Capitulo 1

Protocolo de Control de Transmisión (TCP) .

1. INTRODUCCIÓN

TCP es un protocolo de transporte orientado a conexión. Esto hace que los datos se entreguen sin errores, sin omisión y en secuencia.

Tiene las siguientes características:

- Protocolo orientado a conexión. Es decir, las aplicaciones solicitan la conexión al destino y luego usan están conexión para entregar los datos, garantizando que estos serán entregados sin problemas.
- Punto a punto. Una conexión TCP tiene dos extremos, emisor y receptor.
- Confiabilidad. TCP garantiza que los datos transferidos serán entregados sin ninguna perdida, duplicación o errores de transmisión.
- Full duplex. Los extremos que participan en una conexión TCP pueden intercambiar datos en ambas direcciones simultáneamente.
- Conexión de inicio confiable. El uso de three-way handshake garantiza una conexión de inicio confiable y sincronizada entre los dos extremos de la conexión.
- Conexión de finalización aceptable. TCP garantiza la entrega de todos los datos antes de la finalización de la conexión.

TCP es un protocolo de nivel 4 (transporte) en la capa del OSI, por eso necesita valerse de IP para el envío de sus segmentos o mensajes. De esta manera IP trata el mensaje TCP como la información que debe entregar y en ningún momento intenta interpretar su contenido, como generalmente se hace al pasar un mensaje de una capa inferior a otra. Por eso un router o cualquier dispositivo de nivel 3 del OSI solo puede observar los encabezados IP para el reenvío de los datagramas. El encargado de interpretar los mensajes TCP, después de recibirlos de la capa de red, es el TCP de la maquina de destino.

1.1. Función de TCP

TCP es un protocolo de tamaño considerable, que cumple con una gran número de funciones:

- Asociar puertos con conexiones.
- Establecer conexiones usando un acuerdo en tres pasos.
- Realizar un arranque lento para evitar sobrecargas.
- Dividir los datos en segmentos para su transmisión.
- Numerar los datos.
- Manejar los segmentos entrantes duplicados.
- Calcular las sumas de control.
- Regular el flujo de datos usando las ventanas de envío y recepción.
- Terminar las conexiones de manera ordenada.
- Abortar conexiones.
- Marcar datos urgentes.
- Confirmación positiva de retransmisión.
- Calculo de los plazos de retransmisión.
- Reducir el trafico cuando la red se congestiona
- Indicar los segmentos que llegan en desorden.
- Comprobar si las ventanas de recepción están cerradas.

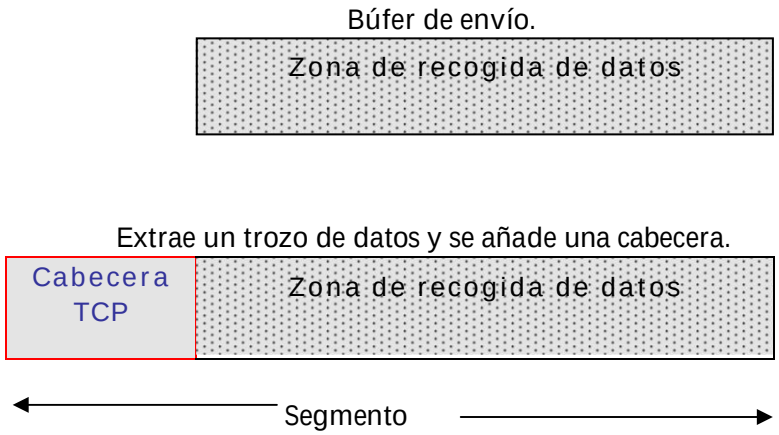
1.2. CONCEPTOS

1.2.1. Flujos de entrada y salida

El modelo conceptual de una conexión es que una aplicación envía un flujo de datos a otra aplicación pareja. Al mismo tiempo, recibe un flujo de datos de la otra. TCP proporciona un servicio dúplex que maneja simultáneamente los dos flujos de datos.

1.2.2. Segmentos

TCP debe convertir los flujos de datos salientes de una aplicación de forma que se puedan entregar como datagramas. La aplicación traslada los datos a TCP y este los sitúa en un búfer de envío. Toma un trozo de los datos, le añade una cabecera, creando un segmento.



TCP traslada el segmento a IP para que lo entregue como un único datagrama. El empaquetado de datos en trozos del tamaño adecuado permite usar de manera más eficiente los servicios de transmisión, por lo que TCP debería esperar a recoger una cantidad razonable de datos antes de crear un segmento.

1.2.3. Push

En algunas ocasiones el programa cliente necesita que TCP entregue a la aplicación del servidor remoto unos datos lo antes posible, para esto utiliza una la función {push}. Imaginemos que un programa cliente ha iniciado una sesión interactiva con un servidor remoto y el usuario ha tecleado un comando y pulsado *retorno de carro*. El programa cliente usara esta función para decir a TCP que entregue lo antes posible dichos datos.

1.2.4. Puertos de aplicación

El interfaz que esta entre TCP y el proceso local se llama puerto. Para que una aplicación pueda acceder a la red y pueda enviar datos a través de ella lo debe hacer a través de un puerto. La aplicación utiliza un número de puerto para enviar el flujo de datos y el otro extremo de la conexión los recibe a través de otro número de puerto.

Los puertos se identifican mediante un número decimal que va desde el 0 hasta el 65.535, tanto en TCP como en UDP. Los fabricantes que implementan TCP disponen de una gran libertad para asignar números de puertos a los procesos, aunque la Autoridad de Números Asignados de Internet (IANA) ha dedicado un rango que va desde el 0 al 1.023 a una serie de procesos comunes (RFC 1700) como telnet, ftp, pop3, smtp, etc.

Cada vez que un cliente quiere una conexión pide al sistema operativo que le asigne un número de puerto en desuso, sin reservar. Al finalizar la conexión, el cliente devuelve el puerto al sistema y lo puede utilizar otro cliente.

Puerto	Aplicación	Descripción
9	Discard	Descartar todos los datos entrantes
19	Chargen	Intercambiar flujos de caracteres
20	FTP-Data	Puerto de transferencia de datos para la transferencia de archivos
21	FTP	Puerto de dialogo para la transferencia de datos
23	Telnet	Puerto de conexión remota mediante Telnet
25	SMTP	Puerto del Protocolo simple de transferencia de correo
110	POP3	Servicio de recuperación de correo electrónico
119	NNTP	Servicio de publicación de noticias de red

Listado de puertos TCP estandarizados.

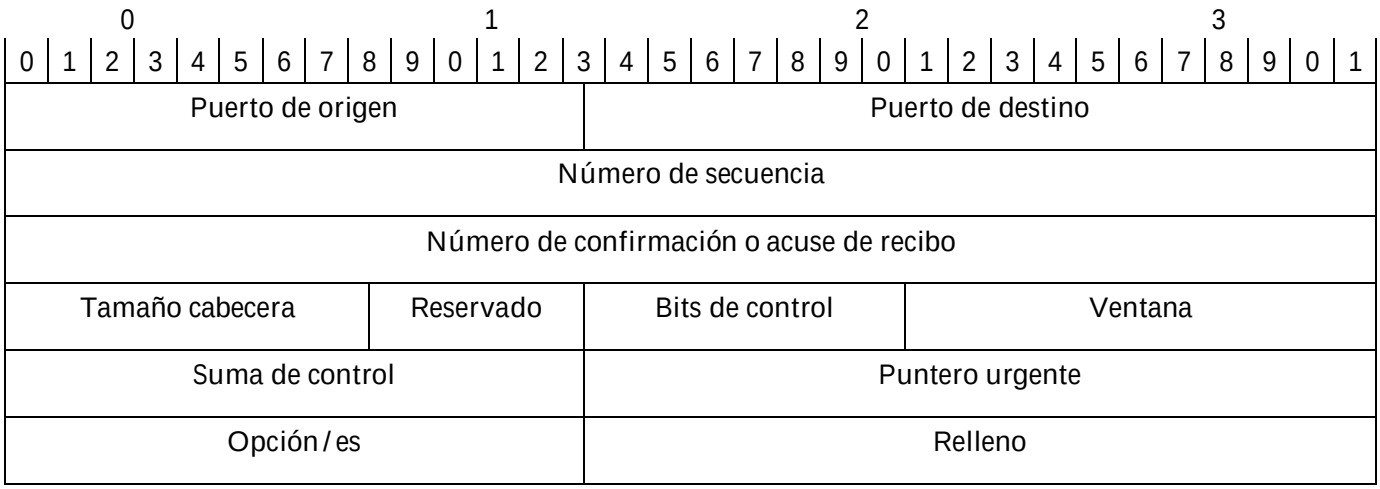
Para especificar plenamente una conexión, la dirección IP del host se añade al número de puerto. Esta combinación se denomina *socket* (*enchufe*). Por tanto, un número de socket es único en toda la interred. Una conexión entre dos host queda totalmente descrita por los sockets asignados a cada Terminal de la conexión. La conexión entre dos sockets proporciona una ruta de comunicación bidireccional (duplex total) entre los dos procesos.

TCP utiliza dos tipos de sockets:

- Sockets de corriente. Se utilizan con TCP para lograr un intercambio de datos fiable, secuencial y bidireccional.
- Sockets de datagrama. Se utilizan con UDP para lograr transferencias de datos no fiables y bidireccionales.

Los sockets constituyen una interfaz de programa de aplicación (API) entre TCP, los procesos y las aplicaciones. Esta API permite a los programadores que sus aplicaciones accedan a TCP.

