Capa de transporte

1. Servicio de la capa de transporte

La capa de transporte es la que facilita la comunicación lógica entre procesos ubicados en diferentes hosts. Principalmente, en el lado del emisor se "trocea" el mensaje en segmentos y en el lado del receptor se reensamblan los segmentos y se interpreta la lógica.

Los principales protocolos del nivel de transporte son **TCP** y **UDP**.

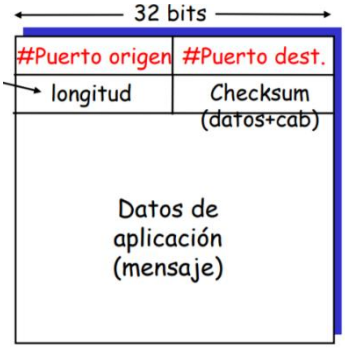
TCP es un protocolo encargado de enviar paquetes de manera ordenada, mientras que UDP es un protocolo encargado de enviar paquetes de manera desordenada.

2. Multiplexacion y demultiplexación

La **multiplexación** se encarga de encapsular el tráfico convenientemente y enviarlo al nivel de red.

La **demultiplexación** se encarga de distribuir los elementos recibidos al socket correcto.

3. UDP – User Datagram Protocol



UDP es un servicio "Best effort" donde los segmentos UDP pueden ser perdidos o entregados en desorden.

Se usa en aplicaciones multimedia las cuales son tolerantes a perdidas, DNS…

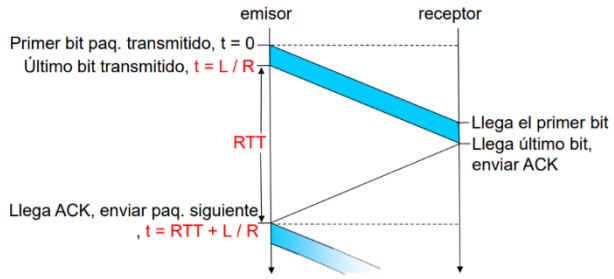
UDP es un protocolo simple que envía un paquete de A a B especificando el puerto de entrada y salida para la multiplexación. Se denomina a UDP como un *conectionless protocol*, debido a que no hay un handshake antes de enviar cualquier paquete, UDP envía los paquetes sin "preguntar".

4. Transferencia fiable

Para poder tener una transferencia de datos fiable son necesarios los siguientes elementos:

- Un *checksum* para detectar errores.
- Temporizadores y numeradores para detectar pérdidas.

Existe un mecanismo **llamada** y **espera** (pkt y ack) con el cual se pueden detectar las perdidas. Este mecanismo tiene un rendimiento pobre.



$$tiempo_{transmision} = \frac{tamaño\ a\ transmitir(L)}{ancho\ de\ banda(R)}$$

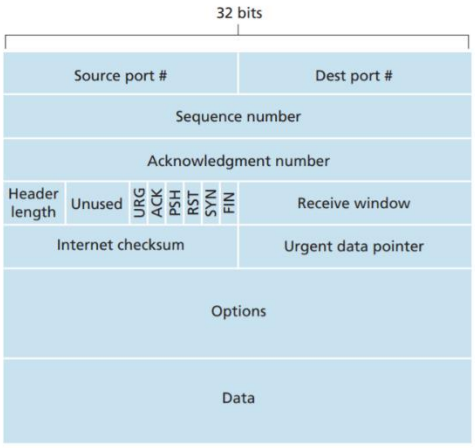
$$Tiempo_{trans.emisor} = \frac{\frac{tamaño\ a\ transmitir}{ancho\ de\ banda}}{RTT + \frac{tamaño\ a\ transmitir}{ancho\ de\ banda}}$$

El protocolo de transporte limita el uso de los recursos de la red.

El mecanismo consiste en cada vez que se envía un paquete, el receptor tiene que enviar una respuesta, dependiendo de si hay respuesta o no, se reenviara el paquete u otra cosa.

Hay otro mecanismo denominado **Pipelined**, aquí se permite que el emisor envíe múltiples paquetes antes de recibir los ACK. Para esto se requiere un buffer en ambos extremos (emisor y receptor).

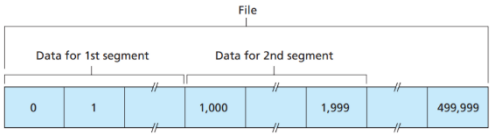
5. TCP



Los segmentos deben tener una estructura:

- Punto a punto, solo hay un emisor y solo hay un receptor.
- Confiable, hay un flujo de bytes ordenados.
- Pipelined, tiene que haber buffers en ambos extremos.
- Full duplex, flujos bidireccionales en la misma conexión con un MSS (máximum segment size).

Los paquetes tienen un **número de secuencia** que hacen referencia al flujo de bytes transmitidos. El número corresponde a el número del primer byte del segmento dentro del flujo. Estos corresponderían a los paquetes enviados, en el caso de haberlos recibido esta el **número de reconocimiento** (ACK)



TCP también usa un mecanismo de **timeout** para reconocer perdidas, pero ¿Cómo fijamos el valor de timeout en TCP?

- Podemos hacerlo mayor que el RTT, pero tenemos que tener en cuenta que el RTT varía.
- Podemos hacerlo pequeño, pero en este caso se producirán retransmisiones innecesarias.
- Podemos hacerlo largo, pero esto producirá una reacción lenta a la pérdida de paquetes.

La solución sería la de tener un RTT de muestra el cual sería algo parecido a una media. Para esto se mide el tiempo desde la transmisión de un segmento hasta que se recibe el ACK.

TCP da un servicio de transferencia de datos **fiable** ya que usa un temporizador de transmisión y las retransmisiones suceden cuando se cumple el temporizador o hay reconocimientos duplicados.

El temporizador puede resultar lento dependiendo de la situación, por esta razón existe la **retransmisión rápida**, la cual en vez de esperar a que se pase el *timeout*, envía 3 ACK's duplicados para los mismos datos, los ACK's corresponderían a el paquete anterior al que se ha perdido.

Tampoco podemos permitir que el emisor envíe paquetes demasiado rápido ya que entonces se llenará el buffer de recepción. Es por esta razón que TCP tiene un servicio de adaptación de la velocidad que se adapta a la tasa de lectura de la aplicación. El funcionamiento de esto es simple, el receptor informa del espacio libre incluyendo el valor del segmento, el emisor limita el volumen de datos enviados, pero no confirmados a este tamaño de ventana.

Es necesaria una **gestión de la conexión** ya que el emisor debe establecer la conexión antes de intercambiar segmentos con datos. Para esto TCP tiene un acuerdo en 3 fases:

- El cliente envía un segmento TCP SYN al receptor, en este paquete se especifica el número de secuencia inicial, y está vacío.
- El servidor recibe el paquete TCP SYN y responde con un SYNACK
- El cliente recibe el SYNACK y responde con un segmento ACK que ya puede contener datos.

También es necesario un protocolo para cerrar la conexión:

- El cliente envía un segmento TCP con la bandera FIN.
- El servidor recibe FIN, y responde con un ACK y con otro paquete FIN.