Escuela Politécnica Superior
Universidad de Huelva

TERCER CURSO. TECNOLOGÍA DE REDES

Tema 2: Protocolo TCP. Nivel de Transporte

Manuel Sánchez Raya
Versión 0.1
5 de Febrero de 2004

ÍNDICE

BIBLIOGRAFÍA

Apuntes año 2002-2003. Estefanía Cortés Ancos.
Apuntes Universidad de Oviedo. J.A.Sirgo, Rafael C. González
Academia de Networking de Cisco Systems. Guía del Primer Año.
Internetworking with TCP/IP. Vol. I. D.E. Comer.

1.- Introducción a TCP (Transmission Control Protocol) y UDP (User Data Protocol).

La capa de transporte define la conectividad de extremo a extremo entre dos aplicaciones host. Los servicios de la capa de transporte permiten el *transporte de datos fiable* entre los extremos finales. Un trasporte fiable pretende conseguir lo siguiente:

- Asegurar que los segmentos entregados sean confirmados por el receptor
- Proporcionar retransmisión para cualquier segmento no confirmado
- Devolver los segmentos a su secuencia correcta en el destino
- Proporcionar control e impedir la congestión

Para ello, se utiliza una relación orientada a la conexión entre los sistemas finales de la comunicación. La unidad de transporte se conoce como **segmento.** Sus principales funciones son:

- Segmentación de los datos de aplicación de la capa superior
- Ofrece una conexión lógica entre dos extremos de la red
- Control de flujo extremo a extremo, proporcionado por las ventanas deslizantes
- Fiabilidad, gracias a los acuses de recibo y los números de secuencia.

El **control de flujo** evita el problema que se produce cuando un host en un extremo de la conexión de red inunda los buffers del host del otro extremo. Esto podría provocar la pérdida de datos. Se consigue haciendo que el destinatario de la información confirme la recepción de cada uno de los segmentos de datos. Sin embargo, si el emisor tiene que esperar un acuse de recibo antes de volver a enviar otro segmento, la transmisión se vuelve ineficiente. Por tanto, la mayoría de los protocolos fiables y orientados a conexión permiten el envío de más de un segmento a la vez.

El número de paquetes que se permite emitir sin que haya sido recibido un acuse de recibo se conoce como ***ventana.*** Para obtener un servicio de transporte de datos **fiable**, se utiliza una relación orientada a la conexión entre los sistemas finales de comunicación. De modo que cada vez que se reciba un paquete envíe al receptor un ***acuse de recibo***.

El emisor guarda un registro de cada paquete de datos que envía, iniciando un temporizador por cada uno de ellos y espera un acuse de recibo. Si transcurrido el tiempo no lo recibe, retransmite el paquete. Internet cuenta con dos protocolos principales en la capa de transporte:

- **TCP (Transmission Control Protocol)**: orientado a conexión.
- **UDP (User Data Protocol)**: no orientado a conexión.

