Protocolo TCP

Matías Ariel Ramirez, Agustín Cucurull y Héctor Ulloa

I. RESUMEN

El siguiente informe se tratará sobre el protocolo TCP. Describiendo el ¿por qué se creó junto a su modelo TCP/IP? Detallando las características del protocolo como el encabezado del segmento y la ventana deslizante. Como es su modelo de servicio, que puertos están protegidos, por ejemplo. De qué forma se administra, establece o termina una conexión. La administración de temporizadores como el seguir con vida o de persistencia, entre otros. Y como se controla la congestión con, por ejemplo, Reno o Tahoe.

II. PALABLAS CLAVE

TCP; Tahoe; Reno; Capa de Transporte; Modelo; Protocolo; Congestión; Temporizador; Ventana deslizante; Conexión.

III. INTRODUCCIÓN

El Departamento de Defensa de Estados Unidos, decidió reemplazar la red ARPANET, que conectaba redes de satélites, de radio, y también cientos de universidades e instalaciones gubernamentales mediante el uso de líneas telefónicas rentadas. Esto, debido a la preocupación de que alguno de sus valiosos hosts, enrutadores y puertas de enlace de interredes pudieran ser volados en pedazos en cualquier momento por un ataque de la antigua Unión Soviética. Quería que las conexiones permanecieran intactas mientras las máquinas de origen y de destino estuvieran funcionando, incluso aunque algunas de las máquinas o líneas de transmisión en el trayecto dejaran de funcionar en forma repentina. Además, como se tenían en mente aplicaciones con requerimientos divergentes que abarcaban desde la transferencia de archivos hasta la transmisión de voz en tiempo real, se necesitaba una arquitectura flexible. Esta nueva arquitectura es conocida como modelo TCP/IP.

IV. MODELO TCP/IP

No existe un modelo oficial de protocolos TCP/IP. En este informe se organizará en cuatro capas las cuales se muestran en la figura 1.

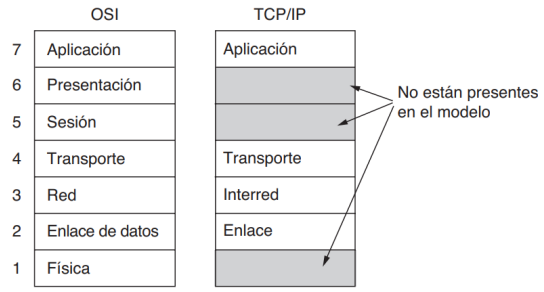


Fig. 1: Comparación entre modelo OSI y modelo TCP/IP

Nosotros solo describiremos la capa de transporte:
Está diseñada para permitir que las entidades pares, en los nodos de origen y de destino, lleven a cabo una conversación, al igual que en la capa de transporte de OSI. Aquí se definieron dos protocolos de transporte de extremo a extremo. El primero siendo UDP (Protocolo de Datagrama de Usuario), es un protocolo sin conexión, no confiable para aplicaciones que no desean la asignación de secuencia o el control de flujo de TCP y prefieren proveerlos por su cuenta.
Y el segundo protocolo, llamado TCP (Protocolo de Control de la Transmisión), el cual será descrito en las siguientes secciones.

V. INTRODUCCIÓN A TCP

Se diseñó para proporcionar un flujo de bytes confiable de extremo a extremo a través de una interred no confiable, para adaptarse de manera dinámica a las propiedades de esta y sobreponerse a muchos tipos de fallas. Una interred difiere de una sola red debido a que sus diversas partes podrían tener diferentes topologías, anchos de banda, retardos, tamaños de paquete y otros parámetros.
TCP se definió formalmente en el RFC 793 en septiembre de 1981. Con el paso del tiempo se han realizado muchas mejoras y se han corregido varios errores e inconsistencias. Se público un documento llamado RFC 4614, que sirve como guía a las mejoras y corrección de errores e inconsistencias que ha tenido el TCP a lo largo del tiempo.
Cada máquina que soporta TCP tiene una entidad de transporte TCP, ya sea un procedimiento de biblioteca, un proceso de usuario o (lo más común) sea parte del kernel. En todos los casos, maneja flujos TCP e interactúa con la capa IP. Una entidad TCP acepta flujos de datos de usuario de procesos locales, los divide en fragmentos que no excedan los 64 KB, y envía cada pieza como un datagrama IP independiente. Cuando los datagramas que contienen datos TCP llegan a

una máquina, se pasan a la entidad TCP, la cual reconstruye los flujos de bytes originales.
El TCP debe proporcionar un buen desempeño con la confiabilidad que la mayoría de las aplicaciones desean y que IP no proporciona. Ya que este último, no ofrece ninguna garantía de que los datagramas se entregarán de manera apropiada, ni tampoco una indicación sobre qué tan rápido se pueden enviar los datagramas. También le corresponde reensamblar los datagramas que lleguen en orden incorrecto en mensajes con la secuencia apropiada y a su vez enviar los datagramas con la suficiente rapidez como para hacer uso de la capacidad sin provocar una congestión.

