Redes Tema 3. Capa de transporte

Martín González Dios

31 de octubre de 2024

En el **origen** prepara los mensajes de las aplicaciones para ser transmitidos y en el **destino** recupera los mensajes y los entrega a las aplicaciones. Solo está implementada (la capa de transporte) en los sistemas finales.

En TCP/IP prepara los mensajes para transmitirlos por un canal no fiable (red de datagramas, IP, servicio de mejor esfuerzo) usando los protocolos de transporte **TCP** (fragmenta los mensajes en segmentos a los que les añade su cabecera) o **UDP** (solo añade la cabecera).

Los mensajes pasan de la capa de aplicación a la de transporte a través de **sockets**. Los procesos escriben y leen del socket, la capa de transporte recorre todos los sockets abiertos, procesa los mensajes y los envía a la capa de red (**multiplexación**) y recoge los segmentos de la capa de red para reconstruir los mensajes y colocarlos en su socket destino (**demultiplexación**), el cual se identifica por los números de puerto de la cabecera del segmento (int 16 bits).

En sockets sin conexión (**UDP**) este se identifica por la IP y el puerto destino, haciendo que segmentos de distinto host con mismo puerto destino sean recogidos por el mismo proceso.

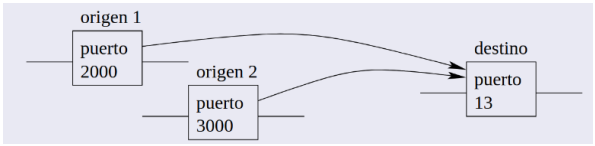

Figura 1: Multiplexación con sockets sin conexión (UDP)

En sockets orientados a conexión (**TCP**) el socket se identifica por la tupla IP origen, IP destino, puerto origen y puerto destino, por lo que segmentos de distinto host o puerto de origen con igual puerto destino irán a sockets distintos y pueden ser atendidos por procesos/hilos distintos. En TCP hay 1 socket servidor (espera conexiones de clientes) y varios de conexión (transmiten los datos).

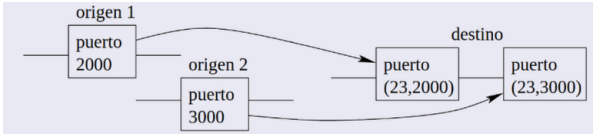

Figura 2: Multiplexación con sockets orientados a conexión (en TCP)

1. UDP(Protocolo de datagramas de usuario)

Es **simple** y **poco sofisticado**, hace casi lo mínimo (multiplexación/demultiplexación y comprobación de errores).

En origen **añade** una **cabecera de 4 campos** (puerto origen, puerto destino, bytes del segmento: cabecera + datos y suma de comprobación) al mensaje **formando un segmento**. En destino comprueba si el paquete llegó sin errores para entregarlo o no.

Es un **protocolo sin conexión** (sin acuerdo previo emisor-receptor → no hay retardo estableciendo la conexión y sin retransmisión en caso de error) y **sin estado** (cada segmento es independiente, no hay segmentación ya que no hay información para reconstruir los mensajes, lo que hace más rápido el procesamiento y usa menos recursos). Usa **cabeceras más pequeñas** y controla más la aplicación.

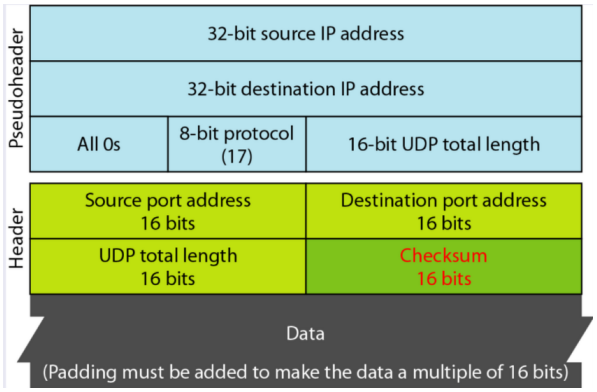

Figura 3: Estructura de UDP

2. Transmisión fiable

Se basa en los protocolos de **retransmisión de paquetes con errores**. Usan protocolos **ARQ** (Automatic Repeat reQuest) que pueden **pasar y esperar** (se envía un paquete y se espera la confirmación) o de **ventana deslizante** (se envían varios paquetes antes de la confirmación). Se consideran **enlaces bidireccionales** (full duplex) y se numeran los paquetes de 0 a $2^n - 1$ (n bits).

2.1. Protocolo ARQ parar y esperar

- Sin errores: receptor devuelve ACK con el no del siguiente paquete.
- Pérdida de paquetes/paquete con errores: receptor no devuelve ACK, timeout y retransmisión.
- Pérdida ACK: timeout y retransmisión, el receptor recibe un duplicado que descarta y envía ACK.
- Timeout: si paquetes y ACKs llegan con retraso se reenvían ignorándolos por duplicados.

