Capa de Transporte

Esta capa se encarga de la transferencia libre de errores de los datos entre el emisor y el receptor, aunque no estén directamente conectados, así como de mantener el flujo de la red. La tarea de esta capa es proporcionar un transporte de datos confiable y económico de la máquina de origen a la máquina destino.

Protocolo de datagrama de usuario (UDP)

En la capa de red, una dirección destino identifica un anfitrión; no se hace ninguna otra distinción con respecto a qué usuario o qué programa de aplicación recibirá el datagrama. Ahora ampliaremos el grupo de protocolos TCP/IP al agregar un mecanismo que distingue entre muchos destinos dentro de un anfitrión, permitiendo que varios programas que se ejecutan en él puedan enviar y recibir datagramas en forma independiente.

Ahora en vez de pensar el proceso como un destino final, imaginaremos que cada máquina contiene un grupo de puntos abstractos de destino que llamaremos puertos de protocolo. Cada uno de ellos se identifica por medio de un número entero positivo. Entonces para comunicarnos con algún puerto externo necesitaremos saber la dirección IP del anfitrión y además el número de puerto de protocolo destino.

El protocolo UDP proporciona un mecanismo primario que utilizan los programas de aplicación para enviar datagramas a otros programas de aplicación. Utiliza IP para transportar el mensaje de una máquina a la otra y proporciona una semántica de entrega sin conexión y no confiable. No emplea acuses de recibo, no ordena los mensajes, ni proporciona retroalimentación para controlar la velocidad de la información entre máquinas. Por lo tanto los mensajes UDP se pueden perder, duplicar o llegar en cualquier orden.

Muchos programas de aplicación que utilizan UDP trabajan bien en un ambiente local pero fallan drásticamente cuando se utilizan en una red de redes más grande.

Formato de los mensajes UDP

Cada mensaje UDP se conoce como datagrama de usuario.

Puerto UDP origen	Puerto UDP destino
Longitud del mensaje UDP	Suma de verificación UDP
Datos	
....	

Como vemos la cabecera esta formada por cuatro campos de 16 bits cada uno.

- Puerto UDP origen: contiene el número de puerto del protocolo origen. Este es opcional, y cuando se utiliza, especifica la parte a la que se deben enviar las respuestas de lo contrario puede valer cero.
- Puerto UDP destino: contiene el número de puerto del protocolo destino.
- Longitud: contiene un conteo de los octetos en el datagrama UDP.

Incluyendo el encabezado y los datos. Por lo tanto el mínimo valor sera 8 (longitud del encabezado).

- Suma de verificación: es opcional y no es necesario utilizarla. Un valor cero significa que la suma no se computó. Pero es la única forma de garantizar que los datos lleguen intactos a destino.

Pseudo-encabezado UDP

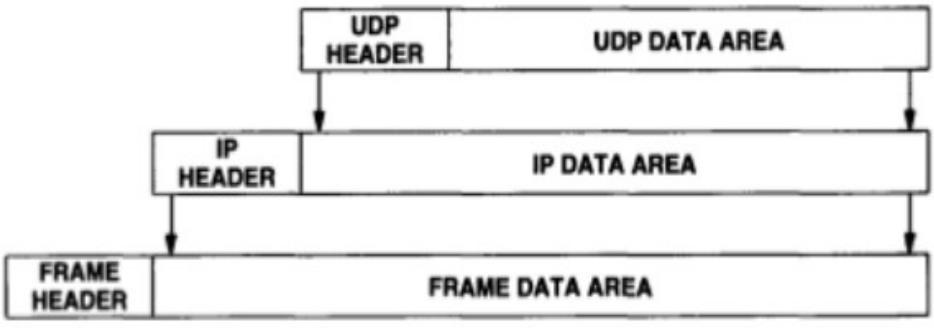
Para computar la suma de verificación, el UDP añade un pseudo-encabezado al datagrama. Añade un octeto de ceros para rellenar el datagrama y alcanzar una longitud múltiplo de 16 bits, y computa la suma de verificación sobre todo el conjunto. El octeto de relleno y el pseudo-encabezado no se transmiten con el datagrama UDP, ni se suman en su longitud.

El propósito del uso del pseudo-encabezado es para verificar la correcta llegada del datagrama UDP a destino.
El encabezado UDP sólo especifica el puerto protocolo destino. Por lo tanto el UDP de la máquina transmisora computa la suma de verificación que cubre tanto la direccionan IP del destino como el datagrama UDP. Luego en el destino final el software UDP revisa la suma de verificación utilizando la IP destino obtenida del encabezado del datagrama IP que transporto el mensaje UDP. Si la suma concuerda, debe ser verdad que el datagrama llego al anfitrión deseado y al puerto correcto.