1.1.- Puertos y sockets

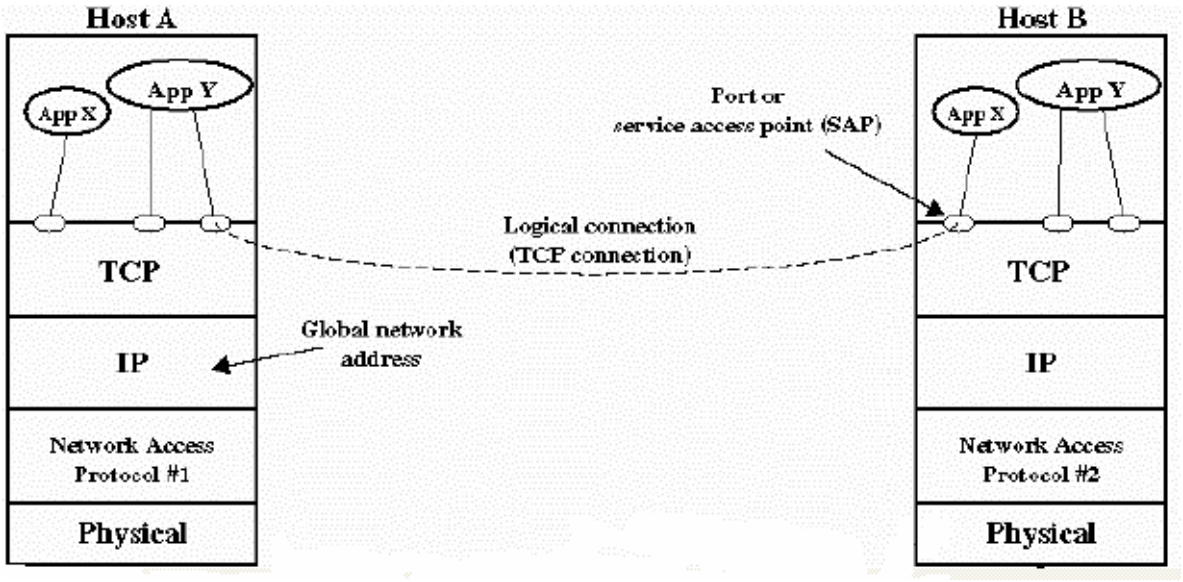
El servicio que ofrecen TCP y UDP se obtiene haciendo que, tanto el emisor como el receptor, creen puntos terminales llamados **sockets**. Los sockets son puntos de acceso a las comunicaciones. Definen un servicio de sesión-transporte. El usuario de la capa de transporte debe establecer una sesión orientada a la conexión entre un sistema de iguales.

Para que comience la transmisión de datos, las aplicaciones del emisor y el receptor informan a sus respectivos sistemas operativos de que se va a iniciar una conexión. Un extremo debe iniciar la conexión y el otro aceptarla. El software de protocolo modula la comunicación entre los dos extremos enviando mensajes a través de la red, para verificar que se ha autorizado la transferencia y que ambos lados están preparados. Así, debe establecerse una conexión entre un socket de la máquina transmisora y un socket de la receptora. La información será enviada a un determinado puerto (parte del software de aplicación) que se encontrará a la escucha en el extremo receptor. Se manejan como descriptores de ficheros y el S.O. los asigna como puntos terminales de comunicaciones bidireccionales.

Cada socket tiene un número que consiste en la dirección IP del host y un número local a dicho host , llamado **puerto**. Cuando se recibe un datagrama se analiza si el valor del campo protocolo de la cabecera es TCP o UDP. Luego, se mira el número de puerto y se entrega el contenido a la aplicación que esté conectada a ese puerto. Los puertos multiplexan conexiones a nivel de transporte, permitiendo a varias aplicaciones acceder simultáneamente a las comunicaciones.

Ejemplo:

Puerto 21: FTP
Puerto 23: TELNET



1.2.- Asignación de puertos.

Los puertos se identifican por números de 16 bits. Los de TCP y los de UDP se manejan por separado. Son asignados con el siguiente criterio:

- **Puertos bien conocidos:**

Los números de puerto 1 a 255 están reservados, y tienen una asignación universal según la RFC 1700.

20 y 21	FTP command y FTP transfer
22	ssh
23	telnet
25	SMTP (email)
53	DNS
80	WWW
110	POP3
119	NNTP (news)
6000	X (despliegue gráfico)

