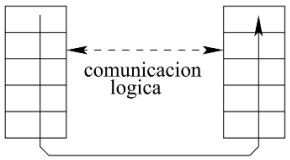
Tema 3: Capa de Transporte

Introducción

- La CAPA DE TRANSPORTE tiene dos funciones { en el origen → prepara los mensajes de las aplicaciones para ser transmitidos por un canal no fiable. / en el destino → recupera los mensajes y los entrega a las aplicaciones.
  - Sólo está implementada en los **sistemas finales** y no en los routers intermedios.
  - Proporciona una **comunicación lógica** entre procesos → gracias al modelo de capas, no se preocupa de cómo es el canal por el que transmite la información, sólo sabe que produce muchos errores.

Capa de aplicacion
Capa de transporte
Capa de red
Capa de enlace
Capa fisica



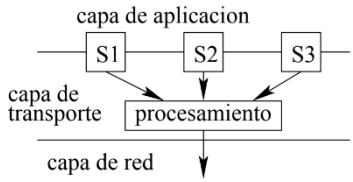
Protocolos de comunicación

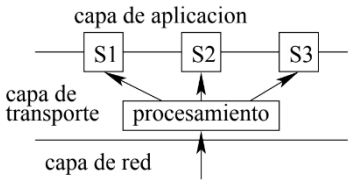
- En TCP/IP se usan los protocolos de transporte:
  - TCP → en el origen fragmenta los mensajes en **segmentos** y les añade la cabecera de transporte.
  - UDP → en el origen sólo le añade la cabecera de transporte al mensaje completo.

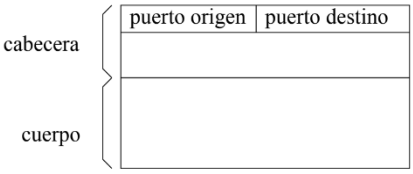
Interacción entre capa de aplicaciones y capa de transporte

- Los mensajes pasan de la capa de aplicación a la de transporte a través de un **socket** → los procesos escriben o leen de él como si fuese un archivo y se desentienden del resto.

1. **Multiplexación** → en el origen puede haber muchos sockets abiertos (pues hay muchos procesos transmitiendo), así que la capa de transporte recorre cada uno de ellos, procesa sus mensajes y envía los **segmentos** generados a la capa de red.
2. **Demultiplexación** → en el destino, la capa de transporte recoge los segmentos que llegan de la capa de red, reconstruye los mensajes y los reparte entre los sockets destino.
   - La identificación del socket destino al que debe ir un segmento se realiza a través de los NÚMEROS DE PUERTO que se colocan en su cabecera.
     - Los números de puerto son enteros de 16 bits.
     - Los números de puerto $0 - 1023$ están reservados por el administrador para servicios bien conocidos.

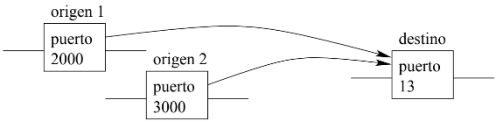






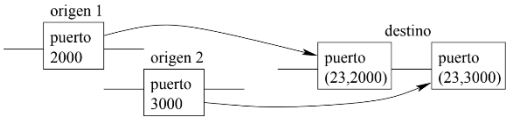
Multiplexación con sockets sin conexión (en UDP)

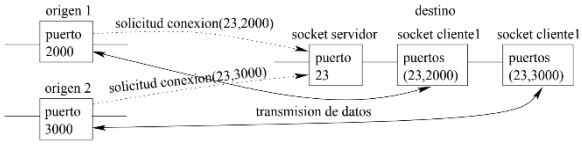
- Estos sockets se identifican sólo por la pareja dirección IP de destino y puerto de destino .
  - Por tanto, si dos segmentos de distintos hosts y distintos puertos origen se entregan al mismo puerto destino, los recoge el mismo proceso.
  - El **puerto origen** se usa para que el proceso sepa a quién responder.



Multiplexación con sockets con conexión (en TCP)

- Estos sockets se identifican por la 4-tupla de direcciones IP de origen y destino y puertos origen y destino.
  - Por tanto, si dos segmentos de distintos hosts (o distintos puertos origen) se entregan al mismo puerto destino, van a sockets distintos.
  - Esto permite que varias conexiones al mismo puerto puedan ser atendidas por **distintos procesos**.
- En estas conexiones hay 2 **tipos de socket**:
  - Un socket de servidor → lo único que hace es esperar a las conexiones de los clientes, una vez hechas las transfiere a un socket de conexión.
  - Varios sockets de conexión → se encarga de la transmisión de los datos.