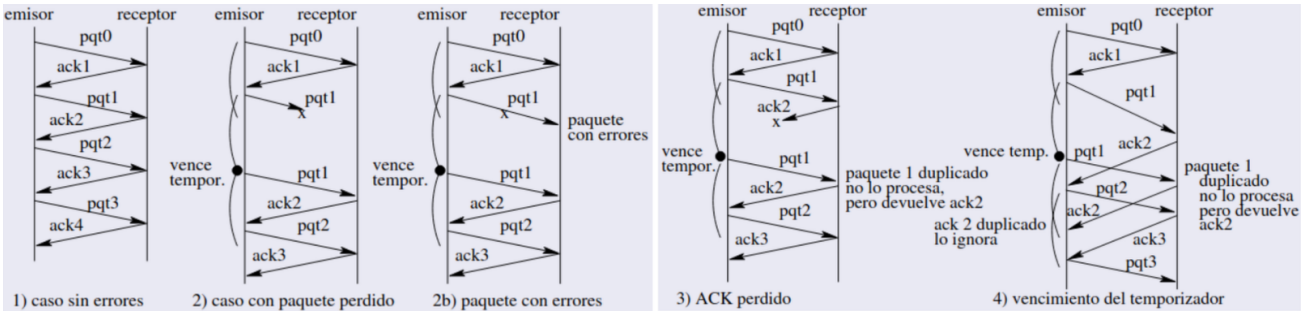

Figura 4: Casos ARQ parar y esperar

Hay **variantes** como **NAK** que indica la recepción de un paquete con errores y no se espera timeout. Otra es que 2 ACKs iguales equivalgan a un NAK: un paquete con errores devuelve el ACK del último paquete correcto. O que 3 ACKs equivalgan a un NAK, resolviendo ciertos conflictos.

El principal **inconveniente** de este protocolo es la **poca utilización del enlace**.

2.2. Protocolo ARQ de ventana deslizante

El emisor envía N paquetes antes de recibir los ACKs.
Utilización del enlace por parte del **emisor**: $U = \frac{t_{trans}}{RTT+t_{trans}}$
Tiempo útil en el emisor: $N * t_{trans}$
Tiempo total: $t_{trans} + RTT$
U = 1 si $N * t_{trans} \geq t_{trans} + RTT \rightarrow N \geq 1 + RTT/t_{trans}$

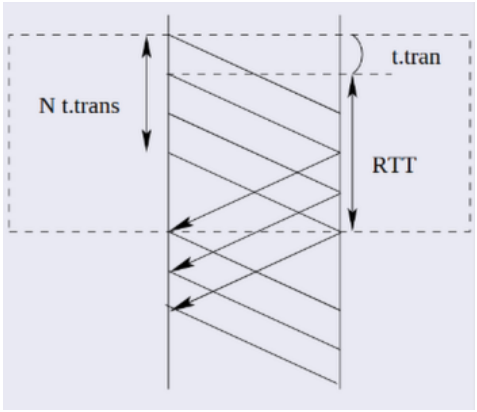

Figura 5: Utilización del enlace

Se requiere que el **rango de números de secuencia** abarque al menos el **doble del tamaño de la ventana emisora** (conjunto de N paquetes que el emisor puede enviar o que están pendientes de confirmación) y que **emisor y receptor** puedan **almacenar más de un paquete**.

La **ventana receptora** es el conjunto de N paquetes que el receptor puede aceptar o está procesando.

- **Go Back N** (Retroceder N): el receptor **sólo acepta paquetes en orden**. Si un paquete llega con errores o no llega se descartan los siguientes y si expira un timeout se retransmiten ese y los siguientes. Los ACKs implican a todos los paquetes previos (acumulativos).
- **Repetición selectiva**: el receptor **acepta paquetes fuera de orden**, retransmitiendo solo los erróneos o los que no llegan. Se debe enviar el ACK de cada paquete recibido.

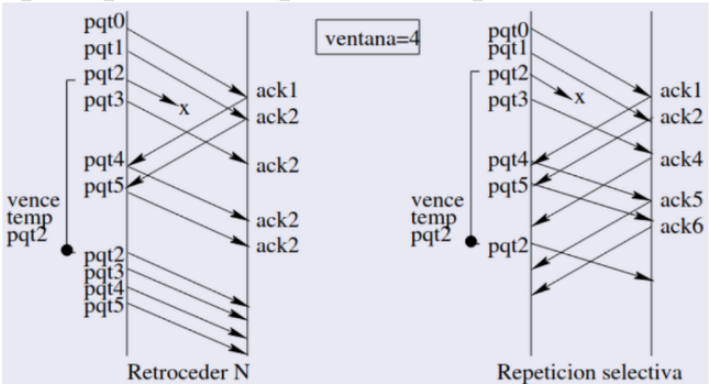

Figura 6: Retroceder N y repetición selectiva

3. TCP(Protocolo de control de transmisión)

Aplica los principios de **transmisión fiable** y usa números de secuencia (32 bits) que **identifican bytes, no segmentos**. Empiezan por un número aleatorio x que incrementa en x + n bytes. **Los ACKs indican el siguiente byte a recibir** y también pueden transmitirse con datos (superposición o piggybacking). Hay un tiempo máximo de espera para mandar datos, si no envía solo el ACK.

