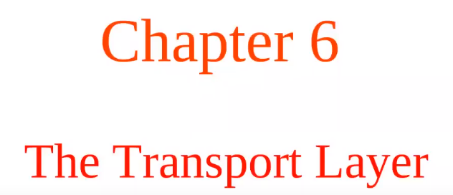
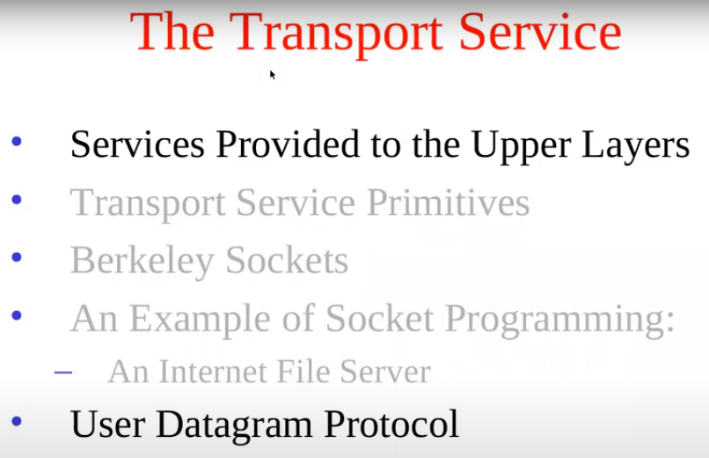
*-Clase del 07-09-2021. “Capa 4, de TRANSPORTE”*

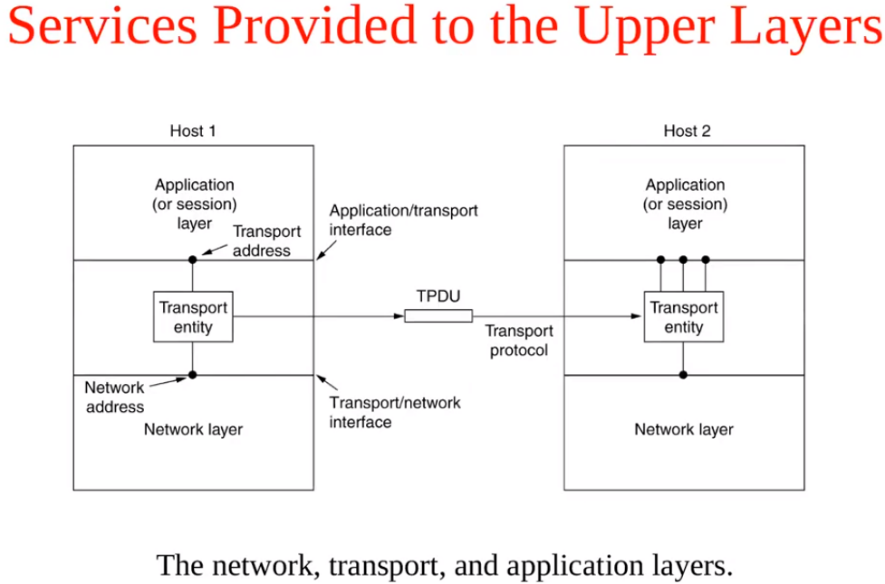
Este es el temario del libro, el profe se va a saltear varias cosas. Porque primero vamos ver los protocolos de la capa de transporte y después vamos a ver Sockets Berkeley y las primitivas de transporte. Hoy vemos los puntos negros y los grises después. Que se refieren a como yo desde un proceso o programa puedo utilizar la red.

Vamos a ver qué servicio da la capa de transporte, un ejemplo.

Básicamente en capa de transporte hay dos protocolos, uno es orientado a conexión “TCP” y el otro no “UDP”.

En la imagen se observan las capas de RED, Transporte y aplicación.

En capa de red la unidad de información eran los datagramas o paquetes, en la C. de Transporte la “unidad de protocolo de transporte” TPDU se llaman segmentos.

Un par de cosas interesante, son los puntos de entrada entre la capa de transporte y red, sería la dirección de RED. La c. de transporte utiliza la dirección de red, para indicar desde donde y hacia donde se dirige. Y la C. de Aplicación utiliza una dirección de transporte para indicar de que proceso origen a que proceso destino va la información.

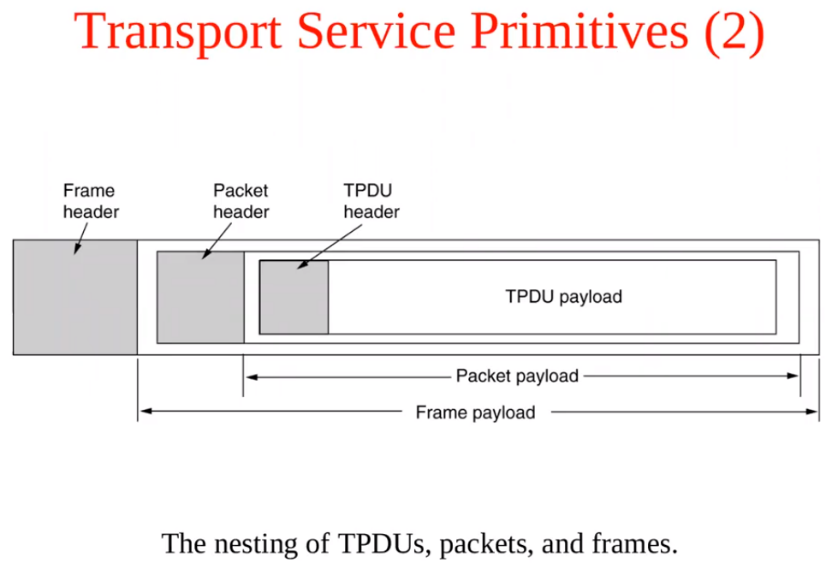
En capa de red interconecto maquinas. Su objetivo es que un paquete pueda atravesar distintas capas de enlace para llegar desde un Host origen a un Host destino. Como podían haber distintas tecnologías, en capa de Red, me abstraigo de las capa de enlace y le puse un plan de numeración homogéneo a toda las maquinas que estén conectadas en las distintas redes o capas de enlace, Ethernet, PPP o lo que sea. Se les pone una dirección IP homogénea. Entonces la capa de red lograba que un paquete llegue de una maquina a otra no importa donde estuviera, siempre y cuando estuvieran interconectadas por algún tipo de tecnología de capa de enlace.

El problema que esto es un IPC, yo busco interconectar procesos no maquinas. Entonces el tema es cuando me llega un paquete a la capa de red ¿a cuál de los 50 procesos que puedo tener corriendo se lo mando?