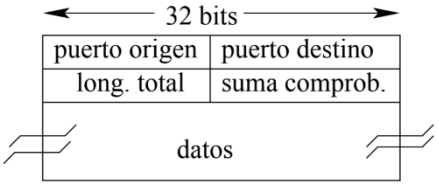


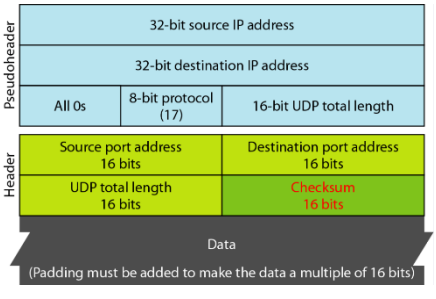
UDP: Capa de transporte SIN conexión

- UDP (protocolo de datagramas de usuario) es un protocolo de transporte **simple** y **poco sofisticado**.

Funcionamiento

1. En el origen, UDP añade la cabecera al mensaje para formar un segmento.
   - La **cabecera** está formada por 4 campos de 16 bits cada uno:
     - Puerto origen.
     - Puerto destino.
     - Longitud total del segmento (cabecera + datos) en bytes.
     - Suma de comprobación.
       - La **suma de comprobación** se calcula primero en el origen y luego en el destino. En el origen se procede así:
         1. Se suman las palabras de 16 bits de todo el segmento (incluida la cabecera UDP y algunos campos de la cabecera IP, la pseudocabecera).
            - Si la longitud de los datos del segmento no es múltiplo de 16 bits se rellenará con 0s hasta que lo sea.
            - Si hay desbordamiento, el bit sobrante se le suma al LSB (como en C2).
         2. Se le realiza el complemento a 1 al resultado.
2. En el destino, UDP comprueba que el segmento llegó sin errores volviendo a calcular la suma de comprobación.
   - En el destino, se vuelve a hacer la suma incluyendo el campo con el resultado del origen. No se hace el complemento a 1. El mensaje llegó bien si el resultado es todo 1s.
   - Si llegó bien → se pasa el segmento al socket.
   - Si llegó mal → normalmente se descarta el segmento.





CARACTERÍSTICAS

- Es un protocolo **sin conexión**:
  - No hay acuerdo previo entre emisor y receptor → por tanto, no hay retardo de establecimiento de conexión.
  - No hay retransmisiones en caso de recibir segmentos con errores.
- Es un protocolo **sin estado**, es decir, cada segmento se envía con independencia de los demás.
  - Por tanto, **los mensajes no se dividen** pues no habría información para reconstruirlos. Si el mensaje es demasiado grande para un segmento se produce un error.
    - Entonces, el procesamiento de los segmentos es más rápido y requiere menos recursos → los servidores pueden atender a más clientes.
- **Cabeceras más pequeñas** → los segmentos ocupan menos espacio de almacenamiento y se leen y escriben más rápido.
- **Más control de la aplicación** → la aplicación puede añadir su propia cabecera con información de control para realizar las funciones que necesita, pero UDP no proporciona.
  - Por ejemplo, la aplicación podría encargarse de la retransmisión del segmento en caso de error.
- **Usos** → aplicaciones en las que es preferible la velocidad frente a la fiabilidad.

APLICACIONES

- Aplicaciones que usan **TCP** { correo electrónico → SMTP; web → HTTP; acceso a terminales remotos → telnet, SSH; transferencia de archivos → FTP, SFTP }.
- Aplicaciones que normalmente usan **UDP** { traducción de nombres → DNS; protocolos de encaminamiento → RIP; administración de red → SNMP; servidor de archivos remoto → NFS }.
- Aplicaciones que usan TCP o UDP { flujos multimedia; telefonía por internet }.
  - Cada vez se usa más **TCP** en aplicaciones **multimedia**.
    - Las aplicaciones multimedia toleran pérdidas y responden mal a mecanismos de control de congestión.
    - Sin embargo, hay organizaciones que bloquean el tráfico UDP porque no tiene mecanismos de control de congestión.
      Esto provoca un desbordamiento de paquetes en los routers y el estrangulamiento del tráfico TCP, que sí tiene mecanismos de control de congestión.

TRANSMISIÓN FIABLE

- La TRANSMISIÓN FIABLE se basa en los protocolos de retrasmisión, los protocolos ARQ (solicitud automática de repetición), que retransmiten los paquetes cuando hay errores.

