Protocolo UDP

Redes

UNIVERSIDAD SIGLO 21 | MIEMBRO DE LA RED ILUMNO

Capa de Transporte

User Datagram Protocol o protocolo de datagrama de usuario trabaja en la capa de Transporte. Esta capa define una comunicación entre procesos de dos computadoras, denominadas cliente y servidor. Mientras que la capa de enlace permite la comunicación entre dos dispositivos físicos interconectados y la capa de red hace lo propio entre dos dispositivos distantes (entre redes), esta capa permite que dos aplicaciones interactuen como si estuvieran formando parte del mismo dispositivo físico, cuando en realidad están distantes.

Un switch se encarga de procesar y reenviar tramas Ethernet en una red LAN. Un Router enruta paquetes para que viajen hacia entre redes. Finalmente, los clientes y servidores crean una conexión lógica en la capa de transporte.

Otros protocolos de esta capa serán abordados en otras lecturas.

Paradigma cliente-servidor

En el modelo cliente-servidor, dos computadoras interactuan: un server o servidor ejecuta un proceso de forma continua, mientras que los clients o clientes se conectan desde cualquier ubicación alcanzable a través de un protocolo de Red (IP en general). El modelo permite que múltiples clientes puedan conectarse en forma simultánea para utilizar un servicio ofrecido por el servidor. Estos servicios son provistos por la capa de Aplicación y algunos ejemplos son: servicio de acceso remoto (SSH), servicio de transferencia de archivos (FTP) o el servicio para sitios web (WWW).

Puertos

En este momento puede surgir la duda: ¿cómo se identifica el proceso al que quiere acceder el cliente? ¿Cómo es posible que múltiples usuarios accedan al mismo proceso?

Tanto el proceso que corren en el servidor, como el que se crea en el cliente para poder acceder al servicio ofrecido por el servidor se identifican con un número que ocupa 16 bits, lo que da lugar a números entre 0 y 65535. Para que un cliente y un servidor se comuniquen entre si e intercambien mensajes es necesario conocer no solo las direcciones IP de cliente y destino la IP del servidor, sino que será también necesario conocer el puerto que usa la aplicación del servidor y el puerto que usa la aplicación del cliente.

Para evitar tener que averiguar que puerto utiliza cada aplicación, se han definido entre el 1 y el 1023 los “puertos bien conocidos”, es decir que las aplicaciones más utilizadas ya tienen un número de puerto asignado, por lo

que los clientes lo conocen de antemano. Estos números son controlados por ICANN.
En el caso del número de puerto que usan los clientes, estos son asignados por el sistema operativo siendo su número mayor a 49152. Los números entre 1024 y 49151 no son asignados ni controlados por ICANN, pero pueden ser registrados para evitar duplicaciones.

Figura 1: Algunos puertos "bien conocidos"

Port	Protocol	UDP	TCP	SCTP	Description
7	Echo	√	√	√	Echoes back a received datagram
9	Discard	√	√	√	Discards any datagram that is received
11	Users	√	√	√	Active users
13	Daytime	√	√	√	Returns the date and the time
17	Quote	√	√	√	Returns a quote of the day
19	Chargen	√	√	√	Returns a string of characters
20	FTP-data		√	√	File Transfer Protocol
21	FTP-21		√	√	File Transfer Protocol
23	TELNET		√	√	Terminal Network
25	SMTP		√	√	Simple Mail Transfer Protocol
53	DNS	√	√	√	Domain Name Service
67	DHCP	√	√	√	Dynamic Host Configuration Protocol
69	TFTP	√	√	√	Trivial File Transfer Protocol
80	HTTP		√	√	HyperText Transfer Protocol
111	RPC	√	√	√	Remote Procedure Call
123	NTP	√	√	√	Network Time Protocol
161	SNMP-server	√			Simple Network Management Protocol
162	SNMP-client	√			Simple Network Management Protocol

Fuente: Forouzan, 2013, p. 737

En la figura 1 se observan algunos puertos "well known". Es interesante notar que un mismo número de puerto puede ser utilizado por diferentes protocolos de la capa de transporte (UDP/TCP/SCTP). Las celdas grisadas significan que la aplicación no admite ese protocolo de la capa de transporte. Por ejemplo, SMTP no admite protocolo UDP.

Protocolo UDP

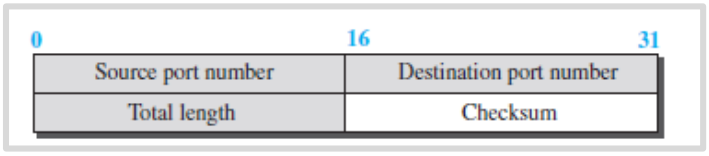
El protocolo UDP ofrece un tipo de servicio a la capa de aplciación: es no orientado a la conexión, y no confiable. Estas características son similares al protocolo IP por lo que no le agrega valor agregado y solamente cumple su función para hacer posible la comunicación proceso a proceso.

¿Para qué fue diseñado un protocolo así? Para ciertas situaciones resulta ventajoso que el protocolo sea simple y veloz. Por ejemplo, para enviar mensajes cortos o información en tiempo real, la cual pierde sentido si debe ser retransmitida. En cualquier caso, las tareas relacionadas a un eventual control de flujo o detección de pérdida de paquetes por ejemplo deberán ser realizadas por la capa superior (Aplicación).

Encabezado del datagrama

El encabezado del los datagramas UDP es muy simple. Se identifica a los puertos de origen y destino (16 bits cada campo), la longitud total (otros 16 bits) y finalmente una suma de verificación (16 bits).

Figura 2: Encabezado UDP



Fuente: Forouzan, 2013, p. 738

Checksum

El cálculo del checksum es diferente al realizado en el encabezado IP. Este último realiza una suma de verificación que abarca todo el encabezado IP (incluyendo su propio campo checksum que es puesto en cero para hacer posible dicho cálculo). En el caso de UDP, la suma de verificación cubre tres áreas:

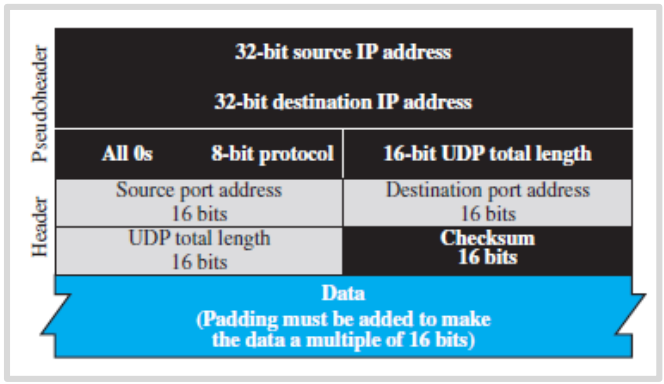
- El encabezado UDP
- El área de datos de UDP
- Un pseudoencabezado

¿Qué es y por qué se incluye este pseudoencabezado en el checksum?

Cómo en la capa de transporte es necesario identificar tanto direcciones IP como puertos, si se corrompen ciertos valores del encabezado IP y esta información no está disponible en el checksum UDP, el paquete IP pudo haber sido entregado a un host de destino diferente al que se suponía (cambió por ejemplo el campo IP de Destino del paquete IP) y UDP no detectaría problema alguno.

Los campos que se incluyen en el pseudoencabezado se observan en la figura 3. Están incluídas las direcciones IP de origen y de destino como así también el campo protocol, para poder identificar si en el encabezado IP estaba indicado que en la capa de transporte se utilizó UDP en el transmisor.

Figura 3: Checksum UDP



Fuente: Forouzan, 2013, p. 739

El checksum en UDP es opcional y puede suprimirse en el caso de que la aplicación considere que es beneficioso no utilizar tiempo de CPU para este cálculo. En el caso de que el checksum no se calcule, el campo debe ser rellenado con ceros.
En el caso de que si se calcule el checksum y su resultado sea 0, se invierten todos los bits a 1.

Funcionamiento

Los servicios que provee UDP son: comunicación proceso a proceso, sin conexión, sin control de flujo ni congestión y sin control de errores (solo se detectan con Checksum pero nada se hace).

Sin conexión significa cuando una aplicación utiliza UDP para enviar sus mensajes, estos serán enviados al receptor sin mediar intercambio alguno previo para que tanto transmisor como receptor puedan tomar conocimiento de la situación.

Sin control de flujo significa que si la velocidad con la que el transmisor envía sus datagramas es mayor a la velocidad con la que el receptor puede procesarlas, este último se saturará y UDP nada hará al respecto.
Algo similar puede ocurrir cuando existe congestión en la red. UDP no podrá detectarlo.

UDP solo realiza un checksum para detectar errores y descartar tramas, pero puede haber otro tipo de errores como por ejemplo datagramas duplicados o perdidos en la red. Estos inconvenientes no son detectados por UDP y junto a los anteriores deben ser solucionados por la capa de aplicación.

Manejo de mensajes

UDP implementa queues (colas) asociadas a los procesos que son abiertos por el Sistema Operativo tanto en el receptor como en el transmisor y son asociadas a los puertos de origen y destino. En la cola de salida del transmisor son colocados todos los mensajes que se envían o reciben.
Además, UDP multiplexa y demultiplexa ya que muchos servicios pueden requerir el uso del protocolo en el mismo dispositivo.

UDP-Lite

UDP-Lite (Lightweight User Datagram Protocol) o protocolo de datagrama de usuario ligero) es un protocolo basado en UDP pero que permite que un datagrama que ha sufrido cierta cantidad de daño en su área de datos sea considerado como válido en lugar de ser descartado por el error en el checksum.

Para lograr su objetivo, la cantidad de datos que cubre el checksum es variable por lo que la aplicación puede decidir cuánta información estará cubierta.

Este protocolo fue diseñado para ser utilizado por aplicaciones multimedia en donde es preferible recibir parte de la información a recibir nada. Pensemos en llamadas telefónicas que usan Voz sobre IP o Streaming de video. Recibir información incompleta reduce la calidad de la llamada o el video, pero no recibir nada es aún peor.

Referencias

Tanenbaum, A (2012). Los protocolos de transporte en Internet en Redes *de Computadoras.* Madrid: Editorial Pearson Education

Forouzan, B (2013). Introduction to Transport Layers en *Data Communications AND Networking.* Estados Unidos: McGraw-Hill

Forouzan, B (2013). Transport-Layer Protocols en *Data Communications AND Networking.* Estados Unidos: McGraw-Hill

IETF (1980). RFC768. User Datagram Protocol. Estados Unidos. https://www.ietf.org/rfc/rfc768.txt

IETF (1999). RFC768. Protocolo de Datagrama de Usuario (User Datagram Protocol). Estados Unidos. https://www.ietf.org/rfc/rfc768.txt

IETF (2004). RFC3828. The Lightweight User Datagram Protocol (UDP-Lite). Estados Unidos. https://tools.ietf.org/html/rfc3828

Protocolo TCP

Redes

UNIVERSIDAD SIGLO 21 | MIEMBRO DE LA RED ILUMNO

Protocolo de Controlde Transmisión (TCP)

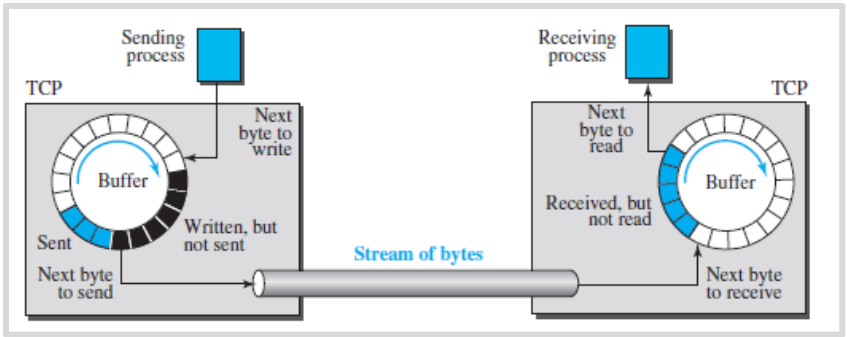
Las características principales del protocolo UDP son: no orientado a la conexión, sin manejo de errores, sin control de flujo y congestión. Si bien esto sirve para algunas aplicaciones, hay muchas otras que requieren lo opuesto: un protocolo orientado a la conexión y confiable. El protocolo TCP cumple con estas características y además provee mecanismos para control de flujo y congestión.

Servicios ofrecidos por TCP

TCP ofrece los siguientes servicios a la capa de aplicación: comunicación proceso a proceso y al igual que UDP, se utilizan puertos para identificar dichos procesos. Servicio de entrega de flujos de información, es decir que organiza la información en grupos de bytes, los cuales son enviados por el transmisor y recibidos por el receptor. Esto difiere de UDP, que solo se limita a agregar un encabezado a los datos y pasar el datagrama de usuario a la capa de red.

TCP utiliza buffers de transmisión y recepción, lo que significa que es posible enviar cierta cantidad de información mientas otra queda en espera en el buffer. En el receptor sucede algo similar: cuando la aplicación toma datos, los extra del buffer.

Figura 1: buffers en TCP



Fuente: Forouzan, 2013, p. 745

En la figura 1 se observa el funcionamiento de los buffers del transmisor (a la izquierda) y el receptor (a la derecha), ambos unidos a través de una tubería imaginaria por donde circulan los flujos de bytes.

En el transmisor, el buffer tiene 3 estados:

- Bytes que ya fueron enviados (celeste)

- Bytes que fueron escritos por la aplicación pero todavía no fueron enviados (negro)
- Bytes disponibles para que la aplicación coloque bytes (blanco).

Los bytes en celeste continúan en el buffer porque están esperando que el receptor confirme su recepción, con un denominado ACK (Acknowledge). Una vez que el transmisor recibe la o las confirmaciones, puede liberar ese espacio para que la aplicación pueda escribir, es decir que pasan de celeste a blanco.
Los bytes en negro serán enviados a la red hasta que se llegue al límite; en el ejemplo de la figura, 6 bytes pueden ser enviados hasta esperar ACK.

Este mecanismo permite que TCP sea un protocolo confiable. Los mensajes podrán perderse en la red, pero el transmisor lleva un timer cada vez que envía un mensaje, y si transcurrido cierto tiempo no recibe confirmación desde el receptor, se dará cuenta del problema y retransmitirá ese mensaje. Además, como estudiaremos mas adelante, el receptor puede disminuir el tamaño del buffer para evitar la saturación.

El buffer en el receptor está dividido en dos partes:

- Bytes recibidos pero no leídos por la aplicación (celeste)
- Bytes disponibles para ser recibidos (blanco)

Cuando los bytes son recibidos, el buffer pasa de blanco a celeste y continúa hasta que se complete el espacio. Al leer bytes la aplicación, va enviando ACK y liberando espacio (de celeste a blanco).

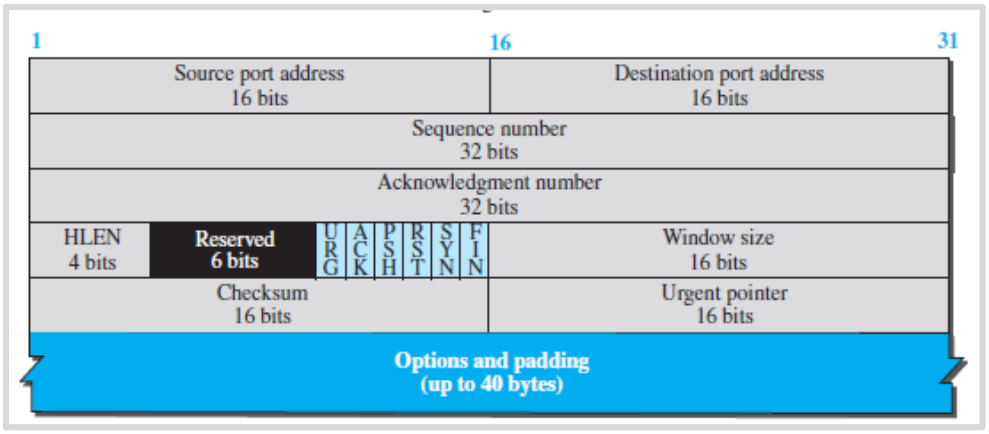
Otros servicios ofrecidos por TCP son:

- Comunicación full-duplex: emisor y transmisor manejan sus buffers y pueden transmitir y recibir en forma simultánea.
- Servicio orientado a la conexión: antes de poder intercambiar datos, TCP establece una conexión donde se gestiona información relevante para poder realizar el control de flujo y congestión. Cuando una de las partes decide terminar la comunicación, se envían otro tipo de mensajes.

Segmento TCP

Así como existe una trama Ethernet, un paquete IP y un datagrama de usuario, TCP define una estructura de paquete denominada segmento. Su estructura es mas compleja que la de UDP para que sea posible cumplir sus funciones.

Figura 2: segmentoTCP



Fuente: Forouzan, 2013, p. 748

Al igual que UDP, TCP contiene en su paquete el número de puerto d eorigen y de destino, y realiza un Checksum. Analicemos los demás campos:

Número de secuencia (Sequence Number).

Los bytes que van a ser enviados primero se numeran, comenzando con algún número aleatorio entre 0 y 2^{32}. Luego de seleccionado el número para los bytes que van a enviarse, TCP asigna un número de secuencia al segmento.

- El número de secuencia para el primer segmento es su número de secuencia inicial. Este es un número aleatorio.
- El siguiente número de secuencia es igual al número de secuencia anterior mas el número de bytes enviados por el segmento anterior.

Ejemplo: se desea transferir un archivo de 2000 bytes y el primer bytes se numera con 30001. Si se utilizan 5 segmentos, los mismos serán numerados de la siguiente forma:

Tabla 1: números de secuencia para 5 segmentos

Segmento	Número de secuencia	Rango de bytes
1	30001	30400
2	30401	30800
3	30801	31200
4	31201	31600
5	31601	32000

Fuente: elaboración propia

Número de confirmación (Acknowledge Number).

El número de secuencia ocupa la misma cantidad de bits que el número de confirmación. Esto es así ya que un número de confirmación es enviado por el receptor y le avisa al transmisor cuál es el siguiente número de secuencia que espera recibir, indicando al mismo tiempo que ha recibido correctamente los bytes anteriores.

Continuando con el ejemplo del envío de 2000 bytes, cuando el receptor recibe correctamente el primer segmento, enviaría un segmento cuyo número de ACK será 30401. Esto no significa que recibió correctamente 30.401 bytes, porque el primer byte no era 0, sino 3001; entonces le está indicando que recibió correctamente 400 bytes.

Control

El campo de control está subdivido en 6 bits o flags diferentes. Es posible activar uno o más al mismo tiempo. Sirven para abrir, cerrar y abortar conexiones entre otras funciones.

Tamaño de ventana

Aquí se almacena un número que especifica el tamaño de la ventana (buffer) que tendrá el dispositivo para recibir bytes. Se lo suele denominar "ventana del receptor" o rwnd por sus siglas en inglés. El dispositivo receptor envía esta información al transmisor, quién no podrá entonces excederse de el tamaño anunciado cuando haga el envío de bytes.

Puntero urgente

El transmisor establece un puntero de urgencia cuando el receptor debe leer primero cierta información que está en el segmento. El número indicado en este campo se suma al del número de secuencia, y allí están los datos que deben ser leídos.

Puede profundizar tus conocimientos sobre el segmento TCP leyendo el RFC original (https://tools.ietf.org/html/rfc793) o la versión traducida al español (https://www.rfc-es.org/rfc/rfc0793-es.txt)

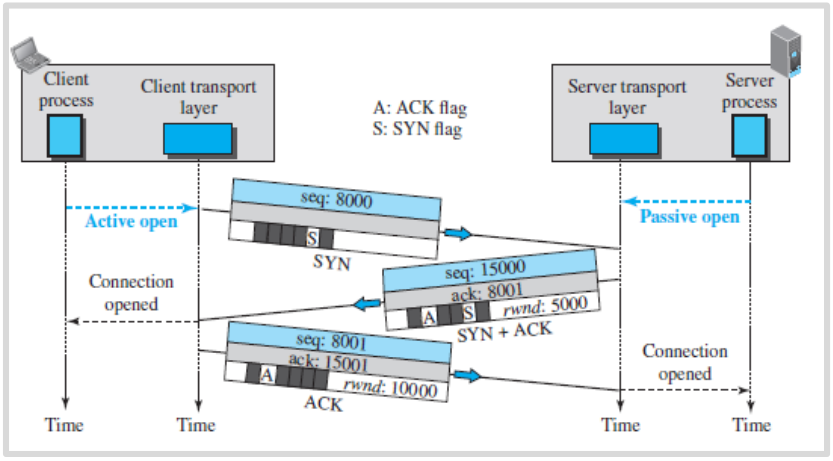
Establecimiento y cierra de conexiones

Como TCP es un protocolo orientado a la conexión, antes de poder intercambiar datos de usuario es necesario establecer una conexión en donde se negocian características que evitan por ejemplo saturar al receptor con un flujo de bytes que no puede procesar.

Establecimiento de una conexión

El proceso para establecer se observa en la figura 3, en lo que se denomina el saludo de 3 vías.

Figura 3: Establecimiento de conexión



Fuente: Forouzan, 2013, p. 751

El dispositivo cliente desea establecer la conexión y envía un segmento TCP con el Flag SYN activado. El dispositivo servidor recibe ese segmento, y envía otro con los flags ACK y SYN activados. Esto significa que está confirmando la recepción y además enviando el flag de apertura. Además, envía el tamaño de ventana de recepción, siendo de 5000 en el ejemplo de la figura. Notar los número de secuencia y ack que envía el receptor: 15000 y 8001. 15000 es su propio número de secuencia, diferente al que maneja el transmisor. 8001 significa que recibió correctamente los bytes anteriores y espera que el próximo sea 8001.
Finalmente, el cliente confirma que recibió correctamente el segmento del server enviando un segmento con ACK=15001 y flag ACK activado. Su ventana de recepción es de 10000, diferente a la del servidor que es 5000. Esto es posible porque la comunicación es full duplex y cada dispositivo maneja su propia ventana de recepción.

A partir de este momento, comienza el intercambio de datos de usuarios entre el cliente y el servidor.

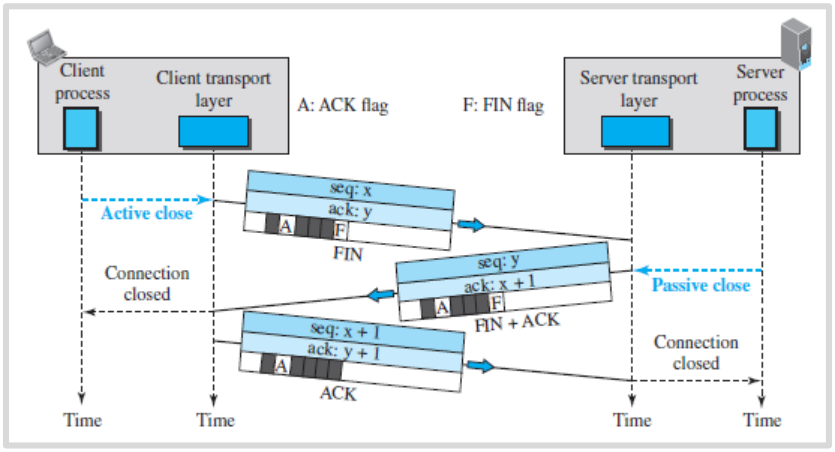
Una vulnerabilidad denominada "inundación SYN" puede producirse por como está diseñado el saludo de 3 vías. Si un host emite continuamente segmentos SYN y luego no confirma con el último ACK, el servidor deberá memorizar todos los números de secuencia que fue recibiendo mientras espera la confirmación que nunca llegará. El problema y las formas de solucionarlo se explica en el RFC 4987. https://tools.ietf.org/html/rfc4987

Cierre de conexión

El cierre de conexión puede ser iniciado tanto por el cliente como por el servidor. Existen dos métodos para realizarlo: mediante un saludo de 3 vías similar al de inicio de conexión, o con un saludo de 4 vías con opción de medio cierre.

En la figura 4 se observa un ejemplo con saludo de 3 vías. El cliente envía un segmento con el Flag FIN activado. El servidor recibe ese segmento y envía entonces un segmento con dos flags: ACK y FIN. Finalmente el cliente recibe el segmento enviado por el servidor y confirma el cierre enviando un segmento con flag ACK activado.

Figura 4: Cierre de conexión con saludo de 3 vías



Fuente: Forouzan, 2013, p. 756

Referencias

Tanenbaum, A (2012). Los protocolos de transporte en Internet en Redes *de Computadoras.* Madrid: Editorial Pearson Education

Forouzan, B (2013). Transport-Layer Protocols en *Data Communications AND Networking.* Estados Unidos: McGraw-Hill

IETF (1981). RFC793. Transmission Control Protocol. Estados Unidos. https://tools.ietf.org/html/rfc793

IETF (2007). RFC793. TCP SYN Flooding Attacks and Common Mitigations. Estados Unidos. https://tools.ietf.org/html/rfc4987

Control de flujo y congestión

Redes

UNIVERSIDAD SIGLO 21 | MIEMBRO DE LA RED ILUMNO

Control de flujo

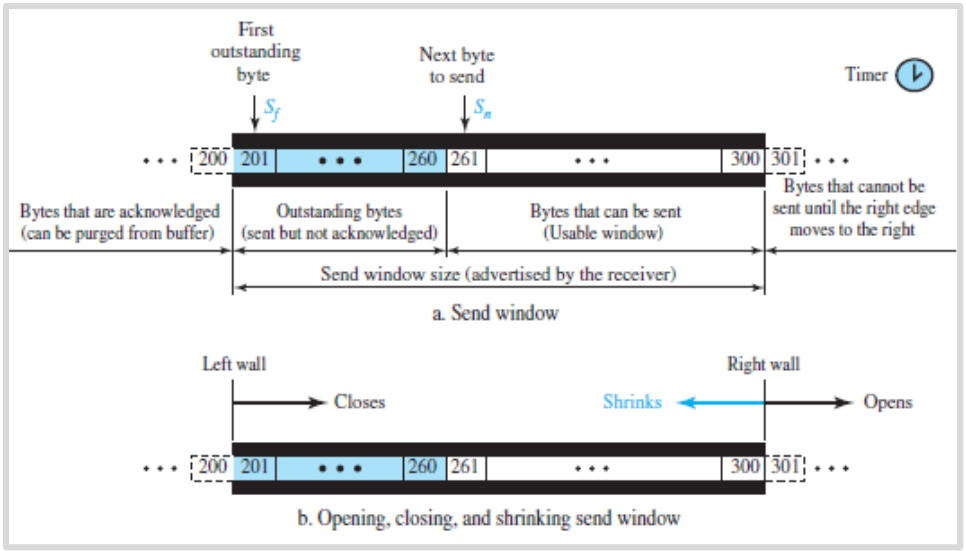
El control de flujo evita que el receptor sea saturado por segmentos provenientes del transmisor. ¿Cómo lo logra? Mediante un mecanismo denominado ventanas deslizantes, lo que es equivalente a un buffer variable. Si el tamaño del buffer es muy pequeño, se llenará rápidamente con bytes del transmisor, el cual deberá dejar de transmitir.
En el caso contrario, si el buffer del receptor es grande, el mismo será capaz de recibir una mayor cantidad de bytes, y el transmisor podrá seguir transmitiendolos.

El tamaño no es fijo, por lo cual el receptor puede controlar al transmisor permitiéndole el envío de mas o menos bytes. Para que esto sea posible se utilizan dos ventanas: una en transmisor y otra en el receptor.

Ventanas deslizantes

En la figura 1 se observa la ventana deslizante en el dispositivo transmisor. A la izquierda de la ventana (200) están los bytes que ya fueron transmitidos y confirmados por el receptor. La ventana comienza con los bytes 201 al 206 en celeste, bytes que ya fueron enviados pero todavía no fueron confirmados mediante un ACK. El byte 261 es el siguiente que será enviado y el límite de la ventana se ubica en 300. El tamaño de esta ventana es de 100, lo cual significa que el transmisor puede enviar 100 bytes sin recibir ACK. Para poder transmitir el byte 301, primero debe recibirse un ACK del 201.

Figura 1: Ventana del transmisor

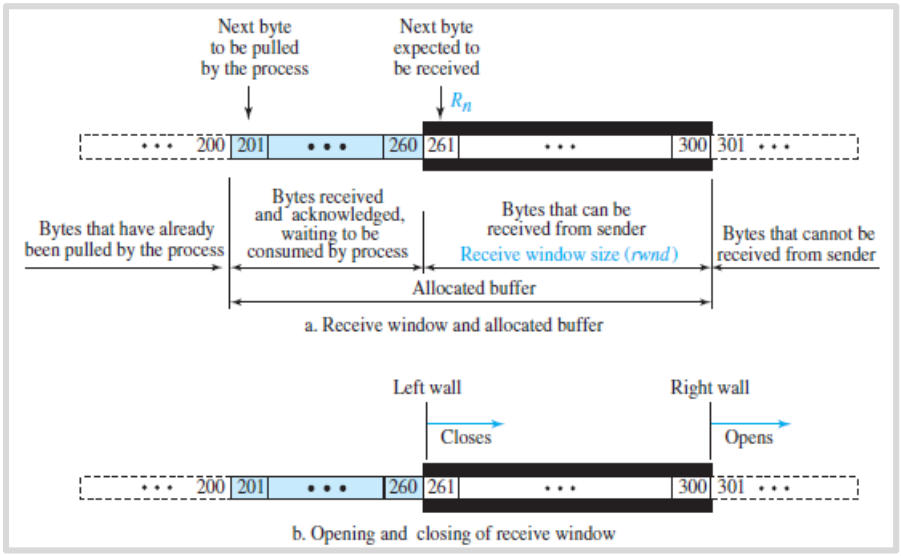


Fuente: Forouzan, 2013, p. 761

El receptor es el encargado de enviarle un tamaño de ventana al transmisor. ¿Cómo determina el tamaño que necesita?

El receptor utiliza una ventana deslizante similar a la del transmisor, la cual puede observarse en la figura 2.

Figura 2: Ventana del receptor



Fuente: Forouzan, 2013, p. 762

En la ventana del receptor, los bytes a la izquierda del comienzo de la ventana (200 hacia la izquierda) ya han sido extraídos por el proceso. La ventana comienza comienza en el byte 201 y hasta el 260 son bytes que fueron recibidos y su correspondiente ACK fue enviado al transmisor. Entre los bytes 261 hasta el 300, son bytes que el receptor puede recibir. El tamaño de la ventana es de 100, el mismo tamaño que maneja el transmisor. Cuando se reciben bytes, la ventana se va cerrando hasta que sean consumidos por el proceso.

Control de flujo

Luego de analizar las ventanas deslizantes, veamos como utiliza este mecanismo TCP para realizar control de flujo.

El objetivo es ajustar el tamaño de las ventanas de acuerdo a la capacidad del receptor. La ventana del receptor se achica a medida que va recibiendo bytes del transmisor pero se agranda cuando el proceso extrae bytes de la ventana y libera esos espacios.

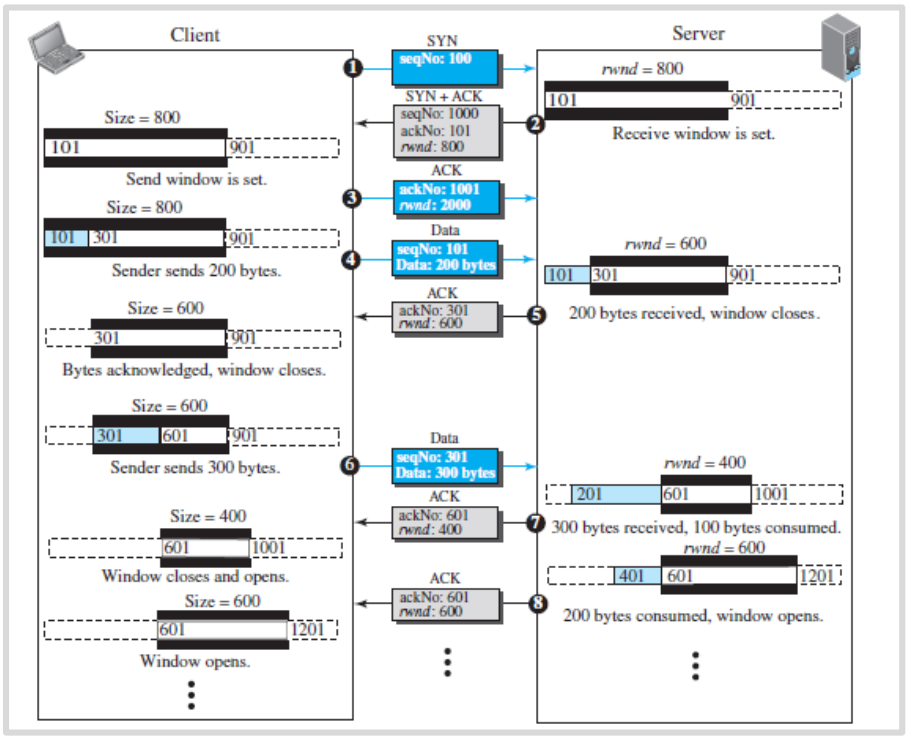
La ventana del transmisor es controlada por la del receptor de esta forma: se achica cuando un ACK llega desde el receptor, mientras que se abre cuando el tamaño de ventana del receptor anunciado por el receptor lo

permite. Finalmente, al ventana se achica cuando no llegan ACK y el receptor no anuncia un tamaño de ventana mayor.

El mecanismo se observa en la figura 3. El saludo de 3 vías comienza con el cliente enviando un segmento con flag SYN con número de secuencia aleatorio 100. El servidor responde a ese mensaje con un segmento cuyo número de secuencia es 1000, anuncia una ventana de recepción de 800 y al mismo tiempo devuelve el ACK 101 indicando el siguiente número de secuencia que espera recibir. Para completar el saludo, el cliente envía un ACK al servidor sin incrementar el número de secuencia y anunciando su porpia ventana de recepción ya que la comunicación puede ser full duplex.

Luego de finalizado el saludo, el cliente envía 1 segmento con 200 bytes. El número de secuencia es 101. La ventana del cliente sigue siendo 800, pero ahora muestra que 200 bytes fueron transmitidos pero esperando confirmación.
Esos 200 bytes son recibidos por el servidor, el cual los almacena en buffer achicando su ventana en 200 bytes, por lo que el nuevo tamaño es de 600. En ese momento el servidor envía un ACK (301) al cliente anunciando su nuevo tamaño de ventana: 600. El cliente recibe ese mensaje y achica la ventana en 200 bytes al eliminar los que se encontraban en estado “enviados pero esperando confirmación”.

Figura 3: control de flujo



Fuente: Forouzan, 2013, p. 764

El cliente nuevamente manda un segmento, ahora con 300 bytes (número de secuencia 301). El servidor los recibe y los almacena en su buffer junto a los 200 bytes recibidos en el segmento anterior; la aplicación consume 100 bytes de los 500, por lo que ahora la ventana tiene un tamaño de 400. Este valor surge de restar 500 de los 800 de la ventana inicial y sumar 100 de los consumidos.
Cuando el servidor envía el ACK, anuncia un tamaño de ventana de 400 y el cliente modifica su ventana removiendo los bytes que fueron confirmados y agrandando la ventana en 100 unidades.

El último segmento es un nuevo ACK por parte del servidor cuya aplicación ha consmido 200 bytes adicionales. Entonces el nuevo tamaño de ventana de recepción es de 600 y el cliente puede agrandar su ventana en 200 unidades cuando reciba el mensaje.

El mecanismo garantiza que si la aplicación consume bytes más lento de la velocidad de recepción, la ventana se achicará y al anunciarlo al transmisor, dicha velocidad disminuirá evitando saturar al receptor.

Síndrome de la ventana tonta

La consecuencia de la “silly window” es la ineficiencia de TCP debido a la utilización de ventanas muy pequeñas. Se produce porque el transmisor genera datos a una velocidad muy baja o bien porque el receptor las procesa muy lentamente. Para profundizar tus conocimientos sobre este síndrome, consulta el capítulo 6 de Tanenbaum.

Control de congestión

El control de flujo de TCP evita saturar al receptor y optmiza la transmisión, sin embargo nada puede hacer ante la congestión de los dispositivos intermedios de la red. Si bien las redes se basan en el protocolo IP, este no posee un mecanismo para controlar la congestión por lo que TCP también se hace cargo de ello.
Además de las ventanas analizadas durante el control de flujo, para el control se congestión se utiliza la denominada ventana de congestión o cwnd por sus siglas en inglés. Las ventanas de congestión y recepción definen juntas el tamaño de la ventana del transmisor.

$$Tamaño\ de\ ventana\ real = mínimo\ (rwnd, cwnd)$$

Detección de congestión

Para poder controlar la congestión, primero es preciso detectarla. Esta tarea es llevada a cabo por el transmisor de dos maneras:

- Time out: cuando el transmisor envía un segmento activa un timer. Si el timer llega a cero sin haberse recibido el ACK correspondiente, se asume que el segmento se perdió.
- 3 ACK duplicados (4 ACK con el mismo número): cuando el receptor envía un ACK duplicado significa que hubo una demora, pero si envía 3 significa que el segmento se perdió. Sin embargo, esta congestión es menos severa que el Time Out, ya que si bien un segmento se perdió, otros 3 llegaron y por esa razón se generaron 3 ACK.

Una vez que el transmisor detecta la congestión ya sea por un Time Out o por recibir 3 ACK duplicados, deberá emplear algún algoritmo que la evite.

Políticas para control de congestión

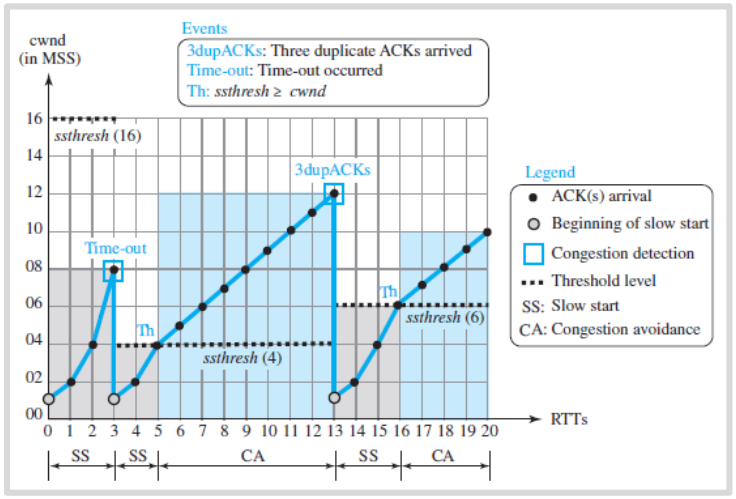
Las políticas par controlar la congestión se basan en tres algoritmos: slow start (arranque lento), Congestion Avoidance (evitación de la congestión) y Fast Recovery (recuperación rápida).

Aprende sobre estos algoritmos en el capítulo 6 de Tanenbaum, y profundiza en la RFC 2581 (https://tools.ietf.org/html/rfc2581)

A partir de estas políticas surgen diferentes implementaciones de TCP. La primera fue la Taho TCP, la cual utilizaba Slow Start y Congestion Avoidnace. Esta implementación no distingue entre congestiones severas y leves.

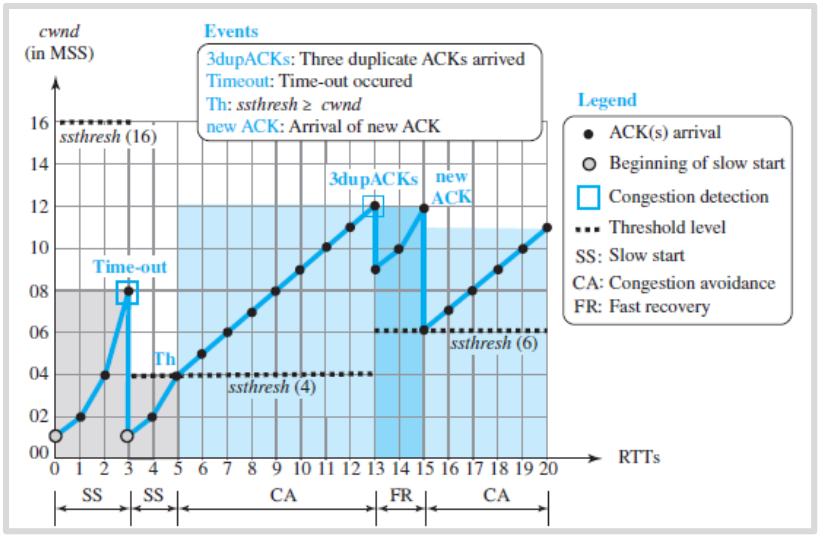
Otra implementación TCP se denomina Reno, la cual si tiene en cuenta los dos diferentes tipos de congestión, y logra de esta forma un mejor rendimiento. Las figuras 4 y 5 comparan las implementaciones Taho y Reno de TCP, mostrando gráficamente como Reno mejora el rendimiento.

Figura 4: Taho TCP



Fuente: Forouzan, 2013, p. 764

Figura 5: Reno TCP



Fuente: Forouzan, 2013, p. 785

Profundiza sobre las implementaciones en el capítulo 6 del libro de Tanenbaum, o en el capítulo 24 del libro de Forouzan (en inglés).

Referencias

Tanenbaum, A (2012). Los protocolos de transporte en Internet en Redes *de Computadoras.* Madrid: Editorial Pearson Education

Forouzan, B (2013). Transport-Layer Protocols en *Data Communications AND Networking.* Estados Unidos: McGraw-Hill

IETF (1999). RFC2581. TCP Congestion Control. Estados Unidos. https://tools.ietf.org/html/rfc2581

IETF (2015). RFC7414. A Roadmap for Transmission Control Protocol (TCP). Estados Unidos. https://tools.ietf.org/html/rfc7414

Protocolo SCTP

Redes

UNIVERSIDAD SIGLO 21 | MIEMBRO DE LA RED ILUMNO

Protocolo SCTP

Si bien TCP es sumamente útil para muchas aplicaciones de Internet, para otras es una limitación por lo que han tenido que utilizar sus propios protocolos encapsulados en UDP.

Supongamos una aplicación multimedia en tiempo real, más precisamente una conversación telefónica que utiliza IP. En el caso de que TCP fuera utilizado como protocolo de transporte, al perderse algún segmento el receptor solicitaría retransmisión. Esta situación no es admisible para comunicación en tiempo real, donde es preferible perder un paquete a retransmitirlo.

La IETF vió el problema y diseñó un nuevo protocolo de transporte denominado SCTP (Stream Control Transmision Protocolo o Protocolo de control de transmisión de flujos).

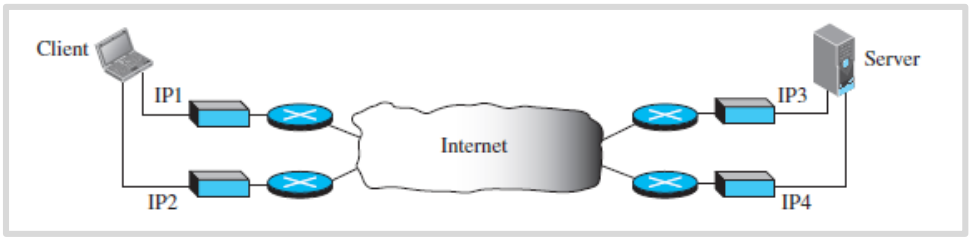
Servicios ofrecidos por SCTP

Al igual que UDP o TCP, SCTP ofrece sus servicios a las aplicaciones que deseen utilizar este protocolo.
Sin diferenciarse de los otros protocolos de transporte, SCTP permite comunicación proceso a proceso. Y al igual que TCP, permite comunicaciones full-duplex, es un servicio confiable que utiliza confirmaciones (ACK) y es orientado a la conexión aunque la conexión en SCTP se denomina asociación.

¿Qué lo diferencia de los otros protocolos de transporte?

- Múltiples flujos: a diferencia de TCP, que utiliza un flujo de datos entre cliente y servidor, SCTP permite múltiples. Esto significa que si uno de los flujos se bloquea, la información continua llegando a destino por otro flujo. Esta característica es sumamente útil para streaming de audio y video en tiempo real.
- Multihoming: la conexión TCP se basa en una IP de origen y una IP de destino (además de los puertos). Una asociación SCTP (recordemos que asociación SCTP es el equivalente a la conexión TCP) permite múltiples direcciones IP. Si algún camino falla, es posible utilizar otro y no interrumpir el flujo. Un ejemplo se observa en la figura 1.

Figura 1: Multihoming



Fuente: Forouzan, 2013, p. 792

Características del protocolo

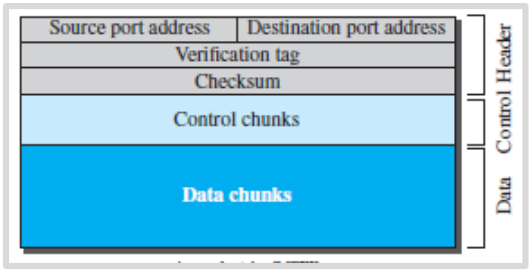
La unidad de datos del protocolo SCTP se denomina *chunk* y las transferencias se controlan numerandolos. Se denomina número de secuencia de transmisión (TSN) a los chunks numerados. El sistema es análogo a la utilización de números de secuencia en TCP los cuales numeran segmentos.

Como puede haber más de un flujo o stream en cada asociación, estos deben identificarse con un SI (identificador de stream).

Paquete SCTP

El paquete SCTP está dividido en tres secciones: un encabezado, un área de control y el área de datos.
Varios chunks de control y de datos pueden juntarse en un mismo paquete SCTP y por lo tanto compartir el encabezado. El paquete SCTP se observa en la figura 2. Este paquete SCTP es análogo a un segmento TCP.

Figura 2: Paquete SCTP



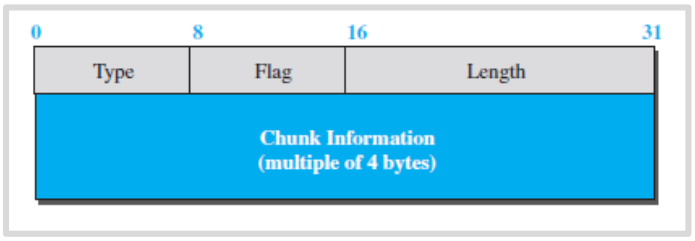
Fuente: Forouzan, 2013, p. 793

El encabezado SCTP es simple y solo posee cuatro campos. Los puertos de origen y destino son iguales que en UDP/TCP. El campo Verification Tag almacena un número que identifica a la asociación, es decirque todos los paquetes enviados en una asociación usarán el mismo número. El campo

Checksum utiliza el doble de bits que en UDP/TCP ya que se calcula un CRC-32.

Todos los chunks de control tienen el formato de la figura 3. El campo Type identifica que tipo de chunk es, por ejemplo un 0 indica que es un tipo de chunk de datos. Para conocer todos los tipos de chunks, consultar la sección 3.2 de la RFC4960: https://tools.ietf.org/html/rfc4960#section-3.2. El campo Flag es usado para definir particularidades. El campo Length almacena el tamaño total en bytes incluyendo estos tres campos.

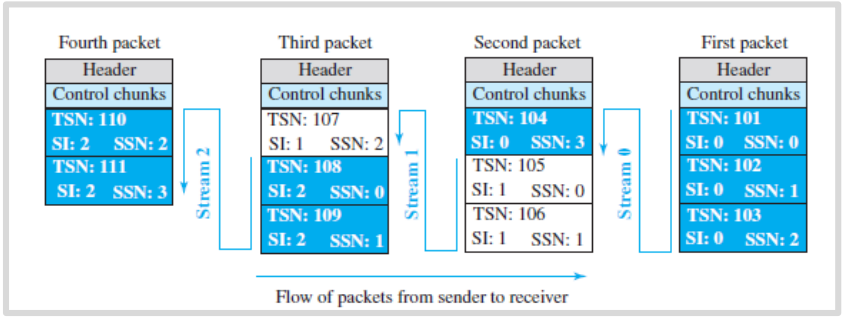
Figura 3: Formato chunk de control



Fuente: Forouzan, 2013, p. 796

Resumiendo, en SCTP existen chunks de datos, flujos y paquetes. Una asociación puede enviar muchos paquetes, cada paquete puede contener varios chunks, y los chunks pueden pertenecer a diferentes flujos. Este funcionamiento se ejemplfiica en la figura 5, en donde una aplicación debe enviar 11 mensajes a otro host en 3 flujos. Como la capacidad máxima por paquete es de 3 chunks de datos por paquete, debe enviar 4 paquetes en total. Los flujos tienen 4, 3 y 4 chunks respectivamente.
Entonces, el primer flujo completa el primer paquete con 3 chunks y coloca el último chunk en el segundo paquete. El segundo flujo coloca 2 de sus 3 chunks en el mismo paquete hasta completarlo, y su último chunk en el tercer paquete. Finalmente, el últilmo flujo completa el tercer paquete y debe utilizar un cuarto para poder enviar sus 4 chunks a destino.

Figura 4: 3 paquetes SCTP



Fuente: Forouzan, 2013, p. 794

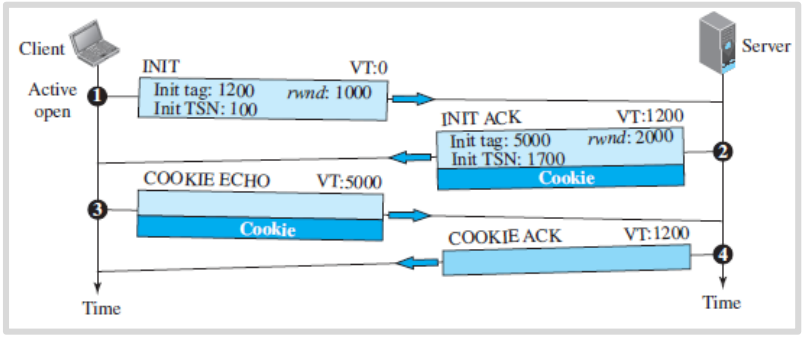
Para identificar los chunks de datos se utiliza TSN, SI y SSN. TSN es un número incremental que identifica a cada Chunk (similar al número de secuencia TCP). Si identifica a cada flujo, por lo se repite para todos los chunks de un mismo flujo. Finalmente SSN define el orden de los chunks en el flujo.

Establecimiento y terminación de una asociación

Establecer una asociación es análogo a establecer una conexión en TCP. Mientras que TCP utiliza el denominado saludo de 3 vías (se intercambian 3 mensajes entre el cliente y el servidor), SCTP utiliza un saludo de 4 vías.

En la figura se ejemplifica el saludo de 4 vías. El saludo comienza cuando el cliente envía un paquete con un chunk de iniciación (INIT Chunk). Además se anuncia el tamaño de ventana que es 1000 en el ejemplo y el TSN inicial (100). Init tag es el tag de iniciación que deberá usar el servidor.
El servidor responde con un paquete INIT ACK cuyo tag de verificación (VT) es el valor de Init tag que envió el cliente en el primer mensaje. Además anuncia su tamaño de ventana, su TSN inicial y el Init tag que deberá usar el cliente.
El cliente luego de recibir el segundo paquete envía un chunk denominado Cookie Echo que responde al Cookie enviado por el servidor en el mensaje anterior.
Para finalizar el saludo, el servidor envía un Cookie ACK indicando que recibió correctamente el Cookie Echo. A partir de este momento, ambos dispositivos pueden intercambiar datos en modalidad full duplex.

Figura 5: Establecimiento de una asociación



Fuente: Forouzan, 2013, p. 797

Cuando se usan múltiples direcciones IP (multihoming), durante el establecimiento de la asociación se define cuál de todas será la primaria. La IP por defecto es definida por el extremo opuesto, es decir que la fuente define cuál será la IP por defecto del destinatario. La transferencia de datos utiliza la IP por defecto hasta que no esté disponible, en tal caso se utilizará una alternativa.

Consulta la página 799 de Forouzan para aprender sobre la terminación de las asociación.

Utilización de SCTP

Actualmente el protocolo SCTP no es ampliamente utilizado a pesar de sus grandes ventajas en relación a UDP y TCP. ¿Cuáles son las razones?

Principalmente que el protocolo SCTP es posterior a TCP. Esto significa que TCP es ampliamente utilizado por las aplicaciones que requieren un protocolo de transporte orientado a la conexión. Para que esas aplicaciones puedan soportar SCTP, deberían ser modificadas lo cual conlleva una inversión de tiempo y dinero. El problema es similar a la implementación de IPv6. Este último soluciona los problemas de IPv4 pero requiere actualizaciones.

A este problema se suma la falta de soporte de algunos sistemas, siendo el caso más emblematico el de Microsoft.

El desconocimiento de los especialistas en tecnología de la información y la falta de difusión no contribuyen a la creación de una demanda que ejerza presión a los fabricantes.

Referencias

Tanenbaum, A (2012). Los protocolos de transporte en Internet en Redes *de Computadoras.* Madrid: Editorial Pearson Education

Forouzan, B (2013). Transport-Layer Protocols en *Data Communications AND Networking.* Estados Unidos: McGraw-Hill

IETF (2007). RFC4960. Stream Control Transmission Protocol. Estados Unidos. https://tools.ietf.org/html/rfc4960

Hogg, Scott. (2012). Network World: What About Stream Control Transmission Protocol (SCTP)? Recuperado de: http://www.networkworld.com/article/2222277/cisco-subnet/what-about-stream-control-transmission-protocol--sctp--.html