El objetivo fundamental de la C. de Transporte es lograr que la comunicación sea del proceso de un Host, al proceso de otro Host. En vez de entre host o maquinas solamente. Añade una abstracción más agregando direcciones de transporte que se asocian a los procesos, se crean nuevas direcciones que se vinculan a los procesos.

De alguna manera, se puede ver cómo, que la C. de transporte multiplexa el canal, supongamos que vemos la C. de red como un medio donde no importa lo que yo mando debería llegar a destino y si me llega algo no importa de donde viene, pero todos mis procesos deberían utilizar ese canal, pero debería de alguna manera multiplexarlo para saber desde que proceso se envían las cosas, y si me llegan a que proceso deben ir. La capa de transporte de alguna manera multiplexa el canal subyacente, que sería la capa de red, entre los distintos procesos.

Con respecto a las direcciones, en TCP/IP se llaman “PUERTOS”. Los puertos son direcciones de la capa de transporte, donde yo puedo asociarle aplicaciones a un puerto.

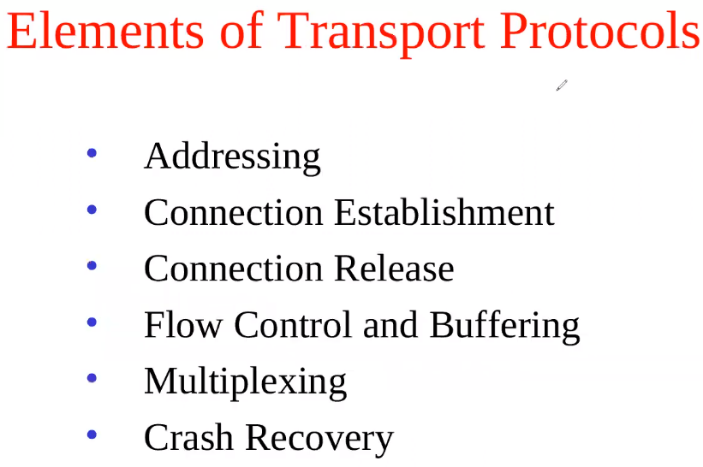
Se observa cómo es la encapsulación o anidado de los distintos protocolos.

Trama: de capa de enlace, que viaja por el medio físico, ej: Ethernet, 802.3, PPP, etc.

En su Payload se encuentra el paquete IP, con su encabezado.

Dentro se encuentra el “Segmento” de c. de transporte, con su encabezado.

Ejemplo pelar una cebolla.



Algunos de los temas no se ven en UDP porque es muy sencillo y no hace estas cosas, sino que lo vemos en TCP.

Elementos de la capa de transporte:

* Esquema de direcciones distinto del de capa de red

-Si es con conexión, un Mecanismo para establecer la conexión

-y un mecanismo para liberarla.

-Podría tener control de flujo y Buffer

-Multiplexación: donde se toma la capa de red y la comparto entre varios procesos que quieren enviar o recibir datos.

-Recuperación ante caídas, para protocolo con conexión. (NO para UDP)

Hay un parecido entre protocolos de transporte y de capa de “ENLACE de datos”.

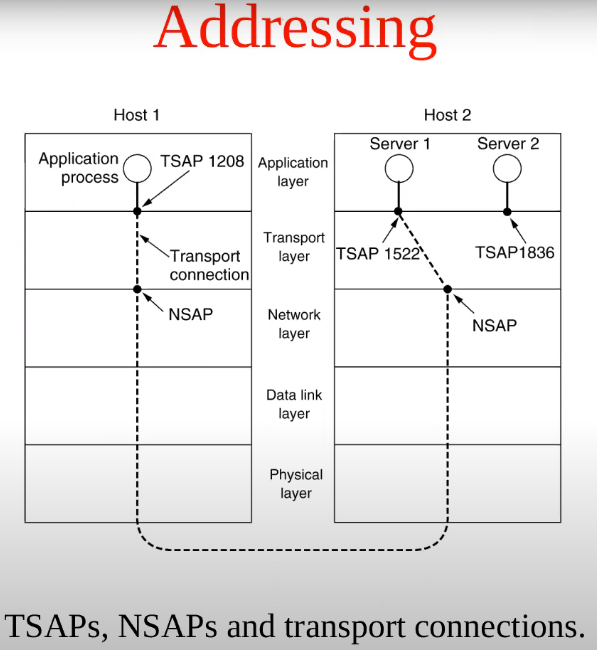
¿Por qué volvemos a ver control de error, de flujo, que vimos en la capa de enlace?

En capa de enlace el profe se salteo una parte que habla de protocolo de ventana, ventanas deslizantes porque lo vemos acá.

Se podría presentar en los 2, pero es más normal que se presente acá.

¿Por qué son similares?

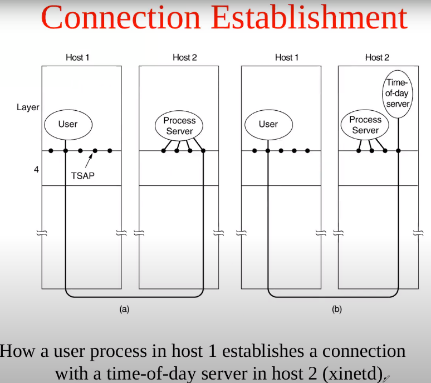
Si comparo, antes desde el punto de visto de la C. de enlace, teníamos en la capa física un Host conectado con otro host en el mismo medio. Y manejo de error, secuencia, y flujo.

Ahora en la capa de transporte, si vemos lo host como están conectados, tenemos un host que pasa por “n” redes de internet y se conecta a otro host. Podemos asumir que todas las redes de internet son como si fueran una capa de enlace (o física), en concreto la capa de transporte se comporta como la capa de enlace. Yo estoy utilizando un protocolo común debajo de la capa de transporte, que es el protocolo IP, si bien es una abstracción, porque hay un montón de capas de enlace intermedias, no una sola. Pero de alguna manera el protocolo de RED los “HOMOGENIZA” y lo podría ver desde capa de transporte como si fuera una única capa de enlace, por eso volvemos a tratar los mismos temas de extremo a extremo que vimos antes, manejo de errores, secuencia, y flujo.

Con respecto a las direcciones de la capa de transporte. Antes en la de red teníamos los NSAP Network Access Servicie Poit, que sería punto de acceso a la capa de red. Que serían las direcciones.

Ahora me encuentro con los TASP. “Trasport Access Servicie Poit” son numeritos, los PUERTOS. Asociados con un proceso.

