# Concepte de port

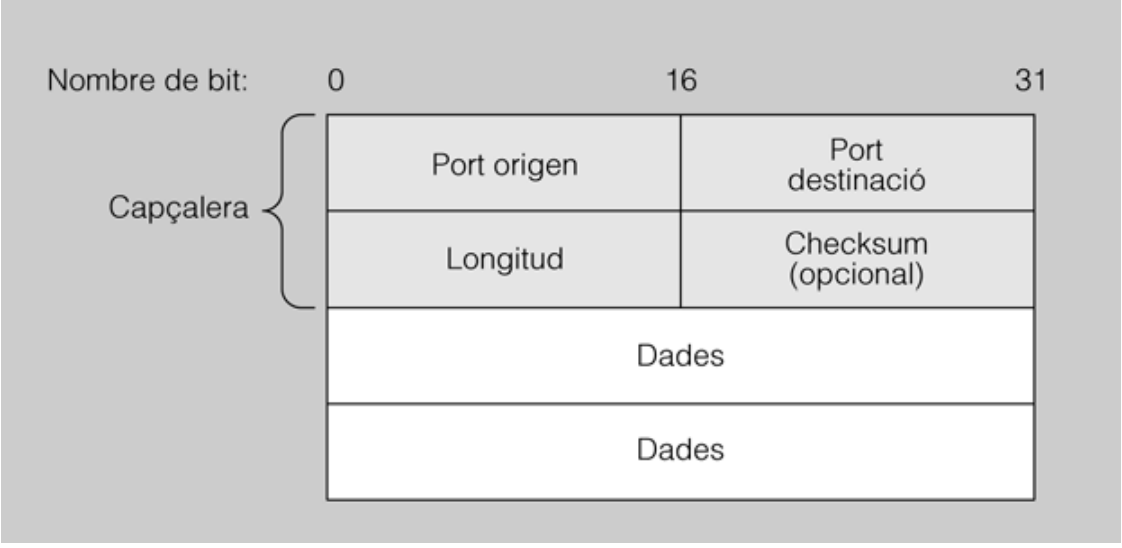
Un port no és més que un nombre, entre 0 i 65535 (escollit automàticament pel software de cada extrem) que serveix per identificar un canal de comunicació concret entre ambdós, i establir-lo de forma independent i separada de qualsevol altre canal que puguin tenir oberts els mateixos extrems en aquell mateix moment. Això és el que s'anomena **"multiplexació"**.

Quan un extrem fa de client (és a dir, quan sol·licita un recurs), el nº concret de port obert sol ser irrellevant (de fet, sol canviar en cada nova petició). En canvi, quan un extrem fa de servidor (és a dir, quan ofereix un recurs) sol utilitzar un nº de port estàndard (però diferent per a cada tipus de servei ofert; d'això en parlarem a l'estudiar el següent nivell) per a què els clients puguin accedir a aquest port ja conegut per defecte sense necessitat d'haver-lo d'indicar manualment.

