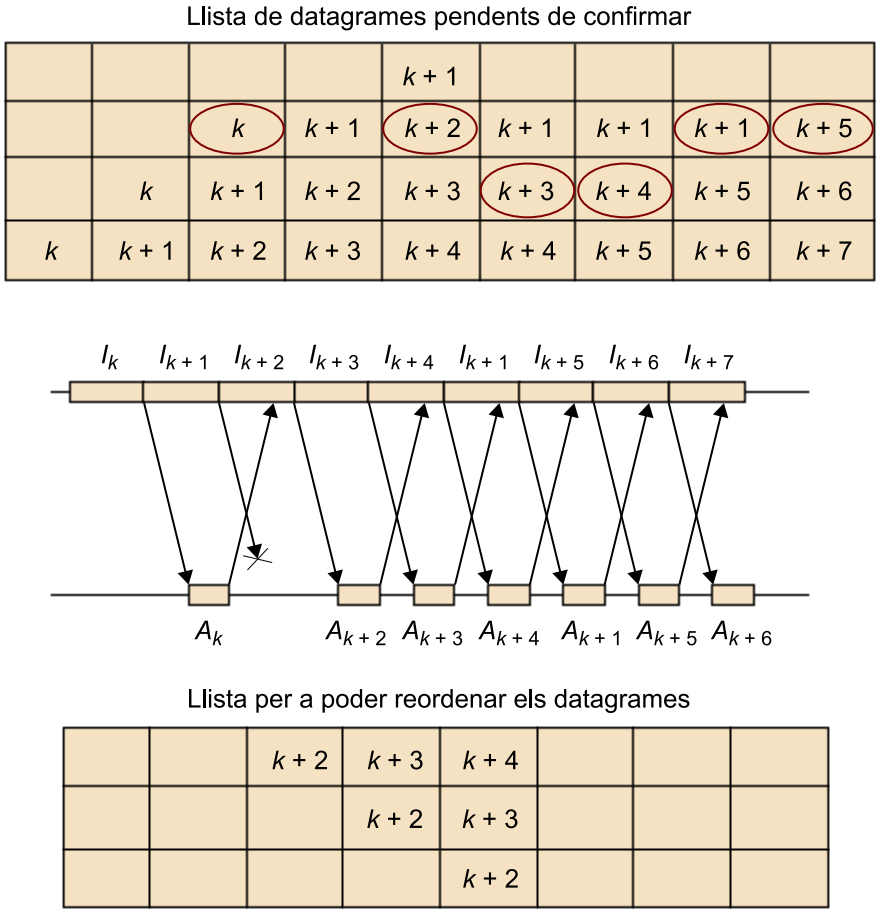
Figura 19



4.11.3. Implementació de la repetició selectiva implícita

En aquesta implementació les confirmacions no són acumulatives. En aquest cas, el receptor només envia confirmacions positives d'aquells datagrames rebuts correctament. Si l'emissor rep un datagrama de confirmació no consecutiu dedueix que hi ha hagut un error i retransmet el datagrama intermedi. Un temporitzador T_{out} s'utilitza per a evitar que es perdin les trames pendents, ja que l'emissor només enviaria la primera perduda a partir de l'última confirmada.

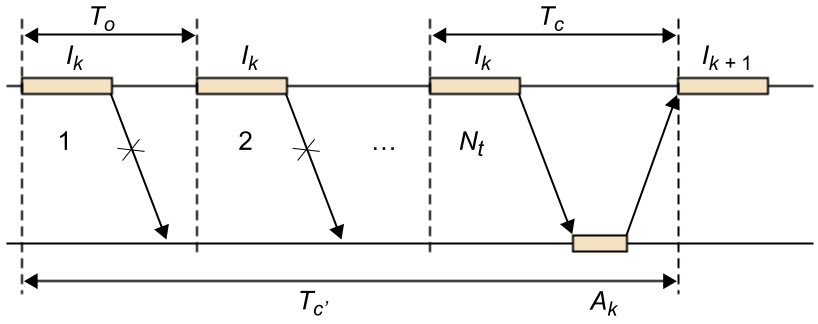
Figura 20



4.12. Càlcul de l'eficiència en presència d'errors

4.12.1. Stop & Wait

Figura 21



En les N_t –1 primeres retransmissions, el datagrama I_k es retransmet després d'un temps T_{out} (quan salta el temporitzador de retransmissió), mentre que en l'última la transmissió del datagrama es fa durant un temps T_c:

$$Eficiència = \frac{t_{Trama}}{T'_c} = \frac{t_{Trama}}{(N_t - 1)T_{out} + T_c}$$

Si ajustem el temporitzador per no tenir retransmissions innecessàries, és a dir, quan $T_o > \approx T_c$, obtenim l'eficiència màxima:

$$T_c' = N_t T_c = N_t(2t_p + t_{ack} + t_t) \approx N_t(1 + 2a)t_t$$

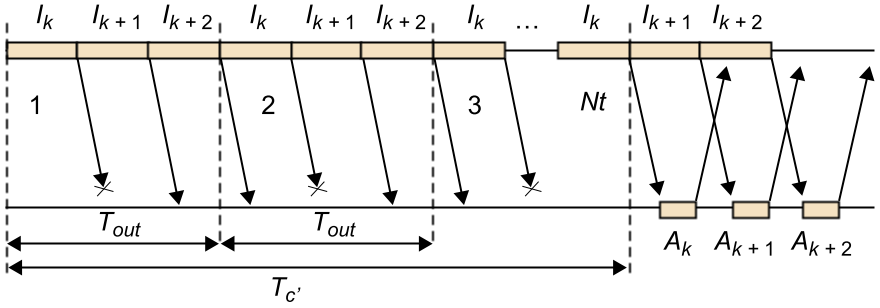
$$Eficiència = \frac{t_{Trama}}{T'c} = \frac{t_{Trama}}{N't T_c} = \frac{t_{Trama}}{N_t(2t_p + t_{ack} + t_{Trama})} \overset{T_{Trama} << t_{ack}}{\approx} \frac{1}{N_t(1+2a)}$$

L'eficiència disminueix en un factor N_t.

4.12.2. Go-Back-N

Cada vegada que hi ha un error es perden els datagrames transmesos i s'avorta la transmissió:

Figura 22



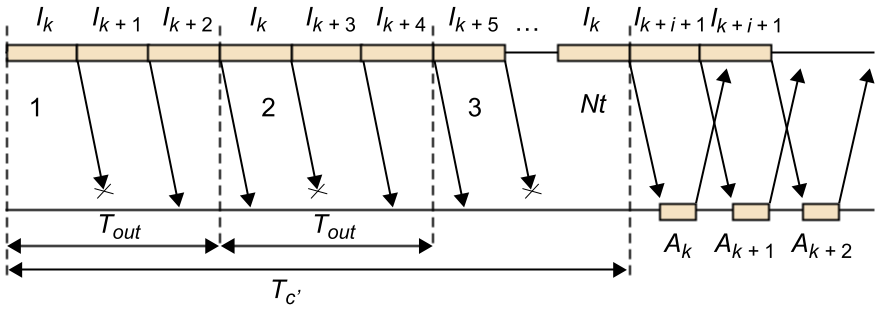
$T_{c'} = (N_t - 1)T_0 + t_{Trama} \approx (N_t - 1)T_c + t_{Trama}$, en què $T_c = t_{Trama} + 2t_{prop}$

$$Eficiència = \frac{t_{Trama}}{(N_t - 1)T_c + t_t} = \frac{t_{Trama}}{(N_t - 1)(t_{Trama} + 2tp) + t_{Trama}} \approx \frac{1}{2a(N_t - 1) + N_t} = \frac{1}{N_t(1+2a) - 2a}$$

4.12.3. Retransmissió selectiva

A diferència de Stop & Wait i Go-Back-N, el temps que queda lliure mentre es fan les N_t transmissions d'un mateix datagrama d'informació s'aprofita per a transmetre altres datagrames. L'eficiència és independent del temporitzador de retransmissió.

Figura 23



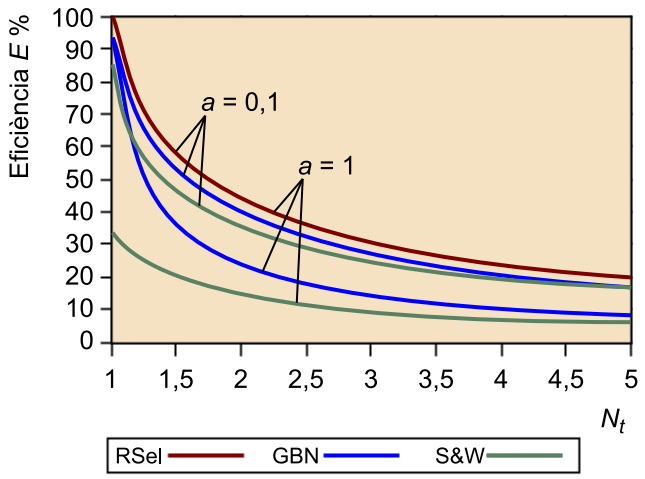
El temps invertit en la transmissió és $N_t \cdot t_{Trama}$:

$$Eficiència = \frac{t_{Trama}}{N_t \cdot t_{Trama}} = \frac{1}{N_t}$$ L'eficiència és independent d'a .

4.12.4. Comparacions i conclusió

- Per a $a < 1$, el comportament dels tres protocols és similar.
- Si P_{error} és alta ($N > 1$), l'eficiència obtinguda amb els tres protocols és semblant.
- L'eficiència de Stop & Wait decreix en augmentar a, independentment de la P_{error}.
- La retransmissió selectiva i el Go-Back-N tenen una $E \approx 100\%$ si la probabilitat d'error és petita ($P_{error} \approx 0 = N \approx 1$).
- Si la probabilitat d'error és petita ($N \approx 1$) la retransmissió selectiva i el Go-Back-N tenen una eficiència semblant.
- La retransmissió selectiva té més eficiència que Go-Back-N per a una probabilitat d'error no gaire alta.

Figura 24. Comparació de l'eficiència de Stop & Wait, Go-Back-N i retransmissió selectiva en funció de N per a dos valors de a



4.13. Càlcul de *N*

Calcularem la relació que hi ha entre N (la mitjana del nombre de transmissions necessàries per a la transmissió amb èxit d'un datagrama i P_{bit} (probabilitat d'error en el bit P_{bit}). Suposarem que cada bit del datagrama té una probabilitat d'error independent dels altres bits i igual a P_{bit}. Si el datagrama té L bits, la probabilitat que el datagrama tingui algun error (P_p) val:

$$P_p = 1 - P\{\text{bit sense error}\}^L = 1 - (1 - P_{bit})^L$$

La probabilitat de transmetre un datagrama k vegades és la probabilitat que les $k - 1$ primeres vegades tingui error (P_p^{k-1}) multiplicada per la probabilitat que l'última transmissió no tingui error ($1 - P_p$):

$$N_t = \sum_{k=1}^{\infty} k \cdot Prob\left\{k \ transmissions\right\} = \sum_{k=1}^{\infty} k \cdot P_p^{k-1}(1-P_p) =$$

$$= (1-P_p)\sum_{k=1}^{\infty} k \cdot P_p^{k-1} = \frac{(1-P_p)}{(1-P_p)^2} = \frac{1}{(1-P_p)}$$

5. Control de flux

Els protocols ARQ fan recuperació d'errors i també control de flux.

El control de flux consisteix a adaptar la velocitat de transmissió eficaç entre l'emissor i el receptor, de manera que sempre hi hagi recursos.

La velocitat de transmissió es controla pels motius següents:

- Saturació de les memòries intermèdies de transmissió, quan hi ha trames pendents de confirmar.

- Saturació de les memòries intermèdies de recepció. Un emissor que envia informació a una velocitat eficaç major que la que pot consumir el receptor satura la memòria intermèdia de recepció del receptor i es perd la informació.

Tècniques per a controlar el flux

Hi ha diferents tècniques per a controlar el flux:

- X-OFF / X-ON: quan el receptor no admet més caràcters, li envia X-OFF per a bloquejar el transmissor i X-ON per a reprendre la transmissió.
- En Stop & Wait no hi ha problema. Només manté un datagrama sense confirmar.
- En transmissió contínua: **mecanisme de finestra lliscant**[14].
- *Echo checking*: fa control d'errors i control de flux. Quan les memòries intermèdies s'omplen s'atura la tramesa d'ECO i el transmissor es bloqueja fins que rep un ECO.
- Control de flux fora de banda: protocol de nivell físic RS232, per a comunicació entre port sèrie d'un PC i un mòdem. Dues línies serveixen per al control de flux: RTS[15] i CTS[16]. El funcionament és el següent:
  - RTS = 1. L'emissor està llest per a transmetre dades.
  - CTS = 1. El receptor pot rebre les dades. L'emissor li envia dades si en té.
  - CTS = 0. L'emissor no pot transmetre.

(14) En anglès, *sliding window*.

(15) *RTS* és la sigla de *request to send*.