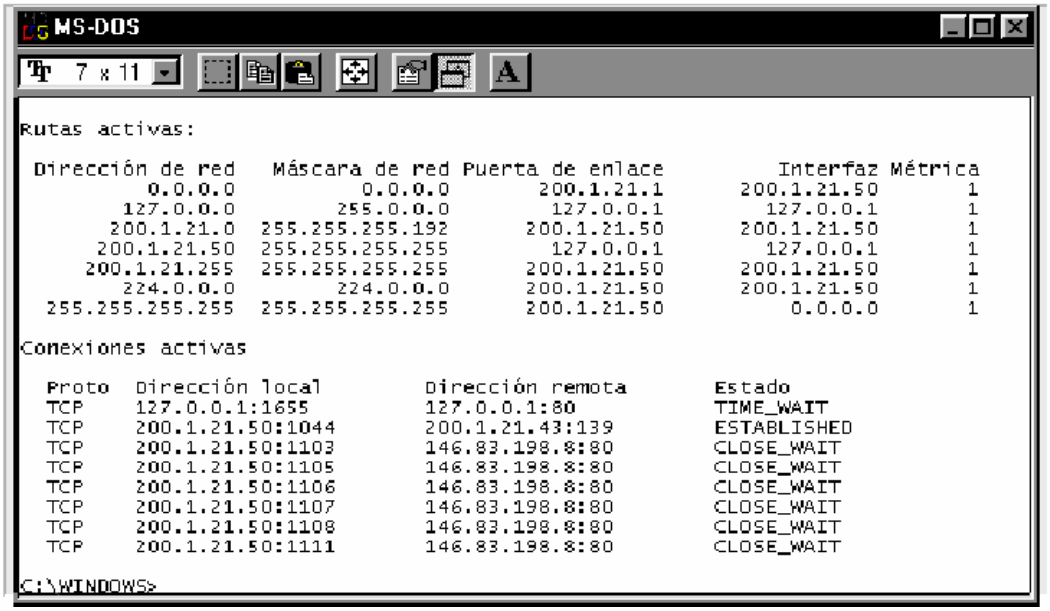
- **Asignación dinámica:**

El resto de los números los gestiona el software de red (integrado en el S.O.), y los asigna dinámicamente según son solicitados por las aplicaciones.

1.3.- Enlace entre puertos y aplicaciones

Una **apertura pasiva** es la que se produce cuando una aplicación informa al sistema operativo de que está dispuesta a aceptar una comunicación entrante. Se produce una asignación de puerto para ella, y queda en espera. Una **apertura activa** genera una serie de mensajes por la red, a fin de conseguir una respuesta del otro extremo mediante un protocolo de conexión.

netstat -rn



2.- El protocolo UDP (User Data Protocol).

Cuando un datagrama llega a su destino se ha de determinar a cual de las aplicaciones existentes en la máquina se ha de hacer llegar la información contenida en el. Esto no lo hace el protocolo IP directamente, sino que es misión del Protocolo de Transporte. El protocolo UDP proporciona la funcionalidad necesaria para identificar al destinatario final de un datagrama de una manera simple. Como no proporciona ningún mecanismo para el acuse de recibo, secuenciación de mensajes ni control de flujo, proporciona el mismo servicio no fiable que el protocolo IP. Los mensajes UDP pueden perderse o llegar fuera de secuencia, y además, los paquetes pueden llegar más rápido de lo que es receptor es capaz de procesar.

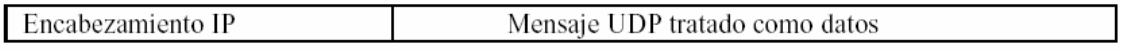
El protocolo UDP permite a varios programas de aplicación en una misma máquina comunicarse simultáneamente y demultiplexa el tráfico que se recibe entre las distintas aplicaciones. El protocolo UDP incorpora los denominados puertos, que identifican el último destino dentro de una máquina, el programa de aplicación que está haciendo uso de ese puerto. Cada puerto tiene asociado un entero para identificarlo. Dado que el número de puerto es único dentro de una máquina, el destino último queda perfectamente determinado por la dirección internet del "host" y el número de puerto UDP en ese "host".