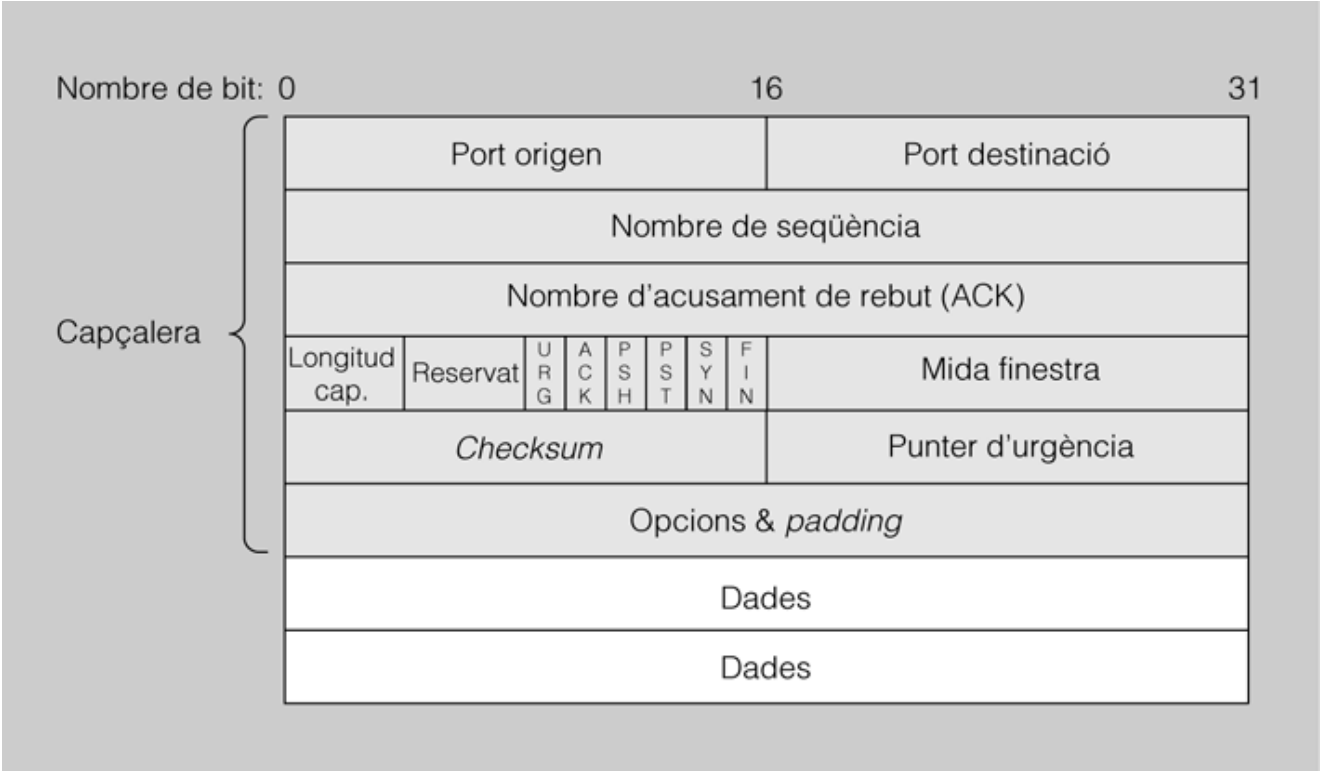
# UDP vs TCP

| UDP v/s TCP | | |
|---|---|---|
| **Characteristics/ Description** | **UDP** | **TCP** |
| General Description | Simple High speed low functionality "wrapper" that interface applications to the network layer and does little else | Full-featured protocol that allows applications to send data reliably without worrying about network layer issues. |
| Protocol connection Setup | Connection less data is sent without setup | Connection-oriented; Connection must be Established prior to transmission. |
| Data interface to application | Message base-based is sent in discrete packages by the application. | Stream-based; data is sent by the application with no particular structure |
| Reliability and Acknowledgements | Unreliable best-effort delivery without acknowledgements | Reliable delivery of message all data is acknowledged. |
| Retransmissions | Not performed. Application must detect lost data and retransmit if needed. | Delivery of all data is managed, and lost data is retransmitted automatically. |
| Features Provided to Manage flow of Data | None | Flow control using sliding windows; window size adjustment heuristics; congestion avoidance algorithms |
| Overhead | Very Low | Low, but higher than UDP |
| Transmission speed | Very High | High but not as high as UDP |
| Data Quantity Suitability | Small to moderate amounts of data. | Small to very large amounts of data. |

# UDP: capçalera



# TCP: capçalera



on destaquem els següents camps, a més del **port d'origen** i **port de destí** del paquet...