En este protocolo el emisor usa **temporizadores para la transmisión**, que se recomienda que sea único (al llegar un ACK o retransmitir un segmento se reinicia). **Usa ventana deslizante, ACKs acumulativos y solo retransmite segmentos no confirmados**.

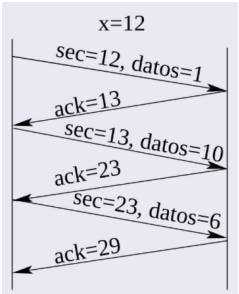

Figura 7: Números de secuencia

La estimación del tiempo de espera sigue las siguientes **fórmulas**:
Temporizador = EstimacionRTT + 4DevRTT
EstimacionRTT = (1-α)EstimacionRTT + α MuestraRTT
DevRTT = (1-β)DevRTT + β |MuestraRTT + EstimacionRTT|
DevRTT es una medida de la variación del RTT, y en general α=0.125 y β=0.25
La estimación del RTT es en base a segmentos transmitidos y no confirmados (no retransmitidos)
*Al expirar el temporizador, el emisor duplica el tiempo de espera.

Cuando el **receptor** recibe un segmento con mayor número del esperado **envía un ACK duplicado** del último segmento correctamente recibido. Al recibir **3 ACKs duplicados** (el cuarto) se interpreta como un **NAK** y se realiza una retransmisión rápida. En este caso el problema es **leve**, **si expira el temporizador** se pierden los segmentos y los ACKs y se vuelve **grave**.

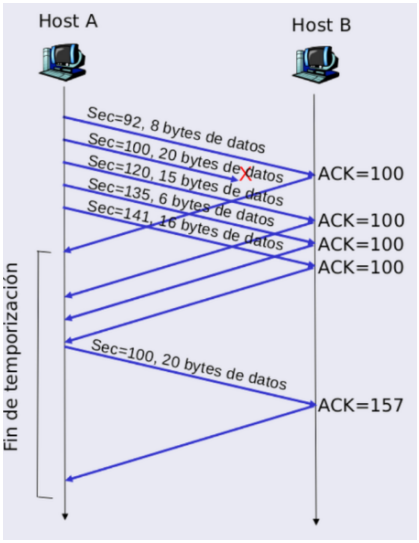

Figura 8: Retransmisión rápida

El TCP tiene **control de flujo** que indica el **ritmo al que puede recibir datos**. El receptor indica el tamaño de su ventana de recepción, el emisor fija su ventana de envío a ese valor para lo que existe un campo en la cabecera TCP. **El tamaño de la ventana puede modificarse en cada envío**.

Conexión: acuerdo en **tres fases**:

- SYN=1, cuando se envía por primera vez x o y (se consume un número de secuencia)
- ACK=1, cuando se confirma un segmento
- En la tercera fase ya se pueden enviar datos

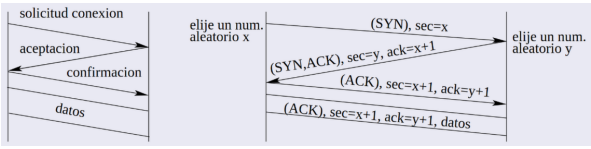

Figura 9: Conexión

Desconexión: en **dos fases**: cada una para desconectar la transmisión en un sentido. Se podría desconectar en un sentido y seguir transmitiendo en el otro.
FIN=1, solicitud de desconexión (se consume un número de secuencia) ACK=1, aceptación

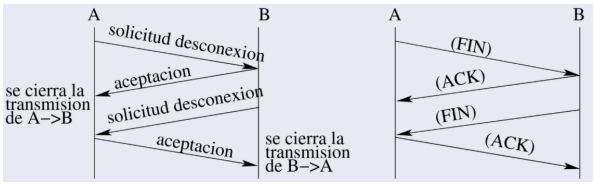

Figura 10: Desconexión

Una vez establecida la conexión comienza la **transmisión**. La aplicación pasa los datos a la capa de transporte (escribe en el socket) que los va acumulando. Cuando el número de bytes supera el **MSS** (maximun segment size), la aplicación fuerza el envío (activa el flag PSH) o un temporizador llega a 0, se dispara la transmisión de un segmento (TPC genera el segmento y se lo pasa a IP).