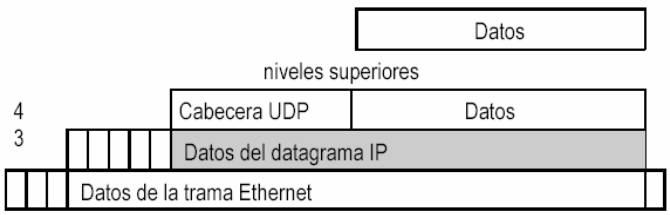
Características:

- No orientado a conexión: los mensajes se pueden perder, duplicarse o llegar desordenados.
- No es fiable. El servicio cede esta responsabilidad a la capa de aplicación.
- Utiliza mensajes de transmisión (datagramas de usuario)
- No proporciona software de verificación de entrega de los mensajes.
- No reensambla los mensajes entrantes
- No utiliza acuse de recibo
- No proporciona control de flujo

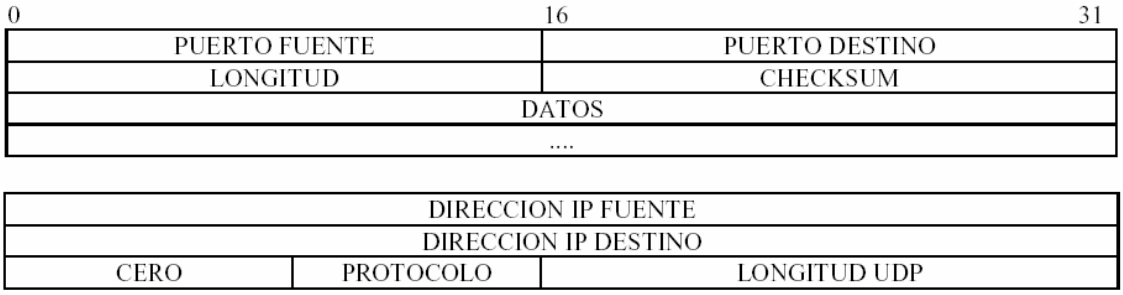
2.1.- Formato del mensaje UDP.

Los mensajes UDP se denominan *datagramas de usuario* y viajan en la porción de datos de los datagramas IP, como se muestra en la figura:





El protocolo UDP entrega al servicio IP el segmento que contiene los datos, que es el que se transmite realmente, y una *pseudo-cabecera* que no se trasmite, pero que permite al protocolo IP completar los datos del o los datagramas que va a generar. El formato del segmento y de la pseudo-cabecera son los que se muestran en la figura siguiente.



A continuación se describe el significado de los distintos campos del segmento.

- PUERTOS FUENTE Y DESTINO: Estos dos campos contienen los números de los puertos UDP que identifican los programas de aplicación en las máquinas fuente y destino

- LONGITUD: Es el número total de octetos que forman el datagrama UDP incluida la cabecera.

- CHECKSUM: Para calcular el "checksum", la máquina fuente antepone la pseudo-cabecera al datagrama y añade al final de los datos bytes conteniendo ceros hasta conseguir una longitud del segmento múltiplo de 16 bits. El "checksum" se calcula una vez hechos los cambios según el siguiente algoritmo: se considera el segmento formado por enteros de 16 bits y se suman todos los complementos a uno de esos enteros utilizando la aritmética de complemento a uno. A efectos de hacer los cálculos se supone que ese campo tiene valor cero. Los ceros añadidos para rellenar, así como la pseudo-cabecera no se cuentan en la longitud del datagrama y no son transmitidos.

  El “checksum” es opcional. Si no se utiliza, su contenido es cero. La razón de usar la pseudo-encabecera es permitir a la máquina destino comprobar que el datagrama ha alcanzado su destino correcto, ya que incluye la dirección internet del "host" destino, así como el número de puerto UDP de la conexión. La máquina destino puede conseguir la información usada en la pseudo-cabecera a partir del datagrama IP que transporta al datagrama UDP.

- DATOS: Representa los datos del datagrama UDP.