VI. EL MODELO DEL SERVICIO TCP

El servicio TCP se obtiene al hacer que tanto el servidor como el receptor creen puntos terminales, llamados sockets. Cada socket tiene un número (dirección) que consiste en la dirección IP del host y un número de 16 bits que es local para ese host, llamado puerto. Podemos usar un socket para múltiples conexiones al mismo tiempo (dos o más conexiones pueden terminar en el mismo socket).
Los números de puerto menores que 1024 se llaman puertos bien conocidos. Están reservados para los servicios estándar que, sólo los usuarios privilegiados pueden iniciar.
Se pueden registrar otros puertos del 1024 hasta el 49151 con la IANA para que los usuarios sin privilegios puedan usarlos, pero las aplicaciones pueden y seleccionan sus propios puertos.
Todas las conexiones TCP son full dúplex (el tráfico puede ir en ambas direcciones al mismo tiempo) y de punto a punto (cada conexión tiene exactamente dos puntos terminales). TCP no soporta la multidifusión ni la difusión. Una conexión TCP es un flujo de bytes, no un flujo de mensajes. Los límites de los mensajes no se preservan de un extremo a otro.
Cuando una aplicación pasa datos a TCP, éste decide entre enviarlos de inmediato o almacenarlos en el búfer (recolectar una mayor cantidad y enviar todos los datos al mismo tiempo). Sin embargo, algunas veces la aplicación necesita que los datos se envíen de inmediato, usándose la bandera PUSH que se transporta en los paquetes. Pero las aplicaciones no pueden establecer literalmente la bandera PUSH cuando envían datos. En cambio, los distintos sistemas operativos han desarrollado distintas opciones para agilizar la transmisión (por ejemplo, TCP-NODELAY en Windows y Linux).
También existe una característica que se usa muy raras veces, llamada datos urgentes. La aplicación emisora usara la bandera URGENT, provocando que TCP deje de acumular datos y transmita de inmediato todo lo que tiene para esa conexión. La aplicación receptora se interrumpe para poder parar lo que estaba haciendo y leer el flujo de datos para encontrar los datos urgentes. La aplicación solo sabe cuando terminan los datos urgentes y tiene que averiguar cuando comienza.

VII. EL PROTOCOLO TCP

Cada byte de una conexión TCP tiene su propio número de secuencia de 32 bits. Los números de secuencia separados de 32 bits se transmiten en paquetes para la posición de ventana deslizante en una dirección, y para las confirmaciones de recepción en la dirección opuesta.
La entidad TCP emisora y receptora intercambian datos en forma de segmentos. Un segmento TCP consiste en un encabezado fijo de 20 bytes (más una parte opcional), seguido de cero o más bytes de datos. El software de TCP decide qué tan grandes deben ser los segmentos. Puede acumular datos de varias escrituras para formar un segmento, o dividir los datos de una escritura en varios segmentos. Hay dos límites que restringen el tamaño de segmento. Primero, cada segmento, incluido el encabezado TCP, debe caber en la carga útil de 65515 bytes del IP. Segundo, cada enlace tiene una MTU (Unidad Máxima de Transferencia). Cada segmento debe caber en la MTU en el emisor y el receptor, de modo que se pueda enviar y recibir en un solo paquete sin fragmentar. En la práctica, la MTU es por lo general de 1500 bytes (el tamaño de la carga útil en Ethernet) y, por tanto, define el límite superior en el tamaño de segmento.
Las implementaciones modernas de TCP realizan el descubrimiento de MTU de la ruta mediante el uso de una técnica que usa mensajes de error de ICMP para encontrar la MTU más pequeña para cualquier enlace en la ruta. Después, TCP ajusta el tamaño del segmento en forma descendente para evitar la fragmentación.
El protocolo básico que utilizan las entidades TCP es el protocolo de ventana deslizante con un tamaño dinámico de ventana. Cuando un emisor transmite un segmento, también inicia un temporizador. Cuando llega el segmento al destino, la entidad TCP receptora devuelve un segmento (con datos si existen) que contiene un número de confirmación de recepción igual al siguiente número de secuencia que espera recibir, junto con el tamaño de la ventana remanente. Si el temporizador del emisor expira antes de recibir la confirmación de recepción, el emisor transmite de nuevo el segmento.
Pero, los segmentos pueden llegar fuera de orden sin que se envíe su respectiva confirmación de recepción. Además, los segmentos se pueden retardar tanto tiempo en tránsito que el temporizador del emisor expirará y este último retransmitirá los segmentos. Las retransmisiones podrían incluir rangos de bytes diferentes a los de la transmisión original, por lo cual se requiere una administración cuidadosa para llevar el control de los bytes que se han recibido de manera correcta en un momento determinado. Sin embargo, esto es factible ya que cada byte del flujo tiene su propio desplazamiento único.

VIII. EL ENCABEZADO DEL SEGMENTO TCP

Cada segmento comienza con un encabezado de formato fijo de 20 bytes. El encabezado fijo puede ir seguido de encabezado de opciones. Después de las opciones, si las hay, pueden continuar hasta 65535-20-20 = 65495 bytes de datos, donde los primeros 20 se refieren al encabezado IP y