El envío A $\rightarrow$ B también **debe incluir el ACK** (piggybacking).

- Si el receptor recibe un segmento en orden, el anterior fue confirmado y no tiene datos para enviar, retrasa el ACK hasta recibir otro segmento o hasta que transcurra cierto tiempo.
- Si llega un segmento esperado y no se confirmó el anterior se envía el ACK.
- Si el segmento que llega tiene un no de secuencia mayor del esperado se envía un ACK con el número de secuencia esperado.
- Si llega un segmento que faltaba se envía un ACK con el siguiente esperado (acumulativo).
- Si llega un segmento duplicado se descarta y se envía un ACK con el esperado.

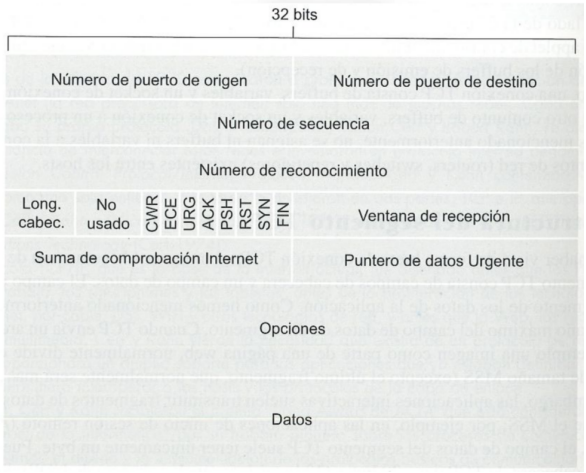

Figura 11: Cabecera TCP

4. Control de congestión

Los recursos de la red deben repartirse entre las **peticiones**. Si hay demasiados paquetes en la red se producen **retardos en las transmisiones** y se **pierden paquetes**. Esto se produce usualmente por el **desbordamiento de la memoria de los routers**. Ante estas congestiones los elementos de la red realizan esfuerzos como pre-reservar recursos para evitarla o dejar que ocurra y resolverla después. Las **congestiones** pueden originarse por **diversas causas**:

- **2 emisores, 1 router con capacidad infinita y enlace compartido de velocidad C**. Mientras la tasa de entrada λ_{in} sea $\leq$ C/2 la tasa de salida λ_{out} será igual a la de entrada.
  Si $\lambda_{in} >$ C/2 el enlace no puede responder, los paquetes se acumulan en la cola del router y aumenta el retardo.
- **2 emisores, 1 router con memoria finita y enlace compartido de velocidad C**. Siendo λ'_{in} la carga ofrecida al enlace con datos originales y retransmitidos, mientras sea $\leq$ C/2 el router puede manejar la carga. Si excede C/2 la memoria del router no llega, causando pérdidas y aumentando el retardo por la retransmisión de paquetes.
- **Varios emisores, routers de memoria finita y varios enlaces**. Mientras la tasa de transmisión es baja los paquetes se envían y procesan correctamente con un retardo finito. Si aumenta demasiado los buffers de los routers se llenan, perdiendo paquetes y disminuyendo la tasa de entrega efectiva λ_{out} que puede llegar a 0 por la saturación.

Por esto se necesitan **mecanismos de control de congestión**, que en TCP/IP están principalmente en TCP. Estos se basan en la **adecuación** de la **tasa de envío** del **emisor según la congestión** que percibe (considera congestión al expirar un temporizador o recibir 3 ACKs duplicados).
En el emisor se definen la **ventana de congestión** (se actualiza según el caso) y el **RTT** (se estima periódicamente calculando el tiempo desde el envío de un segmento hasta su ACK).
Tasa de envío = $\frac{ventanaDeCongestión}{RTT}$(bytes/segundo)

Para **actualizar la ventana de congestión** se determina la capacidad inicial de la red con el **inicio lento** (comienza con una tasa de envío conservadora que va aumentando exponencialmente). En cierto punto se aplica el **AIMD** (incremento aditivo/decremento multiplicativo) que va aumentando o disminuyendo la ventana para mantener la tasa controlada. En caso de congestión se aplica la **recuperación rápida**, que reduce la ventana sin llegar al inicio lento.

Existe también la **notificación explícita de congestión** (**ECN**). Se da cuando un router congestionado activa unos bits en la cabecera IP, haciendo que el receptor TCP active el bit ECE (Eco de ECN) y que el emisor reduzca la ventana de congestión y active el bit CWR.

La compartición de un enlace de capacidad limitada no es imparcial.

- Dos conexiones TCP irán repartiendo la capacidad del enlace de forma variable.
- Entre una conexión TCP y una UDP, la UDP acaparará la mayor parte de la capacidad.
- Entre una conexión TCP y 9 conexiones TCP paralelas, la primera usará 1/10 y la segunda 9/10.