1.3. FORMATO DE UNA CABECERA TCP



DATOS

Formato de la cabecera TCP

Bits de control					
URG	ACK	PSH	RST	SYN	FIN

TCP da formato a una cabecera para cada segmento transmitido a IP. Cuando IP construye un datagrama IP, la cabecera TCP sigue a la cabecera IP.

Los segmentos IP se organizan en palabras de 16 bits. Si un segmento contiene un número impar de octetos se rellena con un octeto final compuesto por ceros en su totalidad.

Los campos de la cabecera TCP son los siguientes:

- Puerto de origen (16 bits). Especifica el puerto del modulo TCP de origen.
- Puerto de destino (16 bits). Especifica el puerto del modulo TCP de destino.
- Número de secuencia (32 bits). Especifica la posición secuencial del primer octeto de datos del segmento. Cuando el segmento abre una conexión *(bit SYN activado),* el número de secuencia es el número de secuencia inicial (ISN) y el primer octeto del campo de datos tiene como número ISN+1.
- Número de acuse de recibo (32 bits). Especifica el siguiente número de secuencia que espera el emisor del segmento. TCP indica que este campo se activa estableciendo el bit ACK, lo cual sucede siempre que se establece una conexión.

- Tamaño de la cabecera (4 bits). Especifica el número de palabras de 32 bits que componen la cabecera TCP. El campo Opciones se rellena con ceros para formar una palabra completa de 32 bits si es necesario.

- Reservado (6 bits). Debe tener el valor cero. Reservado para usos futuros.

- Bits de control (6 bits). Son 6:

    - ACK. Si esta a 1, indica que el campo número de acuse de recibo es significativo.

- URG. Si esta a 0 indica que debe ignorarse. Si esta a 1 indica que los datos son urgentes.
- PSH. Inicia una función {push}. Indica si TCP debe entregar inmediatamente los datos a la aplicación.
- RST. Indica un error, también se usa para abortar una sesión.
- SYN. Sincroniza los contadores de secuencia de la conexión. Este bit se activara (1) cuando un segmento solicita la apertura de una conexión.
- FIN. Final de la transmisión y cierre de la conexión. Se pone a 1 durante la terminación correcta.

- Ventana (16 bits). Especifica el número de octetos que el destinatario del segmento puede aceptar comenzar por el octeto especificado en el campo de acuse de recibo.

- Suma de verificación (16 bits). Una suma de verificación basada en la cabecera y los campos de datos. No incluye el relleno utilizado para que un segmento contenga un número par de octetos. La suma de verificación también se basa en una pseudocabecera de 96 bits.

- Puntero urgente (16 bits). Identifica el número de secuencia del octeto que sigue a los datos urgentes. El puntero urgente es un desplazamiento positivo desde el número de secuencia del segmento.

- Opciones (variable). Las opciones pueden cumplir varias funciones: final de la lista de opciones, tamaño máximo del segmento, datos opcionales del tamaño máximo de segmento, etc,etc.

- Relleno (variable). Octetos con valor cero que se añaden a la cabecera para redondear su longitud a 32 bits.

Una pseudocabecera TCP de 12 octetos incluye la dirección de origen y destino, el protocolo y la longitud del segmento. Esta información se envía con el segmento a IP para proteger a TCP de los segmentos erróneamente encaminados. El valor de campo de longitud incluye la cabecera y los datos TCP, pero no tiene en cuenta la pseudocabecera.

<table>
<tr><td colspan="32">0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1</td></tr>
<tr><td colspan="32">Dirección de origen</td></tr>
<tr><td colspan="32">Dirección de destino</td></tr>
<tr><td colspan="11">Cero</td><td colspan="11">Protocolo</td><td colspan="10">Longitud TCP</td></tr>
</table>

1.3.1 Opciones de tamaño máximo de segmento

La opción de tamaño máximo de segmento (MSS- maximum segment size) se usa para indicar el tamaño del mayor trozo de datos que se puede recibir (y reensamblar) de un flujo de datos. El tamaño máximo de segmento del sistema se define como:

El tamaño del mayor datagrama que se puede recibir – 40

Dicho de otra manera, el MSS informa del mayor tamaño de carga útil de datos del receptor cuando las cabeceras de IP y de TCP son de 20 bytes. Si existe cualquier número de opciones de cabecera, hay que restar su tamaño. Es decir, para calcular el tamaño de los datos que se pueden empaquetar en un segmento, TCP necesita calcular:

MSS anunciado + 40 – (tamaño de las cabeceras de IP y TCP)

Normalmente, los extremos intercambian sus correspondientes valores de MSS junto con el mensaje inicial {SYN} en el establecimiento de una conexión. Si el sistema no indica su tamaño de segmento, se supone un valor de MSS por defecto de 536 bytes y como máximo 65.535 bytes.

El MSS impone una restricción límite en el tamaño máximo que TCP puede enviar, el receptor no puede manejar nada que sea mayor. Pero el TCP emisor podría enviar segmentos menores debido al tamaño del MTV de la ruta de conexión.

1.3.2. La cabecera en la solicitud de una conexión

El primer segmento que se envía para iniciar una conexión tiene la bandera {SYN} a 1 y la bandera {ACK} a 0. El segmento {SYN} inicial es el único segmento que tiene el campo {ACK} a 0. Los cortafuegos lo utilizan para controlar las solicitudes entrantes de sesiones TCP.

El campo {*número de secuencia*} (sequence number) contiene un número inicial de secuencia. El campo {*Ventana* } (Window) contiene el tamaño inicial de la ventana de recepción.

La única opción que se ha definido actualmente en TCP es el tamaño máximo de segmento que podría recibir TCP. El tamaño máximo de este es de 32 bits y, normalmente,

se incluye en la solicitud de conexión, en el campo {*opciones*} (si no se indica el MSS, el valor por defecto es de 536). El tamaño de la cabecera {SYN} de TCP que contiene la opción MSS es de 24 bytes.

1.3.3. La cabecera en la respuesta de conexión

En una respuesta de aceptación de conexión, las dos banderas {SYN} y {ACK} están a 1. El número inicial de secuencia del que responde se encuentra en el campo {*número de secuencia*} y el de la ventana de recepción está en el campo {Ventana}. El tamaño máximo del segmento que desea recibir quien responde se incluye, normalmente, en la respuesta de la conexión, en el campo {opciones}. Puede ser de tamaño diferente de quien inicia la conexión, no tienen porque ser iguales.

Se puede rechazar una solicitud de conexión poniendo en la respuesta la bandera {RST} a 1.

1.3.4. Elección del número inicial de secuencia

Durante el establecimiento de la conexión, la norma de TCP indica que cada extremo de la conexión debe elegir un {número inicial de secuencia} a partir de un reloj interno de 32 bits. ¿Por qué?. Imagina lo que ocurre cuando un sistema deja de funcionar. Suponga que un usuario ha abierto la conexión justo antes de que dejase de funcionar y ya ha enviado una pequeña cantidad de datos. Tras recuperarse, el sistema no recuerda nada sobre lo que estaba haciendo antes de dejar de funcionar, lo que incluye las conexiones establecidas y los números de puertos asignados. Los usuarios han de volver a comenzar las conexiones de nuevo. Los números de puertos empiezan a asignarse de forma que el primero que lo pide, es el primero a quien se le da, y algunos de dichos puertos podría que estuviesen usándolos otras conexiones hacia tan solo unos segundos. Durante ese intervalo, un sistema lejano de una de las conexiones podría no haberse dado cuenta que el otro extremo había dejado de funcionar y había vuelto hacerlo, podría crearse una gran confusión si datos antiguos que tardan en llegar a la red se entremezclasen con datos de una nueva conexión. Haciendo que los nuevos empiecen con cierto valor de reloj, se impide que se cree este problema. Los datos antiguos, probablemente, estarán numerados con valores fuera del nuevo intervalo de {número de secuencia}

1.3.5. Uso general de los campos

Para preparar la transmisión de una cabecera de TCP se incluye el número de secuencia del primer octeto en el campo {número de secuencia}.

Se rellena el campo {número de confirmación} con el número del siguiente octeto que se espera del otro extremo y se pone el bit {ack} a 1.

El campo {ventana} contiene el tamaño actual de la ventana de recepción, es decir: el número de bytes, empezando con el {número de confirmación}, que se puede recibir. De esta forma se consigue un control de flujo muy preciso. Se dice al otro extremo el estado exacto de la ventana de recepción durante la sesión.

Si la aplicación envía un {push} a TCP, se pone la bandera {push} a 1. El TCP receptor se supone que reacciona ante la bandera {push} entregando los datos a su aplicación lo más rápido que la aplicación esta dispuesta a recibirlos.

Si la bandera {urgent} está a 1, indica que hay datos urgentes pendientes y el apuntador {urgente} apunta al último octeto de datos urgentes.

La bandera {reset} se pone a 1 para abortar una conexión. También se puede poner en respuesta a un segmento que no tiene sentido en la conexión actual que maneja TCP.

La bandera {fin} se pone a 1 en los mensajes que sirven para cerrar una conexión.

1.3.6. Suma de control

La suma de control de IP solo se aplica a la cabecera de IP. La suma de control de TCP se aplica sobre el segmento completo así como a una pseudocabecera que se construye con información extraída de la cabecera de IP.

El tamaño de TCP se calcula sumando el tamaño de la cabecera de TCP más el tamaño de los datos. La suma TCP es obligatoria, no opcional como en UDP. En un segmento entrante se calcula la suma de control de la cabecera TCP. Si los valores no coinciden, el segmento se descarta.