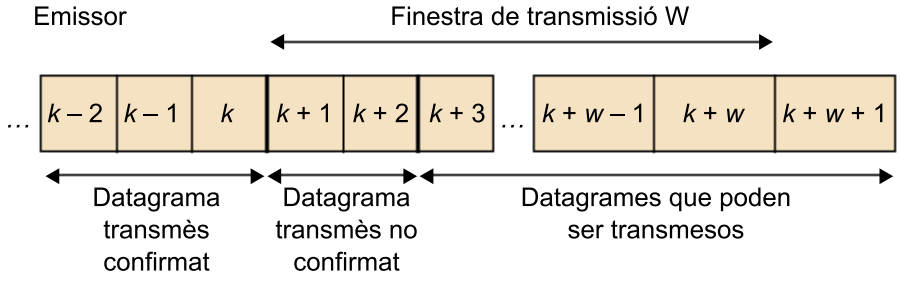
(16) *CTS* és la sigla de *clear to send*.

5.1. Protocols de finestra. Concepte de finestra lliscant

Els protocols ARQ de transmissió contínua requereixen el control del flux de transmissió per tal d'evitar la saturació de les memòries intermèdies d'emissió o recepció, tal com hem vist abans. Una de les maneres d'adreçar aquest control de flux és mitjançant un protocol de finestra lliscant. La finestra lliscant defineix un nombre *W* de datagrames màxim que pot estar sense confirmar.

En aquest sentit, l'emissor pot enviar fins a *W* datagrames d'informació sense confirmar, que queden emmagatzemats en la memòria intermèdia de transmissió.

Figura 25. Finestra de transmissió



Datagrama *k*: últim datagrama confirmat
Datagrama *k* + 2: últim datagrama transmès no confirmat

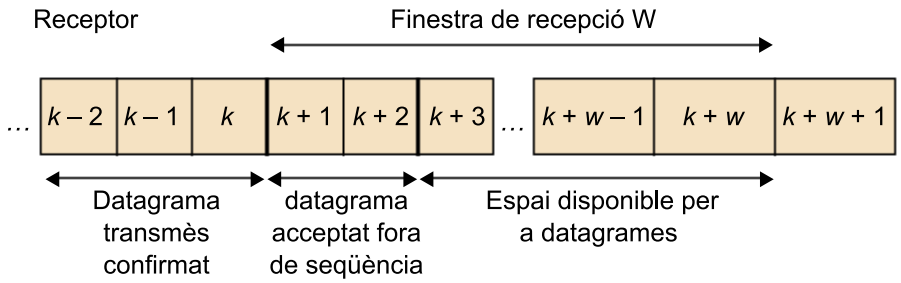
En el protocol de finestra lliscant les estacions han d'estar connectades en mode dúplex. Si I_k és l'últim datagrama confirmat (suposem que hi ha confirmacions acumulatives), i *W* és la mida de la finestra de transmissió, l'emissor pot transmetre fins al datagrama I_{k+w}. Abans de cada transmissió l'emissor comprova que el número de seqüència del datagrama per transmetre menys el número de seqüència de l'últim datagrama confirmat és menor o igual que *W*. En cas contrari, l'emissor es queda bloquejat esperant que arribin confirmacions. Tan bon punt arriben confirmacions l'emissor ja pot tornar a enviar datagrames.

La finestra de transmissió permet dimensionar la mida de la memòria intermèdia de transmissió necessària per a emmagatzemar els datagrames pendents de confirmar, ja que els datagrames confirmats es poden esborrar de la llista.

Stop & Wait és un cas particular, ja que $W = 1$. Després de la transmissió d'un datagrama, l'emissor es queda bloquejat fins que arriba la confirmació (només es pot enviar un datagrama sense confirmar).

En transmissió contínua la mida de la memòria intermèdia de transmissió serà de *W*.

Figura 26. Finestra de recepció



datagrama *k*: últim datagrama rebut i confirmat
datagrama *k* + 2: últim datagrama rebut però no confirmat

La finestra de recepció defineix el nombre màxim de datagrames rebuts sense errors, fora de seqüència. En els protocols en què no s'han de reordenar les trames, com Stop & Wait i Go-Back-N, el valor de la finestra de recepció val 1. En el protocol de retransmissió selectiva, en el qual cal reordenar els datagrames, es podran rebre W datagrames fora de seqüència (des de $k + 1$, $k + 2$, fins a $k + w$), que és el nombre total de datagrames que pot transmetre l'emissor. El valor màxim de la finestra de recepció serà W.

Protocol	Finestra TX	Finestra RX
Stop & Wait	1	1
Go-Back-N	W	1
Retransmissió selectiva	W	W

5.2. Finestra òptima

Establir el valor de la finestra de transmissió és cabdal per a l'eficiència del protocol ARQ. Si la mida de la finestra és massa petita, pot passar que el protocol es quedi ràpidament bloquejat, esperant que arribin les confirmacions. Si la finestra del protocol és massa gran, també pot ser un inconvenient, ja que les memòries intermèdies de transmissió i recepció s'han de dimensionar segons la mida de la finestra.

Es defineix la finestra òptima com la mínima finestra que permet aconseguir una eficiència del 100% en el protocol, és a dir, la mínima finestra que permet aconseguir la velocitat efectiva màxima.

- Si utilitzem una finestra $W < W_{\text{òptima}}$, la velocitat efectiva serà inferior que la que podríem aconseguir amb una finestra més gran.

- Si utilitzem una finestra $W > W_{\text{òptima}}$, no s'augmentarà la velocitat efectiva més enllà del que s'ha aconseguit amb la finestra òptima.

La mida òptima de la finestra és, doncs:

$$W_{\grave{o}ptim} = \frac{T_c}{t_{Trama}}$$

5.3. *Piggybacking*

És una tècnica utilitzada tant per l'emissor com pel receptor quan envien i reben informació. En aquest cas és necessari que hi hagi una confirmació de mida mínima, que s'incorpora dins de la trama d'informació.

5.4. Números de seqüència

Els protocols ARQ necessiten un número de seqüència per a relacionar els datagrames d'informació i les seves confirmacions corresponents. El número de seqüència el porta un camp de la capçalera del datagrama. Si aquest camp té n bits, llavors el número de seqüència podrà tenir un valor en l'interval $[0,2^n - 1]$. És a dir, tenim 2^n números de seqüència diferents. Quan s'arriba al número de seqüència $2^n - 1$, es torna a començar amb el valor 0, i es repeteix el cicle de números de seqüència. La reutilització dels números de seqüència pot crear ambigüitats si no és prou gran. És a dir, quan es confirma el datagrama k, com pot saber l'emissor si es confirma el datagrama del cicle actual o del cicle anterior?

Si els datagrames poden arribar desordenats i amb retards arbitraris, aquesta ambigüitat no es pot resoldre. L'únic que es pot fer és agafar un nombre de bits suficientment gran perquè la probabilitat que això succeeixi sigui pràcticament 0 (això és el que fa TCP).

Si els datagrames arriben en el mateix ordre que s'han enviat, es pot demostrar que necessitem els números de seqüència següents perquè els protocols funcionin sense ambigüitats:

- Amb Stop & Wait n'hi hauria prou amb un sol bit per al número de seqüència.

- Go-Back-N: si volem una finestra de mesura W, necessitarem un nombre de bits n tal que $2^n \geq W + 1$.

- Retransmissió selectiva: $2^n \geq 2W$.

Protocol	Nre. d'identificadors necessaris	Mida de la finestra màxima si s'utilitzen *n* bits per als identificadors
Stop & Wait	2	
Go-Back-N	$W + 1$	2^{n-1}
Retransmissió selectiva	$2 \cdot W$	2^{n-1}

6. Transport orientat a la connexió: TCP

TCP és un protocol de nivell de transport que s'usa en Internet per a la transmissió fiable d'informació (potser és el protocol més complex i important de la pila de protocols d'Internet).

Com hem vist, UDP no garanteix el lliurament de la informació que li proporciona una aplicació. Tampoc no reordena la informació en cas que arribi en un ordre diferent de l'ordre en què s'ha transmès. Hi ha aplicacions que no poden tolerar aquestes limitacions. Per a superar-les, el nivell de transport proporciona TCP.

El TCP dóna fiabilitat a l'aplicació, és a dir, garanteix el lliurament de tota la informació en el mateix ordre en què ha estat transmesa per l'aplicació d'origen. Per a aconseguir aquesta fiabilitat, el TCP proporciona un servei orientat a la connexió amb un control de flux i errors.

TCP és un protocol ARQ, d'extrem a extrem, orientat a la connexió (fases d'establiment de la connexió, tramesa de dades i tancament de la connexió) i bidireccional (dúplex). La unitat de dades TCP és el "segment TCP". TCP, a diferència d'UDP, intenta generar segments de mida òptima que s'anomenen *MSS*[17], generalment la major possible per a minimitzar el sobrecost[18] de les capçaleres, però que no produeixi fragmentació a nivell d'IP. Igual que UDP, TCP utilitza multiplexació mitjançant l'ús de ports, tal com hem vist amb anterioritat.

[17] *MSS* és la sigla de *maximum segment size*.

[18] En anglès, *overhead*.

Proporciona fiabilitat mitjançant el següent:

- **Control d'errors**. TCP és semblant a Go-Back-N, però no descarta segments posteriors quan arriben fora de seqüència. Permet les retransmissions selectives.

- **Control de flux** (finestra advertida). Serveix per a adaptar la velocitat entre l'emissor i el receptor. Vigila que l'emissor no enviï els segments a més velocitat d'aquella a la qual els pot processar el receptor, de manera que es puguin perdre paquets per saturació de la memòria intermèdia de recepció.

- **Control de la congestió** (finestra de congestió). Per a adaptar la velocitat de l'emissor als encaminadors intermedis de la xarxa i evitar així que es col·lapsin les seves memòries intermèdies i es puguin perdre paquets.

En transmissió, TCP s'encarrega de fragmentar les dades del nivell d'aplicació, i els assigna un número de seqüència abans d'enviar els fragments al nivell IP.

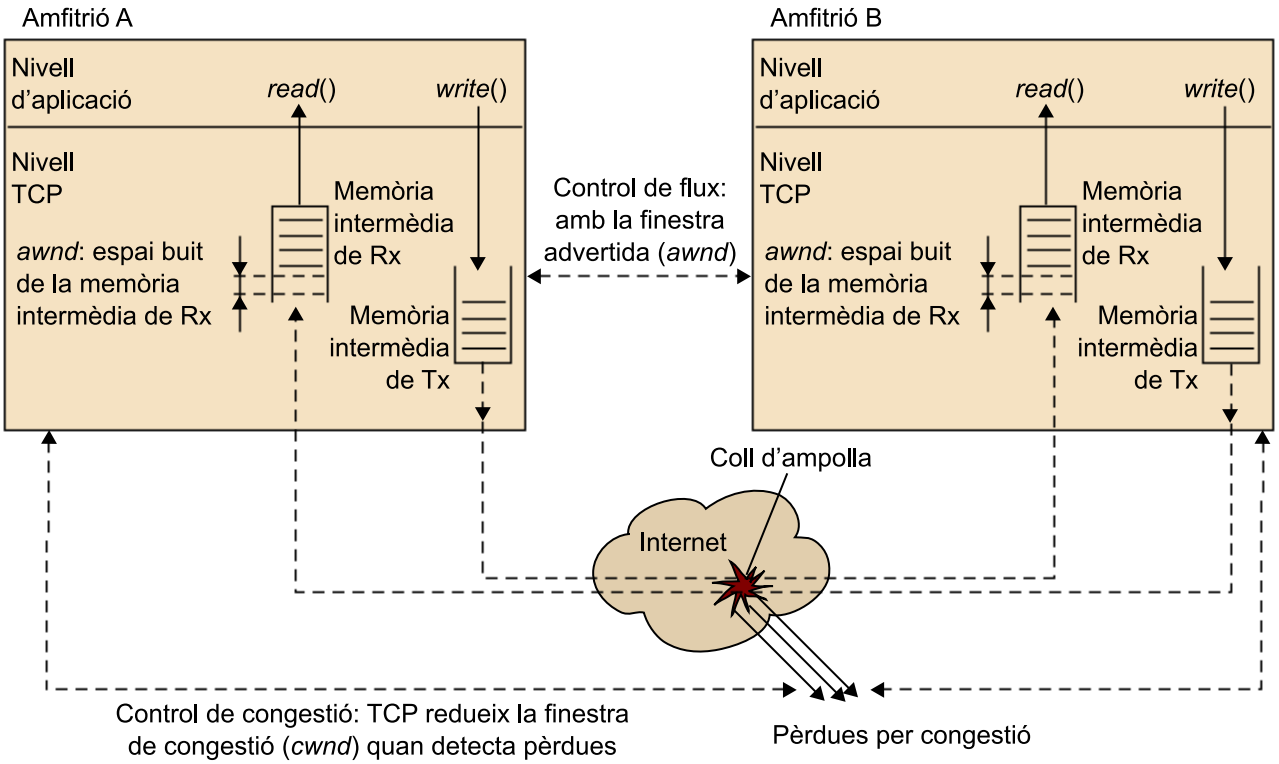
En recepció, com els segments poden arribar fora d'ordre, TCP els ha de reordenar abans de passar-los als nivells superiors.