los segundos al encabezado TCP. Los segmentos sin datos son legales y se usan por lo común para confirmaciones de recepción y mensajes de control. El encabezado TCP se muestra en la figura 2.
Los campos Puerto de origen y Puerto de destino identifican los puntos terminales locales de la conexión.
Los campos Número de secuencia y Número de confirmación de recepción (especifica el siguiente byte en el orden esperado, no el último byte de manera correcta recibido) desempeñan sus funciones normales.
La Longitud del encabezado TCP indica la cantidad de palabras de 32 bits contenidas en el encabezado TCP.
A continuación, vienen ocho banderas de 1 bit. CWR y ECE se utilizan para indicar congestión cuando se usa ECN (Notificación Explícita de Congestión). ECE se establece para indicar una Eco de ECN (ECN-Echo) a un emisor TCP y decirle que reduzca su velocidad cuando hay congestión. CWR se establece para indicar una Ventana de congestión reducida del emisor TCP al receptor TCP.
URG se establece en 1 si está en uso el Apuntador urgente. El bit ACK se establece en 1 para indicar que el Número de confirmación de recepción es válido.
El bit PSH indica datos que se deben transmitir de inmediato (PUSHed data).
El bit RST se usa para restablecer de manera repentina una conexión que se ha confundido debido a una falla de host o alguna otra razón. También se usa para rechazar un segmento no válido o un intento de abrir una conexión.
El bit SYN se usa para establecer conexiones. En esencia, el bit SYN se usa para denotar CONNECTION REQUEST y CONNECTION ACCEPTED, y el bit ACK sirve para distinguir entre ambas posibilidades.
El bit FIN se usa para liberar una conexión y especifica que el emisor no tiene más datos que transmitir.
El control de flujo en TCP se maneja mediante una ventana deslizante de tamaño variable. El campo Tamaño de ventana indica la cantidad de bytes que se pueden enviar, comenzando por el byte cuya recepción se ha confirmado.
En TCP, las confirmaciones de recepción y los permisos para enviar datos adicionales son por completo independientes. En efecto, un receptor puede decir: ”He recibido bytes hasta k, pero por ahora no deseo más”. Esta independencia (de hecho, una ventana de tamaño variable) proporciona una flexibilidad adicional. También se proporciona una Suma de verificación para agregar confiabilidad.
El campo Opciones ofrece una forma de agregar las características adicionales que no están cubiertas por el encabezado normal. Cada opción tiene una codificación Tipo-Longitud-Valor (TLV). Por ejemplo: se puede especificar el MSS (Tamaño Máximo de Segmento) que está dispuesto a aceptar; se puede transmite una estampa de tiempo enviada por el emisor y repetida por el receptor; también indicar al emisor los rangos de números de secuencia que ha recibido con la opcion SACK (Confirmación de Recepción Selectiva).

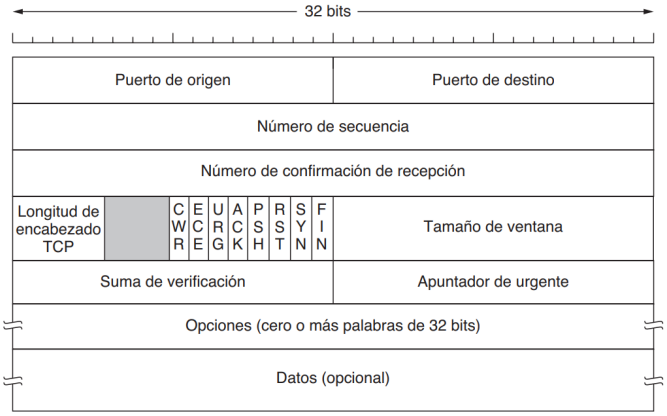
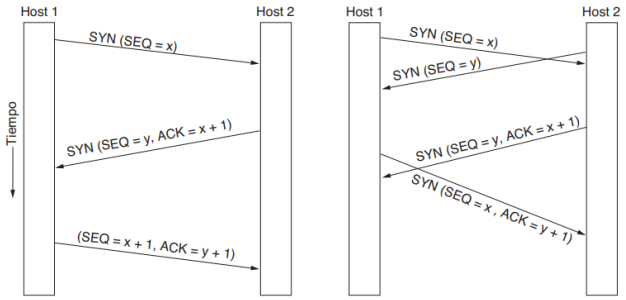


Fig. 2: El encabezado TCP.

IX. ESTABLECIMIENTO DE UNA CONEXIÓN TCP

Las conexiones se establecen mediante el acuerdo de tres vías. Para establecer una conexión, uno de los lados (digamos que el servidor) espera en forma pasiva una conexión entrante mediante la ejecución de las primitivas LISTEN y ACCEPT en ese orden, ya sea que se especifique un origen determinado o a nadie en particular.
El otro lado (digamos que el cliente) ejecuta una primitiva CONNECT en la que especifica la dirección y el puerto con el que se desea conectar, el tamaño máximo de segmento TCP que está dispuesto a aceptar y de manera opcional algunos datos de usuario (por ejemplo, una contraseña). La primitiva CONNECT envía un segmento TCP con el bit SYN encendido y el bit ACK apagado, y espera una respuesta.
Cuando este segmento llega al destino, la entidad TCP de ahí revisa si hay un proceso que haya ejecutado una primitiva LISTEN en el puerto que se indica en el campo Puerto de destino. Si no lo hay, envía una respuesta con el bit RST encendido para rechazar la conexión.

Si algún proceso está escuchando en el puerto, ese proceso recibe el segmento TCP entrante y puede entonces aceptar o rechazar la conexión. Si la acepta, se devuelve un segmento de confirmación de recepción. La secuencia de segmentos TCP enviados en el caso normal se muestra en la figura 3a. Un segmento SYN consume 1 byte de espacio de secuencia, por lo que se puede reconocer sin ambigüedades.
En el caso en que dos hosts intenten establecer al mismo tiempo una conexión entre los mismos dos, sólo se establece una conexión, pues las conexiones se identifican por sus puntos terminales. La secuencia para este caso se muestra en la figura 3b.
Para protegerse contra los paquetes duplicados retardado, el número de secuencia inicial elegido por cada host se debe reiniciar con lentitud en vez de ser una constante tal como 0. En un principio esto se lograba mediante un esquema basado en reloj, en donde éste emitía un pulso cada 4 µseg. Sin embargo, una vulnerabilidad al implementar el acuerdo de tres vías es que el proceso de escucha debe recordar su



(a) Caso normal. (b) Simultánea en ambos lados.