1.4. ESTABLECIMIENTO DE UNA CONEXIÓN

Antes de poder comunicarse, cada parte llama a una subrutina que crea un bloque de memoria para almacenar los parámetros de TCP y de IP durante la conexión, como las direcciones de los conectores (sockets), los números actuales de secuencia, el valor inicial de IP para el tiempo de vida y otros. Cuando un cliente desea acceder al servidor lanza una

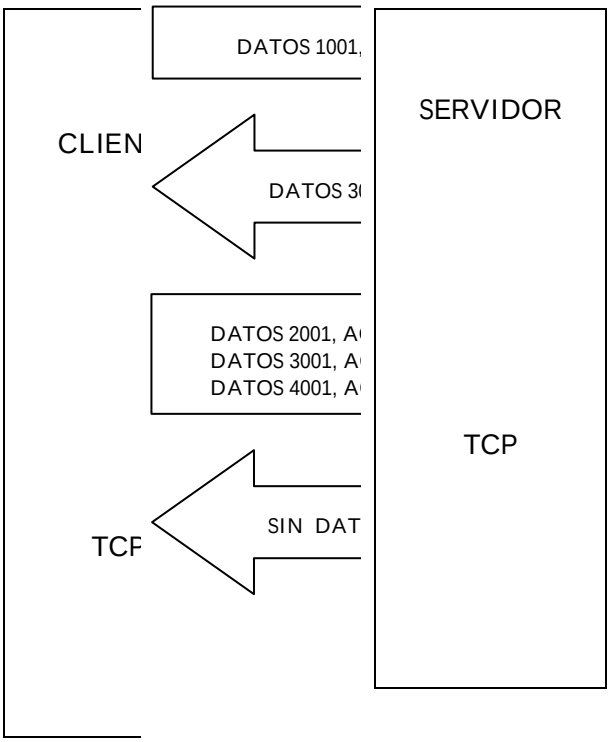
solicitud de conexión mediante la dirección de IP y el puerto del servidor. El procedimiento de conexión se denomina un acuerdo de tres pasos: SYN, SYN y ACK.

Durante el establecimiento de la conexión se intercambian importantes elementos de información. Cada parte notifica a la otra:

- Del espacio disponible en su búfer para recibir datos.
- La cantidad máxima de datos que puede llevar un segmento.
- El número inicial de secuencia que se usara para numerar los datos de salida.

1.5. TRANSFERENCIA DE DATOS

La transferencia de datos comienza después de terminar el establecimiento en tres pasos.



Flujo de datos y sus ACK.

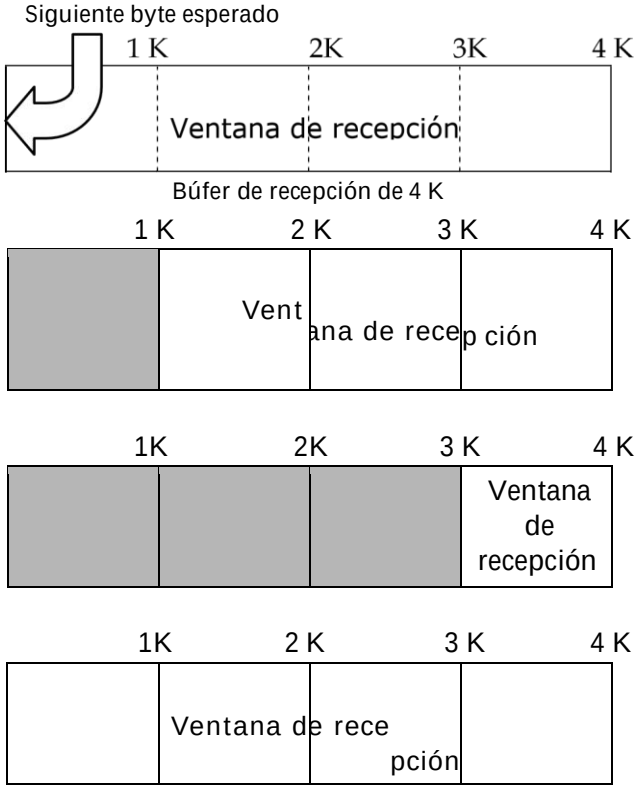
En el ejemplo anterior se muestra un intercambio directo de datos. Para que la numeración resulte sencilla, se usan mensajes de 1000 bytes.

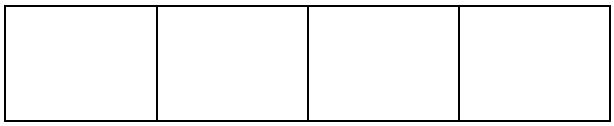
Todas las cabeceras de los segmentos de TCP llevan un campo ACK que identifica el número de secuencia del siguiente byte que se espera del otro extremo. El primer segmento que envía el cliente contiene los bytes del 1001 al 2000. Su campo ACK anuncia que el número de secuencia del siguiente byte que espera del servidor es 3001. El servidor responde con un segmento que contiene 1000 bytes de datos que empiezan en el 3001. El campo ACK de la cabecera de TCP indica que se han recibido correctamente los bytes 1001 a 2000, por lo que el número de secuencia que se espera del siguiente byte del cliente es 2001. A continuación, el cliente envía segmentos que empiezan en los bytes 2001, 3001 y 4001. Hay que tener en cuenta que el cliente no tiene que esperar a que llegue el ACK de cada segmento. Se pueden enviar datos al otro extremo siempre que disponga de espacio no utilizado en el búfer. De este modo el servidor ahorra ancho de banda si se usa un único ACK para indicar que todos los segmentos se recibieron correctamente.

1.6. CONTROL DE FLUJO

El TCP que recibe los datos se encarga del flujo de los datos de entrada. El receptor decide cuantos datos desea aceptar y el emisor debe actuar dentro de estos límites. Durante el establecimiento de la conexión, cada parte asigna espacio para los búfer de recepción para esa conexión y lo comunica, {este es el número de bytes que pueden enviarme}. Este número suele ser múltiplo entero del tamaño máximo del segmento.

El flujo de datos llega al búfer de recepción y permanece ahí hasta que lo recoge la aplicación asociada a ese puerto de TCP.





La aplicación recoge los datos

La ventana de recepción se extiende desde el último byte que se ha confirmado hasta el final del búfer. En el ejemplo anterior, todo el búfer esta libre, por lo que hay una ventana de recepción de 4 K. Llega 1 K de datos y la ventana de recepción se reduce a 3 K. Llegan dos segmentos mas de 1K, con lo que la ventana se reduce a 1 K. Finalmente, la aplicación absorbe los 3 K de datos del búfer, con lo que el espacio libre para los datos aumenta (se puede ver como una ventana *se desplaza* hacia la derecha. En una sesión real el tamaño de la ventana de recepción varía según las necesidades de la aplicación

1.6.1. Ventana de recepción

La ventana de recepción es cualquier espacio del búfer de recepción que no esta ocupado. Los datos permanecen en el búfer de recepción hasta que la aplicación a la que van dirigidos los recoge.

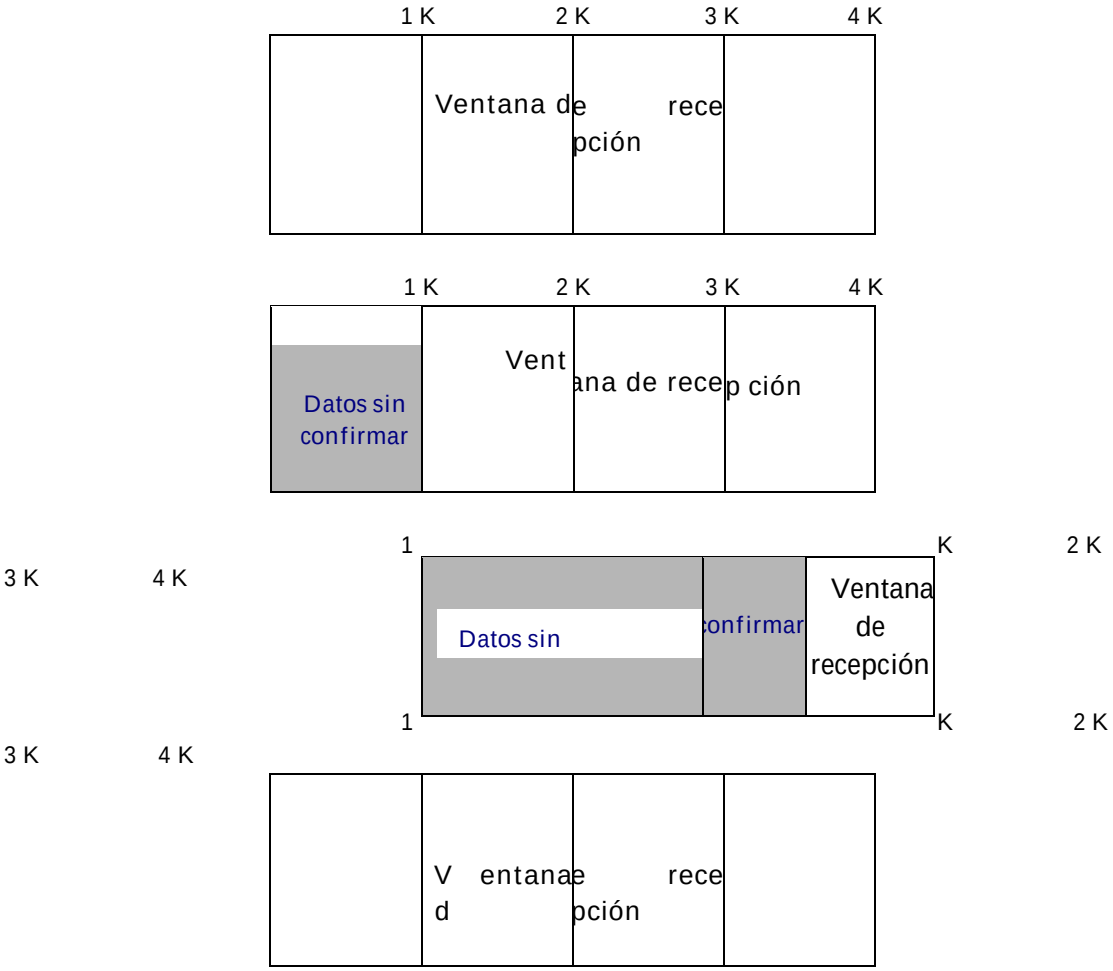
Los {ACK} del receptor contienen una actualización del estado de la ventana de recepción. El flujo de datos desde el emisor se regula de acuerdo con estas actualizaciones de la ventana.