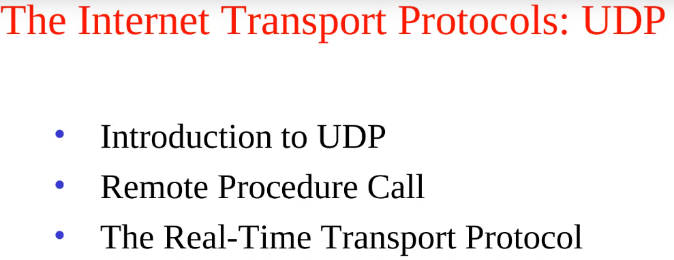
Ahora usos pares de direcciones. Una IP y un Puerto para comunicarme con un proceso.

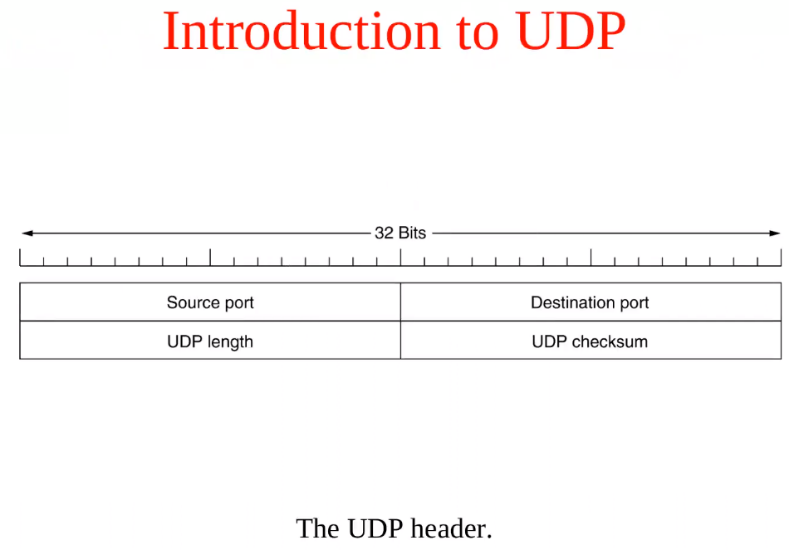


Esto Taffe dijo que no lo miremos, no se usa más.

Básicamente era para ahórrame procesos en memoria, entonces tenía un solo proceso que escuchaba en varios TSAP (puertos), y cuando alguien se conectaba recién ahí creaba un nuevo proceso que atendía al usuario. Hoy en día no se usa, era antes cuando teníamos pocos recursos.

Acá el profe se saltea un par de filminas, dice que primero veamos UDP, un protocolo sin conexión. Y después volvemos.

**UDP: User Datagram Protocol**

Este protocolo no establece ninguna conexión previa, si tengo datos disponibles los mando y listo. Si el proceso al que están dirigidos, no existe o se terminó o no se está ejecutando. No pasa nada, se perderán esos datos, y nadie me avisa que se perdieron o que llegaron bien tampoco, a no ser que el proceso receptor me conteste, si me da una respuesta podría saber si llegaron con éxito, pero depende del programa que escriba el usuario.

Es un protocolo sencillo. Va dentro del Payload del protocolo IP

Tiene solo estos 4 campos.

Puerto Origen: Dirección de capa de transporte. (16 bits)

Puerto destino: que tiene asociado el proceso o aplicación destino.

Largo del datagrama:

Checksum: del encabezado, y es opcional. Podría estar en cero. El proceso que lo recibe no lo descarta necesariamente.

Prácticamente solo coloca puertos origen y destino, que era lo necesario para multiplexar la capa de red.

Esto no establece conexión, no retransmite. ¿Entonces para que sirve? Se usa bastante.

Imaginemos que estamos transmitiendo cosas en tiempo real, como video o audio, no tiene sentido si una parte no llega, retransmitirla más tarde. Porque en tiempo real no puedo buferear, y si llega después me mete ruido o distorsión.

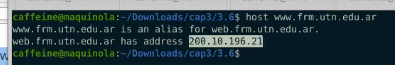
Por lo tanto, es más Rápido.

Depende de las aplicaciones, se usa bastante. Cuando hay que cumplir con restricciones temporales.

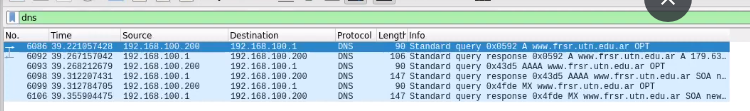
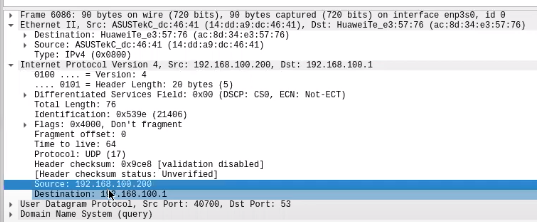
También se utiliza en aplicaciones donde tengo que intercambiar muy pocos datos. Si establecer y finalizar una conexión, me lleva más tiempo y costo, que intercambiar los datos directamente. Si se pierden, se puede volver a realizar un requerimiento, hasta que alguien me conteste.

Protocolo DNS, las direcciones IPv4 son 4 Byte, es más fácil y representativo recordar palabras o nombres en vez de números. Entonces básicamente, lo que hace DNS, es: yo consulto por un nombre y me responde con la dirección IP que le corresponde. Esto es de capa de aplicación. Pero internamente en capa de transporte usa UDP, porque las consultas son muy sencillas y no tiene sentido establecer y finalizar una conexión.

Wireshark, es una herramienta, un esnifer de red, captura todo el tráfico que llega a la placa que le especifiquemos y lo representa de una manera ordenada, de manera anidada muestra la información.

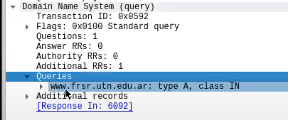


EJ: Zoom usa UDP.

 Se observa que la placa es ethernet, capa de enlace, Se observa la dirección origen y destino, son direcciones MAC, la mitad más alta 8 Bytes de los 16 bytes se asigna a los fabricantes.