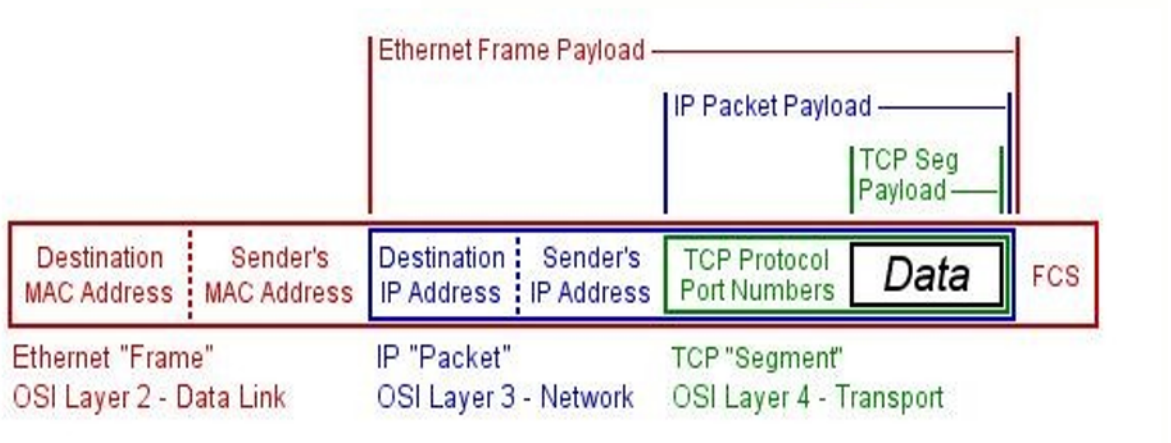
- **Nombre de seqüència**. Indica la posició que ocupa aquest segment en relació amb el conjunt de segments que formen el flux sencer de dades enviades en una connexió concreta. El nombre de seqüència del segment inicial en una connexió és un valor aleatori i, a partir de llavors, els nombres de seqüència dels següents segments que formen part del mateix flux es calculen a partir de la quantitat de dades enviades pel segment anterior (en bytes) + 1. Per exemple, si el nombre de seqüència d'un primer paquet que inicia una connexió suposem que és 1000, per exemple, i aquest primer paquet transporta 234 bytes, el nombre de seqüència del segon paquet que enviarà el mateix origen per la mateixa connexió tindrà un nº de seqüència 1000+234+1=1235.
- **Nombre d'acusament de rebuda (acknowledgment number)**. Indica el nombre de seqüència del segment que s'espera rebre a continuació per part de l'altre extrem.
- **Mida de la finestra**. Indica la mida de la finestra lliscant de recepció. Aquest valor representa el número de bytes que l'emissor serà capaç de processar en rebre el proper segment de part de l'altre extrem. Serveix per regular el trànsit segons les capacitats de l'emissor perquè, en definitiva, el que fa és avisar a l'altre extrem de quina és la quantitat de dades màxima que pot enviar-li en cada moment.

- **Bits de control (URG, ACK, PSH, RST, SYN, FIN)**. Cadascun d'aquests bits o indicadors (flags) és independent de la resta. En destaquem els següents:

| | |
|---|---|
| **SYN** | Quan val 1 indica al receptor que l'emissor vol iniciar una nova connexió |
| **RST** | Quan val 1 indica al receptor que cal que interrompi (a l'igual que farà l'emissor), la connexió mútua de forma immediata i abrupta (perdent-se, doncs, les dades que hi puguin haver en trànsit) |
| **FIN** | Quan val 1 indica que el segment actual és el darrer de la connexió per part de l'emissor (és a dir, que aquest no enviarà més dades). Això no vol dir que l'altre extrem no pugui continuar enviant dades, però, fins que ell no envïi el seu corresponent paquet FIN |
| **ACK** | Quan val 1 indica que el camp "nombre d'acusament" s'ha de tenir en compte |
| **PSH** | Quan val 1 indica que el host emissor ha de passar les dades al nivell inferior (capa 3) immediatament sense esperar a omplir el seu buffer. És l'aplicació generadora de les dades a enviar qui posa a "1" aquest flag |