Dirección IP de origen		
Dirección IP de destino		
Ceros	Protocolo	Longitud UDP (sin P-E)

Encapsulación de UDP y estratificación por capas de protocolos

Estratificar por capas el UDP por encima del IP significa que un mensaje UDP completo incluyendo el encabezado y los datos, se encapsula en un datagrama IP mientras viaja a través de una red de redes.



La capa IP solo es responsable de transferir datos entre par de anfitriones, mientras que la capa UDP solamente es responsable de diferenciar entre varias fuentes o destinos en un anfitrión.

Servicio de transporte de flujo confiable (TCP)

Utilizar un sistema de entrega sin conexión y no confiable para transferencias de gran volumen puede resultar tedioso o molesto y requiere que los programadores incorporen en cada programa de aplicación la detección y corrección de errores.
Debido a que es difícil diseñar, entender o modificar el software que proporciona confiabilidad, se ha encontrado una solución en un protocolo para proporcionar una entrega de flujo confiable.

Características del servicio de entrega confiable

- Orientación de flujo: cuando dos programas de aplicación se transfieren datos, pensamos a los mismos como un flujo de bits, divididos en octetos de 8 bits o en bytes.
- Conexión de circuito virtual: antes de comenzar la transferencia, los programas de aplicación, transmisor y receptor interactúan con sus respectivos sistemas operativos informándoles la necesidad de realizar una transferencia de flujo. Los módulos de software interactúan mandándose mensajes por la red de redes, verificando que la transferencia esté autorizada y estableciendo una conexión.
  Si por cualquier motivo la transferencia falla, ambas máquinas lo sabrán y reportaran los problemas a los programas apropiados para cada aplicación.
- Transferencia con memoria intermedia: para hacer eficiente la transferencia y minimizar el tráfico de red, las implantaciones recolectan datos suficientes para llenar un datagrama razonablemente largo antes de transmitirlo (llenar la memoria intermedia). Lo mismo ocurre si la aplicación genera bloques muy grandes de información, éstos se dividen antes de ser enviados.
  Para aplicaciones en las que los datos se deben enviar aunque no se llene la memoria intermedia, el servicio de flujo proporciona un mecanismo llamado *push* que las aplicaciones utilizan para forzar una transferencia.
- Flujo no estructurado: no es necesario que el flujo sea estructurado, los programas de aplicación de servicio de flujo deben entender el contenido del flujo y ponerse de acuerdo sobre su formato antes de iniciar la conexión.
- Conexión full duplex: este tipo de conexión permite la trasferencia concurrente en ambas direcciones. Son dos flujos independientes que se mueven en direcciones opuestas, sin ninguna interacción aparente.

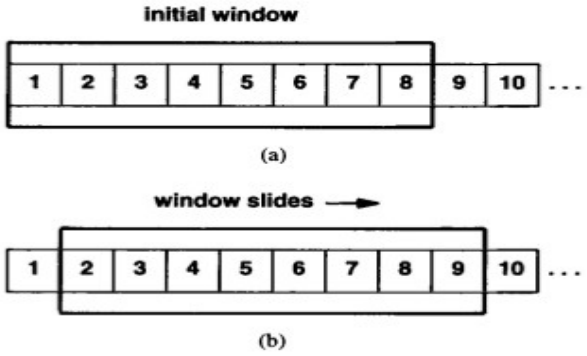
Proporcionando confiabilidad

Los protocolos que aseguran la entrega de datos enviados, utilizan una técnica fundamental llamada *acuse de recibo positivo con retransmisión.* Ésta requiere que el receptor se comunique con el origen y le envíe un mensaje de *acuse de recibo (ACK)* conforme recibe los datos. El transmisor guarda un registro de cada paquete que envía y espera un acuse de recibo antes de enviar el próximo paquete. El transmisor también arranca un temporizador al enviar el paquete y si este expira antes de la llegada del acuse retransmite.