La mayor parte del tiempo, el búfer de recepción que se establece durante el establecimiento de la conexión se mantiene durante la misma, aunque este puede crecer o reducir, siempre que el receptor no {se vuelva atrás} de lo que ha concedido al emisor.

1.6.2. Ventana de envío

El transmisor de los datos controla dos cosas; cuantos datos se han enviado y ya se han asentido y el tamaño actual de la ventana de recepción del otro lado. El espacio de envío activo va desde el primer octeto sin asentir hasta la parte derecha de la ventana de recepción actual. La parte de la ventana de recepción de este espacio indica cuantos datos adicionales se pueden enviar hacia el otro lado.

El número de secuencia inicial y el tamaño inicial de la ventana de recepción se obtienen durante la fase de establecimiento de la conexión.



En el ejemplo anterior:

1. El emisor comienza con una ventana de envío de 4 K.
2. El emisor transmite 1 K y mantiene una copia de dichos datos hasta que llegue el asentamiento, ya que puede ser necesario retransmitirlos.
3. Llega un {ACK} para los primeros 1 K y se envían otros 2 K de datos y mantiene una copia de estos.
4. Finalmente, llega un {ACK} que indica que todos los bytes transmitidos se han recibido. El {ACK} también actualiza la ventana del receptor a 4 K.

Pero hay que destacar algunas características:

- El emisor no tiene que esperar un {ACK} para cada segmento de datos transmitidos. La única limitación para la transmisión es el tamaño de la ventana de recepción.
- Imaginemos que para el emisor tiene que retransmitir datos que se enviaron utilizando segmentos muy pequeños (ej. 80 bytes). Los datos se pueden volver a empaquetar de una manera más eficiente, por ejemplo en un único segmento.

1.7. MANTENIMIENTO DE UNA SESIÓN

1.7.1. Sonda de ventanas

Si existe un emisor activo y un receptor perezoso se puede llegar a tener una ventana de recepción de 0 bytes. Es lo que se llama {ventana cerrada}. Cuando se abre un hueco se envía un {ACK} que actualiza el tamaño de la ventana. Pero, ¿qué ocurriría si se pierde el {ACK}?, pues que ambos extremos podrían esperar indefinidamente. Para evitar que ocurra esta situación, el emisor envía unos temporizadores persistentes cuando la ventana de recepción se cierra. Cuando el temporizador expira, se envía un segmento llamado {sonda de ventana} al otro extremo. En algunas implementaciones la sonda incluye datos. La sonda hace que el otro extremo envíe de vuelta un {ACK} con el estado actual de la ventana. Si la ventana continúa a cero, se duplica el valor del temporizador persistente. Este proceso se repite hasta que el valor del temporizador alcanza un máximo de 60 segundos. TCP continuara enviando sondas indefinidamente cada 60 segundos, o hasta que la ventana se abra o un usuario interrumpa el proceso o el temporizador de la aplicación expire.

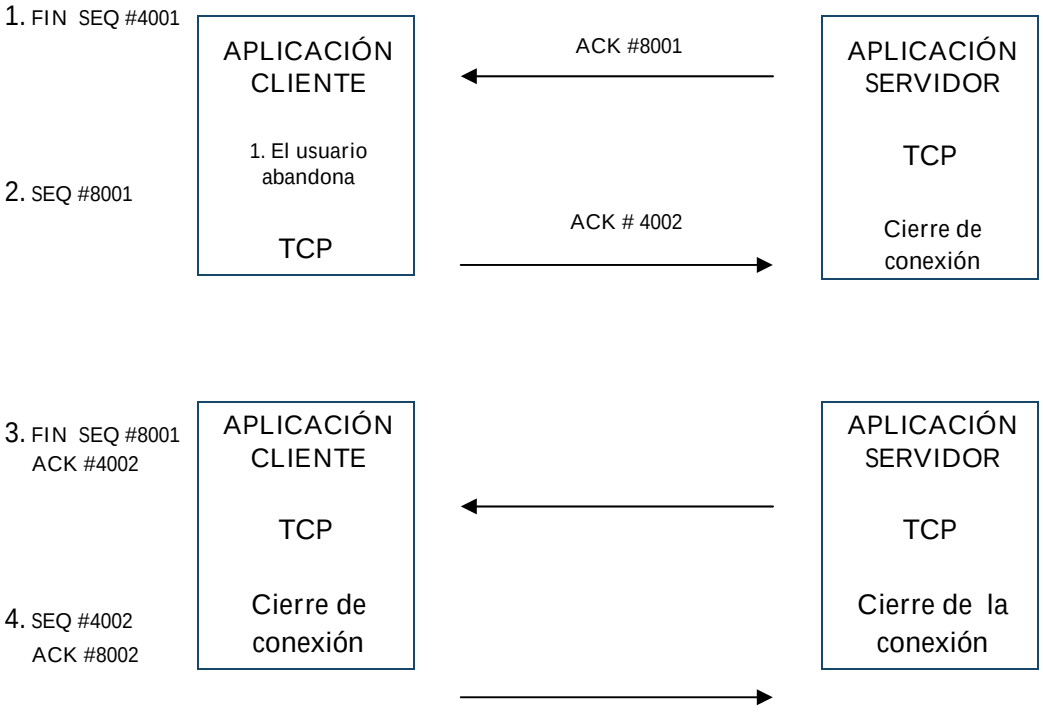
1.7.2. Mantenimiento de la sesión

¿Qué ocurre con una conexión en la que ninguna de las partes tiene datos que enviar durante un largo periodo de tiempo? Durante el periodo en que está libre, la red puede dejar de funcionar o se pueden cortar los cables y volverse a conectar. Basta que la red red este en funcionamiento cuando las dos partes vuelvan a intercambiar datos, la sesión no se pierde. Claro, el problema es que cualquier conexión ocupa bastante memoria en una computadora, por eso muchas implementaciones de TCP envían mensajes {mantener viva (keep-alive)} para comprobar si la conexión esta ociosa. TCP envía este mensaje periódicamente para comprobar que aún existe la conexión. Este mensaje devuelve una respuesta {ACK}. El uso de mensajes es opcional y si el sistema dispone de él, una aplicación puede desactivarlo para sus propias conexiones (el tiempo de plazo sugerido por defecto es de dos horas). Además, las aplicaciones pueden enviar sus propios temporizadores de la capa de aplicación en niveles que tengan sentido para la misma y que la aplicación puede abortar las sesiones ociosas.

1.8. TERMINACIÓN DE UNA SESIÓN

La terminación normal de una conexión se lleva a cabo mediante un proceso en tres pasos, parecido al del establecimiento. Cualquiera de las partes puede lanzar el proceso de terminación.

A. {He terminado. No tengo mas datos que enviar}
B. {Ok} o {Pero yo tengo algunos datos…..}
B. {Yo también he terminado}
A. {Ok}



1. La aplicación servidora indica a TCP que termina la conexión.
2. El TCP servidor envía un Segmento final (final Segment) {FIN}, informando a la otra parte que no va a enviar más datos.
3. El TCP cliente envía un ACK del segmento {FIN}
4. El TCP cliente notifica a su aplicación que el servidor quiere terminar.
5. La aplicación cliente indica a TCP que termine.

6. El TCP cliente envía un mensaje {FIN}
7. El TCP servidor recibe el {FIN} del cliente y responde con un {ACK} 8. El TCP servidor notifica a su aplicación que la conexión se ha terminado.

También se puede dar el caso de que aparezca un fallo de red y haga una terminación abrupta. Cualquiera de las partes puede invocar una terminación abrupta. Esta puede realizarse si una aplicación quiere abortar una conexión o si TCP ha detectado un problema serio en la comunicación que no se puede resolver. Para solicitar una terminación abrupta se envía uno o mas {reset} al otro extremo. El {reset} se indica mediante una bandera de la cabecera de TCP.

1.8.1. Fin de plazo

El sistema del otro extremo puede quedar fuera de servicio o el camino hasta el otro extremo puede cortarse o perderse debido a una pasarela o un enlace. ¿Cómo sabe TCP que no debe retransmitir los datos eternamente?. Existen varios mecanismos.

Tras llegar a un primer umbral en el número de retransmisiones, TCP avisa a IP que compruebe si el problema es un encaminador que ha quedado fuera de servicio y notifica a la aplicación que existe un problema. TCP procede a retransmitir hasta que se alcanza un segundo umbral, después se corta la conexión.

También puede llegar un mensaje ICMP indicando que no se podía alcanzar el destino por alguna razón. En algunas implementaciones, TCP seguirá intentando llegar al destino hasta que fallen los plazos, después de todo, puede que el problema se solucione. Después hace llegar a la aplicación el estado de {*destino inalcanzable*}.

Una aplicación también puede fijar su propio límite en el tiempo de entrega de los datos, junto con la acción que hay que tomar cuando el plazo expire. Normalmente la acción es cortar la conexión.