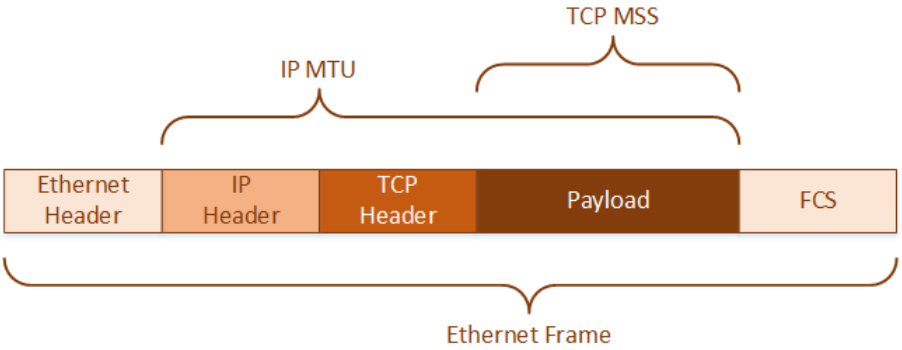
# Estructura paquet Ethernet

Ara que ja hem vist tots els nivells que afegeixen capçaleres respectives a unes determinades dades a transmetre, podem copsar l'estructura completa que té una trama Ethernet en una xarxa TCP/IP. Concretament, una dades a enviar encapsulades via TCP, seguidament encapsulades via IP i finalment encapsulades via Ethernet tindria un aspecte final com el següent (on s'han destacat només els camps més importants de cada capçalera):



# Conceptes de MTU i MSS

El **MSS** (*Maximum Segment Size*) és un valor que pot aparèixer dins del **camp d'opcions** de la capçalera TCP, i serveix per especificar el tamany màxim (en bytes) de dades crues (és a dir, el "payload") que una màquina pot rebre en un sol paquet sense fragmentar. Aquest valor es comunica durant el three-way handshake pels dos extrems (veure més endavant), i pot ser un valor diferent per cadascun. És evident que *MSS màxim possible = Payload màxim possible = MTU – tamany capçaleres IP i TCP* . Tenint en compte que el tamany típic d'una capçalera IP és de 20 bytes i el d'una capçalera TCP és 20 bytes també, tenim que el MSS màxim possible d'un paquet Ethernet per evitar la seva fragmentació és de 1500 – 20 – 20 = 1460 bytes.



# TCP: generalitats

Abans que es comenci una transmissió de dades amb TCP, s'estableix una connexió (o **"circuit virtual"**) entre els dos extrems de la comunicació. Aquest procediment és el que permet al TCP (a diferència de l'UDP, on no s'estableix cap connexió pròpiament dita), poder fer el següent:

- **Controlar l'ordre de recepció dels segments** (gràcies al processament per part del receptor del nombre de seqüència dels diferents segments rebuts)
- **Controlar les pèrdues dels segments** (igualment gràcies al processament per part del receptor del nombre de seqüència dels segments rebuts). D'altra banda, els nombres d'acusament de rebuda de part del receptor informen a l'emissor de la recepció correcta (o no) dels diferents segments.
- **Controlar les possibles duplicitats dels segments** (igualment gràcies al processament del nombre de seqüència)
- **Controlar les eventuals modificacions en el contingut del segments rebuts** (gràcies a l'ús de la suma de verificació; cal dir que aquesta funcionalitat també l'ofereix UDP)

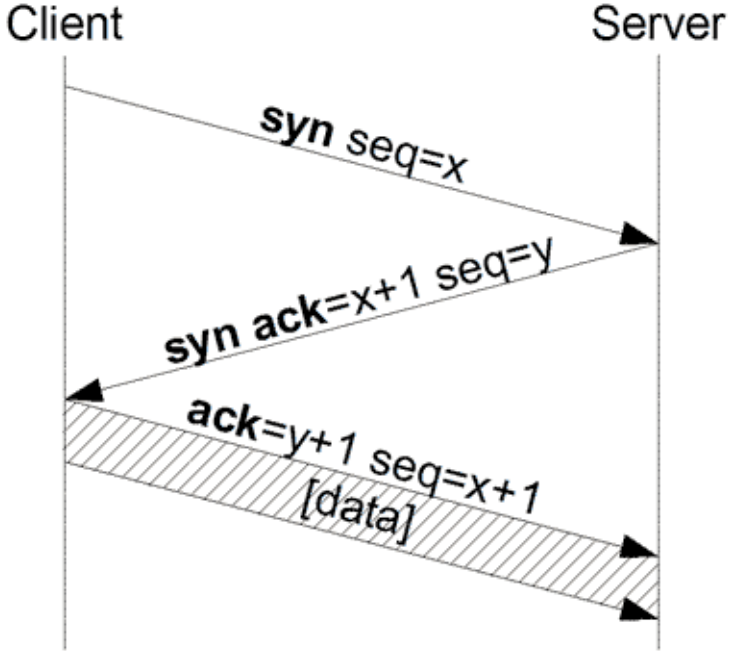
A més a més, el protocol TCP permet **controlar "en temps real" el flux de la transmissió de paquets (la congestió)** gràcies al mecanisme de les finestres "lliscants". Sense aquest control de la congestió una computadora amb una alta capacitat d'enviament d'informació pot saturar fàcilment una computadora més lenta. En aquest sentit, cal saber que cada pors té una memòria intermèdia (**buffer**, en anglès) pròpia, situada entre l'aplicació associada i el nivel inferior de xarxa, de tal manera que quan les aplicacions transmeten la informació als ports, aquests la van emmagatzemant fins que puguin enviar-la finalment per la xarxa (a no ser que vingui marcada pel flag "PSH", on llavors la transmeten immediatament sense acumular-la al buffer); un cop transmesa arribarà al port destí, on s'anirà guardant en el seu corresponent buffer fins que l'aplicació-destinatària concreta estigui preparada per rebre-la (a no ser que vingui marcada pel flag "PSH", on llavors s'entregarà a l'aplicació destinatària immediatament). El desequilibri entre el nivell d'ocupació dels buffers de l'emisor i del receptor és el que genera la congestió, que el protocol TCP és capaç de gestionar mitjançant el mencionat mètode de les finestres lliscants.