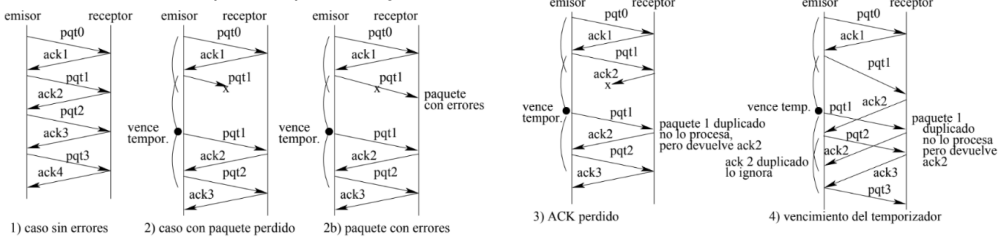
En este apartado veremos dos protocolos ARQ:

- **Parar y esperar** → el emisor envía un paquete y espera a la confirmación del receptor.
- **Ventana deslizante** → se pueden enviar varios paquetes antes de recibir las confirmaciones. Existen varias versiones de este protocolo, como **retroceder N** y **repetición selectiva**.
- Consideraremos **enlaces bidireccionales** (o full dúplex), es decir, que permiten la transmisión en los dos sentidos y de manera simultánea.
- **Numeraremos los paquetes** de 0 a $2^n - 1$ siendo n el número de bits para la numeración.

PROTOCOLO ARQ PARAR Y ESPERAR

Casos que se pueden dar en un protocolo ARQ parar y esperar:

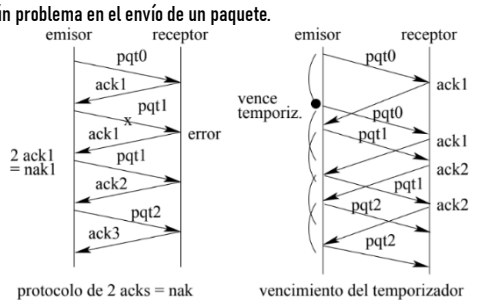
- **Sin errores** → cuando el receptor recibe el paquete devuelve un ACK. En el momento en el que el emisor recibe un ACK, pasa a enviar el siguiente paquete.
  - Por convenio, el número del ACK será el del siguiente paquete a enviar.
- **Pérdida de paquetes** → si se pierde un paquete, el receptor no se enterará y no devolverá un ACK. Para evitar bloquear la transmisión, cuando el emisor envía un paquete pone un **temporizador** a contar. Si vence el tiempo sin que el emisor obtenga un ACK, retransmite el paquete.
- **Paquete con errores** → se responde igual que en el caso anterior.
- **Pérdida de un ACK** → cuando el temporizador del emisor se venza, retransmitirá el paquete. El receptor recibirá un duplicado del paquete, pero lo descartará, y después volverá a enviar el ACK.
- **Vencimiento del temporizador** (timeout) → sucede si el tiempo del temporizador es demasiado breve o si un paquete o ACK se retrasa por problemas de congestión de la red. Como consecuencia, se duplicarán paquetes y ACKs.
  - Cuando el receptor reciba el paquete duplicado actuará como en el caso anterior (lo descarta y vuelve a enviar el ACK).
  - Cuando el emisor recibe el ACK duplicado simplemente lo ignora.



NAKS - VARIANTES

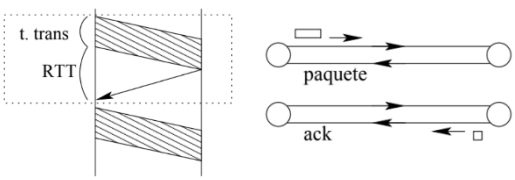
- Los NAKs son reconocimientos negativos y se usan para que el receptor indique al emisor que ha habido algún problema en el envío de un paquete.

- **Variante 1** → cuando el receptor recibe un paquete con errores, envía un NAK al emisor.
  - Así se ahorra tiempo, pues no se tiene que esperar al timeout para que el emisor reenvíe el paquete.
- **Variante 2** → 2 ACKS con el mismo número equivalen a un NAK.
  - Para que funcione se debe considerar que cuando el receptor recibe un paquete con error, devuelve un ACK al último paquete recibido correctamente.
  - El problema es que si se llega al **timeout** se enviarían continuamente duplicados de mensajes, pues cuando el emisor recibe un ACK duplicado ya no lo ignora, si no que reenvía el paquete otra vez.
- **Variante 3** → 3 ACKS con el mismo número equivalen a un NAK.
  - Resuelve algunos de los problemas de las otras variantes, pero no todos.
  - Aun así, una versión de esta variante se emplea en TCP.