Fig. 3: Establecimiento de una conexión TCP.

número de secuencia tan pronto como responde con su propio segmento SYN. Esto significa que un emisor malicioso puede ocupar los recursos en un host si envía un flujo continuo de segmentos SYN y nunca les da seguimiento para completar la conexión.

Para defenderse, se puede usar la técnica SYN cookies. En vez de recordar el número de secuencia, un host selecciona un número de secuencia generado en forma criptográfica, lo coloca en el segmento de salida y se olvida de él. Si se completa el acuerdo de tres vías, este número de secuencia (más 1) se devolverá al host. Así, para regenerar el número de secuencia correcto hay que ejecutar la misma función criptográfica, siempre y cuando se conozcan las entradas para esa función; por ejemplo, la dirección IP y puerto del otro host, además de un secreto local. Existen algunos riesgos, como la incapacidad de manejar opciones TCP, por lo que la técnica SYN cookies sólo se puede usar cuando el host es sujeto de una inundación SYN.

X. LIBERACIÓN DE UNA CONEXIÓN TCP

Aunque las conexiones TCP son full dúplex, para entender la manera en que se liberan las conexiones es mejor visualizarlas como un par de conexiones simplex. Cada conexión simplex se libera de manera independiente de su igual. Para liberar una conexión, cualquiera de las partes puede enviar un segmento TCP con el bit FIN establecido, lo que significa que no tiene más datos por transmitir. Al confirmarse la recepción de FIN, se apaga ese sentido para que no se transmitan nuevos datos. Sin embargo, lo datos pueden seguir fluyendo de manera indefinida por el otro sentido. Cuando se apagan ambos sentidos, se libera la conexión. No hay diferencia entre la liberación secuencial o simultánea por parte de los hosts. Por lo general se requieren cuatro segmentos TCP para liberar una conexión: un FIN y un ACK para cada sentido. Es posible que se reduzca a tres, ya que el primer ACK y el segundo FIN estén contenidos en el mismo segmento.

Para evitar el problema de los dos ejércitos, se usan temporizadores. Si no llega una respuesta a un FIN en un máximo de dos tiempos de vida del paquete, el emisor del FIN libera la conexión. Tarde o temprano el otro lado notará que, al parecer, ya nadie lo está escuchando, y también expirará su temporizador. Aunque esta solución no es perfecta, en la práctica, pocas veces ocurren problemas.

XI. MODELADO DE ADMINISTRACIÓN DE CONEXIONES TCP

Los pasos requeridos para establecer y liberar conexiones se pueden representar en una máquina de estados finitos con los 11 estados. En cada estado son legales ciertos eventos. Al ocurrir un evento legal, debe emprenderse alguna acción. Si ocurre algún otro evento, se reporta un error.

Cada conexión comienza en el estado CLOSED (cerrado) y deja ese estado cuando hace una apertura pasiva (LISTEN) o una apertura activa (CONNECT). Si el otro lado realiza la acción opuesta, se establece una conexión y el estado se vuelve ESTABLISHED (establecida). La liberación de la conexión se puede iniciar desde cualquiera de los dos lados. Al completarse, el estado regresa a CLOSED.

La máquina de estados finitos se muestra en la figura 4.

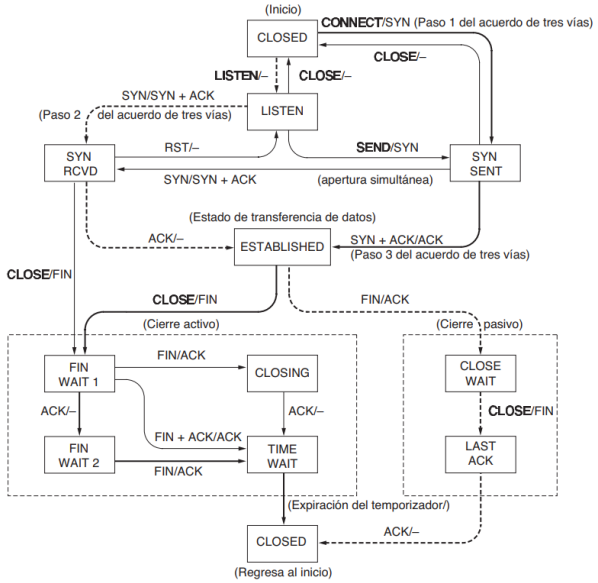


Fig. 4: Máquina de estados finitos para administrar las conexiones TCP. La línea continua gruesa es la trayectoria normal para un cliente. La línea punteada gruesa es la trayectoria normal para un servidor. Las líneas delgadas son eventos no usuales. Cada transición se etiqueta con el evento que la ocasiona y la acción resultante, separada por una diagonal.

Cuando un programa de aplicación en la máquina cliente emite una solicitud CONNECT, la entidad TCP local crea un registro de conexión, lo marca para indicar que está en el estado SYN SENT y envía un segmento SYN. Cabe mencionar que muchas conexiones pueden estar abiertas (o en proceso de apertura) al mismo tiempo como parte de varias aplicaciones, por lo que el estado es por conexión y se graba en el registro de conexiones. Al llegar el SYN + ACK, TCP envía el último ACK del acuerdo de tres vías y

cambia al estado ESTABLISHED. Ahorase pueden enviar y recibir datos.
Cuando una aplicación ha terminado, ejecuta una primitiva CLOSE; esto hace que la entidad TCP local envíe un segmento FIN y espere el ACK correspondiente (recuadro "cierre activo"). Al llegar el ACK, se hace una transición al estado FIN WAIT 2 y se cierra un sentido de la conexión. Cuando cierra también el otro lado llega un FIN, para el cual se envía una confirmación de recepción. Ahora ambos lados están cerrados, pero el TCP espera un tiempo igual al doble del tiempo de vida máximo del paquete para garantizar que todos los paquetes de la conexión hayan expirado, sólo por si acaso se perdió la confirmación de recepción. Al expirar el temporizador, TCP borra el registro de la conexión.
El servidor emite una solicitud LISTEN y se detiene a esperar a que aparezca alguien. Cuando llega un SYN, se envía una confirmación de recepción y el servidor pasa al estado SYN RCVD. Cuando llega la confirmación de recepción del SYN del servidor, el acuerdo de tres vías se completa y el servidor pasa al estado ESTABLISHED. Ahora puede ocurrir la transferencia de datos.
Cuando el cliente termina de transmitir sus datos, emite una solicitud CLOSE; esto provoca la llegada de un FIN al servidor (recuadro punteado con la leyenda "cierre pasivo"). Entonces se envía una señal al servidor. Cuando éste también emite una solicitud CLOSE, se envía un FIN al cliente. Al llegar la confirmación de recepción del cliente, el servidor libera la conexión y elimina el registro de conexión.

XII. VENTANA DESLIZANTE TCP

La administración de ventanas en un TCP separa los aspectos de la confirmación de la recepción correcta de los segmentos y la asignación del búfer en el receptor.
Cuando la ventana es de 0, el emisor no puede enviar segmentos, salvo en dos situaciones. En primer lugar, se pueden enviar datos urgentes. En segundo lugar, para enviar una sonda de ventana, donde el emisor puede enviar un segmento de 1 byte para hacer que el receptor vuelva a anunciar el siguiente byte esperado y el tamaño de la ventana. El estándar TCP proporciona explícitamente esta opción para evitar un interbloqueo si llega a perderse una actualización de ventana.
No se requiere que los emisores transmitan datos tan pronto como llegan de la aplicación, y que los receptores envíen confirmaciones de recepción tan pronto como sea posible.
Para optimizar, se utilizan las confirmaciones de recepción con retardo. La idea es retrasar las confirmaciones de recepción y las actualizaciones de ventana por hasta 500 mseg, con la esperanza de que lleguen algunos datos con los cuales se pueda viajar de manera gratuita.
El algoritmo de Nagle se utiliza para reducir ineficiencia producida por un emisor que envía varios paquetes cortos. Cuando llegan datos en pequeñas piezas al emisor, sólo se envía la primera pieza y el resto se almacena en búfer hasta que se confirma la recepción del byte pendiente. Después se envían todos los datos del búfer en un segmento TCP y nuevamente comienzan a almacenarse en búfer los datos hasta que se haya confirmado la recepción del siguiente segmento. Sólo puede haber un paquete corto pendiente en cualquier momento dado. Este algoritmo se tiene que deshabilitar en, por ejemplo, juegos interactivos que operan a través de Internet, que necesitan un flujo rápido de paquetes cortos de actualización. Además, se puede deshabilitar (lo cual se conoce como la opción TCP-NODELAY) por provocar un interbloqueo temporal al interactuar con las confirmaciones de recepción con retardo (suele retardar las descargas de las páginas web).
El síndrome de ventana tonta puede arruinar el desempeño de TCP, ocurre cuando se pasan datos a la entidad TCP emisora en bloques grandes, pero una aplicación interactiva del lado receptor lee datos sólo a razón de 1 byte a la vez. La solución de Clark obliga a esperar hasta tener disponible una cantidad decente de espacio, y luego lo anuncia, evitando que el receptor envíe una actualización de ventana para 1 byte. El receptor no debe enviar una actualización específica de ventana hasta que pueda manejar el tamaño máximo de segmento que anunció al establecerse la conexión, o hasta que su búfer quede a la mitad de capacidad, lo que sea más pequeño. Además, el emisor también puede ayudar al no enviar segmentos muy pequeños. En cambio, debe esperar hasta que pueda enviar un segmento completo, o por lo menos uno que contenga la mitad del tamaño del búfer del receptor.
El algoritmo de Nagle y la solución de Clark al síndrome de ventana tonta son complementarios y pueden operar juntos, logrando que el emisor no envíe segmentos pequeños y que el receptor no los pida.
El receptor TCP para mejorar el desempeño puede actualizar ventanas en unidades grandes y además tiene la capacidad de almacenar datos en el búfer, por lo que puede bloquear una solicitud READ de la aplicación hasta que pueda proporcionarle un bloque grande de datos. Reduciendo la cantidad de llamadas a TCP (y la sobrecarga), pero aumentando el tiempo de respuesta.
Para solucionar que los segmentos lleguen fuera de orden, el receptor colocará los datos en el búfer hasta que se puedan pasar en orden a la aplicación. No ocurriría nada malo si se descartaran los segmentos fuera de orden, ya que el emisor los retransmitiría en un momento dado, pero sería un desperdicio.
Las confirmaciones de recepción se pueden enviar sólo después de haber recibido todos los datos hasta el byte confirmado. A esto se le conoce como confirmación de recepción acumulativa.

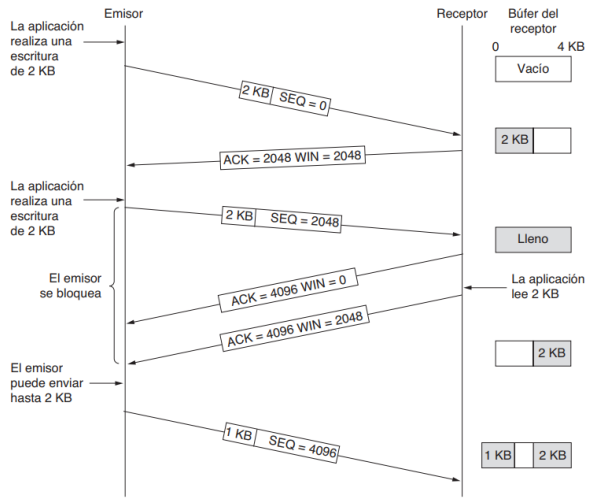


Fig. 5: Administración de ventanas en TCP.