1.9. CONTROL DE ERRORES

TCP es un protocolo orientado a conexión y confiable. Por eso usa diversas técnicas para proveer la entrega confiable de datos. Estas técnicas permiten a TCP recobrarse de errores como paquetes perdidos, duplicados, retardados, diferentes velocidades de transmisión entre nodos y congestión.

- Paquetes perdidos. TCP usa una confirmación positiva con retransmisión para lograr la entrega de datos confiable. De este modo, el receptor envía mensajes de control de confirmación {ACK} al emisor para verificar la recepción exitosa de la información. A su vez, el emisor inicializa un {timer} al transmitir la información. Si el {timer} expira antes que la confirmación llegue, el emisor debe retransmitir la información inicializando un nuevo timer.
- Paquetes duplicados. Si el receptor recibe un paquete duplicado no lo toma en cuenta y procede a su descarte, ya que éste habrá sido tomado y marcado como recibido.
- Retardo de paquetes. Si un paquete no es recibido y el siguiente si, el receptor no mueve la ventana deslizante hasta que el segmento faltante sea recibido. De esta manera el receptor al no recibir el {ACK} reenvía el paquete perdido o retrasado.
- Diferentes velocidades de transmisión. Al establecer la conexión TCP, tanto el emisor como el receptor indican cual es su capacidad de almacenamiento intermedio para acordar cual será la velocidad a la cual la transmisión se llevará a cabo.
- Congestión. TCP implementa una política en la cual mantiene una ventana para medir la congestión, cada vez que un temporizador expira, ésta ventana es reducida. Para la decisión de envío de datos, el emisor toma en cuenta el tamaño de esta ventana para crear el tamaño de la ventana deslizante de datos.

Cada aplicación entrega arbitrariamente toda la información como un flujo de datos, luego TCP se encarga de separar esta información en segmentos, cada uno de los cuales tiene a lo más el tamaño de un paquete IP. El flujo dado por la aplicación es numerado por la cantidad de bytes transferidos y cada uno de estos segmentos contiene un número de secuencia de los bytes de información. Así, el receptor envía un segmento con el número de secuencia de la información confirmada, no de los segmentos. Los {ACK} son acumulativos, de esta manera un {ACK} puede ser la confirmación de varios segmentos.

Una de las causas por las que se pueden perder paquetes es el trafico excesivo. Pero algunos protocolos como TCP utilizan la retransmisión como mecanismo para garantizar la llegada de los paquetes. Este mecanismo es un mecanismo de doble filo ya que una retransmisión excesiva puede contribuir en la congestión.

TCP interpreta la perdida de un paquete como un indicador de congestión. El mecanismo de control de TCP es usado por el nodo emisor para detectar el nivel de congestión y si éste está sobre un cierto nivel de umbral considerado el máximo aceptable, la retransmisión de paquetes es reducida. El mecanismo utilizado consiste en que un host envía un paquete, y si una confirmación llega sin perdida, el emisor envía dos paquetes y comienza a aumentar la ventana en potencia de dos. Cuando TCP envía un número de paquetes igual a la mitad del tamaño de una ventana, la tasa de incremento disminuye hasta recibir las confirmaciones de los paquetes enviados.

1.10. RENDIMIENTO

Hay muchos factores que afectan al rendimiento. Los básicos son los recursos como la memoria y el ancho de banda de la red.

El ancho de banda y los retardos de la red subyacente imponen límites al rendimiento. Una baja calidad de transmisión implica un gran número de datagramas descartados. Al descartar se desencadenan retransmisiones, con lo que se reduce el ancho de banda efectivo.

Un receptor que dispone de un gran búfer de entrada permite que el emisor continúe transmitiendo sin interrupción. Resulta especialmente importante en las redes con grandes retardos, donde pasa mucho tiempo entre el envío y recepción de datos y la actualización de la ventana. Para conseguir un flujo continuado de datos desde el emisor al receptor, el destino necesita una ventana de recepción que sea al menos del tamaño de:

Ancho de banda X retardo

Ejemplo, si se pueden enviar 10.000 bytes por segundo y el {ACK} tarda 2 segundos en llegar, el receptor debe disponer de un búfer de al menos 20.000 bytes para mantener un flujo continuo de datos. Con un búfer que sólo puede guardar 10.000 bytes, el rendimiento se reduce a la mitad.

Otro factor que afecta al rendimiento es la capacidad del host para reaccionar a eventos de alta prioridad y realizar rápidamente {cambios de contexto}, es decir, dejar de hacer una cosa y pasar a hacer otra. Un host puede estar dando servicio a muchos usuarios locales interactivos, procesos de fondo y docenas de conexiones de comunicaciones. Una implementación que integra TCP/IP con el núcleo del sistema operativo puede reducir significativamente el sobrecoste del cambio de contexto. Se necesitan recursos suficientes de CPU para llevar a cabo rápidamente los pasos necesarios para el procesamiento de una cabecera de TCP.

Una CPU que no puede calcular rápidamente los valores de la suma de control puede ralentizar una transmisión de datos.

El software también influye en el rendimiento, si se escriben programas que no realicen pequeños envíos y recepciones innecesarios y con temporizadores que liberen recursos de red cuando no se este realizando ningún trabajo útil, también se mejorara el rendimiento.

Factores de rendimiento de TCP.

Aplicación *Enviar y recibir datos eficientemente.*
Administrador de sistema *Ajuste de los parámetros.*
Fabricante *Software de TCP/IP eficiente y conforme.*
Sistema operativo y Hardware *Disponibilidad de memoria, potencia de CPU, cambios de contexto.*

Red
Ancho de banda, retraso, calidad y capacidad.

1.10.1. Arranque lento

El {arranque lento} previene que una sesión incorpore grandes cantidades de trafico a una red que puede que ya este congestionada. Imagínese una empresa con 500 empleados que llegan a las 8:00 a.m. y empiezan todos a validarse en la red al mismo tiempo. La idea del {arranque lento} es que una nueva conexión empiece despacio, aumentando su tasa de transferencia gradualmente según las condiciones de la red. Durante el establecimiento de la conexión, el otro extremo indica su ventana de recepción. Se calcula el tamaño de una segunda {ventana de congestión} con el propósito de moderar la transmisión de acuerdo con las condiciones de la red. El emisor está limitado por la ventana de congestión en lugar de por la (mayor) ventana de recepción.

La ventana de congestión empieza con un segmento. Por cada segmento del que recibe un {ACK} correctamente, aumenta en un segmento la ventana de congestión, mientras sea menor que la {ventana de recepción}. Si la red no está sobrecargada, la ventana de congestión alcanza pronto el tamaño de la ventana de recepción. Durante el estado normal de la transmisión, el tamaño de la ventana de congestión es idéntico al tamaño de la ventana de recepción.

Hay que tener en cuenta que el arranque lento no es realmente lento. Tras el primer {ACK}, la ventana de congestión es de dos segmentos. Si llegan {ACK} para dos segmentos, la ventana se incrementa a cuatro segmentos. Si estos también se confirman correctamente, la ventana va creciendo exponencialmente.

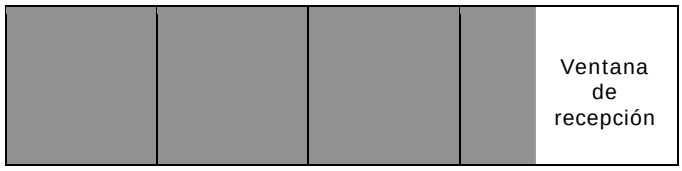
1.10.2. Síndrome de la ventana tonta

En las primeras implementaciones de TCP / IP se observo que ocurría con bastante frecuencia un fenómeno al que se llamo {síndrome de la ventana tonta (SWS)}. Ejemplo:

1. Una aplicación envía datos rápidamente.
2. La aplicación receptora lee atentamente byte a byte de datos del búfer de recepción.
3. Se llena el búfer
4. La aplicación receptora lee un byte, y TCP envía un {ACK} que dice {tengo sitio para un byte de datos}.
5. El TCP emisor empaqueta un byte y lo transmite.
6. El TCP receptor envía un {ACK} que dice {gracias ya lo tengo y no me queda más sitio}.
7. La aplicación receptora lee un byte y envía un {ACK}, repitiéndose el proceso.

La lenta aplicación receptora ya tiene un montón de datos esperando y darse prisa en insertar bytes en la parte derecha de la ventana no resulta útil, sino que añade un tráfico innecesario en la red.

Esta situación no tiene por que ser tan extrema para generar problemas. Se puede producir con un emisor rápido, una aplicación que recibe lentamente y lee trozos de datos de tamaño máximo de segmento y búfer de recepción casi lleno. Ejemplo:



Este búfer podría producir un {*síndrome de ventana tonta*}.

La solución es sencilla. Cuando la ventana se reduce a menos de un determinado tamaño, TCP empieza a mentir. TCP no debe decir al otro extremo que existe espacio adicional en la ventana cuando la aplicación receptora realiza una pequeña lectura. En lugar de ello, TCP debe mantener los recursos extra en secreto hasta que el tamaño objetivo de espacio de búfer quede libre. El tamaño recomendado es de un segmento, excepto en el caso en que en el búfer completo sólo quepa un segmento, en cuyo caso sirve medio segmento. El tamaño objetivo con el que TCP vuelve a decir la verdad es:

Mínimo (1/2 tamaño del búfer de recepción, tamaño máximo del segmento)

TCP empieza a mentir cuando el tamaño de la ventana es menor que dicha cantidad y dice la verdad cuando el tamaño de la ventana es al menos dicha cantidad. Tenga en cuenta que no ocurre nada malo por mantener los envíos, ya que la aplicación receptora no ha recogido la mayor parte de los datos que están esperando. La solución es sencilla,

supongamos que el búfer de recepción puede almacenar varios segmentos. El emisor rápido llenara el búfer de recepción y se indicara al emisor que no queda sitio disponible hasta que quede libre, suficiente espacio para recibir un segmento completo.

1.10.3. Retraso de los ACK

El retraso de los {ACK} es otro de los mecanismos que puede mejorar el rendimiento. Al reducir el número de {ACK} enviados, se ahorra ancho de banda que se puede utilizar para otro tráfico. Si el otro extremo espera un poco antes de enviar un {ACK}, existe la posibilidad de que:

- Pueda confirmar varios segmentos con un único mensaje {ACK}.
- La aplicación del otro extremo puede recoger datos para enviar dentro del plazo del temporizador, en cuyo caso el {ACK} puede ir en la cabecera del mensaje de salida y no haría falta un mensaje separado.

Para evitar el retraso de segmentos completos, por ejemplo cuando se realiza una transferencia de archivos, debería enviarse un {ACK} al menos por cada segmento. Muchas implementaciones usan un plazo de 200 milisegundos. Hay que recordar que retrasar los {ACK} no ralentiza la transmisión. Si llegan pequeños segmentos habrá mucho sitio en el búfer y el transmisor puede seguir enviando aunque las retransmisiones se ralenticen. Si llegan segmentos completos, cada segundo segmento activará inmediatamente un {ACK}.

1.10.4. Plazo de retransmisión

Tras el envío de un segmento, TCP arranca un temporizador y espera un {ACK}. Si el {ACK} no llega en el plazo establecido, TCP retransmite el segmento. ¿Cuál es ese plazo de espera? Si el plazo de retransmisión es demasiado corto, el emisor abarrotara la red con segmentos innecesarios y cargara al receptor con duplicados. Pero si los plazos son demasiado largos se pierde la oportunidad de recuperarse rápidamente cuando realmente se pierde un segmento y se reduce el rendimiento. ¿Cómo se elige el plazo ideal? Porque un valor que funciona bien en una Red de área local (LAN) de alta velocidad podría resultar desastroso en una conexión de larga distancia con muchos saltos, por eso un plazo fijo no puede ser la solución. Por eso existen algoritmos como los de Jacobson, Karn y Partridge que permiten que TCP se adapte a las condiciones cambiantes y así poder elegir el buen plazo de retransmisión.

1.10.5. Barreras al rendimiento

TCP ha demostrado ser muy flexible, funcionando en redes que transportan cientos o millones de bits por segundo. El protocolo ha conseguido buenos resultados en LANS Ethernet, Token-Ring o en la Interfaz de datos distribuidos por fibra (FDDI), así como en enlaces con poco ancho de banda y en enlaces con grandes retardos, como en los enlaces por satélite.

TCP está afinado para que responda en condiciones poco normales como las congestiones. Sin embargo, tiene algunas características que limitan la tasa de TCP usando tecnologías que ofrecen un ancho de banda de cientos o miles de megabytes por segundo.

Supongamos que se está realizando una transferencia de archivos entre dos sistemas y se quieren enviar los datos de la forma más eficiente posible. Imaginemos:

- El tamaño máximo del segmento es de 4 KB.
- La ventana de recepción es de 4 KB.
- Hay suficiente ancho de banda para enviar 2 segmentos por segundo.
- La aplicación receptora recoge los datos en cuanto llegan.
- Los {ACK} llegan tras dos segundos.

El emisor podrá enviar datos rápidamente, pues en cuanto se termina la asignación de ventana, llega un {ACK} que permite enviar otro segmento:

ENVIO SEGMENTO 1.
ENVIO SEGMENTO 2.
ENVIO SEGMENTO 3.
ENVIO SEGMENTO 4.

Han pasado dos segundos:

SE RECIBE EL {ACK} DEL SEGMENTO 1, SE PUEDE ENVIAR EL SEGMENTO 5
SE RECIBE EL {ACK} DEL SEGMENTO 2, SE PUEDE ENVIAR EL SEGMENTO 6
SE RECIBE EL {ACK} DEL SEGMENTO 3, SE PUEDE ENVIAR EL SEGMENTO 7
SE RECIBE EL {ACK} DEL SEGMENTO 4, SE PUEDE ENVIAR EL SEGMENTO 8

Han pasado otros dos segundos:

SE RECIBE EL {ACK} DEL SEGMENTO 5, SE PUEDE ENVIAR EL SEGMENTO 9

…….

Si la ventana de recepción hubiese sido de tan solo 2 KB, el emisor hubiese estado forzado a esperar un segundo de cada dos antes de poder enviar más datos. De hecho, para mantener un flujo continuo, la ventana de recepción ha de tener un tamaño de al menos:

Ventana= ancho de banda X tiempo de ida y vuelta

Aunque el ejemplo era exagerado, demuestra que una ventana pequeña puede ocasionar problemas en los enlaces de satélites con largo retardos. Otro ejemplo es lo que ocurre con conexiones con un gran ancho de banda. Si el ancho de banda y la tasa de de transmisión son de 10 millones de bits por segundo, pero el tiempo de ida y vuelta es de 100 milisegundos (1/10 segundos), la ventana de recepción para mantener un flujo continuo ha de mantener al menos 1.000.000 bits, es decir 125.000 bytes. Pero el mayor numero que se puede escribir en el campo ventana de recepción de la cabecera de TCP es de 65.536 bytes.

1.11. ESTADOS TCP

Una conexión de TCP pasa por cierto numero de estados. Primero se establece la conexión mediante un intercambio de mensajes, después se transmiten los datos y después se cierra la conexión mediante un intercambio de mensajes. Cada paso en el progreso de una conexión se corresponde con un estado de la conexión. El software de TCP en cada extremo de una conexión mantiene un registro del estado de su lado en todo momento.

Estado del servidor	Evento	Descripción
CLOSED		Estado ficticio anterior a una conexión.
	Apertura pasiva por la aplicación servidora	
LISTEN		El servidor espera una solicitud de conexión de un cliente.
	El TCP servidor recibe un {SYN}, envía {SYN / ACK}	El servidor ha recibido un {SYN} y ha enviado un {SYN / ACK}. Espera un {ACK}.
SYN-RECEIVED		
	El TCP servidor recibe el {ACK}	

ESTABLISHED	Se ha recibido el {ACK} y la conexión está abierta.

Transiciones de estado del servidor

Estado del servidor	Evento	Descripción
CLOSED		Estado ficticio anterior a una conexión.
	El cliente solicita una conexión. El TCP cliente envía un SYN.	
SYN-SEN		El TCP cliente ha enviado un {SYN} al servidor.
	El TCP servidor recibe un {SYN / ACK}, envía {ACK}	El cliente ha recibido un {SYN / ACK} del servidor y ha enviado de vuelta un {ACK}.
		Puede empezar la transferencia de datos.
ESTABLISHED		

Transiciones de estado del cliente.

Estado del servidor	Evento	Descripción
ESTABLISHED	La aplicación local solicita cerrar. TCP envía un {FIN / ACK}.	
FIN-WAIT-1		Quien cierra espera la respuesta del otro extremo. Hay que recordar que en este punto todavía pueden llegar datos del otro extremo.
	TCP recibe un {ACK}	
FIN-WAIT-2		Quien cierra ha recibido un {ACK} del otro extremo, pero no un {FIN}. Espera un {FIN} aceptando mientras tanto los datos que le llegan.
	TCP recibe un {FIN / ACK}. Envía un {ACK}.	

TIME-WAIT	La conexión se mantiene en un limbo para permitir que lleguen datos duplicados o {FIN} duplicados que puedan existir en la red. El periodo de espera es el doble del tiempo máximo de vida estimado de un segmento.
CLOSED	Se elimina toda la información sobre la conexión.

Transiciones de estado de quien cierra.

Estado del servidor	Evento	Descripción
ESTABLISHED	TCP recibe un {FIN / ACK}.	
CLOSE-WAIT-1		Llega un {FIN}
	TCP envía un {ACK}	
FIN-WAIT-2		TCP espera a que su aplicación indique que cierre. La aplicación puede, opcionalmente, enviar más datos.
	La aplicación local indica el cierre. TCP envía un {FIN / ACK}.	
LAST-ACK		TCP está esperando el ultimo {ACK}
	TCP recibe el {ACK}	
CLOSED		Se elimina toda la información sobre la conexión.

Transiciones de estado del extremo cerrado.

Se puede utilizar el comando *netstat –an* para examinar el estado actual de las conexiones. Esta herramienta proporciona información estadistica de las conexiones TCP/IP y de los protocolos activos.

Netstat proporciona la siguiente información de cada conexión:

- Proto. El prototipo de transporte utilizado para establecer la conexión.
- Local Address. El nombre o dirección IP de la computadora local y el numero de puerto utilizado por esa conexión.
- Foreign Address. La dirección externa y el numero o nombre del puerto asociado a la conexión.