# TCP: 3-way handshake

El procés d'establiment d'una connexió TCP, anomenat "3-way handshake", consta de les tres accions explicades a continuació (i mostrades al diagrama següent):

1. El host que sol·licita la connexió envia un segment de sol·licitud de connexió al host destinatari (més en concret, el segment és enviat des d'un socket -IP:port- de l'emissor a un socket del receptor). Aquest segment s'identifica perquè té el seu flag **SYN** activat (és a dir, amb el valor 1), i la resta desactivats. D'altra banda, el seu nº de seqüència serà un valor aleatori, el seu valor del nº d'acusament és 0 i no transporta cap dada.
2. El host destinatari retorna un segment de confirmació al host sol·licitant. Aquest segment s'identifica perquè té els seus flags **SYN** i **ACK** activats (i la resta desactivats). El seu nº de seqüència també serà un valor aleatori; el nº d'acusament d'un segment qualsevol sempre és igual al del segment anterior (enviat en el mateix sentit) més la quantitat de dades rebudes pel/s segments/s en sentit contrari des de llavors; així doncs, com que el segment anterior no transportava cap dada (0 bytes), en aquest cas serà 0+1=1. Aquest segment tampoc no transporta cap dada.
3. El host sol·licitant envia un segment amb l'acusament de rebuda del segment de confirmació anterior. Aquest segment s'identifica perquè té el seu flag **ACK** activat. El nº de seqüència d'un segment qualsevol, un cop ja s'han generat els valors aleatoris inicials, sempre és igual al nº d'acusament del segment rebut anterior; en aquest cas,

doncs, serà 1. D'altra banda, el nº d'acusament també serà 1, segons el càlcul descrit al paràgraf anterior



A partir d'aquí, la connexió ja està establerta i, per tant, el host sol·licitant ja podrà començar a enviar dades (començant dins del mateix segment descrit al punt 3 anterior o bé en un de posterior). Mentre duri la transferència de dades, els segments involucrats tindran com a mínim el seu flag ACK activat també i el valor dels seus números de seqüència i acusament respectius es calcularan mitjançant l'algoritme descrit a les frases subratllades anteriors. Un exemple il·lustratiu de possibles valors dels camps mencionats pels diferents segments que intervenen durant l'establiment de la connexió i la posterior transmissió de dades podria ser el següent:

| Remitent | Flags | Quantitat dades | Nº seqüència | Nº d'acusament |
|---|---|---|---|---|
| Extrem A (->) | SYN | 0 | 0 | 0 |
| Extrem B (<-) | SYN-ACK | 0 | 0 | 1 |
| Extrem A (->) | ACK | 725 | 1 | 1 |
| Extrem B (<-) | També ACK | 1448 | 1 | 726 |
| Extrem A (->) | També ACK | 0 | 726 | 1449 |
| Extrem B (<-) | També ACK | 1448 | 1449 | 726 |
| Extrem A (->) | També ACK | 0 | 726 | 2897 |
| Extrem B (<-) | També ACK | 1448 | 2897 | 726 |