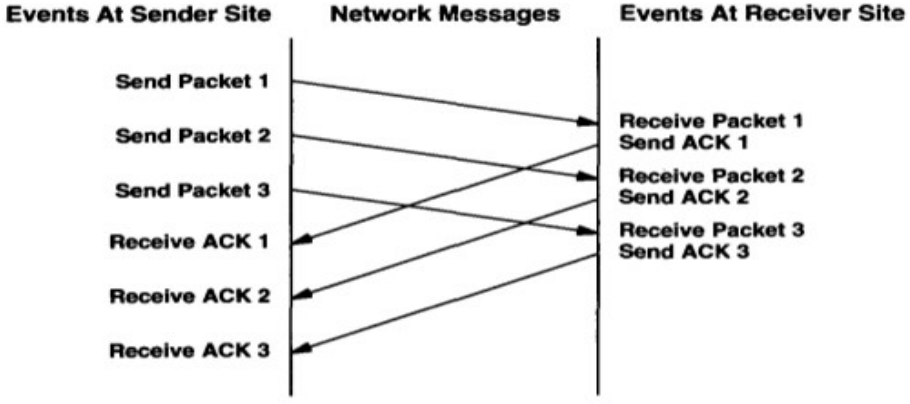
Para la detección de paquetes duplicados, los protocolos confiables los distinguen porque asignan a cada paquete un número de secuencia y obligan al receptor recordar que número de secuencia recibe.
Para la detección de mensajes de acuse de recibo duplicados, los protocoles de acuses de recibo envían los números de secuencia dentro de los acuses para que el receptor pueda asociar correctamente acuses con paquetes.

Ventana deslizable

El concepto de ventana deslizable, sirve de base para la transmisión de flujo. Hace que ésta sea más eficiente. Un protocolo simple de acuses de recibo positivos ocupa una gran cantidad de ancho de banda de red debido a que debe retrasar el envío del siguiente paquete hasta que llegue el acuse de recibo del paquete anterior.
Esta nueva técnica es una forma mas compleja de acuse de recibo positivo y retransmisión que el método mencionado antes. Usa de mejor forma el ancho de banda disponible ya que permiten enviar paquetes sin esperar un acuse de recibo.



El protocolo coloca una ventana pequeña y de tamaño fijo en la secuencia y transmite todos los paquetes que residan dentro de la ventana. Una vez que el transmisor recibe un acuse de recibo para el primer paquete de la ventana, la mueve y envía el siguiente paquete.
El desempeño de los protocolos de ventana deslizable dependerá del tamaño de la ventana y de la velocidad en que la red acepta paquetes.
Con un tamaño de ventana igual a 1 estaremos en el mismo protocolo de acuse de recibo simple. Al aumentar el tamaño de la ventana disminuimos cada vez mas el tiempo ocioso de la red.
Una ventana divide la secuencia de paquetes en tres partes: los de la izquierda de la ventana ya se transmitieron, recibieron y acusaron exitosamente; los paquetes de la ventana están en proceso de transmisión; los de la derecha todavía no se han transmitido.



El protocolo de ventana deslizable siempre recuerda que paquetes tienen acuse de recibo y mantiene un temporizador separado para cada paquete sin acuse de recibo. Si se pierde un paquete el temporizador concluye y el transmisor reenvía el paquete.

Protocolo de control de transmisión

TCP es un protocolo de comunicación, no una pieza de software. La diferencia entre estos dos es análoga a la diferencia entre un lenguaje de programación y un compilador.
TCP:

- Especifica el formato de datos y los acuses de recibo que intercambian dos computadoras para lograr una transferencia confiable.
- Especifica los procedimientos que una computadora utiliza para asegurarse de que los datos lleguen de manera correcta.
- Especifica cómo el software TCP distingue el correcto entre muchos destinos en una misma máquina.
- Especifica cómo las máquinas en comunicación resuelven errores como la pérdida o duplicación de paquetes.
- Especifica cómo dos computadoras inician una transferencia de flujo TCP y cómo se ponen de acuerdo cuando se completa.
- No especifica los procedimientos que usan los programas de aplicación para acceder a las operaciones que realiza TCP. Esto es para darle mayor flexibilidad al implementador.

Puertos, conexiones y puntos extremos

Al igual que UDP, TCP reside sobre el IP en el esquema de estratificación por capas de protocolos.
El TCP permite que varios programas de aplicación en una máquina se comuniquen de manera concurrente y realiza el demultiplexado del trafico TCP entrante entre los programas de aplicación.
También TCP utiliza números de puerto de protocolo para identificar el destino final dentro de una máquina.
El TCP utiliza la conexión, no el puerto de protocolo, como su abstracción fundamental; las conexiones se identifican por medio de un par de puntos extremos.
El TCP define como punto extremo a un par de números enteros (anfitrión,puerto) en donde el anfitrión es la dirección IP de un anfitrión y puerto es un puerto TCP en dicho anfitrión, por ejemplo: (18.26.0.36,1069) y (128.10.2.3,25), mientras que a la vez puede existir la conexión (128.9.0.31,1184) y (128.10.2.3,25).
Esto es posible porque no hay ambigüedad ya que el TCP asocia los mensajes entrantes a una conexión en vez de hacerlo con un puerto de protocolo.
Entonces varias conexiones en una misma maquina pueden compartir el número de puerto TCP.

Aperturas pasivas y activas

Como TCP es un servicio orientado a la conexión, requiere que ambos puntos extremos estén de acuerdo con la comunicación. Para hacerlo, el programa de

aplicación en un extremo realiza una función de *apertura pasiva* al contactar su sistema operativo e indicar que acatará una conexión entrante, en ese momento el S.O asigna un número de puerto TCP a su extremo de la conexión. Por otro lado, el programa de aplicación en el otro extremo debe contacta a su S.O mediante una solicitud de *apertura activa* para establecer la conexión.

Segmentos, flujos y números de secuencia

El TCP utiliza un mecanismo de ventana deslizable un poco mejorado para la transmisión eficiente y el control de flujo.
Permite que el receptor restrinja la transmisión hasta que tenga espacio suficiente en la memoria intermedia para incorporar datos. De esta manera puede controlar el flujo extremo a extremo
El mecanismo TCP de ventana deslizable opera a nivel de octeto, no a nivel segmento ni paquete. El transmisor guarda apuntadores asociados con cada conexión.

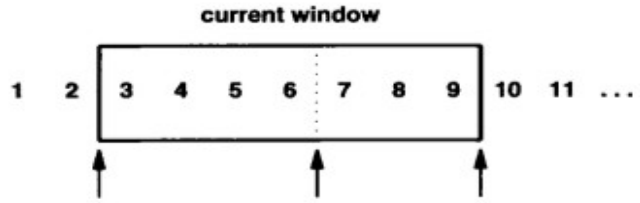


Figure 13.6 An example of the TCP sliding window. Octets through 2 have been sent and acknowledged, octets 3 through 6 have been sent but not acknowledged, octets 7 though 9 have not been sent but will be sent without delay, and octets 10 and higher cannot be sent until the window moves.

Como las conexiones TCP son full duplex, el software TCP en cada extremo mantiene dos ventanas por cada conexión (es decir en total habrá cuatro ventanas) una se desliza a lo largo de los datos que se envían y la otra sobre los que se reciben.

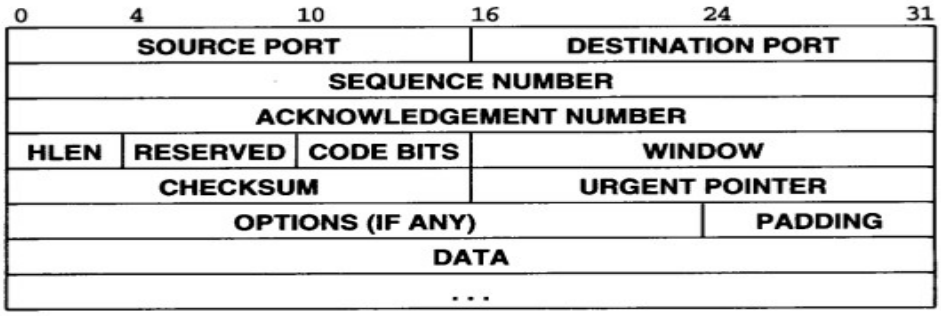
Tamaño variable de ventana y control de flujo

Para controlar el flujo de la comunicación extremo a extremo, TCP emplea un esquema de ventana deslizable. Éste se basa en los acuses de recibo, quienes además de informar cuantos octetos se recibieron, contienen un aviso de ventana que especifica cuántos octetos adicionales de datos está preparado para aceptar el receptor.
Entonces en respuesta a un aumento en el aviso de ventana, el transmisor aumenta el tamaño de su ventana deslizable y procede al envío de octetos de los que todavía no se tiene un acuso de recibo.
En respuesta a una disminución en el aviso de ventana, el transmisor disminuye el tamaño de su ventana y deja de enviar los octetos que se encuentran más allá de la frontera.
Controlar el flujo extremo a extremo es necesario para que TCP pueda garantizar una entrega confiable de los datos.

Formato del segmento TCP

La unidad de transferencia entre el software TCP de dos maquinas se conoce

como segmento. Estos se intercambian para establecer conexiones, transferir datos, enviar acuses de recibo, anunciar tamaños de ventanas y cerrar conexiones.



Cada segmento se divide en encabezado y datos. El encabezado TCP transporta la identificación y la información de control.

- Source port (puerto fuente) y Destination port (puerto destino): contienen el número de puerto TCP que identifica a los programas de aplicación en los extremos de la conexión.
- Sequence number (número de secuencia): identifica la posición de los datos del segmento en el flujo de datos.
- Acnowledgement number (número de acuse de recibo): identifica el numero de octetos que la fuente espera recibir después.
- HLEN: contiene un numero entero que especifica la longitud del encabezado del segmento, medida en multiplos de 32 bits.
- Reserved: esta reservado para usarse en el futuro.
- Code bits: este campo determina el propósito y contenido del segmento. Consta de 6 bits:

Bit (left to right)	Meaning if bit set to 1
URG	Urgent pointer field is valid
ACK	Acknowledgement field is valid
PSH	This segment requests a push
RST	Reset the connection
SYN	Synchronize sequence numbers
FIN	Sender has reached end of its byte stream

- Window: en este campo especifica cuantos datos esta dispuesto a aceptar cada vez que envía un segmento (ó sea en este campo detalla el tamaño de su memoria intermedia).

Datos fuera de banda

Aunque el TCP esté orientado al flujo, a veces es importante que el programa en un extremo pueda enviar datos *fuera de banda*, sin esperar que el otro extremo consuma los octetos que ya están en flujo.
Para incorporar la señalización de fuera de banda, TCP permite que el transmisor clasifique los datos como *urgentes*, dando a entender que se debe notificar su llegada al programa receptor lo antes posible. Asimila estos datos y luego vuelve a su operación normal.
El mecanismo para marcar datos urgentes en un segmento consiste en un bit de código URG y en un campo URGENT POINTER que especifica la posición

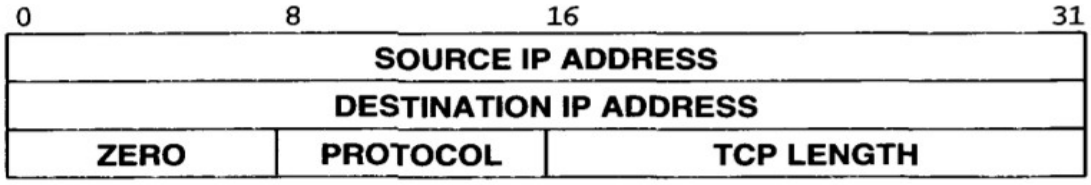
dentro del segmento en la que terminan los datos urgentes.

Opción de tamaño máximo de segmento

El software TCP utiliza el campo OPTIONS para negociar con el software TCP en el otro extremo de la conexión; una de las opciones permite que el software TCP especifique el *tamaño máximo de segmento* (MSS) que está dispuesto a recibir.
Si los dos puntos extremos residen en la misma red física, el TCP por lo general computará un tamaño máximo de segmento de tal forma que los datagramas IP resultantes correspondan con la MTU de la red.
Si los puntos extremos están en distinta red física, pueden intentar descubrir la MTU mínima a lo largo del camino o pueden escoger un tamaño máximo de segmento de 536 (asignado por omisión).
Un tamaño de segmento muy pequeño no aprovechara el ancho de banda de la red. Mientras que un tamaño de segmento muy grande requerirá fragmentación y si uno de los fragmentos no llega correctamente IP requiere retransmisión de todo el datagrama.
El tamaño óptimo de segmento ocurre cuando los datagramas IP que llevan los segmentos son tan grandes como sea posible sin requerir fragmentación en ninguna parte a lo largo del camino entre la fuente y el destino.

Cómputo de suma de verificación TCP

Para computar la suma de verificación, el software TCP en la máquina transmisora sigue un procedimiento igual al de UDP. Coloca un pseudo-encabezado en el segmento, agrega suficientes bits en cero para lograr que el segmento sea múltiplo de 16 bits y calcula la suma de 16 bits sobre todo el resultado. El TCP no cuenta el pseudo-encabezado ni los caracteres de relleno en la longitud del segmento, ni tampoco los transmite.
El propósito del uso del pseudo-encabezado es que permite que el receptor verifique que el segmento llegó a su destino correcto, que incluye tanto una dirección IP de anfitrión como un número de puerto de protocolo.



En el extremo receptor, la información utilizada en el pseudo-encabezado se extrae del datagrama IP que transportó el segmento y se incluye en el cómputo de la suma para verificar que el segmento llegó intacto al destino correcto.

Tiempo límite y retransmisión

Una de las ideas mas importantes y complejas de TCP es la forma en que maneja la terminación de tiempo (time out) y la retransmisión.
TCP esta diseñado para emplearse en un ambiente de red de redes, entonces puede atravesar una sola LAN de alta velocidad o varias redes intermedias y varios ruteadores. Entonces es imposible saber con anticipación que tan rápido regresaran los acuses de recibo al origen, por lo que el tiempo total necesario

para que un segmento viaje al destino y regrese el acuse varia drásticamente de un ejemplo a otro.
El TCP maneja los retrasos variables al utilizar un *algoritmo adaptable de retransmisión*. Monitorea el desempeño de cada conexión y deduce valores razonables para la terminación del tiempo.
E TCP computa el tiempo transcurrido entre que envía el segmento y recibe el acuse de recibo. Este tiempo se llama *ejemplo de viaje redondo*. El software TCP almacena el tiempo estimado de viaje redondo (RTT) como promedio calculado y utiliza nuevos RTT para cambiar lentamente dicho promedio.

$$RTT = (\alpha * OLD\_RTT) + ((1- \alpha) * NEW\_RTT))$$

Donde $0 <= \alpha < 1$ (Si α es cercano a 1 el promedio sera inalterable ante cambios mínimos de tiempo, lo contrario pasa si α es cercano a 0).
Entonces la terminación del tiempo queda dada con:

$$Terminación\ del\ tiempo = \beta * RTT$$

Determinar β puede ser difícil pues por una parte para determinar con rapidez la pérdida de paquetes, el valor de terminación de tiempo debe acercarse a RTT,osea β debe acercarse a 1. Por otra parte si $\beta = 1$ el TCP se vuelve muy ansioso y cualquier retraso pequeño causara una retransmisión innecesaria.

Medición precisa de muestras de viaje redondo

Consideremos una retransmisión: el TCP forma un segmento, lo coloca en un datagrama y lo envía, el tiempo termina y el TCP vuelve a enviar el segmento en un segundo datagrama. Como ambos datagramas llevan exactamente los mismos datos, el receptor no podrá saber si un acuse de recibo corresponde al datagrama original o al reenviado. Este fenómeno se conoce como ambigüedad en acuse de recibo.
Entonces, debe TCP asumir que los acuses de recibo pertenecen a la primera o a la ultima transmisión? Ninguno de los dos casos funciona.
En el primero, puede causar que el tiempo estimado de viaje redondo aumente sin medida en los casos en los que una red de redes pierda el datagrama.
En el segundo, hará que la muestra de viaje redondo sea mucho mas pequeño y resultara en una ligera disminución del RTT.

Entonces que debe hacer TCP? La respuesta es sencilla: no actualizar la estimación de viaje redondo cuando son segmentos retransmitidos. Esta idea es conocida como el *algoritmo de Karn.* Una implantación sencilla de éste algoritmo también induciría fallas, pues si el TCP envía un segmento de una aumento significativo en el retraso, se forzará la retransmisión y si sigue así infinitamente ignorando los segmentos retransmitidos nunca actualizará la estimación.
Para resolver dichas fallas se utiliza la estrategia de anulación *del temporizador.* La técnica computa el tiempo de estimación por medio de una formula, sin embargo si se termina el tiempo y se provoca una retransmisión, el TCP aumenta el valor de terminación de tiempo.

$$NEW\_TIMEOUT = \gamma * OLD\_TIMEOUT$$

Donde por lo general, γ es dos.

Básicamente Karn establece que: computa el valor de terminación de tiempo de la estimación actual de viaje redondo. Pero luego anula la terminación en cada retransmisión hasta que pueda transferir un segmento con éxito. Cuando llega el acuse de recibo del segmento que no requirió retransmisión, el TCP recalcula la estimación de viaje redondo y restablece la terminación de tiempo.

Respuesta al congestionamiento

El congestionamiento es una condición de retraso severo, causada por una sobrecarga de datagramas en uno o más puntos de conmutación (ruters por ejemplo). Se sabe que cada ruter tiene una capacidad finita de almacenamiento. Luego, si esta congestionado, podrá almacenar cierta cantidad de datagramas y el resto deberá descartarlos.

Los puntos extremos no conocen los detalles del congestionamiento, para ellos esto significa un mayor retraso. Y en consecuencia un mayor número de retransmisiones que lo único que hacen es empeorarlo.
Para evitar el congestionamiento, TCP aplica dos técnicas: arranque lento y disminución multiplicativa.
Para cada conexión TCP recuerda el tamaño de la ventana del receptor, ahora para controlar el congestionamiento también sabrá el *limite de ventana de congestionamiento*. En cualquier momento, el TCP actúa como si el tamaño de la ventana fuera:

ALLOWED_WINDOW = MIN (RECEIVER_WINDOW, CONGESTION_WINDOW)