El significado de los campos de la pseudo-cabecera es el siguiente:

- DIRECCION IP FUENTE: Es la dirección internet del "host" que envía el segmento.
- DIRECCION IP DESTINO: Es la dirección internet del "host" al que va dirigido el segmento.
- CERO: Campo que contiene el valor cero.
- PROTOCOLO: Especifica protocolo UDP (es decir, 17).

- LONGITUD UDP: Este campo especifica la longitud total del datagrama UDP.

Alguno de los protocolos que utilizan UDP son:

- Protocolo TFTP
- Protocolo simple de administración de redes (SNMP)
- Protocolo de configuración dinámica del host (DHCP)
- Sistema de denominación de dominio (DNS)
- Protocolo BOOTP

2.2.- Números reservados para puertos UDP.

El protocolo UDP utiliza combina una adjudicación de números de puertos UDP dinámica y estática, usando una serie de asignaciones de puertos conocida para un conjunto de programas que se utilizan comúnmente (por ejemplo, correo electrónico). Sin embargo, la mayoría de los números de puerto están disponibles para que el sistema operativo los utilice a medida que los programas de aplicación los necesiten.

3. El protocolo TCP.

Al más bajo nivel, la comunicación de computadores proporciona un servicio de distribución no fiable de paquetes. Cuando la red física falla, los paquetes pueden perderse, llegar con errores o duplicados.

Al nivel más alto, los programas de aplicación necesitan, frecuentemente, enviar grandes volúmenes de datos. Utilizar el sistema de distribución no fiable anteriormente mencionado requiere que los programadores construyan algoritmos de detección y recuperación de errores en los programas de aplicación. Estos algoritmos son muy complejos, y por tanto pocos programadores tienen los conocimientos necesarios para diseñarlos. Como consecuencia, se desarrollo el protocolo de transporte TCP, que permite la transmisión fiable de datos mediante el establecimiento y liberación de conexiones, de manera que los datos que llegan a los programas de aplicación lo hagan sin errores, pudiendo ser utilizados directamente, sin necesidad de escribir algoritmos de detección y recuperación de errores.

Para conseguir esto, la mayoría de los protocolos utilizan una técnica conocida como "acuse de recibo positivo con retransmisión". Esta técnica requiere que la máquina destino envíe a la fuente un mensaje de acuse de recibo cada vez que recibe datos. La fuente mantiene un registro de paquetes enviados, y espera por un acuse de recibo antes de enviar el siguiente paquete. Asimismo, la fuente inicia un temporizador cada vez que envía un paquete y retransmite el paquete si el temporizador expira antes de recibir el acuse de recibo.

El problema final surge cuando la red física duplica paquetes (por ejemplo, si la red tiene un gran retraso pueden ocurrir retransmisiones prematuras). Para solucionar esto, los mensajes de acuse de recibo tienen números de secuencia, con lo que la fuente puede asociar correctamente acuses de recibo con paquetes.

La unidad de transmisión es el **segmento**. Los segmentos pueden ser de longitud variable. El protocolo TCP presenta las siguientes características:

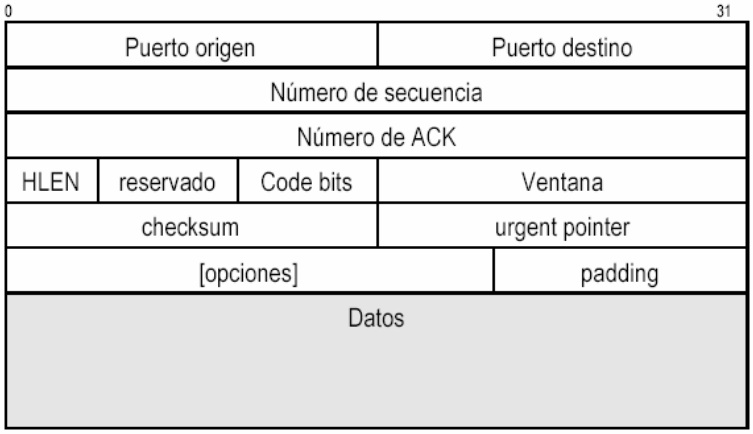
- Orientado a stream (chorro de octetos). TCP ve los datos como un flujo continuo de octetos, no como paquetes independientes. Por tanto, mantiene la secuencia de octetos transmitidos y recibidos.
- Orientado a conexión:
- Es fiable.
- La transferencia se apoya en buffers (TX y RX): ventanas
- Transmisión Full Duplex.
- Piggybacking: Los asentimientos de recepción van en las tramas de envío de datos en el sentido opuesto, para reducir el tráfico.
- Reensambla los mensajes en el destino a partir de los segmentos entrantes
- Vuelve a enviar todo lo que no se ha recibido

