Resumen capa de transporte

miércoles, 6 de noviembre de 2024 09:52

IP es el protocolo de la capa de red que da servicios a la capa de transporte.
IP es débil, no se ocupa de que no se caguen los datos. Solo están las direcciones fuente y destino. Los protocoleos de transporte corren ahí, en los end-points.

Función: Se encarga de permitir la comunicación lógica entre procesos de aplicaciones. Permite que los hosts puedan comunicarse directamente sin tener que preocuparte por como funca.

En el lado del emisor la capa de transporte convierte los mensajes de la aplicqación en **SEGMENTOS** de la capa de transporte que después se encapsulatn en **PAQUETES** de la capa de red y se mandan. Los routers de la capa de red solamente se fijan los campos que correspoonden. Los datos van encapstulados. El receptor hace el desencapsulamiento a la inversa y le manda los datos a la app receptora..

Cada app usa un puerto distinto para poder distinguir quién manda y recibe los datos. Con esto se pued3e hacer que nmuchas apps en una misma compu manden y reciban cositas a la vez.

Sockets: son interfaces para que las apps se conecten a la red. La capa de transporte le da los datos al socket y no a la app. Tienen un id único. Cada segmento sabe a que id va y una vez que llega al receptor la capa de transporte lo manda al que corresponde.

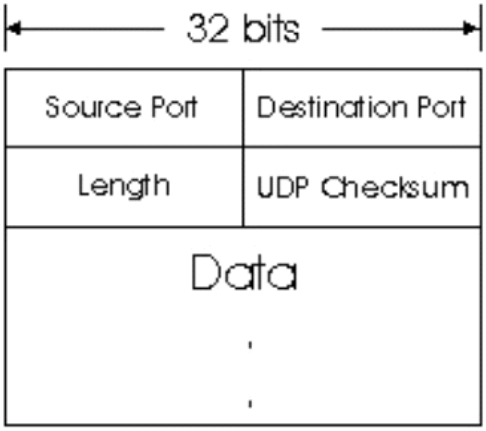
- Multiplexación:
  - Recoger datos de diferentes sockets en el host origen.
  - Encapsular los datos con info de encabezado necesaria para mandarla y después poder identificarla en la capa de transporte.
  - Una vez encapsulados los segmentos pasan a la capa de red para transmitirlos

  EN CRIOLLO ES JUNTAR DATOS DE VARIOS SOCKETS Y MANDARLOS

- Demultiplexación:
  - Es lo que se lleba a cabo en la punta que recibe.
  - Capa de transporte se fija los campos de los segmentos que entran para ver a qué socket van.
  - Hace que el segmento se mande al proceso de la app que corresponda

UDP User datagram protocol: PUERTO 17

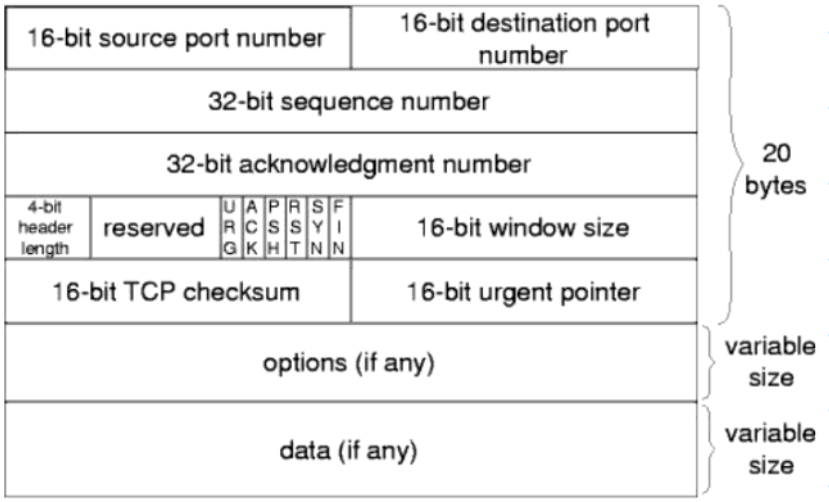
- Menos overhead, best- effort. Pueden llegar los datos pero se pueden perder
- Es no confiable y minimalista
- Mensajes--> **datagramas** son autocontenidos. No tienen un tamaño arbitrario
- Solo provee multiplexación/ demultiplexación
- Tiene básicamente set reset. Mandá estos datos, recibí estos datos.
- Se usa en aplicaciones ágiles. Video, streaming voz
- **No requiere que se establezca la conexión**



- **Length: Longitud del datagrama incluyendo el encabezado. Se usa para que el receptor sepa cuánta info tiene que procesar en el datagrama**
- **Checksum: Es para detectar errores de transmisión. Se calcula en base al contenido del datagrama (+header). El receptor se fija en este valor para ver si llegó bien o no.**
  - **Se puede deshabilitar, detecta errores simples.**
  - **Acá iria cómo se calcula el checksum pero finjamos demencia**

TCP transport control protocol: PUERTO 6

- **Confiable, tiene control de flujo y congestión. Mejor uso de la red**
- **Orientado a streams. Puedo mandar tamaños de datos arbitrarios**
- **Cuando mando algo llega en secuencia ordenada**
- **Se mandan Segmentos. Se segmentan las secuencias de bytes.**
- **+ overhead**
- **Hay timers, RTO**
- **Hay que establecer y cerrrar la conexión**



- **32-bit sequence number: Se usa para mantener un seguimiento del orden de los segmentos tcp. Cada segmento se etiqueta con un nro único.**
- **32-bit acknowledgment number: Indica el próximo número de secuencia que el emisor espera recibir. Este campo confirma la recepción de segmentos previos y ayuda a mantener la confiabilidad de la transmisión. Nro de byte que espera recibir.**
- **4-bit header length: marca la longitud del encabezado. Permite identificar dónde comienza la carga útil de datos en el segmento.**
- **Reserved: reservado para uso futuro???? Tiene que establecerse en 0. Asegurara la compatibilidad con posibles extensiones o mejoras del protocolo**

- Flags: bits usados para controlar y gestionar la comunicación. VER CUÁLES SON Y PARA QUE SIRVEN
- 16-bit window size: Marca el tamaño de la ventana de recepción que el receptor tiene para aceptar datos. Esto se negocia. No manda más del tamaño de ventana sin recibir el ack
- 16-bit TCP checksum: suma de verificación.
- 16-bit urgent pointer: Se usa en la comunicación para marcar la posición de datos urgentes (es opcional)
- Options: es para agregar data extra como mas segment size, ventana de escala, etc. Se usa para ajustar y optimizar la comunicación segúns es necesario

- TCP entrega y envía lo datos agrupados o separados de forma dis-asociada de la aplicación:
  - La aplicación puede enviar 300 bytes en un write y TCP lo podría enviar en 3 segmentos separados de 100 bytes c/u.
  - La aplicación puede enviar 100 bytes y luego otros 200 y TCP esperar para enviarlos todos juntos.
  - La aplicación puede intentar leer 200 bytes del buffer y TCP solo entregar 150 bytes y luego el resto.

Tamaño de la ventana de recepción es el máximo cantidad de datos de recepción que se pueden almacenar en el buffer en una conexión.
No es precdeterminado. Maximun segment size es el tamaño más grande de datos que puede tener un segmento sin ser fragmentado.

Mss determinado por MTU, o la unidad máxima de transmisión, que sí incluye los encabezados TCP e IP (Protocolo de Internet). El MSS es igual a la MTU menos el tamaño de un encabezado TCP y un encabezado IP: MTU – (encabezado TCP + encabezado IP) = MSS
Si un paquete supera la MTU de un dispositivo, se divide en trozos más pequeños, o "se fragmenta." En cambio, si un paquete supera el MSS, se descarta y no se entrega.

Establecimiento de conexión

3 way handhshake. Requiere que tanto el cliente como el servidor intercambien segmentos SYN y ACK antes de empezar la comunicación posta.

Paso 1: Cliente manda el SYN.

El cliente arranca la conexión mandando un segmento TCP al servidor con el bit SYN activado.
Le informa al servidor que quiere arrancar una conexión. El segmento tiene un número de secuencia inicial ISN que se usa para identificar y ordenar los segmentos de datos. Sirve para el seguimiento de la secuencia de bytes.

Paso 2: Servidor responde con SYN/ACK

Cuando el sv recibe el SYN del cliente contesta con SYN y ACK activos.
Se manda también un ISN que genera el servidor para establecer el punto de inicio de su secuencia de bytes.
El ack confirma que llegó el SYN del cliente y el ACK indica el prócimo número de secuencia que quiere recibir el sv.
En caso de que el sv no tenga un proceso en listen, manda un segmento RST para rechazar la conexión.

Paso 3: Cliente envia segmento ack

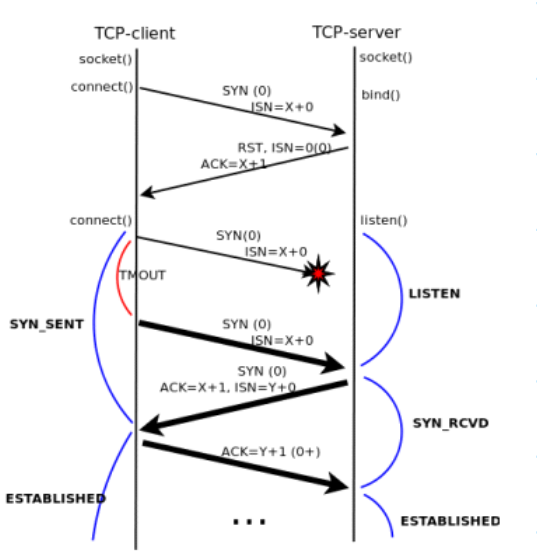
Cuando el cliente recibe el ACK/SYN del servidor, manda un segmento final con el bit ACK activado.

Confirma la recepción del ISN del servidor y el campo de numero de acuse(ack) se incrementa para indicar que el cliente está listo para recibir el siguiente segmento.

Ahí se establece la conexión y ambos extremos pueden arrancar a mandar datos.

Este proceso de tres pasos garantiza que tanto el cliente como el servidor estén sincronizados y que ambos hayan acordado los números de secuencia iniciales antes de iniciar la transferencia de datos. Esto contribuye a:

- **Confiabilidad:** Asegura que ambas partes sepan que la conexión es válida y están listas para recibir datos.
- **Control de flujo:** Al intercambiar números de secuencia y acuses, se puede gestionar el orden de los datos y la detección de pérdidas.



Cierre de Conexión (4-Way Handshake)

Se hace para aseguurarse de que ambas partes terminaron la transferencia de datos.

Paso 1: Envío del segmento FIN

O el cliente o el sv mandan unb segmento tcp con el segmento fin en 1 para avisar que terminó de mandar datos. El extremo que mandó el fin queda en FIN-WAIT-1

Paso 2: respuesta con ack

El otro extremo recibe el segmento FIN y manda un ACK de confirmación para avisar que recibió el FIN de la otra parte.
El iniciador del cierre pasa a FIN-WAIT-2 al recibir el ACK y el otro (el que mandó el ack ahora) queda en CLOSE-WAIT

Paso 3: envío del fin por parte del receptor.

Cuando el otro extremo termina de mandar datos, manda FIN al iniciador para decirle que ya mandó todo. El receptor queda en LAST-ACK esperando una confirmación del FIN. El iniciador del cierre pasa a Time_wait cuando recibe el fin.

Parte 4: ACK final del iniciador

El iniciador del cierre manda un ACK para marcar que recinió el fin del receptor.
El iniciador que da con TIME-WAIT durante un tiempo definido para asegurar que el ACK final llegue al receptor.
Después el iniciador pasa a CLOSED cuando termina el TIME-Wait.
El receptor pasa a CLOSED después de recibir el ACK final

CONTROL DE ERRORES

Algoritmo que ordena los segmentos fuera de orden y se recupera con solicitudes o retransmisiones
Se usa un timer RTO que lo que hace es que si el tiempo pasa hay que retransmitir el mensaje. Al mandajr un segmento inicia el timer y si se pasa se retransmite. Se va ajustando.

El RTT(Round Trip Time) se usa en TCP para ver cuánto tarda en ir y venir uyn paquete. El RTO se calucia cada vez que se manda un msj.

En tcp hay timestamps pero son opcionales, se usan para mantener más preciso el tiempo de ida y vuelta

Stop and wait:

- El emisor manda un paquete y espera confirmación antes de mandar el siguiente
- El emisor arranca timer tipo rto al mandar el segmento, si no recibe ack antes de que termine el timer vuelve a mandar el segmento
- Es ineficiente. No aprovecha el ancho de banda
- La ventana"" es 1, voy mandando de a un dato

Pipelining/sliding window

- Se pueden mandar muchos sin recibir confirmación, quedan in flight.
- El emisor tiene que saber cuántos segmentos puede mandar sin recibir confirmación----> VENTANA
- Se necesita buffer de envio y de recibimiento(tx lo que deja la capa superior y todavía no se confirmó y rx lo que se va recibiendo para entregar a la capa superior)
- Por cada mensaje enviado se arranca un RTO que se mantiene por ráfaga para el segmento más viejo que todavía no se confirmó
- Por cada confirmación se reinicia el rto. Si no se recibe confirmación se vence y se retransmite arrancando nuievo rto.

Go-back n

- hay una ventana estática
- No admite segmentos ni confirmaciones fuera de orden. Solo se confirman por la positiva de los segmentos que se colocaron en el buffer en orden.
- Se pueden confirmar desde N para atróas (ACK acumulativo). No se confirma necesariamente cada segmento
- Se re-transmite si hay timeout o un NAK. buffering extra en el emisor, no se pueden descartar del buffer de com. con la capa usuaria. Si se re-transmite ante un timeout se hace desde N hacia adelante, los que ya se enviaron.

Emisor:

- Si tienme datos en el TxBuf y la ventana lo permite, los transmite
- Si no tiene RTO activo arranca uno nuevo para el primero que manda
- Si recibe ACK en orden desliza la ventana. Si hay segmentos in flight arranca un nuevo rto y sino lo descarta.
- Si se vence el RTO re-envia todo a partir del segmento más viejo que nmo se confirmó.
- Puede haber más de un segmento "in-flight", más de un ACK "in-flight".

Receptor:

- Si recibe segmento en orden confirma indicando el próximo que espera

- Si el segmento está corrupto lo descarta y espera retransmisión
- Si recibe un segmento fuera de orden confirma indicando que espera uno anterior
    - Puede descartar el segmento fuera de orden, ya que ser´a retransmitido.
    - Puede bufferear el fuera de orden, pero no entregar a capa superior, esperando que llegue el/los segmento/s que llena/n el hueco y luego confirmarlo

En Go-Back-N, el emisor y el receptor utilizan una ventana deslizante. La ventana define el número máximo de paquetes que se pueden enviar antes de recibir un ACK (confirmación) del receptor.

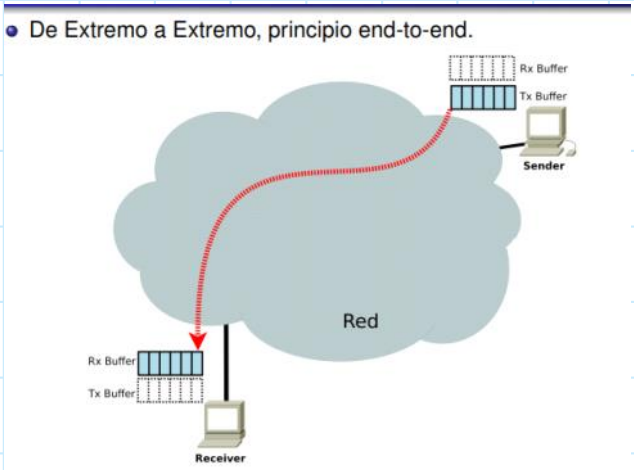
Selective repeat:

- Solo retransmite lo que se haya perdido(no confirmado)
- El receptopr confirma de a uno o usa intervalos de confirmación
- NO SE USAN CONFIRMACIONES ACUMULATIVAS
- No se deben confundir los segmentos de diferentes r´afagas. No se deben reusar #ID/SEQ hasta asegurarse que tiene todos los mensajes previos o estos no están en la red
- La ventana se desliza sin dejar huecos, desde los confirmados m´as viejos

Si uno se pierde el receptor puede seguir aceptando. El receptor almacena lo que recibe hasta que le mandan el que le falta

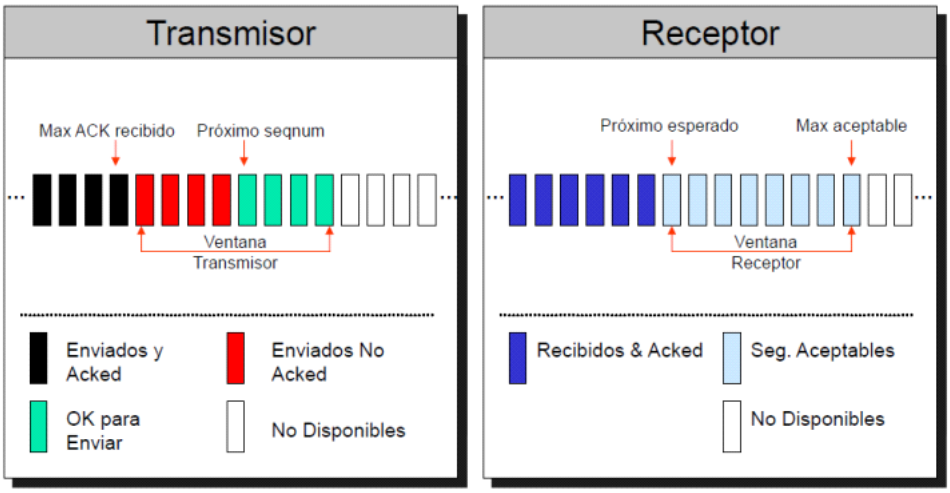
CONTROL DE FLUJO

Algoritmo que permite que el receptor maneje la tasa a la que manda datos el transmisor. Cuále s la capacidad a la que el sv le puede mandar datos al cliente para que el receptor llegue a recibuir y procesar losd atos.
La idea es prevenir que el emisor sobrecargue al receptor haciendo mal uso de la red.
Windows size: capacidad del receptor de recibir datos.
La capacidad de envio es el minimo entre(congestión, flujo, errores).



La idea es que el cliente pueda inducir el tamaño del buffer del cliente para saber cuánto le puede mandar
El receptor indica el espacio del buffer de recepción, Rx buffer, en el campo Advertised window.
Para cada segmento que manda indica el campo de windows. Cant de datos que puede mandar sin esperar recepción. Es el tamaño libre dispnible.
Cuando llega un segmento nuevo se pone en el buffer. Cuando app lee del buffer

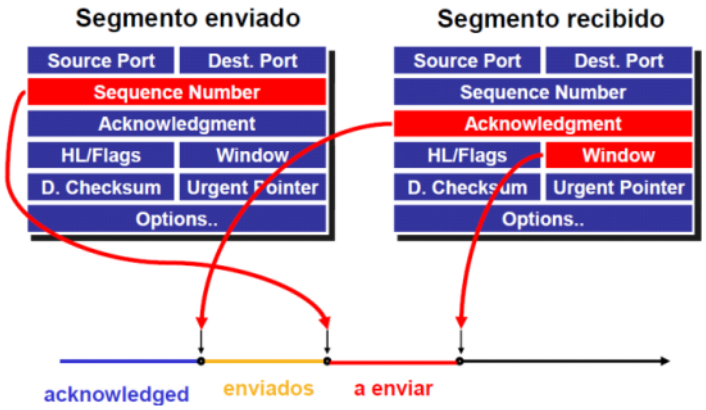
los datos se van del mismo. Si la aplicación no lee, se caga pq el buffer se queda sin espacio. Cuando va leyendo se cambia el tamaño de la ventana



Negro:lo que puedo descartar. Enviado y aknowleadge
Rojo: todavía no los puedo sacar pq no me los confirmaron
Verde: espacio que hay en el buffer del transmisor para recibir

Azul: recibido y aceptado pero capaz app no los leyó así que no los puedo sacar.
Celeste:segmentos aceptados. Próximo esperado y máximo aceptable

Cada vez que llega un ACK en orden se mueve la ventana en el Transmisor, se descartan segmento confirmado de Tx Buffer



Tamaño de ventana efectivo es tamaño de ventana – (menos) los que están en vuelo

El emisor va calculando cuando puede enviar. Básicamente calcula lo que puede seguir enviando sin recibir confirmación. El "control de flujo" se activa cuando se va achicando la ventana.

a. ¿Quién lo activa? ¿De qué forma lo hace?
lo activa el receptor enviando ventanas más chicas. Esto deja en evidencia que el receptor tiene poco espacio Esto se realiza a través del campo de tamaño de

ventana en los encabezados de los segmentos TCP.

b. ¿Qué problema resuelve?
saturación o congestión de los buffers en los endpoints. Al indicar al emisor que reduzca la cantidad de datos que está enviando, evita que el receptor se sobrecargue.

c. ¿Cuánto tiempo dura activo y qué situación lo desactiva?
Cuanto tiempo dura activo depende del receptor (más que nada la velocidad en que lee la aplicación). El control de flujo está activo mientras el receptor envíe ventanas más pequeñas (indicando capacidad limitada). Durará activo hasta que el receptor envíe ventanas más grandes

Gracias agus :)

Control de flujo resumen: no tiene que ver la red. Idea es no sobrecargar al receptor. Intenta regular tasa de transmisión entre receptor y transmisor medainte la tasa de ventana que se pasa al pasar cada segmento
En los routers no se fijan de esto. Routers no hablan tcp

Control de congestión

La idea es no saturar una red. Determinar la cantidad que puedo mandar sin congestionarla.
Intentar descubrir si hay algún router saturado. No puedo preguntarles a los routers, tengo indicios para saber cuál es la potencia a la que puedo mandar sin saturar

Tiene en cuenta el estado de la red a diferencia del control de flujo que se fija en el receptor nada más.

No puedo preguntarles a los routers, tengo indicios para saber cual es la polenta a la que puedo mandar sin saturar la red

¿Por qué hay problemas de congestión?

- Límite de la capacidad de la red:
    - Velocidad de los Routers/Switches (CPU).
    - Capacidad de los Buffers de los Routers/Switches (Memoria).
    - Velocidad de los Enlaces (Interfaces)
- Demasiado tráfico en la red

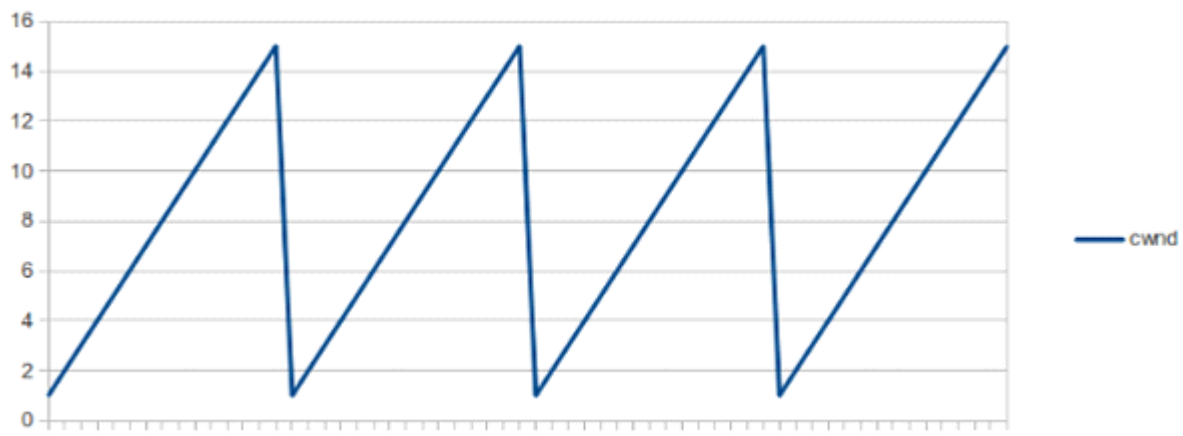
Modelo end to end:

No participa la red
Cwnd ventana de congestión.
Ssthresh Slow Start Threshold (Umbral).
Se calcula: MaxWin = Min(rwnd, cwnd). rwnd era la ventana de recepcion, usada para el control de flujo.

La ventana efectiva es la máxima regida por la de flujo o congestión.

Old version- tcp

No considera control de congestión inteligente.
"Ventana de congestión: cwnd"(no definida), trafico crece hasta que se resetea, buffer overflow o error.



Old tahoe:

Usa slow start. Crece exponencialmente hasta cierto punto. Mando segmento y cuando me aceptan mando el doble y así hasta timeout-

Primer ráfaga es slow start. Crece exponencialmente. Cuando se vence el rto deja de crecer, define el umbral y empieza a crecer lineal con congetion avoidance