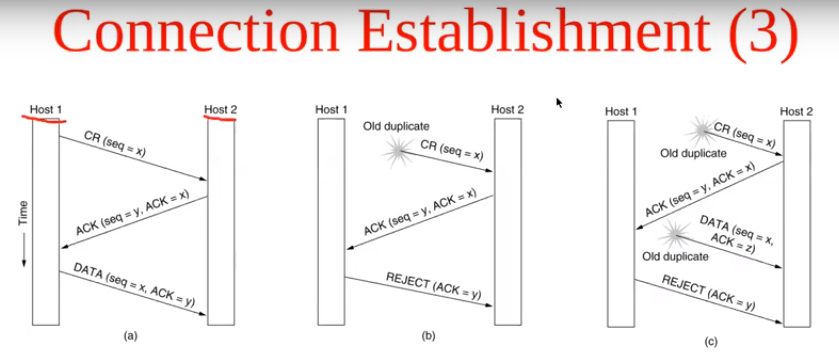


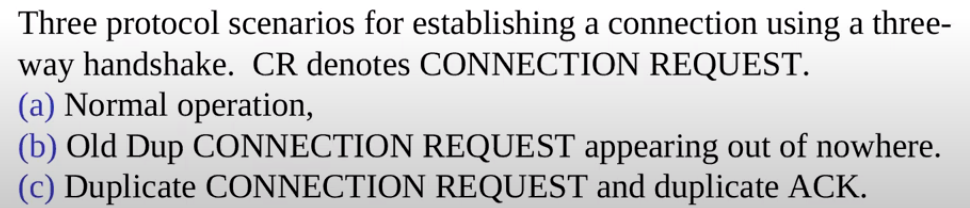
Todos los servidores DNS, están esperando que alguien les consulte en el puerto 53.



**Conexión**:

Vamos a ver, para un protocolo de capa de transporte, como es la conexión, o que cosas se negocian. Como es el requerimiento.

Básicamente se realiza un saludo de 3 vías.



Antes de intercambiar datos, tengo que intercambiar alguna información entre los hosts.

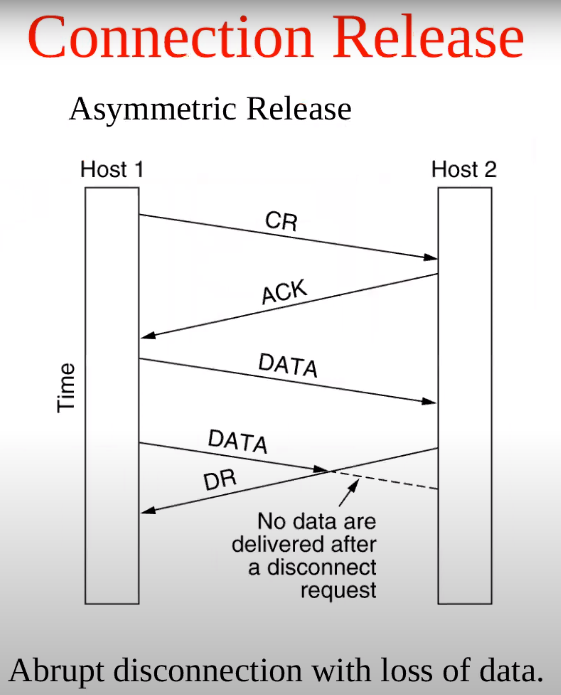
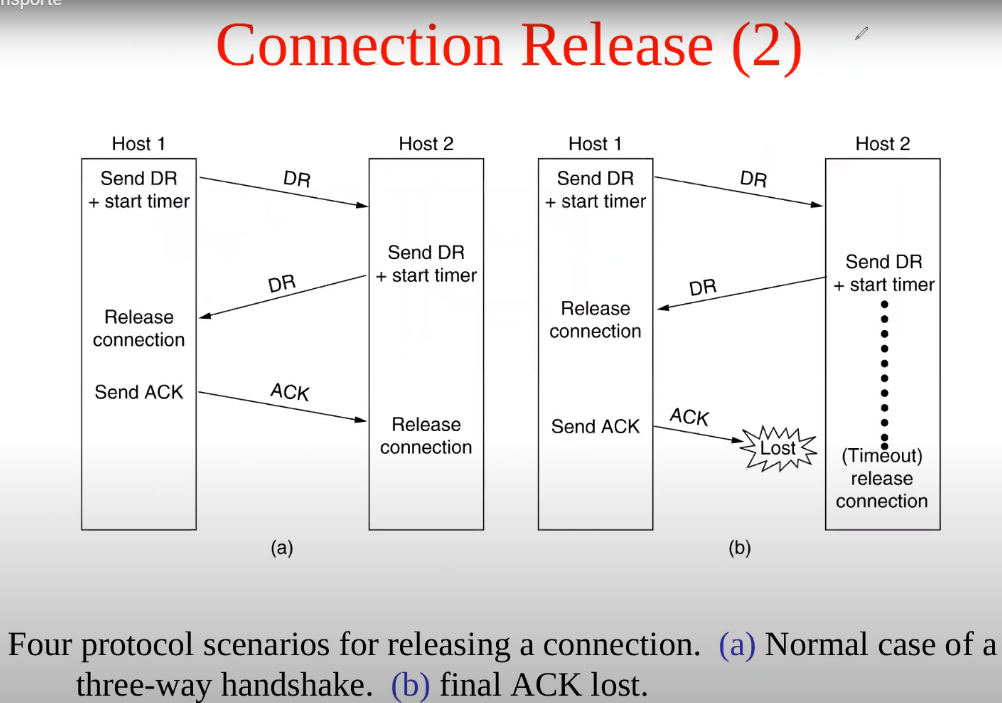
Estos van a ser tramas numeradas.

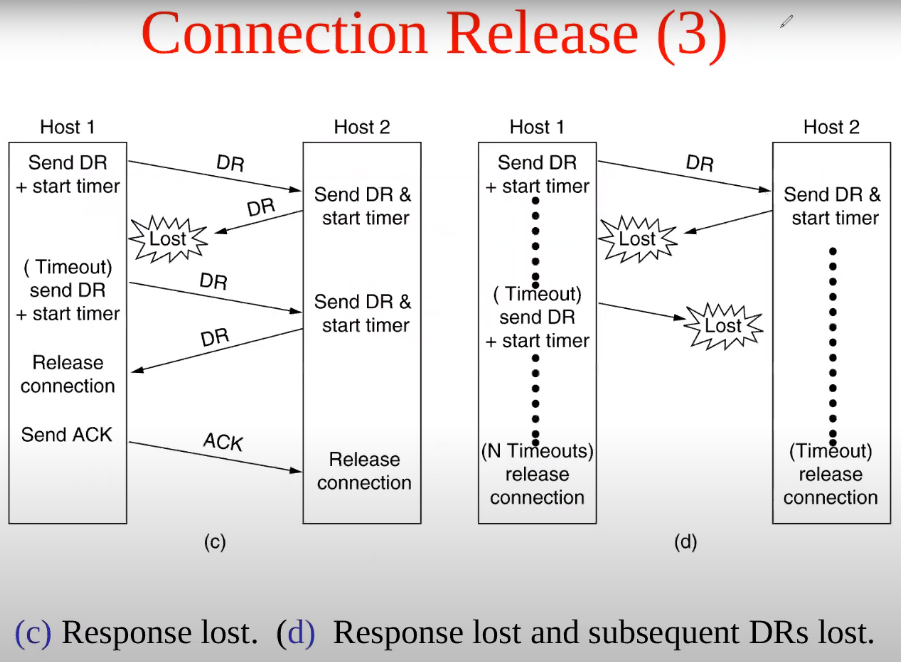
El receptor responde con una cuse de recibo y el número de secuencia del Host 2. Normalmente los números de secuencia empiezan de un numero aleatorio. Por temas de seguridad y evitar ataques.