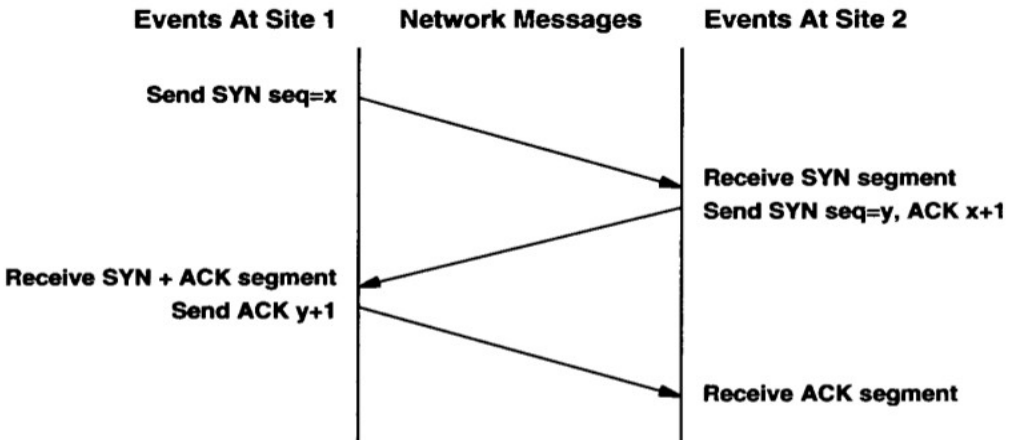
Prevención del congestionamiento por disminución multiplicativa: Cuando se pierda un segmento, se reduce a la mitad la ventana de congestionamiento. Para los segmentos que permanezcan en la ventana permitida, anular exponencialmente el temporizador para la transmisión. Es decir, mientras que el congestionamiento continúe, TCP reducirá la velocidad y la cantidad de tráfico a fin de que los ruteadores tengan tiempo de deshacerse de las colas de espera.
Para recuperarse el TCP luego del congestionamiento, emplea la técnica de *arranque lento*: siempre que se arranque el trafico en una nueva conexión o se aumente el trafico luego de un periodo de congestionamiento, activar la ventana de congestionamiento con el tamaño de un solo segmento y aumentarla un segmento cada vez que llegue un acuse de recibo.
También se le agrega una regla adicional que dice que cuando el tamaño de la ventana de congestionamiento llegue a la mitad de su tamaño original, se hace mas lenta la velocidad de incremento.

Establecimiento de una conexión TCP

Para establecer una conexión TCP implementa un saludo (handshake) de tres etapas. El primer segmento se identifica por tener en 1 el bit SYN. El segundo tiene activo tanto el bit SYN como el ACK. El tercero es solo un acuse de recibo para informar que ambos extremos están de acuerdo en establecer conexión. El saludo esta diseñado para que funcione también en casos en los que ambas máquinas intenten iniciar una conversación al mismo tiempo. Una vez que se

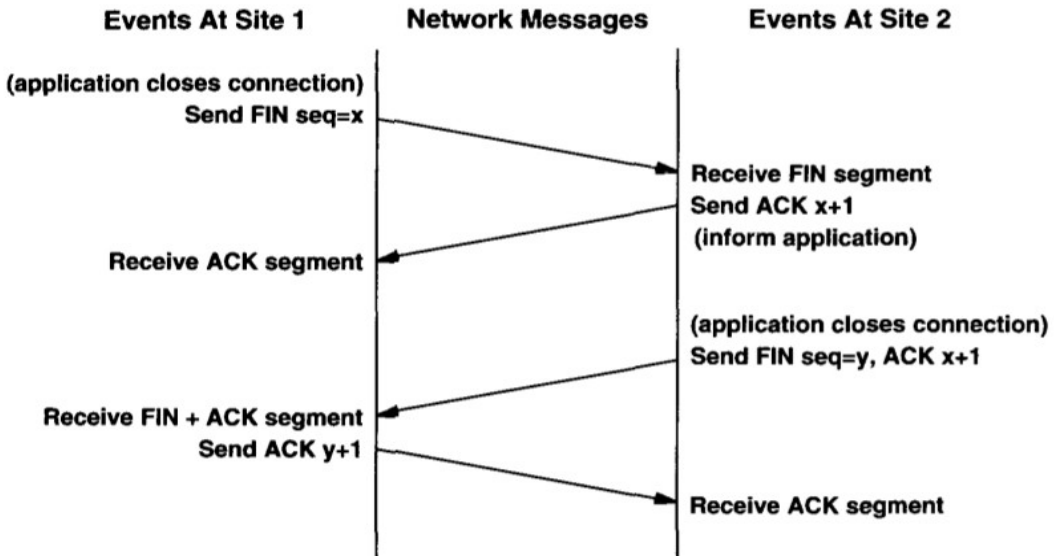
establece la conexión, los datos pueden fluir en ambas direcciones por igual.