# TCP: interrupcions RST

Si s'enviés un segment SYN a un extrem que tingués el port en qüestió filtrat per un tallafocs, el comportament estàndard d'aquest és ignorar-lo (no obtenint el remitent, per tant, cap resposta). No obstant, és possible configurar el tallafocs per rebutjar paquets (és a dir, d'avisar al remitent del seu bloqueig) en lloc d'eliminar-los silenciosament; aquest avís es fa de forma estàndard mitjançant l'enviament d'un segment **RST-ACK**.

D'altra banda, una comportament interessant (que es pot provocar amb eines especialitzades com ara Nmap) és no finalitzar completament el "3-way handshake". És a dir, en comptes d'enviar el darrer segment ACK a l'altre extrem (després d'haver-ne rebut el SYN-ACK) per confirmar l'establiment de la connexió, enviant un paquet **RST** el que es provoca és finalitzar abruptament aquest intent d'inici de connexió. ¿Quin seria l'objectiu? Doncs no establir cap connexió però haver confirmat que el port de l'altre extrem està obert (ja que d'ell hem rebut el SYN-ACK). Aquesta manera de fer és molt comuna a escanejadors de ports per tal d'obtenir la informació que volen sense malgastar recursos i temps en crear connexions que no empraran.

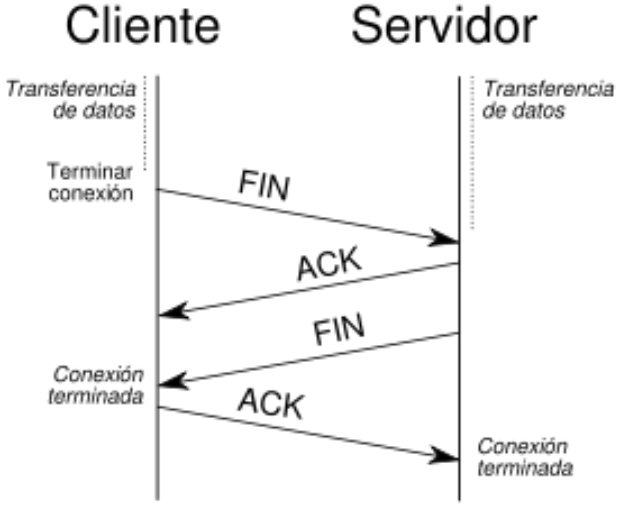
Finalment, si s'enviés un paquet ACK (o SYN-ACK) "no sol·licitat" (és a dir, no pertanyent a cap connexió en marxa) a un port qualsevol d'una màquina (això es podria fer, per exemple, amb eines com Nmap, nping o Scapy), aquesta hauria de respondre amb un altre paquet amb el flag **RST** activat, sigui quin sigui l'estat del port. D'altra banda, si s'enviés un paquet RST "no sol·licitat", aquest és simplement ignorat

# TCP: finalització de la connexió

Un cop totes les dades s'han transferit, cal finalitzar la connexió. Aquest procés de finalització també requereix una "salutació a tres passes":

1. El host que havia iniciat la transmissió (normalment, el client) envia un segment al host receptor indicant que ha acabat d'enviar dades identificat perquè té els seus flags **FIN** i **ACK** activats, (i la resta desactivats). Pot transportar les darreres dades o bé ser un paquet sense dades enviat exclusivament per demanar començar el procés de desconnexió
2. El host receptor confirma la sol·licitud de desconnexió. Això es fa mitjançant l'enviament de dos segments diferents però consecutius: primer un amb el flag **ACK** activat (i la resta desactivats) i després un altre amb els flags **FIN** i **ACK** activats (i la resta desactivats). En el moment en què el host receptor envia el primer segment ACK ja no podrà acceptar més dades del host emissor però les hi podrà continuar transmetent; en el moment en què el host receptor envia el segment FIN, ja no podrà seguir enviant dades.