Este ejemplo está hecho con 2, pero TCP utiliza un 3er acuse de recibo.

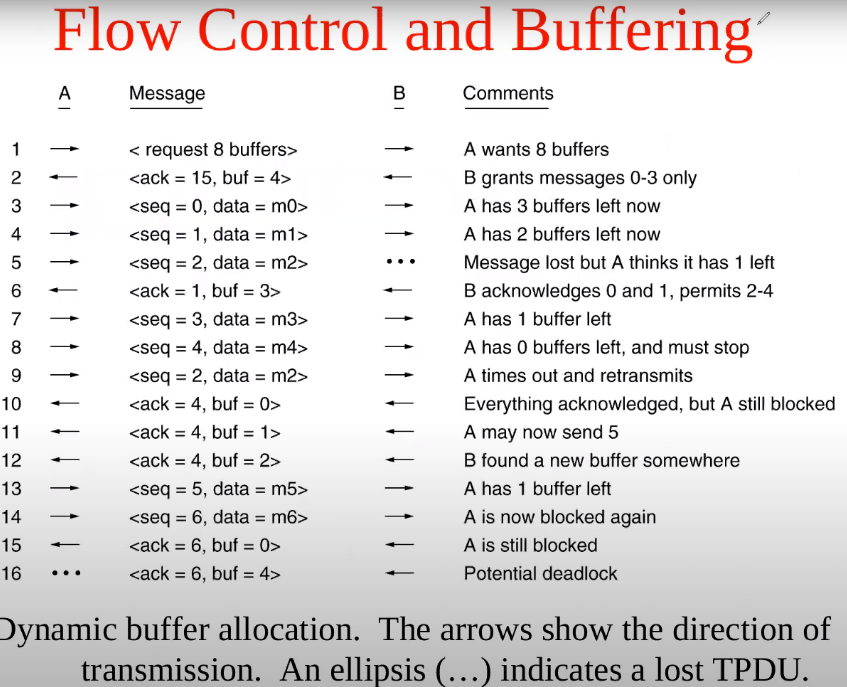
Casos b) y c) paso mucho tiempo, o él no la inicio, corta la conexión el host 1.

Vamos a ver en TCP/IP que hay unos bits en el encabezado que me sirven para iniciar la conexión, hacer el acuse de recibo e incluso para realizar el “Reject”.

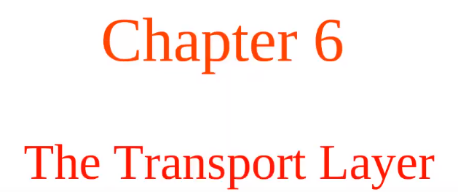


El profe saltea estas filminas y pasa directamente a TCP.

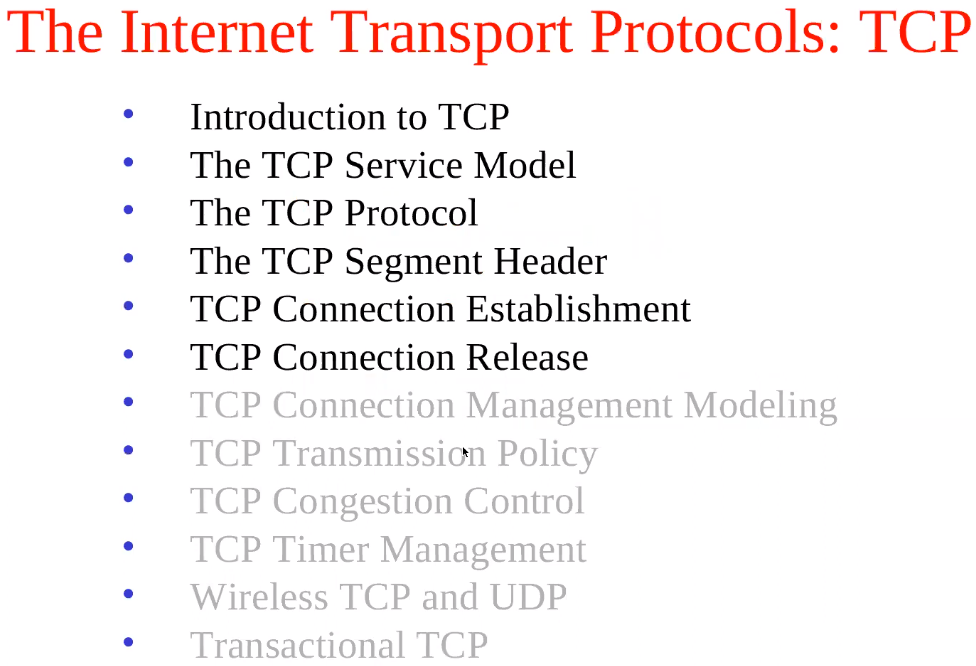
¿Esto hay que estudiarlo?



¿Esto hay que estudiarlo?

**TCP:**

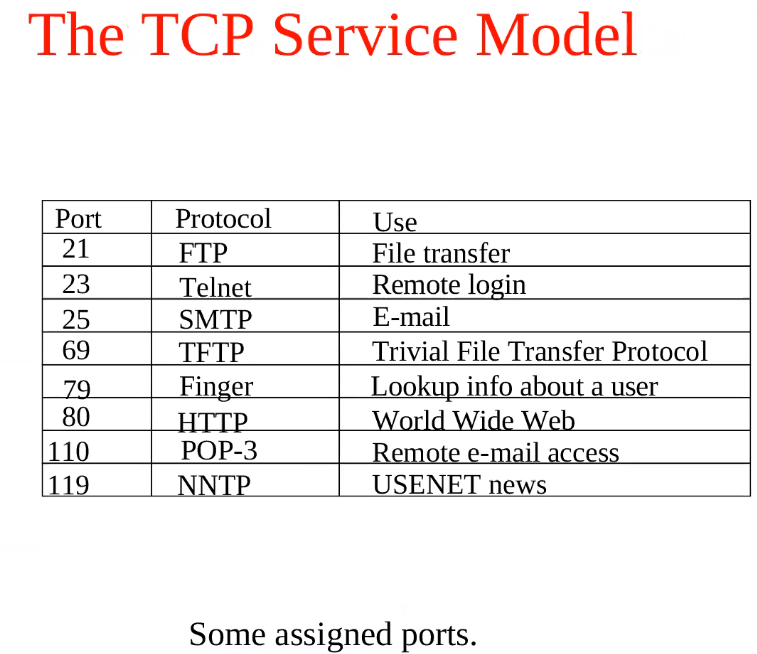
Vamos a ver como es el inicio y fin de la conexión. Y como es el formato del protocolo. Los campos del encabezado.