El saludo además de garantizar que ambos lados estén listos para transferir datos, permite a ambas partes acordar un número de secuencia inicial. Éstos son enviados y reconocidos durante el saludo. Son propios de cada máquina y son elegidos de forma aleatoria. Se utilizaran para identificar octetos en el flujo que se está enviando.



Terminación de una conexión TCP

Dos programas que utilizan TCP para comunicarse, pueden terminar la conversación cortésmente mediante la operación *close.* De manera interna, TCP utiliza una modificación del saludo para cerrar las conexiones.
Recordemos que las conexiones TCP son *full duplex,* es decir que contienen dos transferencias de flujo independientes, una en cada dirección.
Cuando un programa de aplicación informa al TCP que ya no tiene más datos para enviar, éste cerrará la conexión en una dirección. Para cerrar una mitad de la conexión el emisor TCP termina de transmitir los datos restantes, espera el acuse de recibo y entonces envía un segmento con el bit FIN activado. Mientras tanto, los datos pueden fluir en la dirección opuesta hasta que el emisor se cierra. Cuando ambas conexiones se han cerrado el TCP borra los registros de la conexión en cada punto extremo.



Síndrome de ventana tonta

Es un serio problema que se presenta en el desempeño de TCP, cuando las aplicaciones del emisor y receptor operan a velocidades diferentes.

Si el receptor es muy lento procesando la información recibida, enviará acuses de recibo solicitando en el campo ventana tamaños muy pequeños, incluso de un sólo octeto. Si no se hace nada por evitarlo, el receptor adaptará su ventana de transmisión a un octeto y continuará transmitiendo datagramas con un sólo octeto de datos. Esto no viola las normas del protocolo, pero es nefasto para el rendimiento de la red, pues se desperdicia mucho ancho de banda y esfuerzo computacional de las máquinas implicadas.

La ventana tonta también se puede producir por culpa del emisor, por una implementación del protocolo demasiado agresiva, que envíe los datos según le llegan de la capa de aplicación. También un desajuste entre el tamaño de los bloques de datos enviados por una aplicación y el de los segmentos del protocolo puede hacer que último segmento de cada bloque esté casi vacío de datos.

Prevención en el lado del receptor

La solución en el receptor es retener el envío de los segmentos hasta que el espacio libre en la ventana de recepción se haya hecho lo suficientemente grande. Normalmente esta cantidad suele ser la mitad de la ventana o el tamaño mayor de un segmento.

Antes de empezar a retardar el acuse de recibo, ha de enviarse uno indicando tamaño de ventana cero, lo cual suspenderá al emisor, permitiendo así que el receptor se recupere a medida que la aplicación cliente va leyendo datos.

El inconveniente del retardo en los acuses de recibo es que, si se retrasa mucho, el emisor dará por perdidos los datos y los retransmitirá, produciendo una carga innecesaria en la red. Otro efecto negativo es la confusión que se puede producir en el emisor al medir los tiempos de ida y vuelta.

Para minimizar estos efectos, el estándar TCP establece un tiempo máximo de retención de asentimientos de 500 milisegundos, recomendándose que se retorne un asentimiento para cada segmento recibido, con el objetivo de facilitar las estimaciones de tiempos de ida y vuelta.

Prevención en el lado del emisor

La solución equivalente en el lado del emisor es retardar el envío de cada segmento hasta que se hayan acumulado suficientes datos para que el envío sea rentable. A esta técnica se le llama *clumping* (agrupamiento).

La dificultad está en que el protocolo no puede saber cuánto tardará el programa de usuario en llenar suficientemente el buffer, ni conoce cuál es el retardo mínimo que la aplicación requiere. Si espera demasiado, los retardos serán excesivos, si espera poco, los segmentos serán muy pequeños, empeorando el rendimiento.

La solución que enunció Nagle, ampliamente utilizada, es especialmente elegante y flexible, combinando diferentes retardos según las condiciones de la red en cada momento (self clocking, auto-temporización) con la estimación de las necesidades de la aplicación a partir de la llegada de acuses de recibo desde el receptor.

El procedimiento es el siguiente: cuando una aplicación genera datos adicionales para enviarse sobre una conexión por la que se han transmitido datos recientemente, pero de los cuales no hay aún acuse de recibo, deben retenerse éstos en memoria intermedia hasta que se haya alcanzado el tamaño máximo de un segmento, aún cuando la aplicación haya solicitado una operación push. Si en el intervalo de espera llega un acuse de recibo, se debe desencadenar el envío inmediatamente.

A través de esta técnica se consigue la adaptación a combinaciones arbitrarias de retardos de red, tamaños diversos de segmento y velocidades de las aplicaciones, sin disminuir en general el rendimiento.