XIII. ADMINISTRACIÓN DE TEMPORIZADORES EN TCP

TCP usa varios temporizadores. El más importante de éstos es el RTO (Temporizador de Retransmisión). Cuando se envía un segmento, se inicia un temporizador de retransmisiones. Si la confirmación de recepción llega antes de que expire el temporizador, éste se detiene. En caso contrario, se retransmite el segmento (y se inicia de nuevo el temporizador).

El retardo esperado, en la capa de enlace de datos, se mide en microsegundos y es muy predecible, por lo que el temporizador se puede establecer para expirar justo después del momento en que se esperaba la confirmación de recepción. Dado que las confirmaciones de recepción pocas veces se retardan debido a la falta de congestión, la ausencia de una confirmación significa que se perdió la trama o la confirmación de recepción. Se muestra en la figura 6a.

En cambio, TCP, se enfrenta a un entorno radicalmente distinto. Es complicado determinar el tiempo de ida y vuelta al destino, incluso cuando se conoce, es difícil decidir sobre el intervalo de expiración del temporizador. Si se establece demasiado corto (T1), ocurrirán retransmisiones innecesarias e Internet se llenará de paquetes inútiles. Si se establece demasiado largo (T2), el desempeño sufrirá debido al largo retardo de retransmisión de cada paquete perdido. Se muestra en la figura 6b.

La solución es usar un algoritmo dinámico que ajuste de manera constante el intervalo de expiración del temporizador, con base en mediciones continuas del desempeño de la red. El algoritmo utilizado por lo general por el TCP se lo debemos a Jacobson. Por cada conexión, TCP mantiene una variable llamada SRTT (Tiempo de Ida y Vuelta Suavizado). Funciona con una fórmula del tipo EWMA (Promedio Móvil Ponderado Exponencialmente),

$$SRTT = \alpha SRTT + (1 - \alpha)R \quad (1)$$

donde alfa es un factor de suavizado que determina la rapidez con que se olvidan los valores anteriores (suele valer 7/8) y R el tiempo que tardó la confirmación de recepción. Tambien se lo conoce como filtro pasa bajas, que descarta el ruido en las muestras.

Incluso con un buen valor de SRTT, seleccionar la expiración adecuada del temporizador de retransmisión no es un asunto sencillo. Jacobson propuso hacer que el valor de expiración del temporizador fuera sensible a la diferencia en los tiempos de ida y vuelta, así como al tiempo de ida y vuelta suavizado. RTTVAR (Variación de Tiempo de Ida y Vuelta) que se actualiza mediante la siguiente fórmula:

$$RTTVAR = \beta RTTVAR + (1 - \beta)|SRTT - R| \quad (2)$$

Ésta es también una fórmula EWMA (Beta = 3/4). El tiempo de expiración de retransmisión, RTO, se establece así:

$$RTO = SRT + 4RTTVAR \quad (3)$$

Esta economía no es necesaria para los hosts modernos.

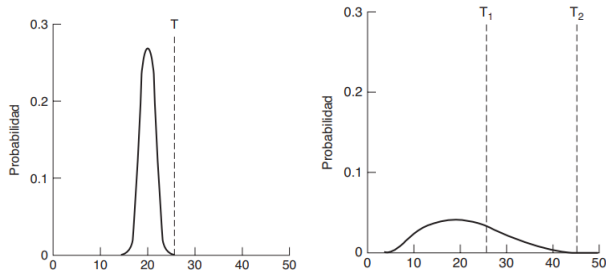
El temporizador de retransmisión se mantiene en un mínimo de 1 segundo, sin importar las estimaciones. Éste es un valor conservador que se seleccionó para evitar retransmisiones espurias con base en las mediciones.

Un problema que ocurre con la recopilación de las muestras, R, del tiempo de ida y vuelta es qué hacer cuando expira el temporizador de un segmento y se envía de nuevo. Cuando llega la confirmación de recepción, no está claro si ésta se refiere a la primera transmisión o a una posterior. Si adivinamos mal se puede contaminar seriamente el temporizador de retransmisión. Karn hizo una propuesta sencilla: no actualizar las estimaciones sobre ninguno de los segmentos retransmitidos. Además, se duplicará el tiempo de expiración con cada retransmisión sucesiva hasta que los segmentos pasen a la primera. Esta corrección se conoce como algoritmo de Karn. La mayoría de las implementaciones de TCP lo utilizan.

El temporizador de persistencia está diseñado para evitar el interbloqueo, donde el receptor envía una confirmación de recepción con un tamaño de ventana de 0 para indicar al emisor que espere. Después el receptor actualiza la ventana, pero se pierde el paquete con la actualización. Ahora, tanto el emisor como el receptor están esperando a que el otro haga algo. Cuando expira el temporizador de persistencia, el emisor transmite un sondeo al receptor. La respuesta al sondeo proporciona el tamaño de la ventana. Si aún es cero, se inicia el temporizador de persistencia una vez más y se repite el ciclo. Si es diferente de cero, ahora se pueden enviar datos.

Cuando una conexión ha estado inactiva durante demasiado tiempo, el temporizador de seguir con vida (keepalive) puede expirar para ocasionar que un lado compruebe que el otro aún está ahí. Si no se recibe respuesta, se termina la conexión. Esta característica es controversial puesto que agrega sobrecarga y puede terminar una conexión saludable debido a una partición temporal de la red.