- State. Muestra el estado de la conexión, solo de TCP

netstat tiene la siguiente sintaxis:

netstat [-a] [-e] [-n] [-o] [-s] [-p proto] [-r] [intervalo]

Sintaxis	Descrpción
-a	Muestra todas las conexiones y puertos de escucha. (Normalmente, el extremo servidor de las conexiones no se muestra).
-e	Muestra estadísticas Ethernet. Se puede combinar con la opción -s.
-n	Muestra números de puertos y direcciones en formato numérico.
-o	Muestra la Id. de proceso asociado con cada conexión.
-p proto	Muestra conexiones del protocolo especificado por proto; que puede ser TCP, UDP, TCPv6 o UDPv6. Si se usa con la opción -s para mostrar estadísticas por protocolo, proto puede ser TCP, UDP, TCPv6 o UDPv6.
-r	Muestra el contenido de la tabla de rutas.
-s	Muestra estadísticas por protocolo. De forma predeterminada, se muestran para IP, IPv6, ICMP, ICMPv6, TCP, TCPv6, UDP y UDPv; se puede utilizar la opción p para especificar un subconjunto de los valores predeterminados.
intervalo	Vuelve a mostrar las estadísticas seleccionadas, haciendo pausas en el intervalo de segundos especificado entre cada muestra. Presione Ctrl+C para detener la actualización de estadísticas. Si se omite, netstat imprimirá la información de configuración una vez.

Netstat sin argumentos proporciona el estado basico de las conexiones.

C:\netstat

Conexiones activas

Proto	***Direcci□ local***	***Direcci□ remota***	***Estado***
TCP	***cia:3012***	***baym-cs186.msgr.hotmail.com:1863***	***ESTABLISHED***
TCP	***cia:3032***	***194.224.100.71:http***	***CLOSE_WAIT***
TCP	***cia:3048***	***a217-75-98-16.deploy.akamaitechnologies.com:http***	***CLOSE_WAIT***

Con la opción –s muestra estadisticas para los protocolos IP,, ICMP, TCP y UDP.

C:\netstat -s

Estadísticas de IPv4

Paquetes recibidos = 5963
Errores de encabezado recibidos = 0
Errores de dirección recibidos = 6
Datagramas reenviados = 0
Protocolos desconocidos recibidos = 0
Paquetes recibidos descartados = 2395
Paquetes recibidos procesados = 3568
Solicitudes de salida = 3771
Descartes de ruta = 0
Paquetes de salida descartados = 0
Paquetes de salida sin ruta = 0
Reensambles requeridos = 0
Reensambles correctos = 0
Reensambles erróneos = 0
Datagramas correctamente fragmentados = 0
Datagramas mal fragmentados = 0
Fragmentos creados = 0

Estadísticas ICMPv4

	Recibidos	*Enviados*
Mensajes	*8*	*3*
Errores	*0*	*0*
Destino inaccesible	*7*	*2*
Tiempo agotado	*0*	*0*
Problemas de parámetros	*0*	*0*
Paquetes de control de flujo	*0*	*0*
Redirecciones	*0*	*0*
Echos	*1*	*0*
Respuestas de eco	*1*	*0*
Fechas	*0*	*0*
Respuestas de fecha	*0*	*0*
Máscaras de direcciones		
Máscaras de direcciones respondidas	*0*	*0*

Estadísticas de TCP para IPv4

Activos abiertos = 221
Pasivos abiertos = 0
Intentos de conexión erróneos = 8
Conexiones restablecidas = 78
Conexiones actuales = 5
Segmentos recibidos = 2288
Segmentos enviados = 2404
Segmentos retransmitidos = 10

Estadísticas UDP para IPv4

Datagramas recibidos = 1270

Sin puerto = 2
Errores de recepción = 16
Datagramas enviados = 1353

Con la opción **–**an muestra las direcciones y el estado de todas las conexiones asi como sus puertos.

C:\ netstat -an

Conexiones activas

Proto	*Dirección local*	*Dirección remota*	*Estado*
TCP	*0.0.0.0:135*	*0.0.0.0:0*	*LISTENING*
TCP	*0.0.0.0:445*	*0.0.0.0:0*	*LISTENING*
TCP	*0.0.0.0:1026*	*0.0.0.0:0*	*LISTENING*
TCP	*0.0.0.0:3012*	*0.0.0.0:0*	*LISTENING*
TCP	*0.0.0.0:3023*	*0.0.0.0:0*	*LISTENING*
TCP	*0.0.0.0:3024*	*0.0.0.0:0*	*LISTENING*
TCP	*0.0.0.0:3032*	*0.0.0.0:0*	*LISTENING*
TCP	*0.0.0.0:3040*	*0.0.0.0:0*	*LISTENING*
TCP	*0.0.0.0:3048*	*0.0.0.0:0*	*LISTENING*
TCP	*0.0.0.0:3056*	*0.0.0.0:0*	*LISTENING*
TCP	*0.0.0.0:3057*	*0.0.0.0:0*	*LISTENING*
TCP	*0.0.0.0:3067*	*0.0.0.0:0*	*LISTENING*
TCP	*0.0.0.0:3201*	*0.0.0.0:0*	*LISTENING*
TCP	*0.0.0.0:3220*	*0.0.0.0:0*	*LISTENING*
TCP	*0.0.0.0:3227*	*0.0.0.0:0*	*LISTENING*
TCP	*0.0.0.0:3228*	*0.0.0.0:0*	*LISTENING*
TCP	*0.0.0.0:3229*	*0.0.0.0:0*	*LISTENING*
TCP	*0.0.0.0:3231*	*0.0.0.0:0*	*LISTENING*
TCP	*0.0.0.0:3236*	*0.0.0.0:0*	*LISTENING*
TCP	*0.0.0.0:3237*	*0.0.0.0:0*	*LISTENING*
TCP	*0.0.0.0:5000*	*0.0.0.0:0*	*LISTENING*
TCP	*127.0.0.1:3001*	*0.0.0.0:0*	*LISTENING*
TCP	*127.0.0.1:3002*	*0.0.0.0:0*	*LISTENING*
TCP	*127.0.0.1:3003*	*0.0.0.0:0*	*LISTENING*
TCP	*213.102.64.17:3012*	*207.46.4.54:1863*	*ESTABLISHED*
TCP	*213.102.64.17:3032*	*194.224.100.71:80*	*CLOSE_WAIT*
TCP	*213.102.64.17:3048*	*217.75.98.16:80*	*ESTABLISHED*
TCP	*213.102.64.17:3187*	*216.12.215.207:80*	*TIME_WAIT*
TCP	*213.102.64.17:3201*	*217.75.98.7:80*	*ESTABLISHED*
TCP	*213.102.64.17:3220*	*130.94.72.189:80*	*ESTABLISHED*
TCP	*213.102.64.17:3227*	*217.75.98.7:80*	*ESTABLISHED*
TCP	*213.102.64.17:3228*	*62.37.226.86:80*	*ESTABLISHED*
TCP	*213.102.64.17:3229*	*62.37.226.86:80*	*ESTABLISHED*
TCP	*213.102.64.17:3231*	*161.58.186.225:80*	*ESTABLISHED*
TCP	*213.102.64.17:3236*	*62.37.236.78:80*	*ESTABLISHED*

```
TCP    213.102.64.17:3237     62.37.236.78:80       ESTABLISHED
UDP    0.0.0.0:135            *:*
UDP    0.0.0.0:445            *:*
UDP    0.0.0.0:500            *:*
UDP    0.0.0.0:1025           *:*
UDP    0.0.0.0:3010           *:*
UDP    0.0.0.0:3022           *:*
UDP    0.0.0.0:3031           *:*
UDP    0.0.0.0:3064           *:*
UDP    0.0.0.0:3065           *:*
UDP    127.0.0.1:123          *:*
UDP    127.0.0.1:1900         *:* UDP
127.0.0.1:2234         *:*
```

Capitulo 2

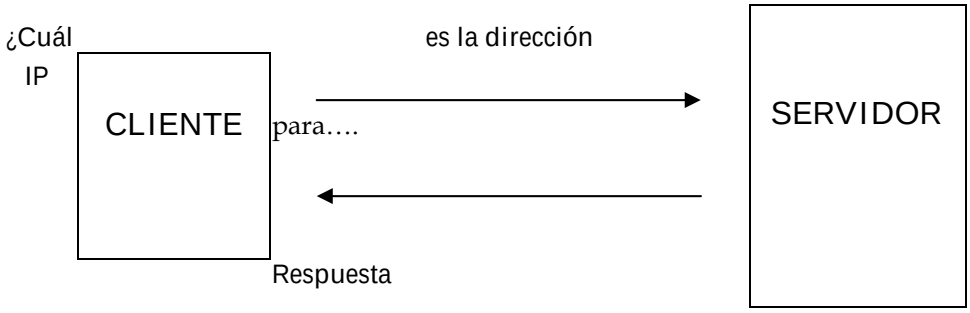
Protocolo de datagrama de usuario (UDP).

2. INTRODUCCIÓN

UDP es un protocolo de transporte de nivel 4 (OSI) no orientado a conexión. UDP es un protocolo de datagrama que no garantiza la entrega de los datos. Si una aplicación envía una petición en un datagrama de UDP y no llega una respuesta en un tiempo razonable, es responsabilidad de la aplicación el retransmitir la petición. UDP simplemente envía el datagrama y no se preocupa de si llega o no. Al no garantizar la entrega, hace que

sea un protocolo mucho más rápido y ligero que TCP ofreciendo un transporte alternativo a aquellos procesos que no requieren una entrega fiable.

Muchas aplicaciones usan UDP para enviarse mensajes entre si o para realizar consultas rápidas a bases de datos o servidores DNS (Domain Name System). UDP es una pieza perfecta para construir funciones de monitor, depuración, gestión y prueba.



Comunicación con UDP.