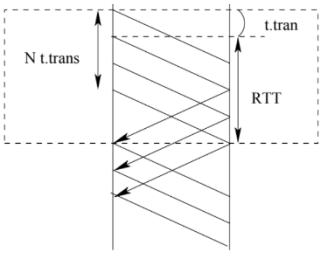


UTILIZACIÓN DEL ENLACE

- La principal desventaja de los protocolos ARQ parar y esperar es que **desperdician muchísimo tiempo** debido a que mientras se espera una respuesta, el enlace puede estar desocupado.
- Utilización del enlace por parte del emisor → $U = \frac{t_{trans}}{RTT + t_{trans}}$, $t_{trans} = \frac{L}{R}$
  - L la longitud del paquete en bits y R es la velocidad de transmisión en bps.



PROTOCOLO ARQ CON VENTANA DESLIZANTE



- La VENTANTA DESLIZANTE (o entubamiento) consiste en que el emisor envía un número N de paquetes antes de recibir los ACKs.
  - N se denomina TAMAÑO DE VENTANA.

UTILIZACIÓN DEL ENLACE

- **No** habrá **tiempo desperdiciado** (es decir, $U = 1$) cuando $N * t_{trans} \geq t_{trans} + RTT \rightarrow N \geq \frac{t_{trans} + RTT}{t_{trans}}$.

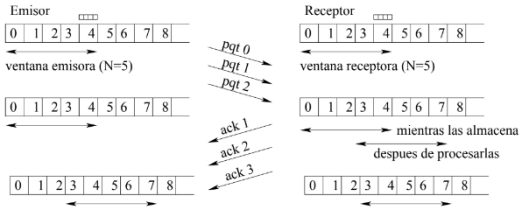
REQUERIMIENTOS

- El **rango de números de secuencia** debe ser por lo menos el doble de N.
- El emisor y el receptor deben poder almacenar **más de un paquete** para poder almacenar una copia de los paquetes que están en su ventana.

TEMPORIZADORES

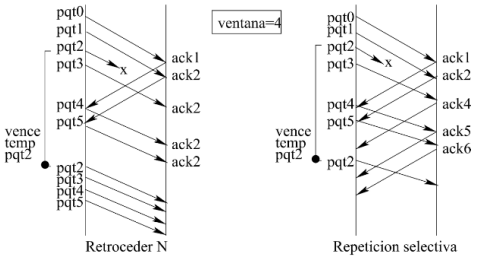
- Igual que los protocolos parar y esperar, el emisor pone en marcha un temporizador cada vez que manda un paquete.

VENTANAS



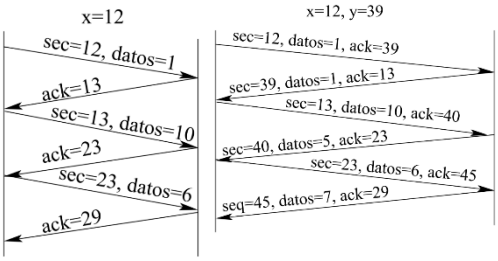
- **Ventana emisora** ⟶ conjunto de N paquetes que el emisor puede enviar o que ya ha enviado, pero están pendientes de confirmación.
  - Se desplaza cuando el emisor recibe la confirmación de un paquete.
- **Ventana receptora** ⟶ conjunto de N paquetes que el receptor puede aceptar o que ya ha aceptado, pero están procesándose.
  - Se desplaza cuando el receptor envía la confirmación de un paquete.

TIPOS



- **Retroceder N** (GBN) ⟶ el receptor sólo acepta paquetes en orden.
  - Hay **ACKs acumulativos** ⟶ un ACK implica a todos los ACKs previos.
  - Si un paquete llega al receptor con errores o no llega, el receptor descarta todos los siguientes.
  - Si ocurre un timeout, el emisor debe reenviar también todos los siguientes paquetes.
- **Repetición selectiva** ⟶ el receptor acepta paquetes en desorden.
  - **No** hay **ACKs acumulativos** ⟶ hay que enviar los ACKs para cada uno de los paquetes recibidos.
  - Sólo se retransmiten los paquetes erróneos o que no llegan.

TCP: CAPA DE TRANSPORTE CON CONEXIÓN



- TCP (protocolo de control de transmisión) es un protocolo de transporte que aplica los principios de la **transmisión fiable**.