El último temporizador que se utiliza en cada conexión TCP es el que se usa en el estado TIME WAIT durante el cierre. Opera durante el doble del tiempo máximo de vida de



(a) Capa de enlace de datos. (b) TCP.

Fig. 6: Densidad de probabilidad de los tiempos de llegada de las confirmaciones de recepción.

paquete para asegurar que, al cerrarse una conexión, todos los paquetes creados por ella hayan desaparecido.

XIV. CONTROL DE CONGESTIÓN EN TCP

Cuando la carga ofrecida a cualquier red es mayor que la que puede manejar, se genera una congestión. La capa de red detecta la congestión cuando las colas crecen demasiado en los enrutadores y trata de lidiar con este problema, aunque lo único que haga sea descartar paquetes. Es responsabilidad de la capa de transporte recibir la retroalimentación de congestión de la capa de red y reducir la tasa del tráfico que envía a la red. En Internet, TCP desempeña el papel principal en cuanto al control de la congestión, así como en el transporte confiable.

El control de congestión de TCP se basa en la ley de control AIMD (Incremento Aditivo/ Decremento Multiplicativo), que en respuesta a las señales de congestión binarias provenientes de la red, converge hacia una asignación de ancho de banda equitativa y eficiente. Lo implementa mediante el uso de una ventana y con la pérdida de paquetes como la señal binaria.

La ventana de congestión se mantiene además de la ventana de control de flujo, la cual especifica el número de bytes que el receptor puede colocar en el búfer. Ambas ventanas se rastrean en paralelo, y el número de bytes que se puede enviar es el número que sea menor de las dos ventanas. Por esto, la ventana efectiva es la cantidad más pequeña de lo que tanto el emisor como el receptor consideran que está bien. TCP dejará de enviar datos si la ventana de congestión o la de control de flujo está temporalmente llena.

El control de congestión moderno, se agregó por Van Jacobson, el cual implementó la corrección de aproximar una ventana de congestión AIMD. Observó que la pérdida de paquetes es una señal adecuada de congestión, para esto es necesario que los errores de transmisión sean relativamente raros. Por lo general esto no es así para los enlaces inalámbricos, lo cual explica por qué incluyen su propio mecanismo de retransmisión en la capa de enlace. Corrigió los temporizadores de TCP que incluye estimaciones de la media y la variación en los tiempos de ida y vuelta, mediante la inclusión del factor de variación. Si se tiene un buen temporizador de retransmisión, el emisor de TCP puede rastrear el número pendiente de bytes que están cargando la red. Simplemente analiza la diferencia entre los números de secuencia que se transmiten y los números de secuencia cuya confirmación de recepción se ha enviado. Todo lo que tenemos que hacer es rastrear la ventana de congestión mediante el uso de los números de secuencia y los que ya se han confirmado, y ajustar la ventana de congestión mediante el uso de una regla AIMD. Pero es más complicado que eso.

Una de las primeras consideraciones a tener en cuenta, es que la forma en que se envían los paquetes a la red, incluso a través de periodos cortos, debe coincidir con la trayectoria de red. En caso contrario, el tráfico provocará una congestión. Podemos usar pequeñas ráfagas de paquetes para nuestra ventaja. Mediante el uso de un reloj de confirmación de recepción (ack clock), TCP regula el tráfico y evita las colas innecesarias en los enrutadores.

Una segunda consideración es que la regla AIMD tardará mucho tiempo para llegar a un buen punto de operación en las redes rápidas si la ventana de congestión se inicia a partir de un tamaño pequeño. Hay que esperar mucho tiempo sólo para llegar a la velocidad correcta para una transferencia.

Jacobson, para manejar ambas consideraciones, eligió hacer una mezcla de incremento lineal y de incremento multiplicativo, conocido como inicio lento (slow start). No es para nada lento (su crecimiento es exponencial), excepto en comparación con el algoritmo anterior que permitía enviar toda una ventana de control de flujo al mismo tiempo. Hace que el emisor siga la sincronización del reloj de confirmación de recepción mientras inyecta nuevos paquetes. Si la trayectoria de la red es lenta, las confirmaciones de recepción llegarán con lentitud (después de un retardo de un RTT). Si la trayectoria de la red es rápida, las confirmaciones de recepción llegarán con rapidez (de nuevo, después del RTT).

Al imponer un espaciamiento mínimo entre los paquetes de datos que llegan al receptor, también se mantiene el mismo espaciamiento cuando el receptor envía confirmaciones de recepción, y también cuando el emisor recibe esas confirmaciones.

Como el inicio lento provoca un crecimiento exponencial, en un momento dado (y más pronto que tarde) enviará demasiados paquetes a la red con mucha rapidez. Cuando esto ocurra, las colas se acumularán en la red. Cuando las colas estén llenas se perderán uno o más paquetes. Una vez que ocurra esto, el temporizador del emisor TCP expirará cuando una confirmación de recepción no llegue a tiempo.

Para mantener el inicio lento bajo control, el emisor mantiene un umbral para la conexión, conocido como umbral de inicio lento. En un principio este valor se establece en un nivel arbitrariamente alto, al tamaño de la ventana de control de flujo, de manera que no limite la conexión. TCP continúa incrementando la ventana de congestión en el inicio lento hasta que expira un temporizador o la ventana de congestión excede el umbral (o cuando se llena la ventana del receptor). Cada vez que se atraviesa el umbral, TCP cambia del inicio lento al incremento aditivo. En este modo, la ventana de congestión se incrementa un segmento por cada tiempo de ida y vuelta. Al igual que el inicio lento, por lo general esto se

implementa mediante un incremento por cada segmento cuya recepción se ha confirmado, en vez de usar un incremento una vez por cada RTT. La idea general es que una conexión TCP pase mucho tiempo con su ventana de congestión cerca del valor óptimo: no tan pequeño como para que la tasa de transferencia real sea baja y no tan grande como para que ocurra una congestión.

En comparación con el inicio lento, la tasa lineal de crecimiento es mucho menor. No hay mucha diferencia para las ventanas de congestión pequeñas, como en este caso, sino una gran diferencia en el tiempo que se requiere para hacer crecer la ventana de congestión hasta 100 segmentos, por ejemplo.

El defecto en el esquema hasta ahora es esperar a que expire un temporizador. Los tiempos de expiración son algo largos debido a que deben ser conservadores. Después de perder un paquete, el receptor no puede confirmar la recepción más allá de ese paquete, por lo que el número de confirmación de recepción permanecerá fijo y el emisor no podrá enviar nuevos paquetes a la red, debido a que su ventana de congestión permanecerá llena. Esta condición puede continuar durante un periodo relativamente largo, hasta que el temporizador se dispare y se retransmita el paquete perdido. En ese punto, TCP vuelve a empezar con el inicio lento.

Una forma rápida de que el emisor reconozca que se ha perdido uno de sus paquetes es que, a medida que llegan al receptor los paquetes subsiguientes al perdido, activan confirmaciones de recepción que regresan al emisor. Estas confirmaciones de recepción contienen el mismo número de confirmación de recepción y, por ende, se denominan confirmaciones de recepción duplicadas. Cada vez que el emisor recibe una confirmación de recepción duplicada, es probable que haya llegado otro paquete al receptor y que el paquete perdido todavía no aparezca.

Puede que los paquetes lleguen fuera de orden, esto activará confirmaciones de recepción duplicadas, incluso aunque no se hayan perdido paquetes. Pero es poco común en Internet. Cuando hay un reordenamiento a través de múltiples trayectorias, por lo general los paquetes recibidos no se reordenan demasiado. Así, TCP asume con cierta arbitrariedad que tres confirmaciones de recepción duplicadas indican que se ha perdido un paquete. La identidad del paquete perdido se puede inferir también del número de confirmación de recepción. Es el siguiente paquete inmediato en la secuencia. Entonces este paquete se puede retransmitir de inmediato, antes de que se dispare el temporizador de retransmisión. Esto se denomina retransmisión rápida.

Una vez que se dispara, el umbral de inicio lento sigue establecido a la mitad de la ventana de congestión actual, justo como cuando expira un temporizador. Para volver a comenzar con el inicio lento, hay que establecer la ventana de congestión a un paquete. Con este tamaño de ventana, se enviará un nuevo paquete después del tiempo de ida y vuelta que se requiere para confirmar la recepción del paquete retransmitido, junto con los datos que se habían enviado antes de detectar la pérdida.

El TCP Tahoe (que incluía buenos temporizadores de retransmisión), ofrecía un algoritmo de control de congestión funcional que resolvió el problema del colapso por congestión.

Jacobson lo mejoró, dando como resultado que TCP evite el inicio lento, excepto cuando la conexión se inicia por primera vez y cuando expira un temporizador. Esto último puede aún ocurrir cuando se pierde más de un paquete y la retransmisión rápida no recupera en forma adecuada. En vez de inicios lentos repetidos, la ventana de congestión de una conexión funcional sigue un patrón de diente de sierra de incremento aditivo (un segmento por cada RTT) y decremento multiplicativo (la mitad en un RTT). Ésta es exactamente la regla AIMD que buscábamos implementar. El TCP Reno es el TCP Tahoe más la recuperación rápida.

La comparación entre TCP Tahoe y TCP Reno se observa en la figura 7.

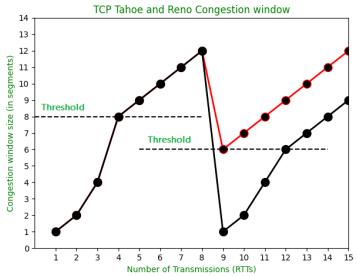


Fig. 7: TCP Tahoe (en negro) y TCP Reno (en rojo).

XV. RESULTADOS

Al finalizar este escrito podemos concluir que es un protocolo confiable orientado a la conexión que permite que un flujo de bytes originado en una máquina se entregue sin errores a cualquier otra máquina en la interred. Este protocolo segmenta el flujo de bytes entrante en mensajes discretos y pasa cada uno a la capa de interred. En el destino, el proceso TCP receptor vuelve a ensamblar los mensajes recibidos para formar el flujo de salida. El TCP también maneja el control de flujo para asegurar que un emisor rápido no pueda inundar a un receptor lento con más mensajes de los que pueda manejar.

XVI. CONCLUSIONES

El protocolo TCP se ha utilizado para muchas aplicaciones y se ha extendido en el transcurso del tiempo para brindar un buen desempeño a través de un amplio rango de redes. Existen muchas versiones en uso con implementaciones un poco distintas de los algoritmos clásicos que hemos descrito, en especial para el control de la congestión y la robustez ante los ataques. Y sin dudas ha sido parte esencial en la revolución causada por el Internet desde su creación por sus características principales descritas en la sección XV.

REFERENCES

[1] Andrew S. Tanenbaum (2012), Redes de computadoras", Quinta edición, de Pearson Educación, México.

[2] Pintu Saini (2022), "TCP Tahoe and TCP Reno", de Geeksforgeeks, India.