3. El host sol·licitant respon amb un segment d'acusament de rebuda del paquet de confirmació de la desconnexió. Aquest segment s'identifica perquè té el seu flag **ACK** activat i serveix per a què el host receptor tanqui definitivament la connexió.




A vegades passa, no obstant, que en comptes de respondre al primer segment FIN amb un segment ACK per continuar amb el procés "3-way handshake" estàndard, el host receptor pugui respondre amb només un segment RST i ja està: si passa això la connexió es tancarà abrupta i immediatament.

# TCP: estats de la connexió

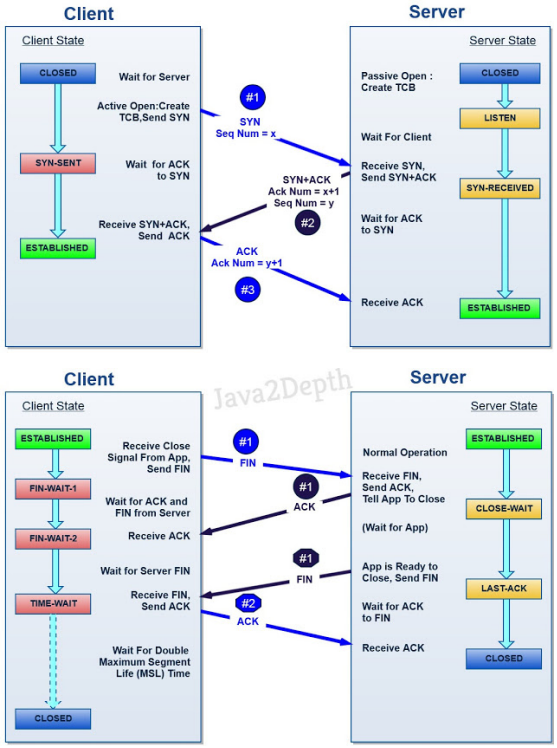
Les connexions TCP passen al llarg del temps per diversos estats, la llista completa dels quals es detalla a continuació:

- **LISTEN**: El socket està a l'espera de possibles peticions de connexió entrants. Típic de servidors que estan esperant eventuals connexions de clients.
- **SYN_SENT**: El socket està intentant establir de forma activa una connexió amb l'altre extrem. O dit d'una altra manera, ha enviat un segment SYN però encara no ha rebut la resposta SYN-ACK. Típic de clients en voler connectar amb un servidor.
- **SYN_RECV**: El socket ha rebut una petició de connexió de la xarxa i hi ha accedit. O dit d'una altra manera, ha enviat a l'altre extrem un segment SYN-ACK, tot i que encara no ha rebut la resposta ACK que confirmi l'establiment de la connexió definitiva. Típic dels servidors quan han rebut una petició de connexió dels clients i els han respost.
- **ESTABLISHED**: El socket està associat a una connexió establerta. O dit d'una altra manera, ja està llest per rebre o enviar dades de/a l'altre extrem perquè s'ha finalitzat el "3-way handshake".
- **FIN_WAIT1**: El socket està tancat (s'acaba d'enviar el segment FIN) però la connexió encara ha de finalitzar perquè tot just comença el "3-way handshake" de comiat

- **CLOSE_WAIT**: La connexió remota ha finalitzat (s'ha rebut el segment FIN) i, per tant, s'envia com a resposta un segment ACK (per tant, encara el socket no s'ha tancat)
- **FIN_WAIT2**: El socket està tancat i s'acaba de rebre el segment ACK de resposta al segment FIN enviat. Per tant, la connexió encara està en procés de finalitzar
- **LAST_ACK**: La connexió remota ha finalitzat (s'ha rebut el segment FIN) i ja s'ha enviat el segment ACK de resposta; aquest estat s'assoleix quan s'envia tot seguit el segment FIN de resposta (per tant, el socket ja es tanca definitivament però roman a l'espera d'un darrer ACK)
- **TIME_WAIT**: El socket està tancat i s'acaba de rebre el segment FIN de resposta al segment FIN enviat. Per tant, la connexió ja ha sigut tancada també en el altre extrem.
- **CLOSED**: La connexió ja no existeix

A excepció dels estats ESTABLISHED, LISTEN i CLOSED, la resta d'estats són temporals i efímers. De fet, el kernel Linux té indicat (mitjançant paràmetres "sysctl" específics) un determinat temps "timeout" per cadascun d'aquests estats temporals, més enllà del qual, si l'estat en qüestió no ha "transaccionat" a un dels tres estats definitius possibles anteriors, automàticament s'allibera la connexió i es tanca el socket per tal d'evitar que quedi penjat indefinidament.