6.1. Funcionament bàsic de TCP

En cada extrem, TCP manté una memòria intermèdia de transmissió i una de recepció. L'aplicació utilitza les crides al sistema operatiu *read*() i *write*() per a llegir i escriure en aquestes memòries intermèdies. Quan l'aplicació escriu la informació que s'ha d'enviar a l'emissor, TCP la desa en una memòria intermèdia de transmissió. Quan la memòria intermèdia de Tx és plena, la crida *write*() queda bloquejada fins que hi torni a haver espai.

Cada vegada que arriben segments de dades al receptor, l'API *read*() els passa al nivell superior i s'envien les seves confirmacions corresponents. Les dades, segons són confirmades pel receptor, s'esborren de la memòria intermèdia de transmissió i queda espai lliure perquè l'aplicació continuï escrivint.

Figura 27



6.2. Capçalera TCP

Figura 28

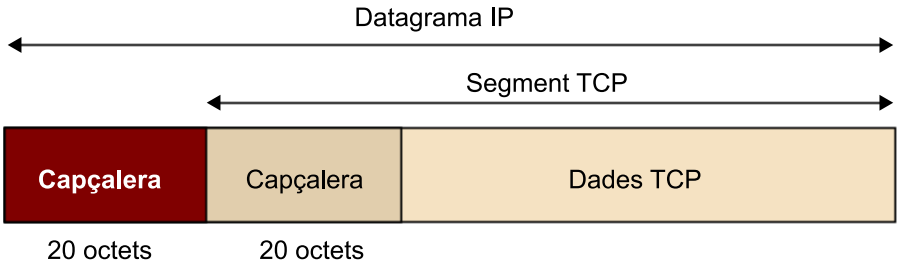
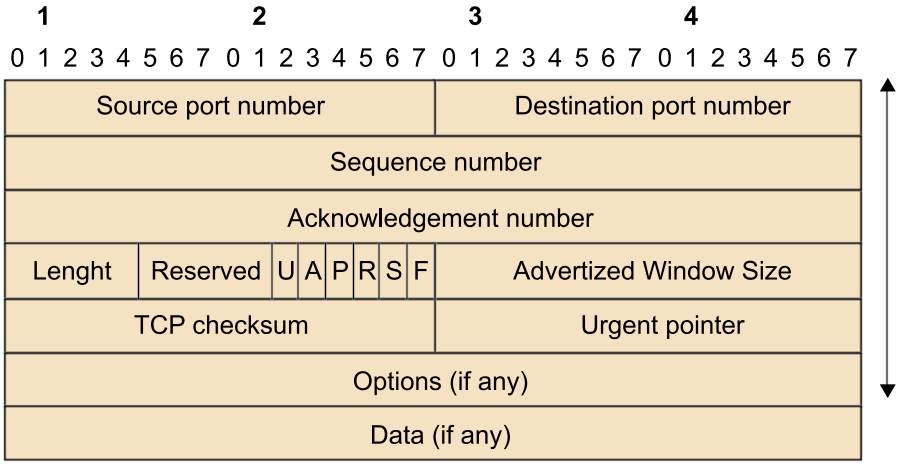


Figura 29



- **Ports d'origen i destinació** (*Source port number* / *Destination port number*), que identifiquen les aplicacions.

- **Número de seqüència del segment** (*Sequence number*): identifica el primer octet dins d'aquest segment de la seqüència d'octets enviats fins aquell moment.

- **ISN** (*Initial sequence number*): primer número de seqüència escollit pel protocol TCP. El número de seqüència serà un nombre a partir d'ISN. Per tant, per a saber quants octets hem enviat cal fer:

  Número de seqüència – ISN

- **Número de reconeixement** (*Acknowledgement number*): conté el pròxim número de seqüència que el transmissor de l'*ACK* espera rebre, és a dir, és el número de seqüència + 1 de l'últim octet rebut correctament.
  TCP és dúplex: cada extrem manté un *Seq. number* i un *Ack number*.

- **Longitud total de la capçalera** (*Header length*) del segment TCP en paraules de 32 bits (igual que el camp *header length* de la capçalera IP).
  - Mida mínima de la capçalera TCP = 20 octets (*header length* = 5)
  - Mida màxima de la capçalera TCP = 60 octets (*header length* = 15).

- **Reservat** (*Reserved*): bits reservats per a possibles ampliacions del protocol. Se'n posen 0.

- **Indicadors** (*Flags*): hi ha sis indicadors (bits) a la capçalera. Són vàlids si estan a 1.

- **URG** (*Urgent*): indica que s'utilitza el camp *apuntador d'urgència*[19] de la capçalera TCP.

- **ACK** (*Acknowledgement*): indica que s'utilitza el camp Número de reconeixement. Vàlid per a tots menys per al SYN inicial.

- **PSH** (*Push*): indica que el receptor ha de passar les dades de la memòria intermèdia de recepció als nivells superiors tan ràpidament com sigui possible, l'operació més ràpida sense omplir la memòria intermèdia. L'activació d'aquest indicador depèn de la implementació. Les implementacions derivades de BSD l'activen quan la memòria intermèdia de Tx es queda buida.

- **RST** (*Reset*): s'activa quan es vol avortar la connexió. Un exemple és quan es rep un segment d'un client dirigit a un port on no hi ha cap servidor escoltant. En aquest cas, TCP contesta amb un segment amb l'indicador de *reset* activat.

- **SYN** (*Synchronize*): s'utilitza en l'establiment de la connexió, durant la sincronització dels números de seqüència a l'inici de la connexió.

- **FINAL** (*Finalize*): s'utilitza en l'acabament de la connexió.

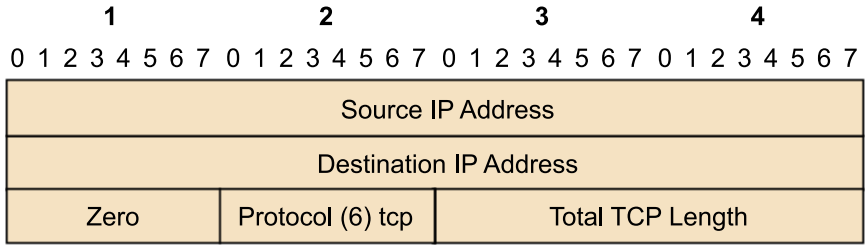
- **Mida de la finestra detectada** (*Advertised window size*): mida de la finestra detectada pel receptor al transmissor (*Sliding window*) per al control de flux. La mida màxima de la finestra és de 65.535 octets.

- **Suma de comprovació TCP** (*TCP checksum*): s'utilitza per a detectar errors en el segment TCP, i per a tenir més certesa que el segment no ha arribat a una destinació equivocada.

  – Igual que en UDP, la suma de comprovació TCP s'aplica conjuntament a una pseudocapçalera + la capçalera TCP + el camp de dades TCP. Aquesta pseudocapçalera té tres camps de la capçalera IP i la mida del segment TCP (capçalera + *payload*).

(19) En anglès, *urgent pointer*.

Figura 30



– La pseudocapçalera només és per a calcular la suma de comprovació: no es transmet en el segment TCP, ni s'inclou en la longitud del paquet IP.

– La mida del segment TCP no es posa a la capçalera TCP, només es té en compte en el càlcul de la suma de comprovació.

– A diferència d'UDP, en TCP el càlcul de la suma de comprovació és obligatori.

- **Apuntador d'urgència** (*Urgent pointer*): camp vàlid si l'indicador *URG* = 1. Implementa un mecanisme per a indicar dades urgents (és a dir, que s'han d'atendre el més aviat possible). És un punter al *Seq. number* que indica la part de dades urgents dins del camp de dades. Les dades urgents aniran del primer octet del segment a l'octet indicat per l'*urgent pointer*. Aquest indicador s'utilitza poques vegades. Un exemple és quan es tecleja un Control-C (interrupció) des de l'aplicació Telnet.

- **Opcions** (*Options*): TCP permet afegir opcions a la capçalera, però a diferència d'IP, les opcions de TCP se solen utilitzar. En podem destacar les següents:

  – **Mida màxima del segment** (*Maximum segment size*): s'utilitza durant l'establiment de la connexió per a suggerir el valor de l'MSS en l'altre extrem MSS = MTU xarxa directament connectada – capçalera IP – capçalera TCP (sense opcions). Si la xarxa és Ethernet (MTU 1.500), llavors MSS = 1.460.

  – **Factor d'escala de la finestra** (*Window scale factor*): s'utilitza durant l'establiment de la connexió per a indicar que el valor de la finestra advertida s'ha de multiplicar per aquest factor d'escala. Això permet advertir finestres majors que 216.

  – **Segell de temps** (*Timestamp*): s'utilitza en el càlcul de l'*RTT*.

– **SACK**: permet que TCP faci retransmissió selectiva (*selective ack*). TCP utilitza el camp *ack* per a indicar fins on s'ha rebut correctament. Amb l'opció *SACK* el receptor pot indicar blocs de segments que s'han rebut correctament més enllà del segment confirmat per l'*ack*. D'aquesta manera, l'emissor pot escollir millor els segments que s'han de retransmetre.

– ***Padding***: octets de farcit afegits perquè la capçalera tingui un múltiple de 32 bits.

Activitat 2

Assumim que un extrem client TCP (emissor) ha triat el 28.325 com a número de seqüència inicial (ISN), mentre que l'extrem receptor TCP (servidor) ha triat com a ISN el 12.555. Què indica un segment client (emissor) TCP amb número de seqüència 29.201, número *ACK* 12.655 i finestra 1.024?

Solució de l'activitat 2

El número de seqüència indica que el client ja ha transmès des de l'octet 28.325 fins a l'octet 29.200 (875 octets en total) i que en aquest segment transmetrà a partir de l'octet 29.201. El número *ACK* indicarà al receptor que l'emissor ha rebut correctament fins a l'octet 12.654 i que espera rebre a partir del 12.655. La finestra indica al receptor que el client només pot acceptar 1.024 octets abans de confirmar-los. Per tant, el servidor TCP actualitzarà la seva finestra de transmissió a 1.024.

6.3. Control de flux en TCP

El control de flux és cabdal per al funcionament de TCP. Amb anterioritat hem vist com els protocols ARQ aconseguien controlar el flux de transmissió mitjançant l'ús de memòries intermèdies d'emissió i de transmissió. TCP també fa servir aquesta tècnica i intenta evitar la saturació de la memòria intermèdia de recepció en el cas que la velocitat de transmissió de l'emissor sigui més gran que la velocitat de processament del receptor. El control de flux és imposat pel receptor, que informa sobre la mida actual de la memòria intermèdia de recepció per mitjà del camp *finestra advertida*[20] en la capçalera de les confirmacions (*acks*). El receptor no pot enviar més octets dels que indica la finestra advertida. D'aquesta manera, sempre hi haurà espai suficient per a desar els octets enviats en la memòria intermèdia de recepció del receptor. Quan la memòria intermèdia de recepció s'omple el receptor advertirà una finestra advertida de valor 0. Mentre la finestra de transmissió sigui zero, l'emissor no podrà enviar més informació. Per tal de regular la transmissió, TCP adapta la finestra a la condició més restrictiva tant de control de flux com de control de congestió.

[20] En anglès, *advertised window*. Per abreujar, *awnd*.

En tot moment TCP utilitza una finestra de transmissió que pot ser definida: Finestra TX $wnd = mín(awnd, cwnd)$, en què *cwnd* és la finestra de congestió[21].

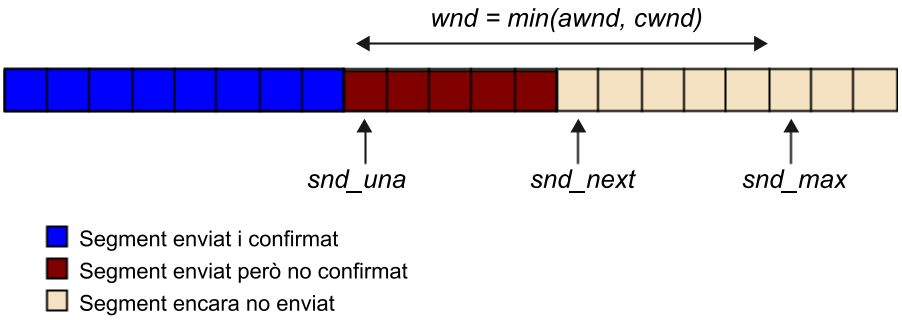
[21] En anglès, *congestion window*. Per abreujar, *cwnd*.

6.3.1. Finestra lliscant en TCP

TCP implementa un protocol de control de flux mitjançant una finestra de mida lliscant.

El control de flux d'extrem a extrem permet que l'emissor enviï segments sense rebre el seu *ack* corresponent gràcies a l'ús d'una memòria intermèdia. El receptor emmagatzema en la memòria intermèdia els segments que li han arribat però encara no han estat utilitzats. Si l'aplicació no consumeix les dades rebudes es deixen de reconèixer segments.

Figura 31



awnd	Finestra advertida
cwnd	Finestra de congestió
snd_una	Primer segment enviat no confirmat
snd_next	Pròxim segment per enviar
snd_max	Últim segment que pot ser enviat (si no no es confirmen noves dades)

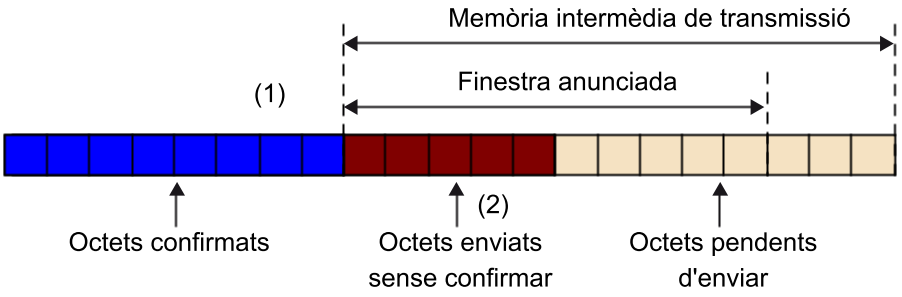
6.3.2. Finestra advertida de transmissió

L'emissor desa en la memòria intermèdia:

- Els segments enviats i no reconegut. Quan les memòries intermèdies s'omplen l'emissor deixa d'enviar segments.

- Els octets que no han estat transmesos encara.

Figura 32



En la figura 32 veiem que:

1) Últim *ack* rebut ≤ últim octet enviat i confirmat (consumit).
2) Últim octet enviat ≤ últim octet consumit (*write*).

També podem veure que la mida de la memòria intermèdia serà:

Mida de la memòria intermèdia *r* ≥ Últim octet escrit – Últim *ACK* rebut.

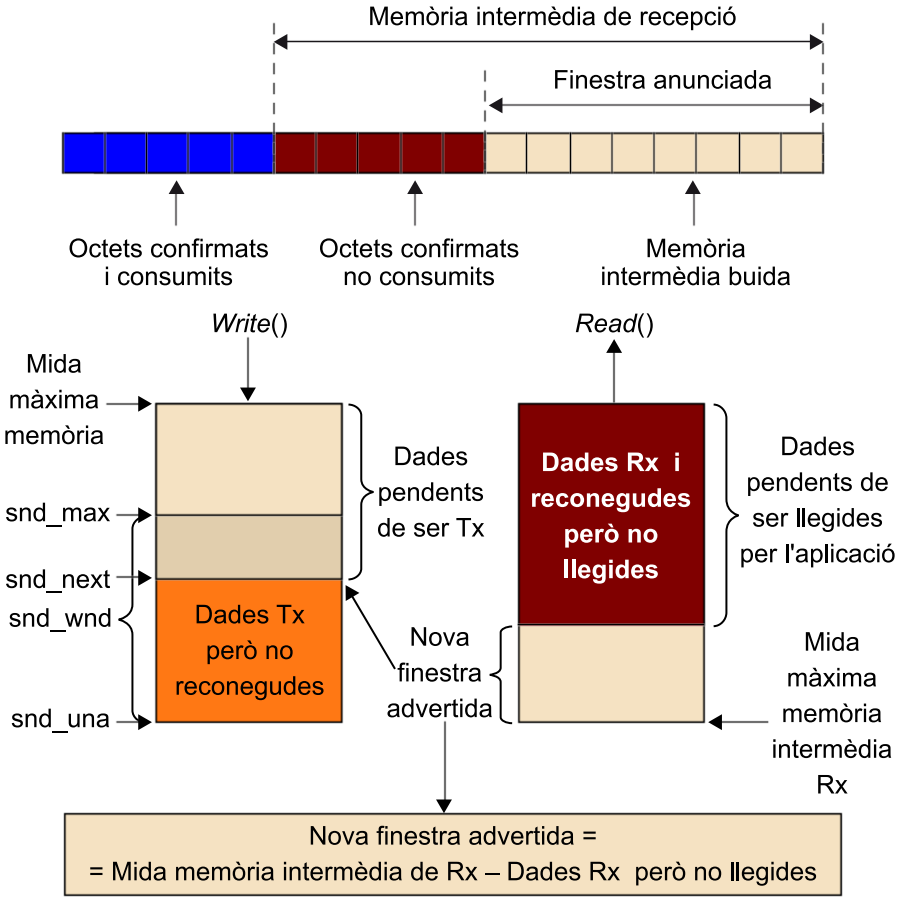
I que la finestra advertida:

Finestra advertida ≥ Últim octet consumit – Últim *ACK*

6.3.3. Finestra advertida de recepció

El receptor desa en la memòria intermèdia els octets rebuts i els pendents de consumir per l'aplicació.

Figura 33



Casos que es poden donar:

- Si el receptor és més ràpid que l'emissor, en aquest cas el receptor estarà esperant contínuament dades i la finestra podria arribar a ocupar tota la memòria intermèdia.

- El receptor és igual de ràpid que l'emissor, i en aquest cas es van transmetent dades i la memòria intermèdia del receptor té dades no consumides i dades que s'han rebut de l'emissor.

- El receptor és més lent que l'emissor, i en aquest cas la finestra advertida s'anirà reduint fins que bloquegi l'emissor, ja que no podrà enviar més dades.

Tipus d'aplicacions que usen TCP

1) Transferència massiva[22]:

- Aplicacions que envien gran quantitat de dades.
- La memòria intermèdia de transmissió sempre és plena i TCP envia segments de mida màxima.
- La finestra de transmissió limita la velocitat efectiva de la connexió.
- Aplicacions que generen aquest tipus de trànsit: FTP, Web, correu electrònic, etc.
- Els segments porten més de 512 octets de dades.
- Són importants els algorismes de control de la congestió (*Slow start*, *Congestion avoidance*, *Fast retransmit* i *Fast recovery*).

2) Aplicacions interactives:

- Aplicacions en les quals l'usuari interactua amb la màquina remota: *rlogin*, Telnet.
- La informació s'envia en segments de pocs octets i de manera discontínua (el 90% dels segments tenen menys de deu octets de dades).
- No és important el control de la congestió. No significa que no l'implementin, només que no es veuen afectades per aquest control de la congestió.

[22] En anglès, *bulk transfer*.

6.3.4. El problema de les aplicacions interactives

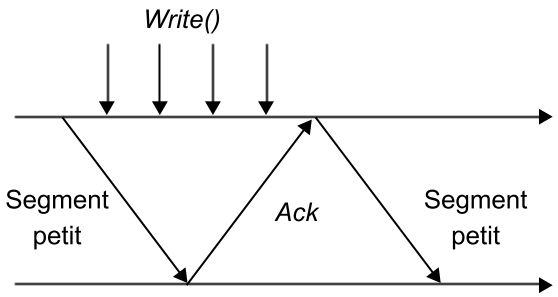
En connexions TCP com Telnet o *rlogin*, les dades són escrites pràcticament d'octet en octet, la qual cosa provoca que es transmetin molts segments petits i les seves confirmacions corresponents per cada tecla premuda. El fet que cada segment hagi de ser confirmat fa que la relació d'octets tramesos d'informació respecte als octets transmesos en confirmacions sigui baixa i que l'eficiència del protocol decaigui. Per a solucionar aquest problema l'algorisme de Nagle proposa:

- Evitar la tramesa d'un segment per cada tecla premuda, en les aplicacions interactives. Els octets són emmagatzemats en la memòria intermèdia de transmissió, entre *ack* i *ack* rebuts, amb l'objectiu d'enviar-los en un únic segment.

- Si la finestra permet enviar un nou segment, només s'envia si: hi ha suficients octets per a enviar un segment de mida màxima; no hi ha octets pendents de confirmar; i si hi ha octets pendents de confirmar, els octets

que van arribant a la memòria intermèdia de transmissió es retenen fins que arriba la confirmació.

Figura 34



6.3.5. *Delayed acknowledgements*

Una altra manera d'adreçar el problema de la transmissió de dades d'aplicacions interactives és mitjançant la tècnica dels *delayed acknowledgements*. Aquesta tècnica redueix el nombre d'*acks* enviats pel receptor. El receptor no transmet l'*ack* immediatament després de rebre noves dades, sinó que espera un temps per a veure si arriben els nous segments d'informació, i després envia un *ack* (que els confirmarà tots).

Algunes implementacions recomanen:

- Retard *delayed acks* ≤ 0,5 s
- Enviar almenys un *ack* per cada dos segments de mida màxima.

En les implementacions de TCP/IP actuals el retard dels *delayed acks* és menor o igual que 200 ms.

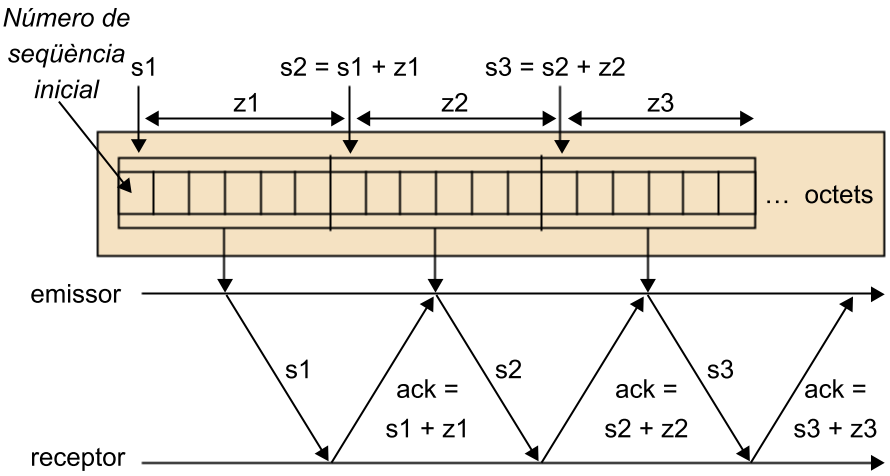
6.4. Números de seqüència en TCP

Com ja hem vist amb anterioritat en aquest mòdul, l'ús dels números de seqüència és cabdal per al funcionament dels protocols ARQ, ja que ens permeten establir correspondències entre les trames trameses i les seves confirmacions.

En TCP, els números de seqüència i les finestres es mesuren en octets; aquests identifiquen el primer octet de dades del segment. Si un segment té número de seqüència S_i i porta MSS octets, els números de seqüència següents seran S_i + MSS, S_i + 2 MSS, S_i + 3 MSS, increments de valor MSS. La confirmació del segment de dades amb número de seqüència S_i que porta MSS octets porta el valor: *ack* = S_i + MSS. (*ack* és el valor del número de seqüència del pròxim segment de dades que espera rebre el receptor o el pròxim octet que falta al receptor.) Les confirmacions són acumulatives: la recepció d'una confirmació

amb *ack* implica que tots els octets identificats amb números de seqüència inferiors al de l'*ack* han arribat correctament al receptor, i que per tant l'emissor els pot esborrar de la memòria intermèdia de transmissió.

Figura 35



6.5. Establiment i acabament d'una connexió TCP

TCP és un protocol orientat a la connexió, tal com hem dit amb anterioritat. Això implica que en tot procés de comunicació hi haurà una fase d'establiment de la connexió en què emissor i receptor se sincronitzen per tal de poder intercanviar dades. En TCP s'usa l'algorisme *encaixada a tres mans*[23], que consisteix en l'intercanvi de tres segments que no porten dades (només és la capçalera TCP).

[23] En anglès, *three-way handshake.*

Establiment de la connexió

[24] En anglès, *well known.*

[25] En anglès, *kernel.*

1) L'emissor envia un segment SYN, amb:

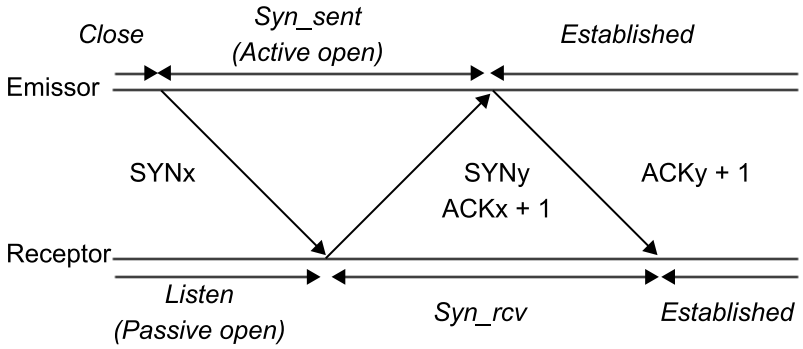
- Port destinació = Port ben conegut[24] d'un servidor (< 1.024).
- Port origen = Port efímer (escollit pel nucli[25], > 1.024).
- *Seq. number* = ISN client escollit a l'atzar pel nucli, s'utilitza per a identificar els octets de dades enviats pel client.
- Indicador SYN = 1 (activat).
- Indicador *ACK* = 0 (no es confirma res).
- És possible negociar (indicar) opcions com l'MSS, *timestamp*, *window scale factor* o SACK, com a indicació que es vol utilitzar.

2) El receptor (servidor) torna un segment SYN + *ACK*:

- *Seq. number* = ISN servidor per a identificar els octets de dades enviades pel servidor. Els segments de SYN, encara que no portin cap octet de dades, consumeixen un número de seqüència.
- *ACK* = ISN client + 1, reconeixent el segment SYN indicant el seu ISN.
- Indicador SYN = 1 (activat).
- Indicador *ACK* = 1 (activat).

3) El client respon amb un segment *ACK* reconeixent el SYN + *ACK*. Finalment, el client confirma la recepció del SYN + *ACK* enviant la confirmació amb el valor ISN servidor + 1. Un cop establerta la connexió es passa a la fase d'enviament de dades.

Figura 36




```
11:27:13.771041 147.83.35.18.3020 > 147.83.32.14.ftp: S
951111901:951111901(0) win 32120 <mss 1460,sackOK,timestamp
86683767 0,nop,wscale 0> (DF)
11:27:13.771491 147.83.32.14.ftp > 147.83.35.18.3020 : S
211543977:211543977(0) ack 951111902 win 10136 <nop,nop,timestamp
104199850 86683767,nop,wscale 0,nop,nop,sackOK,mss 1460> (DF)
11:27:13.771517 147.83.35.18.3020 > 147.83.32.14.ftp: . 1:1(0) ack 1
win 32120 <nop,nop,timestamp 86683767 104199850> (DF)
```

Acabament de la connexió

De la mateixa manera que l'establiment de la connexió, es fa necessari un procés d'acabament de la connexió. El tancament de la connexió pot ser per diverses causes:

- El client o el servidor tanquen la connexió (LLS *close*()).
- Per alguna raó s'envia un *reset* de la connexió (indicador actiu RST).
- Tancament a causa d'una interrupció, un Control-D o un Control-C, etc.

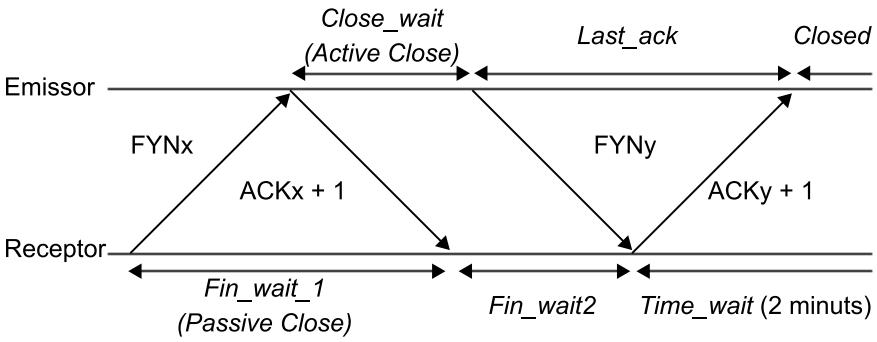
L'acabament també es fa per mitjà de l'enviament de segments TCP. El primer segment de *FINAL* el pot enviar tant el client com el servidor. El tancament normal és a causa d'un *close*() del client, cosa que provoca l'intercanvi de tres o quatre segments TCP.

Com la connexió TCP és dúplex cada participant en la connexió ha de tancar la connexió. Es fan servir els missatges de FIN/*ack* en un sentit, i FIN/*ack* en el sentit contrari. Igual que el SYN, el *FINAL* consumeix un número de seqüència. El segment de *FINAL* pot portar dades, si encara hi ha octets de dades per enviar en la memòria intermèdia de transmissió de TCP quan l'aplicació fa la crida *close*().

És possible que un extrem tanqui el seu costat de la connexió i l'altre no. En aquest cas, l'extrem que no ha tancat pot enviar dades i l'altre extrem les reconeixerà encara que hagi tancat la seva connexió. Aquest fet s'anomena *connexió de tancament parcial*[26].

[26]En anglès, *half closed*.

Figura 37

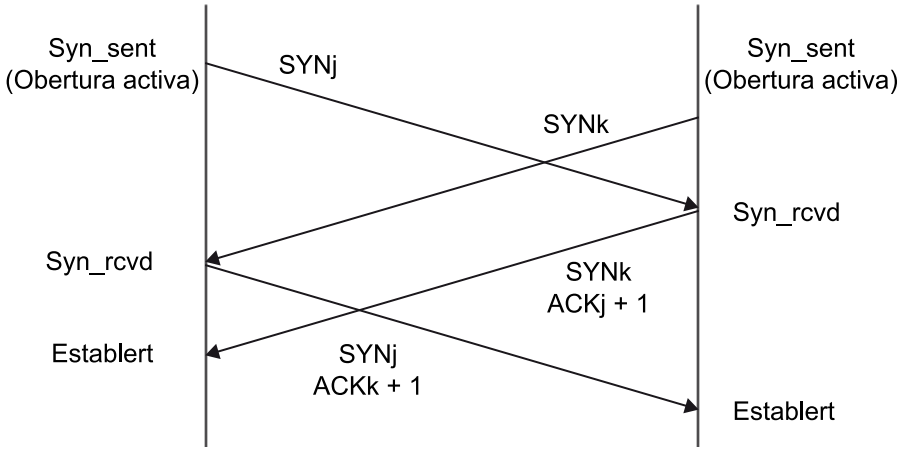



```
11:27:19.397349 147.83.32.14.3020 > 147.83.35.18.ftp: F
3658365:3658365(0) ack 1 win 10136 <nop,nop,timestamp 104200411
86684327> (DF)
11:27:19.397370 147.83.35.18.ftp > 147.83.32.14.3020: . 1:1(0) ack
3658366 win 31856 <nop,nop,timestamp 86684330 104200411> (DF)
```

```
11:27:19.397453 147.83.35.18.ftp > 147.83.32.14.3020: F 1:1(0) ack
3658366 win 31856 <nop,nop,timestamp 86684330 104200411> (DF)
11:27:19.398437 147.83.32.14.3020 > 147.83.35.18.ftp: .
3658366:3658366(0) ack 2 win 10136 <nop,nop,timestamp 104200412
86684330> (DF)
```

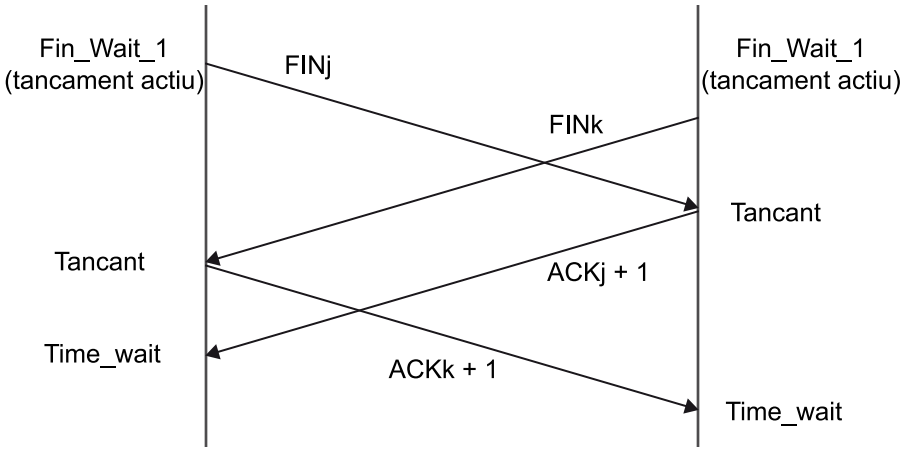
Els diagrames de temps següents mostren situacions diverses que es poden donar en l'establiment o tancament de la connexió TCP.

Figura 38. *Simultaneous open*



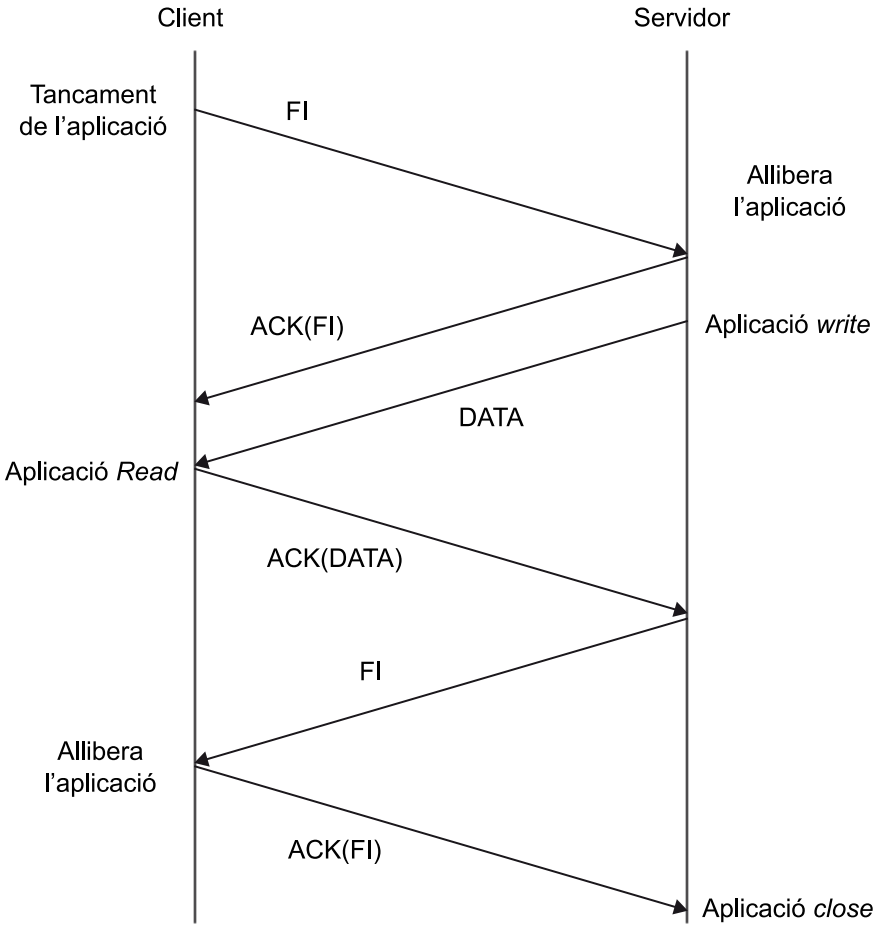
Simultaneous open és una situació en què emissor i receptor intenten establir la connexió de manera concurrent. En aquest cas, el SYN-*ACK* ja confirma la connexió

Figura 39. *Simultaneous close*



Simultaneous close és com l'anterior, una situació en què emissor i receptor decideixen finalitzar la comunicació de manera concurrent. En aquest cas, l'acabament de la comunicació es duu a terme per a cada extrem independent

Figura 40. *Half closed*



Finalment el tancament parcial permet a l'extrem que ha finalitzat la connexió continuar enviant confirmacions a l'altre extrem

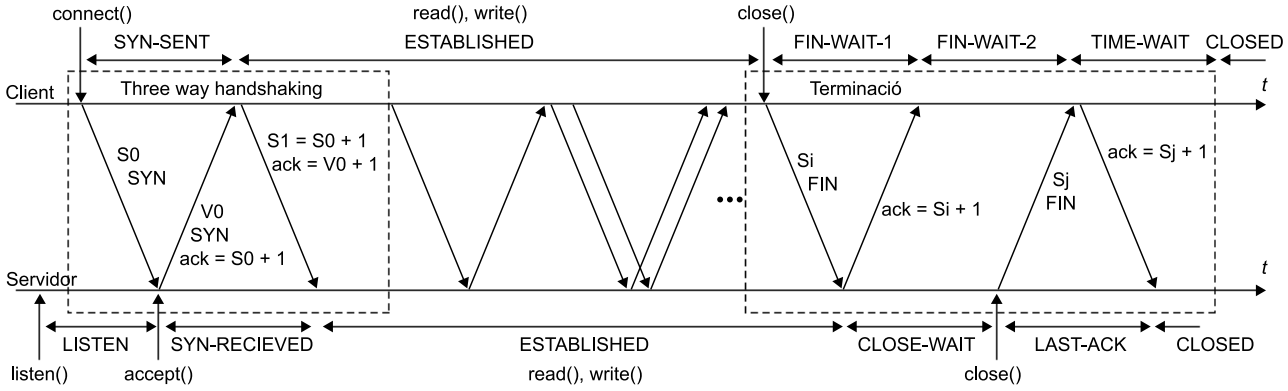
6.6. Diagrama d'estats de TCP

El protocol TCP és orientat a la connexió i, per tant, té diferents estats durant el seu cicle de vida. Concretament, passa per tres fases dividides cada una en un nombre determinat d'estats.

1) Fase d'establiment de la connexió.
2) Fase de transmissió.
3) Fase d'acabament.

Entre aquestes fases podem distingir onze estats diferents:

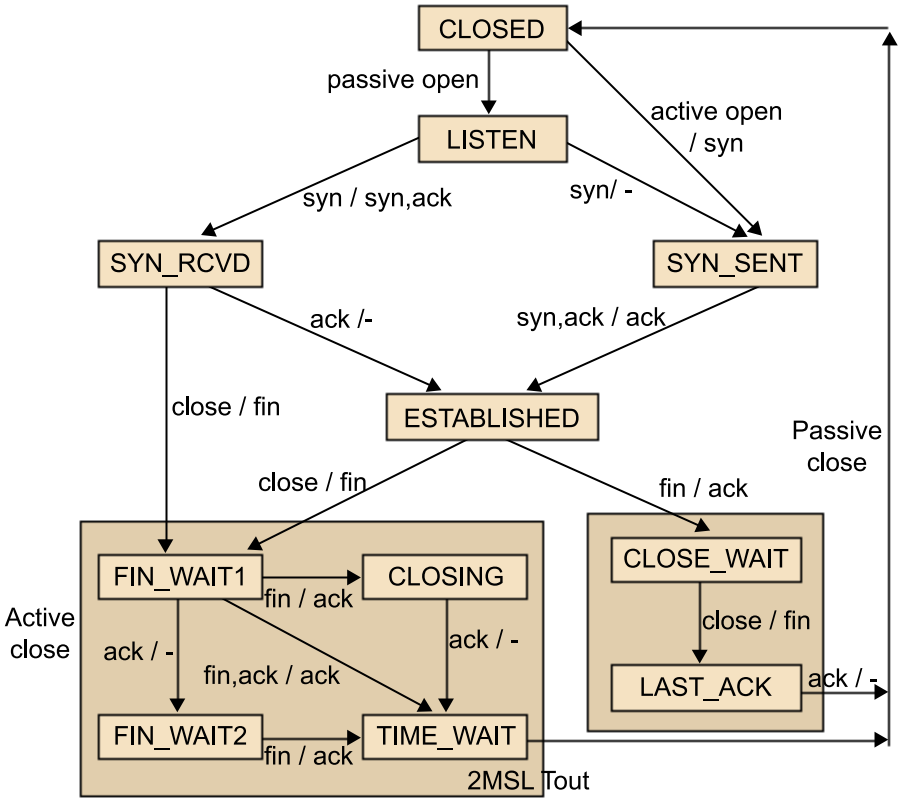
Figura 41



La taula següent presenta els onze estats de TCP:

Active open	***CLOSED***	La connexió no està essent usada.
	SYN_SENT	L'emissor envia un segment SYN després d'executar-se l'API *connect*() i queda en espera de rebre confirmació.
Passive open	***LISTEN***	El receptor espera rebre peticions de clients (emissors).
	SYN_RECEIVED	El receptor rep un segment SYN i accepta la connexió (fa la crida a l'API *accept*()). Envia un segment amb els indicadors de SYN i *ACK* activats.
ESTABLISHED		Connexió establerta. El client (emissor) passa a aquest estat quan rep el segment SYN + *ACK* del (receptor) servidor i respon amb un *ACK*. Quan el servidor (receptor) rep aquesta *ACK* també passa a l'estat d'*ESTABLISHED*.
Active close	***FIN_WAIT_1***	El servidor o client fa un *close*() de la connexió i envia segment *FINAL*. Es produeix el tancament de la connexió origen.
	CLOSED	L'altre extrem, quan rep l'*ack* del *FINAL* que ha enviat, té la certesa que els dos extrems han rebut el segment de *FINAL* i passa directament a l'estat de *CLOSED*.
	FIN_WAIT_2	Es rep confirmació de connexió d'origen tancat per part de l'altre extrem, i es queda en espera de la desconnexió remota.
	***TIME_WAIT* (2 min)**	L'ordinador central que ha iniciat l'acabament, quan rep el segment *FINAL* de desconnexió de la connexió remota, envia l'*ACK* i es queda en l'estat de *TIME-WAIT* durant un temps $T = 2 \times$ MSL (màxim temps de vida del segment) = 2 min. En cas de pèrdua d'aquesta confirmació, l'ordinador central podria contestar a les possibles retransmissions de l'últim *FINAL*, mentre es troba en aquest estat. Aproximadament 1 min és el temps que es pot estar un datagrama com a màxim a Internet.
Passive close	***CLOSE_WAIT***	Recepció del primer segment *FINAL*, tramesa de confirmació i espera per a la desconnexió de la connexió remota.
	LAST_ACK	Inici de desconnexió remota. Queda en espera de rebre reconeixement per part de l'altre extrem.

Figura 42



Exemple de funcionament del protocol de finestra lliscant

Utilitzarem el programa *tcpdump* per a veure com funciona el protocol de finestra lliscant. Assumim que hem efectuat un *rlogin* d'*argos* a *helios* (*argos % rlogin helios*) i ja estem connectats a *helios*. Un cop som a *helios*, executem el comandament *ls*. Aquest retorna per sortida estàndard la llista de directoris del directori de l'usuari[27] en *helios*, que ocupen 811 caràcters (representa l'enviament de 811 octets). *helios % ls*

[27]En anglès, *home directory*.

Les línies que obtenim amb el programa *tcpdump* (numerades de l'1 al 13) són les següents:

1) 15:56:59.506091 argos.1023 > helios.login: P 37:38 (1)*ack* 596 win 31744
2) 15:56:59.516091 helios.login > argos.1023: P 596:597 (1) *ack* 38 win 8760
3) 15:56:59.526091 argos.1023 > helios.login: .*ack* 597 win 31744
4) 15:56:59.846091 argos.1023 > helios.login: P 38:39 (1)*ack* 597 win 31744
5) 15:56:59.856091 helios.login > argos.1023: : P 597:600 (3) *ack* 39 win 8760
6) 15:56:59.866091 argos.1023 > helios.login: .*ack* 600 win 31744
7) 15:57:00.116091 argos.1023 > helios.login: P 39:40 (1)*ack* 600 win 31744
8) 15:57:00.126091 helios.login > argos.1023: P 600:603 (3) *ack* 40 win 8760
9) 15:57:00.136091 argos.1023 > helios.login: .*ack* 603 win 31744
10) 15:57:00.146091 helios.login > argos.1023: P 603:658 (55) *ack* 40 win 8760
11) 15:57:00.156091 argos.1023 > helios.login: .*ack* 658 win 31744
12) 15:57:00.166091 helios.login > argos.1023: P 658:1414 (756) *ack* 40 win 8760
13) 15:57:00.176091 argos.1023 > helios.login: .*ack* 1414 win 31744

La interpretació d'aquestes línies és la següent: *argos* ja ha enviat 36 octets, mentre que *helios* ja n'ha enviat 595 (informació que tots dos han intercanviat des del començament de la connexió, com poden ser *login*, *username*, etc.). Deduïm aquesta informació de la primera línia de l'exemple.

1) *argos* envia el caràcter *l*. L'indicador *P* assenyala *PUSH*. El número de seqüència avança de 37 a 38.

2) *helios* retorna un *echo* del caràcter *l*. El seu número de seqüència avança de 596 a 597 i reconeix l'octet rebut (*ACK* = 37 + 1 = 38).

3) *argos* reconeix l'*echo*: *ACK* = 597 + 1 = 598.

4) *argos* envia el caràcter *s*. El número de seqüència avança de 38 a 39. L'*ACK* no reconeix res perquè val igual que abans: *ACK* = 597.

5) *helios* fa un *echo* que ocupa 3 octets (BS* = 1 + s). El número de seqüència avança tres posicions (de 597 a 600) i reconeix el caràcter *s*, ja que *ACK* = 38 + 1 = 39.

6) *argos* reconeix l'*echo* amb un *ACK* = 600.

7) *argos* envia el retorn de carro (CR). El número de seqüència avança una posició.

8) *helios* fa un *echo* del CR i, a més, retorna un altre CR seguit d'un LF (*line feed*). Això significa l'enviament de 3 octets. Reconeix el CR, ja que *ACK* = 40.

9) *argos* reconeix aquests tres caràcters.

10) *helios* respon a *ls* enviant 55 dels 811 octets que ha d'enviar. El número de seqüència avança de 603 a 658. L'*ACK* resta a 40.

11) *argos* reconeix aquests 55 octets enviant un *ACK* de 659.

12) *helios* transmet la resta dels 811 octets, és a dir, 756 octets.

13) *argos* reconeix aquests octets avançant l'*ACK* a 1.414.

Com podem veure en aquest exemple, TCP divideix la informació per enviar en dos segments: un segment de 55 octets i un altre de 756 octets. Cal remarcar que *rlogin* envia els comandaments caràcter a caràcter i, a més, l'aplicació remota fa un *echo* d'aquests caràcters. Per això en les primeres línies s'envia primer la *l*, després la *s*, després el retorn de carro, etc. El que ens interessa d'aquest exemple és veure com avancen les finestres en emetre i en reconèixer octets. Per tant, no justificarem per què *rlogin* retorna ecos ni per què afegeix un caràcter LF al retorn de carro.

6.7. Control de la congestió

Fins ara hem vist com TCP implementava el control d'errors i el control de flux fent ús de variants dels protocols ARQ que hem introduït al principi d'aquest mòdul. Un dels problemes que poden aparèixer durant la comunicació entre un emissor i un receptor és la congestió: TCP intenta adaptar la velocitat de transmissió de l'emissor a la dels encaminadors intermedis de la xarxa, per evitar la saturació de les seves memòries intermèdies i les pèrdues de paquets. Per a fer-ho, s'utilitza la finestra de congestió (*cwnd*). Aquest control de flux és imposat per l'emissor, ja que és aquest el que regula la finestra de congestió. El protocol intenta disminuir la congestió de la xarxa en el moment en què apareix; quan es perd un paquet o salta un temporitzador la finestra de congestió es divideix per la meitat i fa que la quantitat d'informació que s'envia a la xarxa disminueixi de manera multiplicativa en presència de múltiples pèrdues de paquets. De la mateixa manera, tan bon punt es detecta que ja no hi ha pèrdua de paquets, la mida de la finestra va creixent de manera additiva per tal d'intentar maximitzar la quantitat d'informació enviada.

En cas de congestió la *cwnd* es redueix de manera multiplicativa (és a dir, ràpidament) per evitar agreujar la congestió. En absència de congestió, *cwnd* s'incrementa lentament, i intenta buscar el punt on no hi hagi congestió però en què l'aprofitament de les línies de transmissió sigui màxima.

Per a aconseguir aquest comportament de la xarxa s'han definit diversos mecanismes que intenten regular la congestió. Els mecanismes bàsics usats són:

- *Slow start* / *Congestion avoidance*.
- *Fast retransmit* / *Fast recovery*.

Activitat 3

Busqueu quina és la implementació actual de TCP. Quines són les seves característiques?

6.7.1. Algorismes *Slow start* i *Congestion avoidance*

Tots dos mecanismes funcionen conjuntament. Fan el control bàsic de la finestra de TCP. Aquests algorismes utilitzen les variables següents:

- ***cnwd***: és la finestra de congestió.
- ***snd_una***[28]: és el primer segment no confirmat. És a dir, és el segment que fa més temps que espera en la memòria intermèdia de transmissió per a ser confirmat.
- ***ssthresh***[29]: Límit entre la fase de *slow start* i *congestion avoidance*.

(28) *snd_una* vol dir *unacknowledged*, 'sense confirmar'.

(29) *ssthresh* vol dir *slow start threshold*, 'llindar de començament lent'.

6.7.2. *Slow start*

Slow start augmenta el valor de la finestra de congestió fins que arriba el més ràpidament possible al llindar *ssthresh*, valor per sobre del qual es poden produir pèrdues de paquets. *Slow start* injecta segments a la xarxa, segons va rebent *ack* des de l'altre extrem. Durant *Slow start*, l'increment de la finestra de congestió és molt ràpid, creix de manera exponencial en funció del temps.

A cada *RTT*[30] (el retard que hi ha des que TCP envia un segment fins que n'arriba la confirmació) la *cwnd* es multiplica per 2. En un temps $n \times RTT$, la *cwnd* passa a valer aproximadament 2^n.

(30) *RTT* és la sigla de *round trip time*.

S'envien segments de manera exponencial, fins que se satura el canal de comunicació, omple la memòria intermèdia d'algun encaminador intermedi i produeix pèrdues de paquets.

El nom de *Slow start* és contradictori perquè *cwnd* augmenta ràpidament durant aquesta fase.

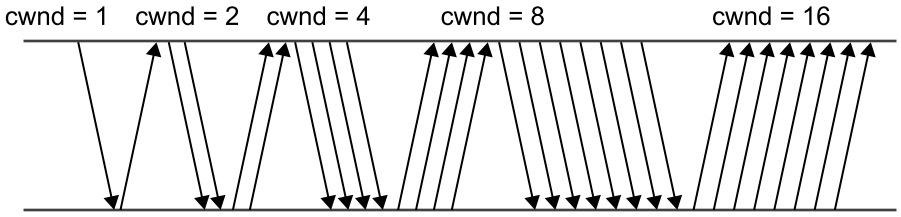
```
Initialització
   cwnd = MSS;
   ssthresh = 216 = 65535 bytes;
```

```
Quan es rep un ack:
   if (cwnd < ssthresh) /* Estem a Slow start (SS)*/
      cwnd = cwnd + MSS; //Creix exponencialment
   else      /* Estem a Congestion avoidance (CA)*/
      cwnd = cwnd + MSS/cwnd; //Creix linealment

Quan es produeix una perduda deguda a time-out o a la recepció de tres ACK duplicats

Retransmet el segment snd_una;
cwnd = MSS; //Això només en el cas que la congestió sigui deguda
      //al salt del temporitzador.
ssthresh = max(2, min(snd_awnd, snd_cwnd) / 2)
      //threshold passa a valer la meitat de la finestra de transmissió però no
      //per sota de 2 segments.
```

6.7.3. Funcionament de *Slow start*

TCP inicia la finestra de congestió amb el valor MSS. Inicialment només pot enviar un segment inicial sense confirmar, i es queda en espera. Mentre no hi ha pèrdues, TCP incrementa *cwnd* en MSS per cada nou *ack*. Quan arriba l'*ACK* del primer segment, *cwnd* augmenta a 2 MSS i s'envien dos segments. Quan arriben els justificants de recepció, la finestra de congestió s'incrementa en MSS per cada acusament rebut i acaba valent 4 MSS. En arribar els justificants de recepció dels quatre segments enviats, la finestra s'incrementa fins a 8 MSS i així successivament, fins que arriba al límit de la finestra de recepció.

Figura 43



6.7.4. *Congestion avoidance*

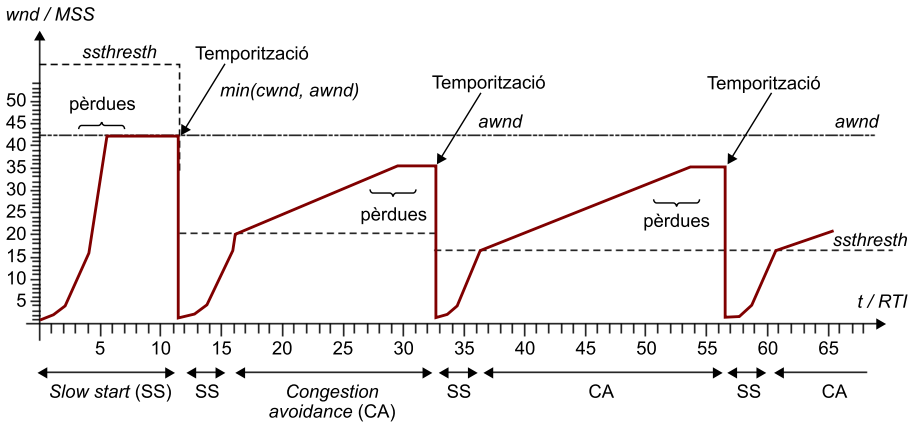
Congestion avoidance evita que es congestioni la xarxa, cosa que provocaria que es deixés de transmetre, saltés el temporitzador i comencés de nou un procés de *Slow start*. El que fa *Congestion avoidance* és intentar mantenir la finestra de congestió *cwnd* en el límit de la transmissió de dades sense tenir pèrdues. Quan la finestra de congestió supera el llindar *ssthresh*, per una banda, *Congestion avoidance* augmenta lentament el valor de *cwnd* (transmissió de manera lineal) per si la finestra ha quedat per sota del seu valor òptim, i l'enllaç s'està infrautilitzant, i per altra banda intenta ajustar la variable *ssthresh* a un valor en què TCP no tingui pèrdues.

6.7.5. Funcionament

Inicialment *ssthresh* val infinit (*Slow start*). Quan es produeix alguna pèrdua, saltarà el temporitzador, es farà la retransmissió, i es posarà *ssthresh* al valor de la finestra de transmissió que hi havia en el moment de la pèrdua dividit per 2: (mín(*awnd, cwnd*) / *2*).

A continuació *Slow start* incrementarà *cwnd* ràpidament fins al valor de *ssthresh*. Quan *cwnd* superi *ssthresh*, TCP entrarà en la fase de *Congestion avoidance*: la finestra augmentarà lentament, aproximadament 1 MSS cada vegada que es reben les confirmacions de tota una finestra, i la finestra passarà a valdre: *cwnd* = *cwnd* + MSS/*cwnd*.

Figura 44



Funcionament de TCP

Cal destacar que si no hi ha un enllaç congestionat (com passa típicament quan el client i el servidor estan dins d'una mateixa LAN), TCP està sempre en *Slow start*. En aquest cas la finestra augmenta fins a la finestra advertida (*awnd*), i a partir d'aquest moment actua només el control de flux: la finestra de TCP és sempre igual que l'advertida.

6.8. Temporitzadors en TCP. Càlcul del temporitzador de retransmissió (*RTO*)

El protocol TCP utilitza diversos temporitzadors, i el més important és el temporitzador de retransmissió *RTO*[31]. El temporitzador de retransmissió *RTO* determina quan s'ha de retransmetre un segment davant de l'absència de reconeixement d'aquest segment. Està activat sempre que hi ha dades pendents de confirmar. Si el temporitzador expira abans de l'arribada d'una confirmació en l'emissor, aquest retransmet el segment sense confirmar.

Cada vegada que arriba una confirmació de noves dades:

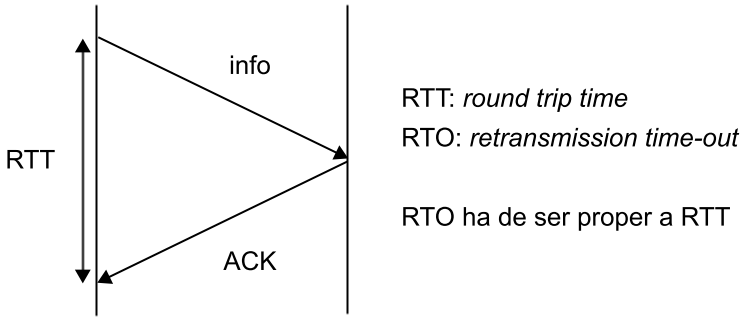
1) TCP calcula el valor de l'*RTO*.

2) Si hi ha més dades pendents de confirmar, llavors el temporitzador s'actualitza al valor calculat; si no el temporitzador es desactiva.

El problema principal consisteix a fixar l'*RTO*.

(31)*RTO* és la sigla de *retransmission timeout*.

Figura 45



Si *RTO* < *RTT* s'envia un paquet duplicat inútilment a la xarxa, ja que el sistema creu que hi ha una pèrdua quan en realitat només hi ha un retard.

Si *RTO* > *RTT* es pot arribar a esperar més temps del que realment és necessari. Com que l'*RTT* varia a cada segment, l'*RTO* no es pot calcular de manera única sinó que per a cada segment es necessita fer el càlcul del seu temporitzador.

6.8.1. Càlcul del temporitzador *RTO* en les primeres implementacions del protocol TCP

Per a calcular el valor del temporitzador, en primer lloc s'havia d'estimar el retard mitjà d'anada i tornada de l'enllaç, és a dir, *MRTT*[32] (valor mitjà de l'*RTT*). Per tal d'aconseguir aquesta mètrica s'utilitzava una estimació ponderada (anomenada *estimació exponencial*) que assigna més pes al còmput en les darreres estimacions de l'*RTT*.

(32)*MRTT* és l'estimació de la mitjana del temps de cicle.

Estimació exponencial

- Valor mitjà de l'*RTT*:

$$MRTT(k) = \alpha \cdot MRTT(k-1) + (1-\alpha) \cdot RTT(k)$$

- Temporitzador de retransmissió:

$$RTO = \beta \cdot MRTT$$

En què $0 \leq \alpha \leq 1$:

- Si α és proper a 1 fa que el pronòstic sigui inalterable als canvis.
- Si α és proper a 0 es respon amb rapidesa als canvis

β = 2 és el valor que es pren com a referència (el doble del valor de la mitjana). Si β = 1 es produïen moltes retransmissions por qualsevol retard.

6.8.2. Algorisme de Jacobsen per al temporitzador TCP

En canals amb alta fluctuació del valor instantani d'*RTT*, el factor 2 pot ser insuficient per a protegir-se de retransmissions innecessàries, mentre que per a canals molt estables aquest factor pot ser excessiu. Per aquesta raó Jacobsen va proposar un algorisme basat en el còmput de la desviació mitjana del valor

(33)*DRTT* és la desviació mitjana de la mitjana del temps de cicle.

instantani de l'*RTT* (*DRTT*[33]) respecte a la seva mitjana (*MRTT*). Com en el cas del còmput de la mitjana de l'*RTT*, per a ponderar en més mesura les estimacions més recents de la desviació, s'utilitza un algorisme de tipus exponencial:

Algorisme de Jacobsen

- Valor mitjà de l'*RTT*:

$$MRTT(k) = \alpha \cdot MRTT(k-1) + (1-\alpha) \cdot RTT(k)$$

- Desviació mitjana de l'*RTT*:

$$DRTT(k) = \gamma \cdot DRTT(k-1) + (1-\gamma) \cdot |RTT(k) - MRTT(k)|$$

- Temporitzador de retransmissió:

$$RTO(k) = MRTT(k) + \beta \cdot DRTT(k) (\alpha = 7/8, \beta = 4 \text{ i } \gamma = 3/4)$$

És a dir, com més gran sigui la variabilitat d'*RTT*, major serà la variància i, per tant, la diferència entre el valor de l'*RTO* i l'estimació de l'*RTT* (per a evitar que l'*RTO* salti prematurament i es facin retransmissions innecessàries).

6.8.3. Mesures dels temporitzadors en TCP

L'*RTO* es calcula en termes de *slow-timer tics* (cada 500 o 200 ms).

L'*RTT* es mesura de diferents maneres:

- En funció dels tics (aproximació dura).
- Usant segells de temps[34]. Aquesta opció consisteix a enviar un segell de temps a la capçalera dels segments (el valor del rellotge de l'ordinador central quan TCP envia el segment), i afegir també un *echo* del segell de temps del segment que confirmen. D'aquesta manera, quan arriba l'*ack*, la diferència entre el rellotge de l'ordinador central i el valor que porta l'*echo* del segell de temps és una mesura precisa de l'*RTT*.

(34) En anglès, *timestamps*.

6.8.4. Algorisme de Karn

En cas de forta congestió de la xarxa, cas en el qual es produeixen pèrdues de paquets o valors de l'*RTT* extremadament elevats, no és convenient mantenir el valor del temporitzador de retransmissió. La temporització s'hauria d'incrementar per a garantir que els paquets es retransmeten una vegada la situació de congestió ha desaparegut.

Per aquesta raó en situacions de retransmissió es fa servir el denominat *algorisme de Karn*, basat en l'algorisme de creixement exponencial de l'*RTO*[35]. En aquest cas de retransmissió d'un segment, el seu funcionament és el següent:

(35)En anglès, *exponential RTO backoff*.

1) No s'ha d'utilitzar l'algorisme de Jacobsen per a estimar el nou valor de l'*RTO*. No s'ha d'actualitzar l'*RTT* en els segments retransmesos. Només es mesura l'*RTT* per als segments que es transmeten per primera vegada.

2) Cada vegada que es produeix un procés de retransmissió d'un segment, el valor de temporitzador *RTO* s'incrementa exponencialment (TCP s'espera el doble del temps esperat en la transmissió anterior, per evitar que la xarxa es pugui tornar inestable):

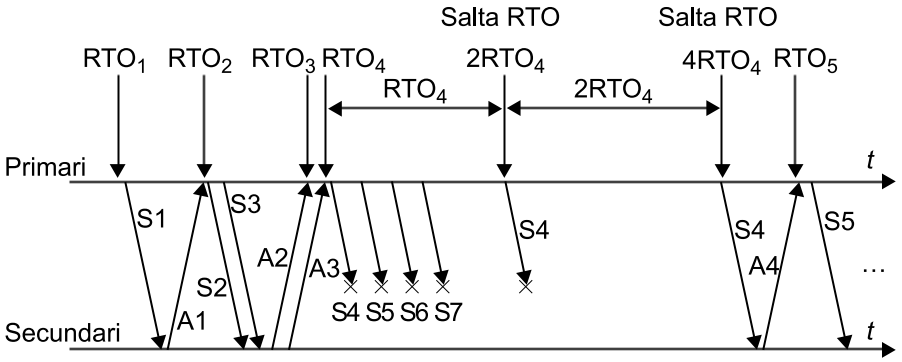
$$RTO(k+1) = q\ RTO(k), \text{ en què habitualment } q = 2$$

3) Aquest còmput de l'*RTO* es manté fins que es rep un segment de reconeixement corresponent a un paquet no retransmès (és a dir, es tornen a confirmar noves dades). A partir d'aquest moment es torna a utilitzar Jacobsen.

Utilització de l'algorisme de Karn o Jacobsen

```
Si la retransmissió:

RTO(k+1) = 2 · RTO(k) (Algorisme de Karn-Creixement exponencial d'RTO)

Si no:

      RTO(k) = MRTT(k) + 4 · DRTT(k) (Algorisme de Jacobsen)
```

Figura 46



A la figura 46 el segment S4 es retransmet dues vegades. *RTO*1, *RTO*2... Són els instants en què s'actualitza el temporitzador, ja sigui perquè es confirmen noves dades, o perquè es produeix una retransmissió. En cas de retransmissió, l'*RTO* es fixa a dues vegades el valor que tenia anteriorment.

6.8.5. Problema dels temporitzadors de retransmissió de TCP

Com hem vist, la problemàtica dels temporitzadors TCP no és trivial. El problema principal és que la determinació de l'*RTO* és imprecisa, ja que es basa en el valor *RTT* de l'últim segment de reconeixement rebut. L'*RTT* depèn de

l'estat de la xarxa, del tipus de reconeixement efectuat pel protocol TCP i els retards en el terminal receptor. Si un segment es perd en transmissió, l'emissor continuarà enviant segments fins que la seva finestra de transmissió s'esgoti. El receptor rebrà aquests segments, però com hi ha un segment previ no rebut no enviarà cap tipus de reconeixement, i esperarà que arribi el segment perdut. La retransmissió del segment perdut es farà quan expiri el temporitzador *RTO*, la qual cosa fa que aquest procés pugui ser extremadament lent.

6.9. Algorismes *Fast retransmit / Fast recovery*

El problema dels temporitzadors ha estat adreçat amb un seguit de mecanismes que permeten la recuperació ràpida de la congestió de la xarxa.

6.9.1. *Fast retransmit*

L'algorisme *Fast retransmit*[36] agilitza el procediment de retransmissió basat en el temporitzador *RTO*. Quan el receptor rep un segment d'informació fora de seqüència, immediatament envia una confirmació del segment que espera rebre. Aquests *acks* es diuen *acks duplicats*. Quan l'emissor rep confirmacions duplicades, desconeix si això és a causa que el segment perdut és simplement retardat a la xarxa (el segment arribarà i, per tant, no és necessari retransmetre'l) o bé el segment s'ha perdut realment (és necessari retransmetre'l el més ràpidament possible).

(36)En català, *retransmissió ràpida*.

Fast retransmit proposa esperar 3 *acks* duplicats seguits (és a dir, 4 *acks* amb el mateix número de seqüència) per a estar raonablement segurs que el segment s'ha perdut i retransmetre'l immediatament, sense esperar que salti el temporitzador. A partir d'aquell moment:

- retransmet *snd_una* (retransmet el primer segment no confirmat).
- *ssthresh* = màx(mín(*awnd*, *cwnd*) / 2,2 MSS). Calcula l'*ssthesh* com la meitat de la finestra actual (igual que quan salta el temporitzador).
- *cwnd* = *ssthresh* + 3 MSS;

i es passa al mode *Fast recovery*.

6.9.2. *Fast recovery*

Un cop es recupera un segment perdut i s'envia la confirmació amb el número de seqüència de l'últim segment rebut correctament, TCP intenta tornar a *Congestion avoidance* sense haver de passar una altra vegada per *Slow start*. D'això s'encarrega *Fast recovery*.

Fins que no es rebi un nou *ack* no duplicat, l'emissor augmenta *cwnd* en un MSS per cada *ack* duplicat rebut, per a poder enviar un nou segment (si el deixa la finestra):

$$cwnd = cwnd + \text{MSS}$$

Quan es confirma el segment retransmès, és a dir, es rep un *ack* nou no duplicat, l'emissor surt de *Fast recovery* i posa:

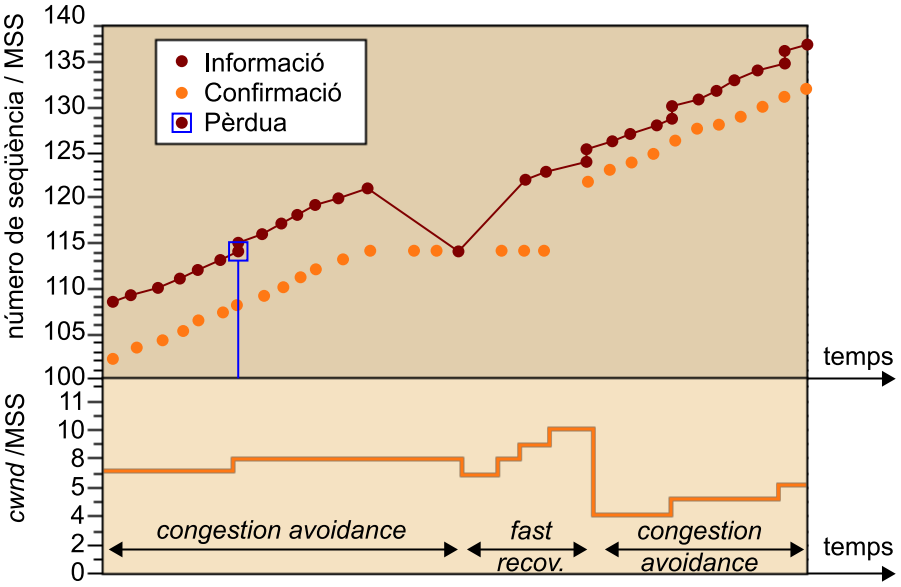
$$cwnd = ssthresh$$

A partir d'aquest moment, l'emissor entra en fase de *Congestion avoidance*.

Pseudocodi explicatiu dels algorismes *Fast retransmit*/*Fast recovery*

```
Cada vegada que es rep un ack
   Si (es rep un ack duplicat) {
      Si (és el 3r. ack duplicat) { //Fast retransmit
         retransmet snd_una; //Retransmet el primer segment no confirmat
         ssthresh = max(min(awnd, cwnd) / 2, 2 MSS);
         // Calcula l&#8217;ssthesh com la meitat de la finestra actual (igual que quan
         // salta el temporitzador).
         cwnd = ssthresh + 3 MSS;
         // La finestra que suposa que hi havia abans de la pèrdua, més els 3 segments
         // que han generat els acks duplicats.
         fast_recovery = CERT ;
      }
      si no { //Fast recovery
         si(fast_recovery == CERT) {
            cwnd += MSS;
            //Per cada ack duplicat s&#8217;incrementa la finestra en un MSS, per a
            //poder enviar un nou segment (si el deixa la finestra).
         }
      }
} altrament { //Es rep un ack nou (no duplicat)
   si(fast_recovery == CERT) {
      cwnd = ssthesh;
      fast_recovery = FALS;
      //Quan es confirmen noves dades se surt de fast recovery i es posa cwnd = ssthesh,
        per a iniciar la fase de congestion avoidance.
   } altrament {
      /* Slow start / Congestion avoidance */
   }
}
```

Figura 47



La part superior de la figura mostra una traça capturada en l'emissor amb l'evolució dels números de seqüència i les confirmacions, en un interval de temps en què es produeix una pèrdua. La part inferior de la figura mostra l'evolució de la *cwnd*.

A la figura 47 veiem que quan l'emissor rep 3 *acks* duplicats entra en *Fast retransmit* (*cwnd* valia 8 *MSS*). En aquest moment l'emissor retransmet el segment perdut, Calcula *ssthresh* = 4 *MSS*, la meitat de la finestra de congestió:

$$cwnd = ssthresh + 3 = 7 \text{ MSS}$$

A continuació entra en fase de *Fast recovery*. L'emissor augmenta *cwnd* en MSS per cada *ack* duplicat. Com que quan entra en *Fast recovery* hi ha 8 segments sense confirmar, l'emissor pot enviar nous segments després de rebre 2 *acks* duplicats més (*cwnd* augmenta fins i tot a 9 MSS).

Quan arriba un *ack* nou no duplicat, l'emissor surt de *Fast recovery* i posa:

$$cwnd = ssthresh = 4 \text{ MSS}$$

Com que en aquest moment només hi ha dos segments sense confirmar, l'emissor en pot enviar dos més. A partir d'aquest moment, l'emissor entra en fase de *Congestion avoidance*.

6.10. Implementacions actuals de TCP

Totes les implementacions actuals de TCP estan obligades a implementar *Slow start* i *Congestion avoidance*. Per a identificar algunes de les diferents versions de TCP que s'ha fet al llarg del temps s'han utilitzat noms de ciutats: Tahoe, Vegas, Reno, etc.

- TCP Tahoe (1988): *Slow start*, *Congestion avoidance*, *Fast retransmit*.
- TCP Reno (1990): Tahoe + *Fast recovery* + *TCP header prediction*.

Actualment les implementacions milloren encara més el comportament de TCP amb algorismes addicionals:

- TCP *Sack*: Reno + *Selective ACK*.
- D'altres: multicàsting, *routing tables*

7. Altres protocols de transport

Tot i que s'ha centrat el mòdul a presentar els protocols de transport per excel·lència, que són TCP i UDP, cal conèixer que n'hi ha d'altres que han estat dissenyats per a suportar millor certs tipus de comunicacions o que han millorat alguns dels punts febles de TCP i UDP:

- SCTP[37] és un protocol de transport dissenyat com a alternativa a TCP i que, a part d'oferir fiabilitat, control de flux i seqüenciament, com TCP, també permet l'enviament de missatges fora d'ordre, i d'aquesta manera es comporta com un protocol no orientat a la connexió com UDP. Permet, a més, el paral·lelisme d'enviament de missatges.

(37) *SCTP* és la sigla de *Stream Control Transmission Protocol.*

- DCCP[38] és un protocol de transport no orientat a la connexió basat, com SCTP, en l'enviament de missatges. DCCP és usat per aplicacions que tenen necessitat de lliurament ràpid de dades. Aquesta categoria d'aplicacions inclou la telefonia a Internet i multimèdia en temps real, entre d'altres. DCCP està pensat per a aplicacions que requereixen el control de flux de TCP però no necessiten el lliurament en ordre ni la confiabilitat que ofereix TCP, o que volen un control de congestió dinàmic diferent del de TCP. De la mateixa manera, DCCP està definit per a aplicacions que no requereixen les característiques especials de SCTP, com per exemple el lliurament seqüencial de flux múltiple.

(38) *DCCP* és la sigla de *datagram congestion control protocol.*

Resum

L'objectiu principal d'aquest mòdul didàctic ha estat presentar en detall els serveis, protocols i funcionalitats del nivell de transport d'una xarxa. En el mòdul s'han introduït els protocols que s'utilitzen en la xarxa Internet. Aquests protocols són l'UDP i el TCP.

S'ha vist que l'objectiu principal del nivell de transport és lliurar la informació als nivells orientats a l'aplicació en els extrems de la xarxa. El mòdul ha presentat els protocols ARQ, protocols usats pel control d'errors i que asseguren la fiabilitat de la xarxa. Aquest protocols han estat descrits de manera genèrica sense entrar en l'especificitat dels protocols a la xarxa Internet. Més endavant però, s'ha posat l'èmfasi en els protocols que implementa la xarxa Internet. Els principals protocols de transport d'aquesta xarxa són:

1) L'UDP, que és un protocol no orientat a la connexió. Per tant, no efectua cap control d'errors ni de flux. Si un datagrama UDP arriba equivocat (l'UDP utilitza un codi detector d'errors), l'UDP el descarta i no el lliura a l'aplicació. Aquesta haurà de ser capaç de respondre a aquest tipus de servei o haurà d'assumir la pèrdua de la informació. Aquest tipus de servei pot ser útil en aplicacions en temps real, en què és més important que la informació arribi quan li pertoca, és a dir, amb un retard delimitat, que no pas que se'n perdi una part.

2) El TCP, que és un protocol orientat a la connexió. Hi haurà una fase d'establiment de la connexió (l'anomenat *procediment three-way handshake*), una fase de transmissió de la informació i una fase de terminació de la connexió. El TCP lliurarà la informació a l'aplicació totalment lliure d'errors. Per a aconseguir-ho, necessita efectuar un control d'errors i de flux. El TCP utilitza un codi detector d'errors juntament amb un protocol de retransmissions per a recuperar la informació errònia. Com que les memòries intermèdies de recepció es poden desbordar, el TCP fa servir un control de flux per finestra lliscant. S'han vist les implementacions específiques d'aquests protocols i s'han relacionat amb els protocols ARQ en què es basen. El TCP ha de dimensionar correctament els temporitzadors de retransmissió. Hi ha diferents algorismes, entre els quals destaca el de Jacobson, basat en una estimació de l'*RTT* (temps d'anada i tornada de la informació entre els extrems TCP), i el càlcul de la mesura i la variància de l'*RTT*. També hem estudiat els algorismes que TCP utilitza per a alleujar la congestió en la xarxa: *Slow start* i *Congestion avoidance*.

Activitats

1. Enviem 8.192 octets travessant quatre encaminadors (assumim que els encaminadors són del tipus *store and forward*, cada un dels quals està connectat a una línia telefònica T1 a 1.544.000 bps). Considerem que el temps de propagació entre el primer i l'últim encaminador és una constant K.

a) Quin és el temps màxim per a transmetre la informació si la longitud del segment de dades és de 4.096 octets?

b) Quin és el temps màxim per a transmetre la informació si la longitud del segment de dades és de 512 octets?

2. Quan s'estableix una connexió entre les estacions *argos* i *helios*, té lloc el procés següent:

```
15:56:54.796091 argos.1023 > helios.login S 3541904332: 3641904331 (0)win
31744 <mss 1460>
15:57:00.796591 argos.1023 > helios.login S 3541904332: 3641904331 (0)
win 31744 <mss 1460>
15:57:13.797035 argos.1023 > helios.login S 3541904332: 3641904331 (0)win
31744 <mss 1460>
15:57:13.797035 helios.login > argos.1023 S 548133143: 548133143 (0)ack
34519043333 win 8760 <mss 1460>
15:56:54.797035 argos.1023 > helios.login .ack 548133144 win 31744
```

El primer temporitzador d'*argos* s'inicialitza a 6 s. Determineu quant valen el segon i el tercer temporitzador d'*argos*.

3. Un TCP *transmissor* ha advertit, durant l'establiment d'una connexió, un MSS de 512 octets, i durant la transferència de la informació adverteix una finestra de 512 octets. Un TCP receptor adverteix finestres de 2.048 octets. Dibuixeu un diagrama de temps en què es percebi l'algorisme *Slow start*.

Bibliografia

Kurose, J. F.; Ross, K. W. (2005). *Computer networking: a top-down approach featuring the Internet*. Addison-Wesley.

Tanenbaum, A. S. (2003). *Redes de computadores* (4a. ed.). Pearson.