TCP es muy usado, es un protocolo orientado a conexión, con confirmación, y retransmisión en caso de que se pierdan los datos.

Me garantiza de alguna manera que no se pierda la información y que llegue en el mismo orden.

UDP no tiene número de secuencia ni nada, no me garantiza ni siquiera que llegue. Puede llegar desordenado o no llegar. No me asegura nada.

La Pregunta seria ¿Cómo eligen las aplicaciones o los procesos que número de puerto usar? ¿Cómo se los asigno?

Obviamente no pueden estar repetidos, es un puerto por proceso. Ej: tabla en un servidor, no se podría acceder.

DNS utiliza el puerto 53

HTTPS utiliza el 433 o 4433

Utiliza una forma llamada “Well know port” Puertos bien conocidos. Lo que hace es asociar todos los servidores a un puerto, siempre el mismo, en cada máquina.

Ej: Si tengo un servidor web funcionando en mi máquina, lo asocio al puerto 80. Entonces cualquier máquina del mundo que me quiera hacer una consulta a un servidor web que este corriendo en mi máquina, lo único que tiene que saber es cual es mi dirección IP o un nombre asociado a la misma. Porque el cliente ya después sabe que tiene que ir a buscar el puerto destino 80.

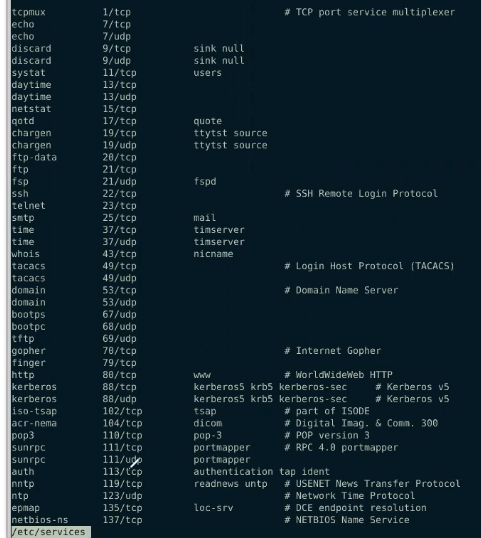
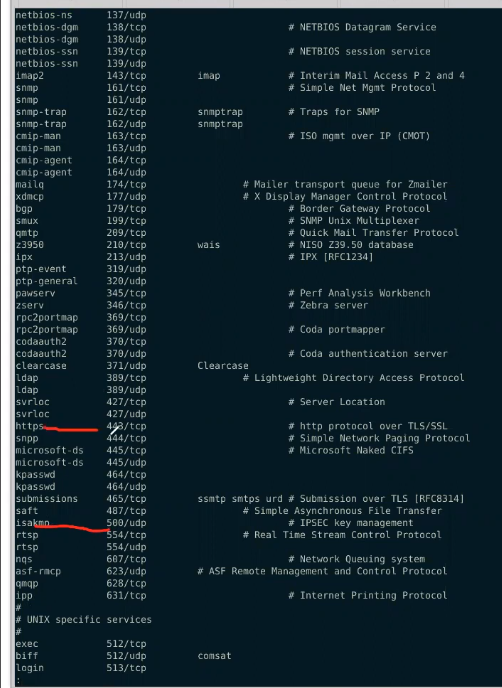
Los servidores, esperan que alguien haga consulta, los clientes que son las otras aplicaciones tienen que, en el caso de TCP, iniciar la conexión, y cuando estén conectados realizar la consulta. Por lo tanto, deberían saber a qué puerto remoto conectarse para iniciar la conexión. Si los puertos remotos estas asignados y bien definidos, asunto solucionado.

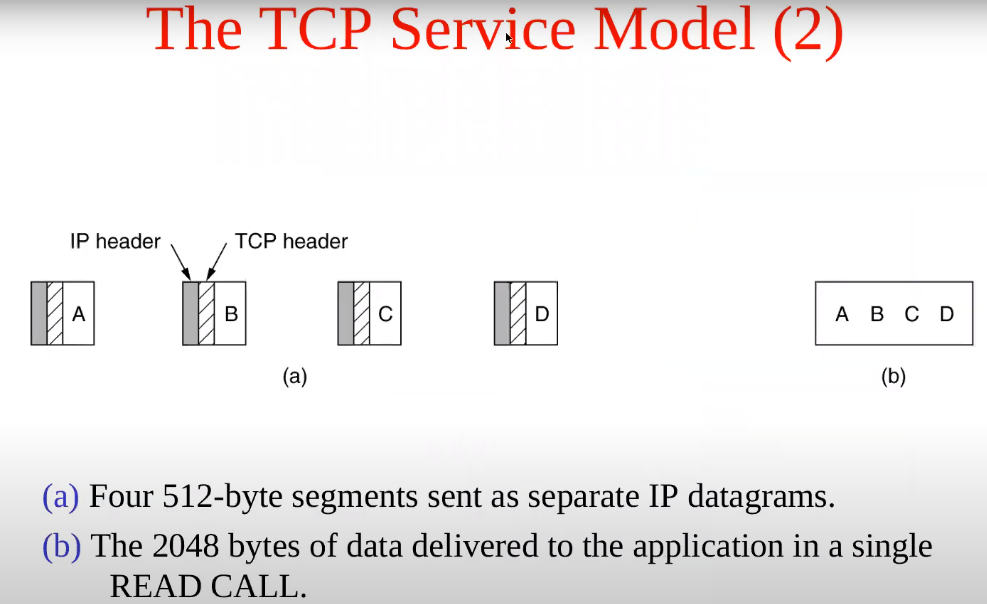
Esto es sencillo, fácil.

Hay varios protocolos, todos de capa de aplicación. Generalmente los puertos “Predefinidos” son del 0 al 1000 y están estandarizados.