El següent diagrama mostra, a mode de resum, la seqüència d'estats tant durant el procés de "3-way handshake" d'establiment de la connexió com durant el de finalització.



# TCP: finestres lliscants

Per "control del flux" s'entén tot un conjunt de procediments que permeten indicar al host emissor quina quantitat de dades (en nombre de paquets o en nombre de bytes) ha de transmetre abans de parar-se a esperar un segment ACK per part del host receptor. L'objectiu del control de flux ése no saturar el buffer d'entrada del host receptor (fet que provocaria pèrdua de paquets i les conseqüents retransmissions, perjudicant de retruc el rendiment de la xarxa). El més habitual per controlar el flux d'una transmissió és l'anomenat de "finestra lliscant", on l'emissor envia un nombre acordat de paquets abans d'esperar un (únic) ACK per a tots ells.

Concretament, cada cop que el host receptor respon amb un nou ACK, hi inclou de nou (actualitzat si cal) aquest nombre en el camp "Mida de finestra" de la seva capçalera per a què l'emissor el tingui en compte a partir de llavors (com a mínim fins que rebi la propera resposta ACK, on hi haurà un nou valor del camp "Mida de finestra" a inspeccionar). La idea és que cada extrem TCP (recordem que són duplex) reguli la quantitat de dades que l'altre extrem li pot transmetre.

El tamany concret que en cada moment pot ser publicat al camp "Mida de finestra" d'un determinat paquet ACK enviat pel receptor sol establir-se igual al tamany màxim disponible per poder ubicar futures dades al buffer d'entrada d'aquest receptor en aquell precís moment. Això vol dir que a la pràctica l'emissor sol enviar fins aquesta quantitat de dades abans d'haver-se d'esperar a una nova notificació d'espai disponible al "buffer" per part del receptor.

¿I què passa si els paquets són enviats a més velocitat que la que pot processar el dispositiu final (o qualsevol dels intermediaris, com "switchos" o "routers")? Ja sabem que els dispositius guarden els paquets rebuts en unes memòries anomenades buffers. Els buffers no tenen, però, un espai il·limitat i si la seva capacitat és excedida aleshores el receptor començarà a eliminar paquets. Tots aquests paquets s'hauran de tornar a transmetre i, per tant, el rendiment/velocitat de la transmissió disminuirà. El mecanisme per evitar la pèrdua de paquets en aquestes circumstàncies és reduir la noció que té el remitent de la finestra del receptor de manera que intenti enviar menys dades. A partir d'aquí, TCP augmenta la seva taxa d'enviament de nou, amb l'esperança que la pèrdua hagués sigut transitòria. La idea, doncs, sempre és la mateixa: la mida de la finestra s'anirà augmentant mentrestant el receptor pugui processar els paquets: si el receptor es satura, pot disminuir la mida de la finestra i fins i tot pot especificar una mida de zero (en aquest cas, l'emissor hauria de parar d'enviar informació fins que la finestra tornés a ser positiva). D'altra banda, la mida de la finestra tampoc pot tenir augmentar indefinidament: si un sol paquet que s'envia dins d'una finestra ha de tornar a ser transmés, aleshores tota la finestra s'ha de tornar a enviar (això pot afectar molt el rendiment).

Existeixen varis **algorismes** diferents que defineixen la manera concreta d'anar augmentant el rendiment de la transmissió de dades en l'emissor un cop aquest ha disminuit degut al desbordament del buffer en el receptor (o dispositius intermediaris): els actuals "htcp" o "cubic" són molt més agressius per tornar a la velocitat màxima que l'antic "reno". En qualsevol cas, es pot triar l'algoritme desitjat configurant el comportament del nucli mitjançant determinats fitxers ubicats a "/proc" i "/sys"

El següent diagrama il·lustra tot això explicat en aquest apartat: