Yeison Melo
melocuentamaster@gmail.com
Si encuentras algun error no dudes en hacermelo saber, gracias. Las faltas ortográficas no cuentan ;)

Capa de transporte:

Ofrece comunicación entre aplicaciones, el puerto de cada aplicación esta identificado con un numero de 16bits y será el mismo tanto en cliente y servidor.

Protocolo UDP: UDP PDU == UDP datagrama

UDP es un protocolo de nivell de transporte “orientado al datagrama”. Es decir, ofrece el mismo servicio que el transporte de datagramas al nivel IP. La cabecera UDP tiene un tamaño fijo de 8 bytes.

UDP es: no seguro(si se pierde un datagrama no se retransmite), sin recuperación de error, sin ack (confirmación), no orientado a la connexión (conenectionless, no hay un período de establecimiento de connexión) y sin control de flujo (datagramas pueden llegar fuera de orden). Sin buffer de transferencia(Tx): cada operación de escritura de cada aplicación genera un datagrama.

Normalmente usado en aplicaciones con poco intercambio de mensajes (dhcp,dns, rip) o aplicaciones de tiempo real donde se lee el buffer de información a un ritmo constante por lo tanto si un datagrama llega más tarde del momento en que se esperaba leerlo es inservible y es descartado.

Protocolos ARQ (Automatic Repeat reQuest):

Se encargan de la comunicación entre destinos finales añadiendo funcionalidades como pueden ser: detección de errores, recuperacción de errores y control de flujo. Es decir, que la inforamación llegue sin errores, sin duplicados y en el mismo orden en que se transmitierón. Tenemos un cliente (secundario) y un servidor (primario) cada uno con su buffer de transmisión (Tx) y recepción (Rx).

El secundario informa de la llegada de la información mediante mensages de ack (acknowledgements) al primario, solo envia ack cuando capa superior lee el datagrama del Rx. En el Tx del primario se guarda la información enviada que aun no se ha confirmado y cuando llega su respectivo ack la borra del buffer y escribe nueva para enviarla. En caso de error el

primario puede reenviar la información porque se encuentra en el buffer. El Rx del segundario almacena la información hasta que es leída por el nivel superior.

Además necesitamos numeros de secuencia para identificar los mensajes y lasrespetivas confirmaciones de estos por razones obvias y es que el sistema sepa en cada momento que mensaje se esta confirmando o que información del conjunto le esta llegando. Donde al mensaje Ik coresponderia el ack Ak.

Stop and wait

El principio de funcionamiento del protocolo es el siguiente: “transmitir una PDU y esperar a que sea confirmada para transmitir la siguiente”. El primario genera el datagrama Ik lo escribe en su Tx y se transmite. Ik llega al Rx del segundario, es pasado a la capa superior y genera su respectivo Ak, acontinuación se lo envia el primario. El primario recibe Ak y borra Ik de su Tx.

Cada vez que el primario envia un datagrama inicia un timeout (TO), si el datagrama es recibido con errores o se pierde, no se envia su ack. El TO expirara en el primario y lo volverá a enviar, generando una retransmisión (ReTx).

Calculos

Se D m la sitancia entre las dos estaciones, Vt bps la velocidad de transmisión o efectiva, Vp m/s la velocidad de transmisión del señal eléctrico e Ik un datagrama de longitud Lt bits con un Ak de longitud La. Tenemos que:

- La transmisión durará Tt= Lt/Vt segundos
- El tiempo de prograpación de cada bit sera Tp = D/Vp
- Tiempo de transmisión del ack Ta = La/Vt
- Normalmente Ta<<Tt

Fijarse en que las unidades sean las mismas (metros, bits, segundos).

Eficiencia del protocolo Stop and wait

Asumiendo que no hay errores(máxima eficiencia)

E= (1/ (1+2a)) donde a = Tp/Tt → a pequeña: eficiencia mayor/máxima, a grande: eficiencia decrece rápidamente

Protocolos de transmisión continua

Permiten transmitir más de una PDU sin confirmar, por lo tanto resuelven el problema de Stop and Wait (es independientemente del tiempo de propogación) , dandonos una eficiencia del 100% en ausencia de errores. Para la recuperación de errores tenemos dos algoritmos.

Go Back N

- Ack's acumulativos: Ak confirma todos los datagramas Ii para dodo i<=k.
- Primario: cuando salta un TO vuelve a retransmitir el datagrama causante y continua el envió desde ese punto.
- Segundario: en caso de detectar error o PDU fuera de orden, deja de enviar ack's y descarta todos los datagramas siguientes, hasta que se le reenvie el que fallo. Las PDU's siempre estarán en orden en el Rx

Retransmissión selectiva

- Ack's acumulativos
- Primario: solo reenvia PDU's de las cuales ha saltado su TO
- Segundario: solo descarta los datagramas que tienen error, acepta todos los siguientes pero sin mandar ack, cuando recibe el causante del error envia ack-acumulado para confirmar hasta la última PDU recibida en sequencia correctamente

  Los acepta aunque esten fuera de orde → complica la implementación. Un datagrama no se puede servir a la capara superior hasta que todos los que lo preceden han sido servidos tambien, si no han llegado se queda en Rx hasta que lleguen.

Control de Flujo - Protocolo de ventanas

El control de flujo consiste en evitar que el primario sature el Rx del segundario. Para ello establecemos un limite de PDU's que se pueden enviar sin estar confirmadas es lo que llamamos "**ventana de transmisión**". De esto deducimos que: Stop and wait es equivalente a un protocolo de venta de tamaño 1 y que, en los protocolos de retrasmisión selectiva en Tx y Rx tanto del primario como del segundario habrá como máximo W PDU's donde W es el tamaño de la ventana.

Ventana Óptima

Ventana minima que permite conseguir la velocidad efectiva máxima = parteSupero(Tt/Tu) y Eficiencia = Tu/Tc, donde Tc es el tiempo que tarda en repertirse el ciclo con ventana no

optima, Tu el tiempo que se estan enviado datagramas del ciclo y Tt el tiempo de envio de un datagrama (mirar pa.19 de las diapositivas).

Protocolo TCP

Recuperación de errores, confirmación de entrega (ack's), orientado a la conneción, control de flujo (primario-seguncario), control de congestión (saturación en la red). Aplicaciones como web, ftp, ssh, telnet, mail. Tamaño de cabecera 20bytes+opciones, total 60bytes como máximo. Con TCP hablamos de segmentos.

Es un protocolo ARQ con tamaño de venta variable: wnd=min(awnd,cwnd).

Advertised Window: para el control de flujo, siempre se envia en la cabecera TCP indica el espacio libre del buffer Rx. Cada vez que el segundario recibe un segmento recalcula el valor de la variable wnd. El primario nunca puede enviar más bytes de los que indica wnd. De esta forma el segundario siempre tiene espacio suficiente para recibir los bytes enviados.

TCP envia acks inmediatamente después de recibir el segmento de información adiferencia de los ARQ basicos vistos antes donde se enviaban cuando la capa superior los leía. Loque implica que en los ARQ básicos si el segundario leí la información más lento tardaba más en enviar los acks y de esta forma se hacia el control de flujo. En TCP en cambio se realiza mediante la wnd ya que se envia el espacio disponible en Rx y por lo tanto al conocer esta información el primario decide cuanta información más puede enviar.

Congestion Windows: para el control de congestión debído a cuellos de botella en la red. Incrementa o decrementa en función de si se detecta perdida de segmentos en el envio.

Conexiones TCP se pueden clasificar en:

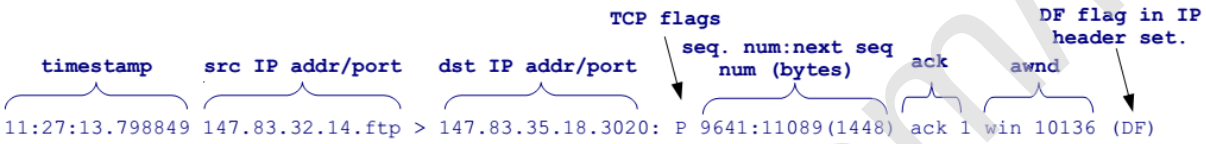
Bulk transfer (massive): la aplicación siempre tiene datos para enviar. Tx del primario siempre lleno y se envian segmentos de tamaño máximo.

Interactives: interacción del usuario con el host remoto, poco tráfico. La cantidad de información enviada esta muy por deabajo de lo que permite la ventana de transmisión. El envio de paquetes tan pequeños reduciría la eficiencia, mecanimos para corregirlo:

- Delayed ack: reduce el numero de ack's. Envia un ack cada 2 segmentos de tamaño máximo o cada 200ms, que los confirma todos. En caso de recibir segmentos fuera de orden siempre se envia.

En la siguiente imagen vemos como awnd es de 10136, awnd < cwnd suponemos, encambio se envian segmentos de 1148<<10136 por lo tanto se confirman cada 2 segmentos de tamaño máximo que suponemos que es 1148

```
...
11:27:13.798849 147.83.32.14.ftp > 147.83.35.18.3020: P 9641:11089(1448) ack 1 win 10136 (DF)
11:27:13.800174 147.83.32.14.ftp > 147.83.35.18.3020: P 11089:12537(1448) ack 1 win 10136 (DF)
11:27:13.800191 147.83.35.18.3020 > 147.83.32.14.ftp: . 1:1(0) ack 12537 win 31856 (DF)
11:27:13.801405 147.83.32.14.ftp > 147.83.35.18.3020: P 12537:13985(1448) ack 1 win 10136 (DF)
11:27:13.802771 147.83.32.14.ftp > 147.83.35.18.3020: P 13985:15433(1448) ack 1 win 10136 (DF)
11:27:13.802788 147.83.35.18.3020 > 147.83.32.14.ftp: . 1:1(0) ack 15433 win 31856 (DF)
...
```



- Naggle algorithm: cada vez que llegan datos nuevos al Tx se permite enviar un nuevo segmento solo si: hay suficientes bytes para enviar un segmento de tamaño máximo o si no hay segmentos pendientes de confirmar. Si hay información pendiente de confirmar la nueva se almacena en el buffer hasta que llega la confirmación.

Conexión TCP

El cliente siempre envia el primer segmento, cliente y servidor intercambia 3 segmentos de payload 0:

- El cliente es quien envía el primer segmento. Flag SYN (S) activado y flag ACK desativado. Numero de secuencia inicial aleatorio, 32 bits.
- Segundo segmento es enviado por el servidor Flag S, ACK activado. Consume un numero de sequencia. Su ack vale = numero de sequencia inicial +1.
- Tercer segmento enviado por el cliente numero de secuencia inicial del servidor +1

Para finalizar la connexión segmento con Flag F puede iniciarlo tanto cliente como servidor y pueden contener datos (payload >0)

Control básico de congestiones

Los algoritmos de slow start/Congestion Avoidance hacen el control básico de la ventana TCP. Durante SS cwnd es incrementado rápidamente. Durante CA cwnd es incrementado lentamente.

cwnd: ventana de congestión
snd_una: primer segmento no confirmado, segmento que lleva más tiempo en Tx para por ser confirmado.
ssthresh: umbral entre las fases de slow start y congestion avoidance

Cuando se inicia la connexión cwnd = MSS, solo se puede enviar un segmento sin confirmar, de aquí el nombre de Slow Start. Mientras no haya perdidas TCP incrementa cwnd en 1 MSS por cada ack recibido. Si salta algun TO tendremos una perdida y

ssthresh = max((min(cwnd,awnd)/2, 2 MMS)
Maximo de , Minimo entre cwnd y awnd divido entre 2, o 2 MMS. Como minimo tendremos el ssthresh a 2 segmentos.

Después vuelve a empezar el Slow Start hasta que lleguemos a cwnd=sstresh. A partir de entonces entramos en la fase de congestion Avoidance y la ventana aumenta lentamente, aproximadamente 1 MSS cada vez que se reciban las confirmaciones de toda una ventana. Por lo tanto si cwnd = 3 y nos llega un ack cwnd se vera aumentado en 3 + ⅓.

Si no tenemos congestion estaremos en Slow Start hasta que llenemos el Rx del segundario y sera entonces este quien dictara el tamaño de la ventana de transmisión siempre. Es lo que suele pasar en connexiones LAN.

La idea es es que aumentemos cwnd rapidamente hasta llegar a sstresh(donde empezamos a tener perdidas) llegados a este punto aumentamos lentamente para detectar con mayor presición cual es el limite real y aprobechar mejor la connexión.

Rount-Trip-Time: período desde el momento en que se envía un segmento y aquel en el que se recibe su ack.

MSS = MTU - 20(cabecera TCP) -20(cabecera IP)
normalmente MSS = 1460 **KB** → MTU = 1500 **KB**
RTT==Tp=tiempo de ping=retardo en recepto y transmisor

tt=tiempo de transmisión de una PDU=Lt/Vef

Velocidad Eficaz
Vef= awnd/RTT

Vef=aprox=Wop/RTT ← todo a segundos y bits o bytes, de mili a seg → mili/1000

Ventana Optima en Bytes

Wop= Vef*RTT ← todo a segundos y bits o bytes, de mili a seg → mili/1000 RTT== Tiempo de ping

Wop = ParteSuperior Tt/tt = ParteSuperior ((tt + 2tp+Tack)/tt)
El Tack normalmente no se tiene en cuenta así que no tenerlo en cuenta a menos que no s lo digan

Ventana optima en segmentos
partesuperior (Wop en (b,B,..) / MSS)
total de informacion de ventana optima /maximo tamaño por segmento

tienen que estar en la misma unidad si MSS suele ser 1460 KB Wop tiene que estar en KB tambien!

Aplicaciones

Email

Componentes: capa de transporte puerto well-known 25. Utiliza TCP. Uitliza el protocolo SMTP para el envio de los mensajes. Los mensajes de respuestadel servidor contienen un código de estado de 3 dígitos.

Protocolos de recuperacion: permiten obtener los mensajes de correo electrónico almacenos en un servidor de internet para su gestión. IMAP, POP, HTTP.

MIME (Multipurpose Internet Mail Extensions): formato que añade funcionalidades. Adjuntar archivos de datos no-ASCI (archivos,imagenes, audio, video). Mensajes con diferentes partes diferenciadas. “Prácticamente todos los mensajes de correo electrónico escritos por personas en Internet y una proporción considerable de estos mensajes generados automáticamente son transmitidos en formato MIME a través de SMTP”

Post Office Protocol (well-known 110): usado normalmente para borrar correo.
Intener MessageAcces Protocol: wellknown143: proporciona comandos para crearcarpetas, mover mensajes descagar parte de los mensajes. Los mensajes se mantienen en el servidor a menos que el usuario especifique que desea borrarlos
Web Base Email (HTTP): servidor web que se encaga de gestionar el buzón de correo. El agente usuario es un navegador web, usando HTTP para enviar y recuperar los emails.

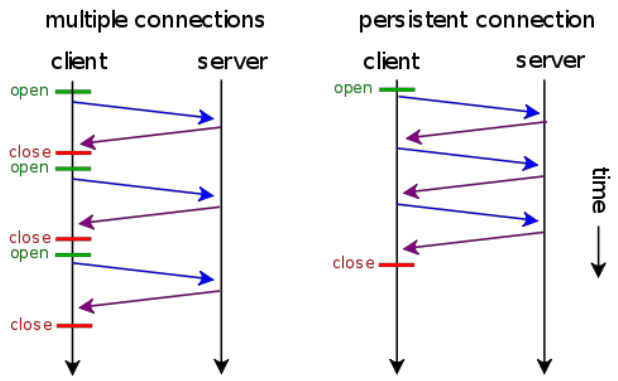
World Wide Web

Sistema de distribución de documentos de hipertexto.

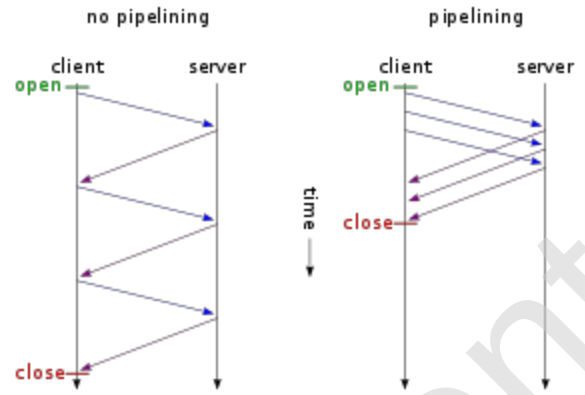
Capa de transporte: TCP, puerto well-known 80.
Protocolo HTTP (HyperText Transfer Protocol) de la capa de aplicación usado en cada transacción de WWW.
HyperText Markup Languag (HTML): lenguaje utilizado para formatear documentos web.

No persistete (default in HTTPP/1.0) el servidor cierra la connexió TCO después de enviar cada objeto. Ejem: una página html con 10 images, 11 conexiones TCP son sequencialmente abiertas.

Persistente (default in HTTP/1.1): el servidor mantiene la conexión TCP abierta hasta que un pase un tiempo máximo de inactividad. Los 11 objetos serán enviados a travez de la misma connexión TCP.

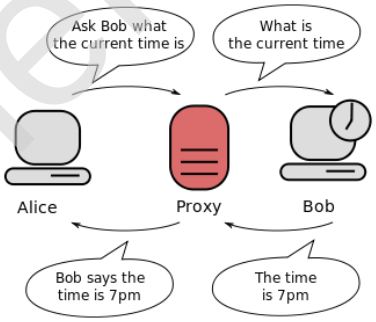


Connexiones persistentes con pipelining (soportado en HTTP/1.1 solo): El cliente emite nuevas solicitudes tan pronto como encuentra nuevas referencias, incluso si los objetos no han sido completamente descargado.



Caching: el cliente almacena las páginas descargadas en cache local. Se utila un GET condicional en función de un parametro/fecha de caducidad para saber si es neceario volver a descargar la página.

Proxy server: servidor intermediario entre el cliente y el servidor que tiene la página. Proporciona seguridad, logs caching, ahorrar direcciones ip publicas.



HTML

HTML objetivo de mostrar páginas de hipertexto formateadas (incluso links a otros documentos) en navegadores webs. Características: formularios que permiten al usuario introducir información para enviarla al servidor. Scripting permite añadir programas que son ejecutados en la maquina del cliente cuando se carga o cuando un link es activado.

Cascading Style Sheets (CSS): permite especificar la distribución y estilo visual de la pagina en un documento separado de esta, que puede ser usado por diferentes webs.