Donde cada aplicación, de acuerdo a que protocolo implemente, es el puerto que va a usar.







Otra cosa respecto al modelo TCP, que UDP no lo tiene es: ¿Cómo es la comunicación?

La máquina origen podría enviar 4 segmentos, pero en el destino llegan en distinto orden, o se pierde alguno y se retransmite después, o pasar cualquier cosa. El tema es que la maquina destino cuando lo quiera leer o consumir los tiene ordenados.

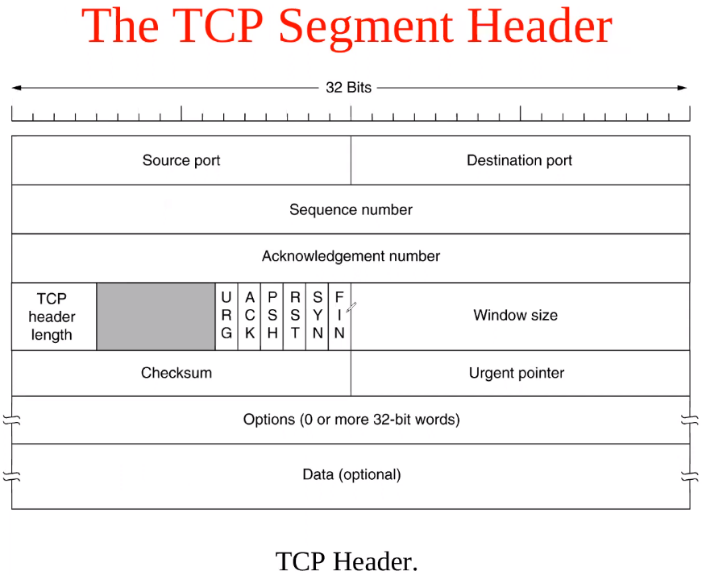
Tiene una similitud a un PIPE, escribo varias cosas y en el destino se leen en el orden que fueron escritas.

Desde el modelo semántico de cómo se usa, sería como un pipe bidireccional.

Video de la practica:

-Todas las conexiones TCP son full dúplex y punto a punto.

-TCP no soporta multidifusión ni difusión, pero UDP si, vimos el ejemplo de DHCP que utiliza UDP para obtener la IP del equipo. Video de la práctica ejemplo de UDP con telefonía por IP.



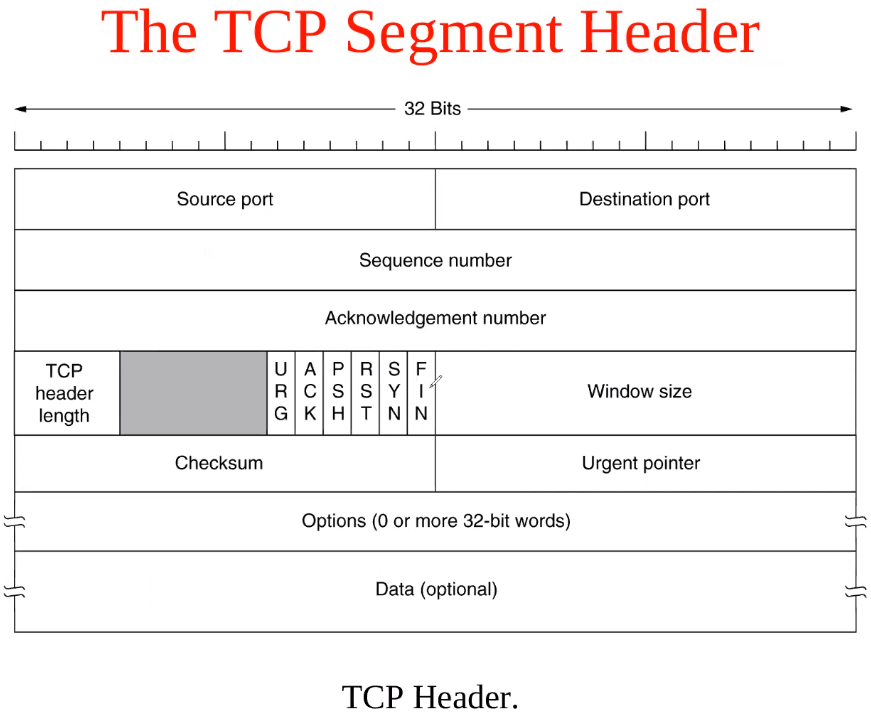
FORMATO: Tamaño máximo total encabezado + datos 64KB. Divide la información recibida desde la c. de enlace en fragmentos.

-Usa 16 bits para indicar Puerto origen y destino.

-Tiene un numero de secuencia que se le asigna a cada segmento, de 32 bits. Se incrementa a medida que se envían Segmentos. Pero no cuenta los segmentos, sino la cantidad de Bytes de datos que manda.

-El número de acuse de recibo son 32 bits, y lo envía el receptor al emisor de la información, informando el número de secuencia que previamente fue recibido.

En TCP/IP el ACK se incrementa en 1, Nº de secuencia +1, porque es el número del siguiente segmento que espera el receptor.

-Después tiene algunos Bits para realizar el inicio y fin de la conexión.

**SYN:** es un bit que se pone en alto para indicar que está iniciando el saludo de 3 vías.

**FIN:** Es para finalizar o liberar la conexión.

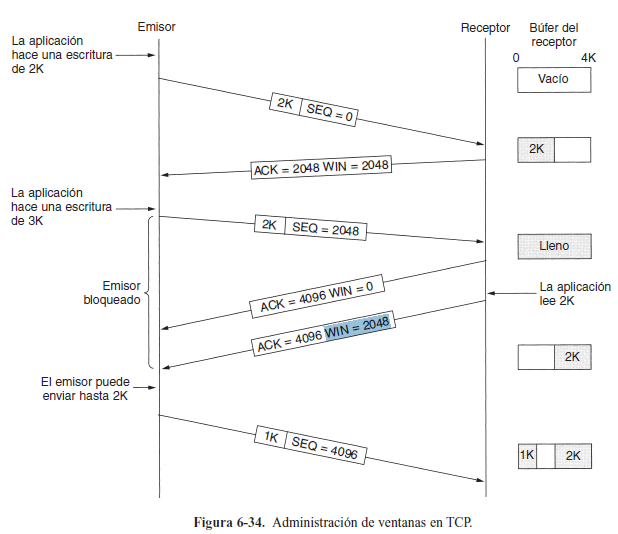
**RST:** En caso de que no haya podido iniciar el saludo de 3 vías.