NÚMEROS DE SECUENCIA

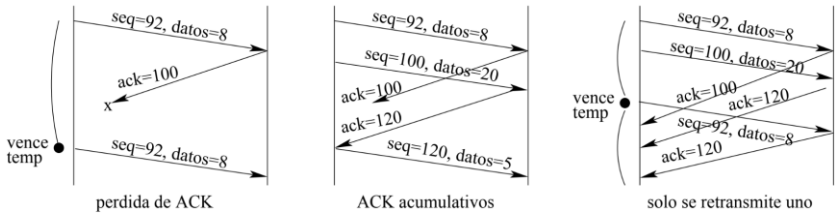
- Se usan números de **32 bits**.
- Se comienza por un **número aleatorio** x.
- En lugar de incrementar una unidad, en cada segmento se **incrementa en el número de bytes enviados**, $x + bytes$.
- Los **ACK** indican el **siguiente byte** que se desea recibir, $x + bytes + 1$.

TRANSFERENCIA FIABLE DE DATOS

- TCP es un ARQ de **ventana deslizante** intermedio entre **retroceder N** y **repetición selectiva:**
  - De retroceder N toma los **ACKs acumulativos**.
  - De la repetición selectiva toma que se aceptan **segmentos fuera de orden** y se **retransmiten** sólo los segmentos **necesarios**.

TEMPORIZADORES

- El emisor utiliza **temporizadores** para retransmitir los segmentos de los que no se han obtenido ACKs.
- Como la gestión de temporizadores consume recursos, se recomienda usar un **temporizador único** para toda la conexión que se reinicia cuando llega algún ACK.
  - Cuando se produce un timeout:
    1. Se retransmite sólo el segmento no confirmado.
    2. Se reinicia el temporizador.
    3. Se espera a recibir algún ACK antes de seguir.



SUPERPOSICIÓN

- La SUPERPOSICIÓN (o piggybacking) consiste en transmitir datos del receptor al emisor aprovechando el segmento ACK.
- Cuando llega un segmento desde el emisor, en lugar de enviar directamente el reconocimiento, el receptor espera hasta que tenga que enviar datos y los envía juntos.
  - Se debe seleccionar un **tiempo de espera** máximo hasta que haya datos en el receptor. Si este se sobrepasa, se envía el ACK solo.

GENERACIÓN DE ACKs

- Si el receptor no tiene datos a enviar, recibe un segmento esperado y ya envió el ACK del anterior ⟶ retrasa el ACK hasta que reciba otro segmento o transcurra cierto tiempo.
- Si llega un segmento esperado y no envió aún el ACK del anterior ⟶ envía el ACK del segmento recibido.
- Si llega un segmento con un número de secuencia mayor del esperado ⟶ envía el ACK con el número de secuencia esperado.
- Si llega un segmento duplicado ⟶ descarta el duplicado y envía el ACK con el número de secuencia esperado.
- Si llega un segmento que faltaba ⟶ envía el ACK con el número de secuencia del siguiente esperado (ACKs acumulativos).

ESTIMACIÓN DEL TIEMPO DE ESPERA

$$\begin{aligned} \text{Temporizador} &= \text{EstimacionRTT} + 4\text{DevRTT} \\ \text{EstimacionRTT} &= (1-\alpha)\text{EstimacionRTT} + \alpha\text{MuestraRTT} \\ \text{DevRTT} &= (1-\beta)\text{DevRTT} + \beta|\text{MuestraRTT} - \text{EstimacionRTT}| \end{aligned}$$

- $EstimacionRTT$ se calcula con segmentos transmitidos y confirmados (sin tener que ser retransmitidos).
- $DevRTT$ es una medida de la variación del RTT.
- En general, $\alpha = 0'125$ y $\beta = 0'25$.

- Cuando se produce un vencimiento del temporizador, además de retransmitir el segmento, **se duplica el tiempo de temporizador**.

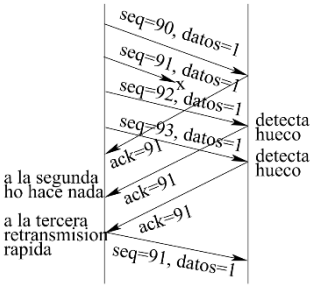
PÉRDIDAS

TCP distingue dos tipos de eventos de pérdida:

- **Vencimiento del temporizador** → se considera un problema grave, pues se han perdido tanto los segmentos como los ACKs.
- **ACKs triplicados** → se considera un problema leve, puesto que al menos han llegado los ACKs.

RETRANSMISIÓN RÁPIDA

- 3 ACKs duplicados se interpretan como un NAK (en total, se recibe 4 veces el mismo ACK).
  - Cuando el receptor recibe un segmento con un número mayor al esperado, envía un ACK duplicado.
    El emisor no hace nada hasta que recibe 3 ACKS duplicados, momento en el que reenvía el segmento solicitado.
  - Así se evita tener que **esperar al fin del temporizador** para retransmitir paquetes.



CONTROL DE FLUJO

- El CONTROL DE FLUJO es el mecanismo que permite al receptor indica al emisor el ritmo en el cual puede recibir datos.

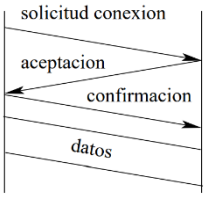
1. En el momento de la conexión el receptor le indica al emisor el **tamaño** de su **ventana receptora** en bytes.
2. El emisor fija el **tamaño** de su **ventana de envío** a este valor.

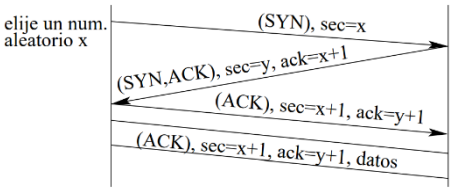
- El tamaño de la ventana **se puede modificar en cada transmisión**.

CONEXIÓN

- La conexión se realiza **en 3 fases**:
  1. **Solicitud** de conexión → el emisor escoge un número aleatorio x.
  2. **Aceptación** de conexión → El receptor escoge un número aleatorio y y acepta el número del emisor enviando el ACK $x+1$.
  3. **Confirmación** de conexión → el emisor acepta el número del receptor enviando el ACK $y+1$.
     - En esta fase ya se pueden comenzar a enviar datos.

- Estos segmentos de conexión son segmentos normales que tienen activados **bits especiales de cabecera**:
  - SYN se activa cuando se envía por primera vez x o y.
    - Se consume un número de secuencia.
  - ACK se activa cuando se confirma algún segmento.

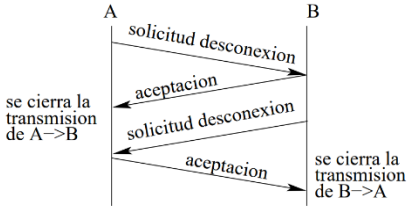


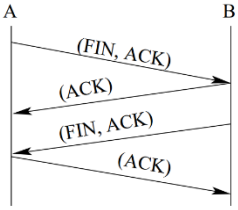


DESCONEXIÓN

- La desconexión se realiza **en 2 fases**:
  1. Desconexión de la transmisión en un sentido y aceptación de esta.
  2. Desconexión de la transmisión en otro sentido y aceptación de esta.
     - Después de que se cierre la transmisión en un sentido, aún se pueden seguir enviando datos en el otro.

- Estos segmentos de desconexión son segmentos normales que tienen activados **bits especiales de cabecera**:
  - FIN se activa en la solicitud de conexión.
    - Se consume un número de secuencia ¿?
  - ACK se activa cuando se acepta una desconexión.



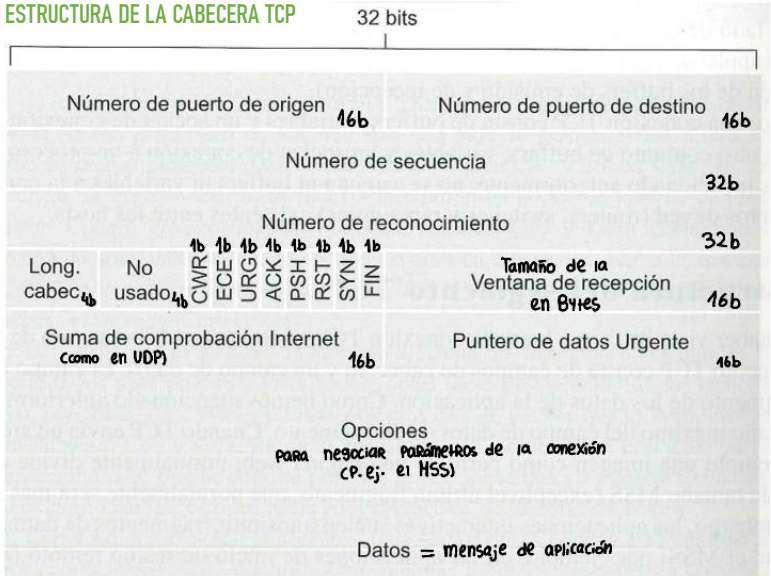


MECANISMOS PARA DISPARAR LA TRANSMISIÓN DE UN SEGMENTO

El socket va acumulando los mensajes que le envía la aplicación para que la capa de transporte forme con ellos segmentos para enviar a la capa de red.
Eventos que disparan la división de los datos acumulados en segmentos y su transmisión:

- **Segmentación** → sucede cuando el número de bytes en el socket supera el MSS.
- **Push** → sucede cuando la aplicación fuerza el envío.
  - Se activa el bit PSH en la cabecera del segmento para indicarle al receptor que debe entregar los datos inmediatamente a la aplicación.
- **Temporizador** → sucede cuando un temporizador llega a 0.

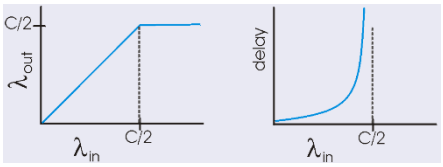
ESTRUCTURA DE LA CABECERA TCP



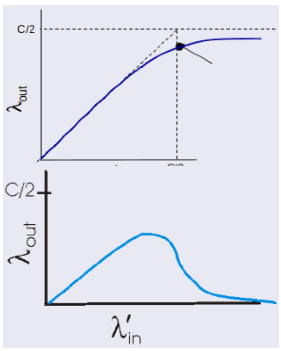
CONTROL DE CONGESTIÓN

- Que una red sufra de CONGESTIÓN significa que hay demasiados paquetes en ella, lo cual causa $\begin{cases}\text{retardos en las transmisiones}\\ \text{muchos paquetes perdidos}\end{cases}$.
  - ↳ Se suele producir por el **desbordamiento de las memorias de los routers**.
- El CONTROL DE CONGESTIÓN son los esfuerzos realizados por los elementos de la red para prevenir o responder ante situaciones de congestión.
  - Puede realizarse de dos formas distintas $\begin{cases}\text{prerreservando recursos para evitar la congestión.}\\ \text{dejando que la congestión ocurra y resolviéndola entonces (que es lo que hace TCP/IP).}\end{cases}$
  - En TCP/IP el control de congestión recae principalmente en la capa de transporte (TCP).

ORIGEN DE LA CONGESTIÓN



- Escenario 1 → dos emisores, un router con capacidad infinita y un enlace compartido de velocidad C.
  - Tasa de transmisión entre 0 y $C/2$ → todo se recibe bien y con retardo finito.
  - Tasa de transmisión mayor que $C/2$ → el enlace no puede proporcionar paquetes a esa velocidad, por lo que los paquetes se van acumulando en la cola del router y aumenta el retardo.
    - ↳ El retardo producido en el envío de un paquete origina **retardos acumulativos en los siguientes**.
- Escenario 2 → dos emisores, un router con capacidad finita y un enlace compartido de velocidad C.
  - Tasa de transmisión entre 0 y $C/2$ → todo se recibe bien y con retardo finito.
  - Tasa de transmisión mayor que $C/2$ → el router tendrá que descartar paquetes cuando la memoria esté llena, así que la tasa entregada disminuye porque algunos de los paquetes enviados son duplicados



- Escenario 3 → varios emisores, varios router con capacidad finita y varios enlaces.
  - Tasa de transmisión baja → todo se recibe bien y con retardo finito.
  - Tasa de transmisión elevada → la tasa entregada disminuye porque los buffers de los routers se llenan.
    - ↳ Los routers tendrán que descartar paquetes cuando la memoria esté llena.
      Cuando se descarta un paquete a mitad de camino el trabajo realizado por los routers anteriores se pierde, por lo que **la tasa entregada tiende a 0**.

▷ λ_{out} es la tasa entregada, λ_{in} es la tasa ofrecida y λ'_{in} es la tasa ofrecida con datos originales y retransmitidos.

CONTROL DE CONGESTIÓN EN TCP

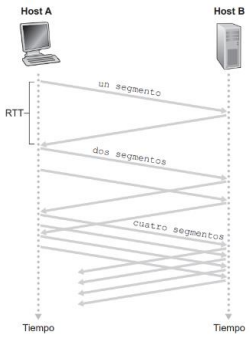
- El mecanismo utilizado por TCP para el control de la congestión consiste en que el emisor **adecúe su tasa de envío en función de la congestión de red** que percibe: disminuye su tasa de envío cuando nota que hay congestión y la aumenta cuando nota que no la hay.
  - ↳ El emisor considera que hay congestión cuando o bien **expira un temporizador** o bien se reciben **3 ACKs duplicados**.

PROCEDIMIENTO DE AJUSTE DE TASA DE ENVÍO

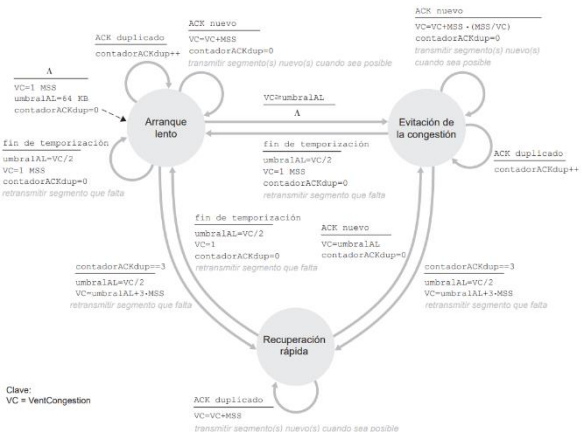
- TCP define periódicamente en el emisor la VENTANA DE CONGESTIÓN (vc) y el RTT para poder actualizar la tasa de envío:
  - El **RTT** es estimado periódicamente por el emisor midiendo el tiempo que pasa entre que envía un paquete y recibe su ACK.
  - La **ventana de congestión** se actualiza periódicamente siguiendo un algoritmo.
  - La tasa de envío se calcula como $tasa\ de\ envío = \frac{ventana\ de\ congestión}{RTT}$ (bytes/segundo).

ALGORITMO PARA ACTUALIZAR LA VENTANA DE CONGESTIÓN

A lo largo de la conexión, TCP considera 3 fases:



- **Arranque lento** → determina la capacidad inicial de la red.
  1. Se establece la vc al valor más pequeño posible, 1 MSS.
  2. Se va aumentando la vc exponencialmente hasta que se produce una pérdida.
     - ↳ Este aumento exponencial consiste en incrementar la vc en 1 MSS cada vez que se recibe un ACK pertinente.
     - Si se reciben 3 ACKs duplicador ir a recuperación rápida.
     - Si se produce un timeout volver a empezar el arranque lento y continuar hasta la mitad del valor de vc cuando se produjo el timeout. Luego ir a AIMD.
- **Incremento aditivo/Decremento multiplicativo** (AIMD) → situación normal.
  1. Al entrar en esta etapa, vc se reduce a la mitad (decremento multiplicativo).
  2. El emisor va sumando 1 MSS a vc cada RTT (aumento aditivo).
     - Si se reciben 3 ACKs duplicados ir a recuperación rápida.
     - Si se produce un timeout ir a arranque lento.
- **Recuperación rápida** → evita la fase de inicio lento cuando hay congestión.
  1. Se le suma a vc 1 MSS por cada ACK duplicado recibido correspondiente al segmento que falta.
  2. Cuando llega un ACK para el segmento que falta, vuelve a AIMD.
     - Si llega el ACK del segmento que faltaba ir a AIMD.
     - Si se produce un timeout ir a arranque lento.



NOTIFICACIÓN EXPLÍCITA DE CONGESTIÓN

- Cuando un **router** se encuentra congestionado **activa los bits** $\boldsymbol{ECN}$ (notificación explícita de congestión) **de la cabecera IP** de los paquetes que tramita.
  - ↳ Se recomienda que sólo haga esto cuando tiene una **congestión persistente**.
- El **receptor** de esos paquetes observa que se activó ECN y en consecuencia **activa el bit** $\boldsymbol{ECE}$ (eco de congestión encontrado) de la cabecera TCP en el ACK del segmento.
- Cuando el **emisor** recibe estos ACK, observa que se activó ECE y en consecuencia **ajusta la ventana de congestión** y **activa el bit** $\boldsymbol{CWR}$ (ventana de congestión reducida) de la cabecera TCP del siguiente segmento.

IMPARCIALIDAD

Considerando 2 aplicaciones que comparten un enlace de capacidad limitada, observemos las siguientes posibilidades:

- Las **dos** aplicaciones usan una conexión **TCP** → se reparten a medias la capacidad del enlace.
- Una aplicación usa una conexión **TCP** y la otra una conexión **UDP** → la conexión UDP, al no tener control de congestión, acaparará la mayor parte de la capacidad del enlace.
- Una aplicación usa **una** conexión **TCP** y la otra **9** conexiones **TCP** paralelas → la primera recibe $1/10$ de la capacidad del enlace y la segunda un $9/10$.