Aunque UDP es menos fiable que TCP en algunas ocasiones es más recomendable que TCP ya que ofrece ventajas en determinadas situaciones:

- Los mensajes entre host son esporádicos. SNMP (Simple Network Management Protocol) vuelve a ser un buen ejemplo. Sus mensajes se envían a intervalos irregulares. La carga de trabajo necesaria para abrir y cerrar la conexión TCP de cada mensaje retrasaría su transmisión y penalizaría el rendimiento.
- Mensajes que no requieren acuse de recibo. UDP contribuye a reducir el tráfico de la red. Los avisos de SNMP pertenecen a esta categoría. En una red extensa, se genera una gran cantidad de avisos SNMP cuando los dispositivos SNMP transmiten sus actualizaciones de estado. Sin embargo, la pérdida de un mensaje SNMP no suele ser crítica y la red se libera de una importante carga de trabajo al utilizar UDP para SNMP.
- La fiabilidad se implementa al nivel del proceso. El sistema de archivos de red (NFS) es un buen ejemplo de proceso que implementa su propia función de fiabilidad y se ejecuta sobre UDP para mejorar el rendimiento de la red.
- Lo pueden usar aplicaciones que necesitan enviar mensajes de difusión o multidifusión, por ejemplo un cliente de BOOTP.

2.1. PUERTOS DE APLICACIÓN

El interfaz que esta entre UDP y el proceso local, se llama puerta. Para que una aplicación pueda acceder a la red y pueda enviar datos a través de ella lo debe hacer a través de un puerto.

1. Un usuario invoca un programa cliente, como por ejemplo *nslookup*
2. El proceso cliente ejecuta una rutina del sistema que dice: *{Quiero una comunicación de UDP. Dame un puerto.}*
3. La rutina del sistema le asigna un identificador de 16 bits llamado *{numero de puerto}* que este libre.

Los puertos se identifican mediante un número decimal que va desde el 0 hasta el 65.535. Los fabricantes que implementan UDP disponen de una gran libertad para asignar números de puertos a los procesos, aunque la Autoridad de Números Asignados de Internet (IANA) ha reservado un rango de puertos que va desde el 0 al 1.023 para servicios estándar, (Well-Known) como DNS, SNMP, o Netbios.

Puerto	Aplicación	Descripción
7	Eco	Eco del datagrama de usuario de vuelta al emisor.
9	Discard	Descartar el datagrama de usuario.
13	DayTime	Indica la hora de forma sencilla para el usuario.
17	Quote	Devuelve una {cita del día}.
19	Chargen	Intercambiar flujos de caracteres
53	DNS	Servidor de nombres de dominio.
67	Bootps	Puerto del servidor para descargar la información de configuración.
68	Bootpc	Puerto del cliente para recibir recibir información de configuración.
69	TFTP	Puerto del Protocolo trivial de transferencia de archivos.
161	SNMP	Usado para recibir las peticiones de gestión de red.
162	SNMP-trap	Usado para recibir los informes de problemas de red.

Listado de algunos puertos UDP públicos.

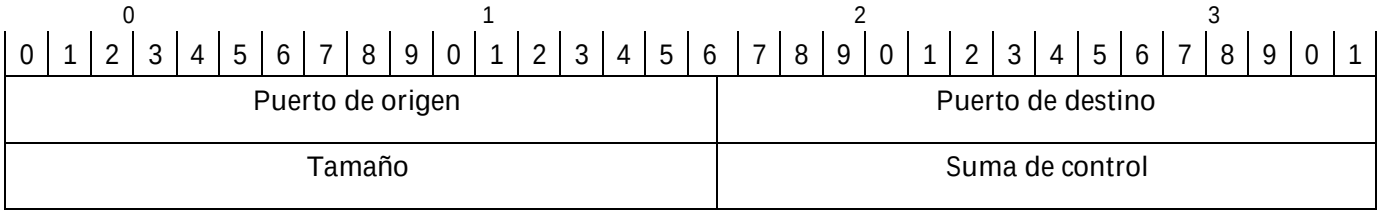
Algunos de estos servicios proporcionan bloques para probar, depurar y medir, como el servicio {eco} en el puerto 7 donde devuelve cualquier datagrama que se le envía. Otro puerto como el 9 (Discard) tira los datagramas que recibe. El puerto 19 es un generador de caracteres donde responde a cualquier mensaje con un datagrama que

contiene entre 0 y 512 bytes. El número se elige aleatoriamente. El servicio cita del día que escucha por el puerto 17, responde a cualquier datagrama enviando de vuelta un mensaje con alguna frase histórica o de sabiduría popular al cerrar la sesión o ejecutar el comando fortune. El puerto 13 (DayTime) responde a cualquier datagrama con un mensaje que contiene la fecha y hora actual en formato ASCII legible. El servidor y el cliente de BOOTP se usan para inicializar dispositivos sin configuración (terminales tontos). Una estación de trabajo puede descubrir su dirección de IP, su Mascara de red, la situación de su encaminador por defecto, la dirección de servidores importantes y hasta el nombre y ubicación de un archivo de descarga de software desde el servidor de arranque. El software de la estación de trabajo se descarga mediante el protocolo trivial de transferencia de archivos (TFTP; UDP 69).

Los puertos TCP y UDP son independientes unos de otros. Un proceso puede enviar mensajes por el puerto 1700 de UDP a la vez que otro está manteniendo una sesión en el puerto 1700 de TCP. Algunos servicios necesitan acceder tanto a TCP como a UDP por eso el IANA intenta asignar el mismo número de puerto a los servicios que necesitan UDP como TCP, como por ejemplo DNS, que escucha por el puerto 53 de TCP y UDP.

A la combinación de una dirección de IP y de puerto para una comunicación se le llama {*dirección de conector*} o {*socket*}. Un socket ofrece toda la información que necesita un cliente o un servidor para identificar a su otro extremo.

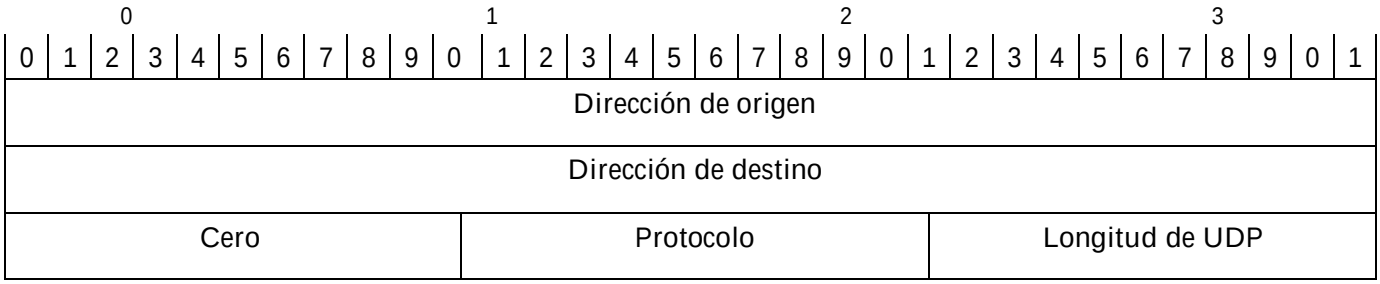
2.2. CABECERA UDP



Cabecera de UDP.

Una cabecera UDP se compone de dos palabras de 32 bits y tiene los siguientes campos:

- Puerto de origen (16 bits). Este campo es opcional y especifica el puerto de origen cuando es necesario permitir al receptor del datagrama enviar una respuesta.
- Puerto de destino (16 bits). El puerto del host IP de destino.
- Longitud (16 bits). La longitud del datagrama en octetos incluyendo la cabecera y los datos. El valor mínimo es 8 para permitir el uso de una cabecera. Por consiguiente la longitud máxima de un datagrama UDP es de 65.535 octetos de los que 65.527 pueden dedicarse a datos.
- Suma de control. El objetivo de la suma de control de UDP es validar el {contenido} de un mensaje de UDP. La suma de control de UDP se calcula sobre la combinación de una pseudocabecera construida especialmente con cierta información de IP., la cabecera de UDP y los datos del mensaje. El uso de la suma de control de UDP en la comunicación en particular es opcional. Si no se usa, el campo vale 0. Si se ha calculado una suma de control y su valor llega a valer 0, se representa con un campo de unos.



Formato de la pseudocabecera de UDP.

Cuando una aplicación adquiere un puerto de UDP, el software de red reservara algunos búfer para contener una cola de los datagramas de usuario que llegan a ese puerto. Un servicio de UDP no tiene ninguna forma de predecir o controlar cuántos datagramas se enviaran en un momento dado.

Si se bombardea el servicio con mas datagramas de los que puede manejar, ese exceso simplemente se descarta.

Ejemplo:

>netstat -a

…….

0 incomplete headers
0 bad data length fields
0 bad checksums
17 sockets overflows

2.4. DIFERENCIAS ENTRE TCP Y UDP

La clave de la capa de transporte es que permite a los desarrolladores de una aplicación optar por la fiabilidad (TCP) o por la eficacia (UDP).

El gran interes de TCP radica en su capacida para aislar los procesos de la capa superior con respecto a la red. Los procesos no necesitan conocer el tamaño de los mensajes, ya que TCP se encarga de su fragmentación y reensamblaje. Además, los procesos no se preocupan por la fiabilidad de la red, ya que TCP garantiza por completo la integridad de los mensajes que transporta.

Por parte de UDP, es un protocolo más despreocupado en la realización de su trabajo. A diferencia de TCP, UDP no necesita establecer circuitos virtuales entre los host, esto permite una comunicación mas agil y eficiente, por eso UDP admite mensajes de difusión y multidifusión.

TCP	UDP
Fiable	No fiable.
Orientado a conexión	No orientado a conexión.
Circuito virtual	Datagramas.
Elevado trafico de control	Trafico de control reducido.
Segmentación de mensajes	Sin segmentación ni reensamblaje de mensajes.
Segmentos de mensajes secuenciales	No secuenciales.

Tabla comparativa entre TCP y UDP.