**ACK:** para indicar que el dato en el campo de acuse de recibo es válido. Generalmente va siempre.

**URG:** Esta asociado al campo de Urgent Pointer, sirve para, en vez de procesar los datos del Payload como vienen, darle mayor prioridad a alguna parte. No se usa mucho.

**Checksum:** Obligatorio en TCP. Distinto del de capa de enlace.

**Length:** Como el encabezado puede tener opciones, el tamaño es variable y se tiene que especificar. (4bits) solo en el encabezado.

**Window size:** Tamaño de ventana, que es el buffer de salida, de datos que tengo esperando. Una vez que el destinatario me manda un acuse de recibo de un dato enviado previamente, lo saco del buffer y puedo seguir transmitiendo nuevos datos.

De esta manera, el tamaño de la ventana es variable y podemos realizar control de flujo con esto. A que me refiero, si mi host no puede procesar todos los datos que le llegan, porque es más lento que el emisor, lo que puede hacer es mandar un acuse de recibo, poniendo que el tamaño de la ventana es cero, por lo cual, el host emisor no va a poder mandar más datos, hasta que el receptor no modifique el tamaño. No pueden mandar DATOS, pero si acuses de recibo.

**PSH:** Es un BIT que tiene que ver con la aplicación. Normalmente cuando desde la aplicación o proceso escribo algunos datos, y le pido al SO que se encargue de mandarlos, a veces, lo que hace el SO, es que no los manda inmediatamente, sino que deja los datos en el buffer y espera un periodo de tiempo (milisegundos), por si hay más datos para enviar. ¿Para qué? Para generar un segmento más grande tratando de minimizar el Overhead, ósea buscando que la relación del tamaño del Payload sea mayor al tamaño del encabezado. De esta manera minimiza el overhead y se evita generar de nuevo todo el encabezado.

Cuando la aplicación pone el bit PUSH en alto, le avisa al operativo que no espere, sino que lo mande inmediatamente, independientemente del tamaño de los datos, puede ser 1 byte por ejemplo. Esto lo usan algunas aplicaciones interactivas, ej: Telnet.

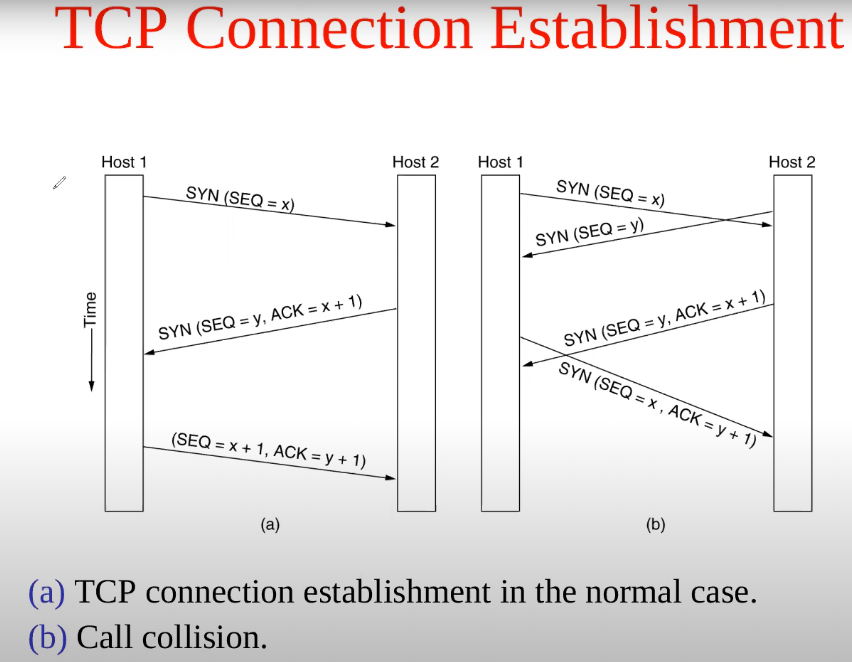
El encabezado se envía completo cada vez que se intercambia algo entre los hosts, sea mensaje de control (handshake) o datos.

Observar que la parte de datos es opcional.



NO lo explico el profe

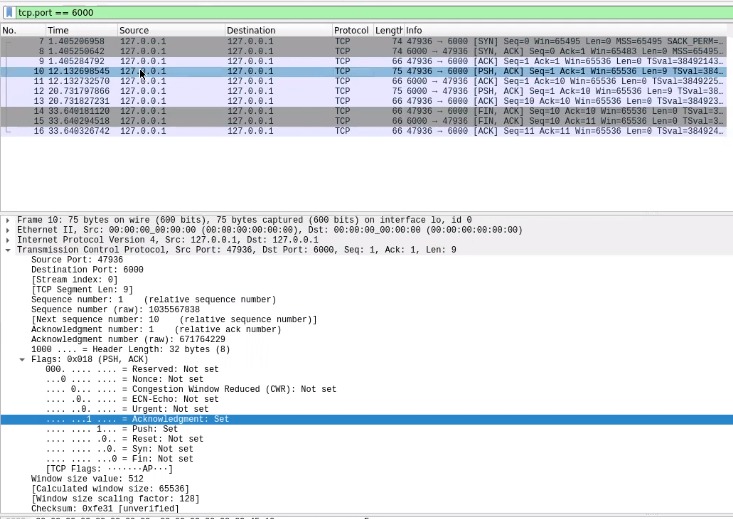
¿Esto hay que estudiarlo?



Ejemplo de inicio de conexión, uno funciona y el otro tiene colisión.

Inicia la conexión, se intercambian datos, depende de la capa de aplicación que tantos datos voy a enviar y recibir, y una vez que ya no solicito más datos corto la conexión desde la aplicación y esto genera un handshake de liberación de la conexión.

Ejemplo



EL modelo TCP me asegura que lleguen los datos en el orden que se enviaron, cosa que UDP, no. ¿Cómo lo hace? Cada segmento tiene específicamente número de secuencia, y el destino va dándole confirmación de recibo a lo que recibió, si algo se pierde en el camino, dato o ACK, lo retransmite y en algún momento llega el ACK correspondiente. Esto le da confiabilidad.

Pero el objetivo fundamental de la capa de transporte no es este, es multiplexar el canal o que varios procesos puedan compartir la misma capa de red, esto lo hace asignando direcciones de la capa de transporte, los puertos.

No puede haber en la misma maquina 2 procesos usando el mismo puerto.

Sockets: es la manera de, desde mi aplicación, utilizar la red. Ósea yo no tengo que escribir los encabezados de las capas, sino tengo unas instrucciones, (primitivas), que son llamadas a sistema del SO, y con eso, el SO se encarga de generar tramas, paquetes, segmentos.