3.1.- Formato del segmento TCP.

Una vez explicados los mecanismos de trabajo del TCP puede verse el formato del segmento TCP, que es la unidad de transferencia de datos con este protocolo. Los segmentos se intercambian para establecer y liberar conexiones, transferir datos, enviar acuses de recibo e informar sobre tamaños de ventana. Un acuse de recibo que vaya de una máquina A a otra B puede viajar en el mismo segmento que lleve datos de la máquina A a la máquina B (*piggybacking*).

Los segmentos TCP viajan en la porción de datos de los datagramas IP, como se muestra en la figura:

El protocolo TCP entrega al servicio IP el segmento que contiene los datos, que es el que se transmite realmente, y una *pseudo-cabecera* que no se trasmite, pero que permite al protocolo IP completar los datos del o los datagramas que va a generar. El formato del segmento y de la pseudo-cabecera son los que se muestran en la figura siguiente.



- **Puerto origen**: nº del puerto que llama. Identifica el proceso de capa superior en el extremo del origen de ese segmento. Puede ser o no el extremo que inició la conexión
- **Puerto destino**: nº del puerto al que se llama, asignado al final del destino para ese segmento.
- **Número de secuencia** es un entero que indica cual es la posición del primer octeto del segmento dentro del stream. Se utiliza para asegurar la secuencia correcta en la llegada de los datos.
- **Número de ACK**: número de secuencia del siguiente octeto que se espera recibir (asiente la llegada de todos los anteriores).
- **HLEN**: Este campo de 4 bits contiene un entero que especifica el desplazamiento de la porción de datos del segmento. Es decir, indica la longitud de la cabecera en unidades de 32 bits. Se necesita porque la longitud del campo OPCIONES varía en longitud según las opciones elegidas.
- **RES**: Los 6 octetos reservados para uso futuro deben estar a 0.

- **Code Bits**: Es un campo de 6 bits que determina el propósito y contenido del segmento. Los seis bits explican cómo hay que interpretar otros campos en el encabezamiento de acuerdo con la tabla que se muestra a continuación.

Bit (de izquierda a derecha)	Significado
URG	El campo puntero urgente es válido
ACK	El campo acuse de recibo es válido
PSH	El segmento requiere un "push"
RST	"Resetear" la conexión
SYN	Sincronizar números de secuencia
FIN	La fuente ha alcanzado el fin de su secuencia de bytes

- **Ventana** : Este campo contiene el "informe de ventana" explicado, es decir, el número de bytes que la máquina está dispuesta a aceptar. Este campo se incluye tanto en los segmentos que llevan datos como en los que sólo llevan un acuse de recibo.

- El **checksum** Este campo contiene un entero de 16 bits usado para verificar la integridad del encabezamiento y los datos del segmento. Para calcular el "checksum", la máquina fuente antepone la pseudo-cabecera al segmento, y añade al final de los datos bytes conteniendo ceros hasta conseguir una longitud del segmento múltiplo de 16 bits. El "checksum" se calcula una vez hechos los cambios según el siguiente algoritmo: se considera el segmento formado por enteros de 16 bits y se suman todos los complementos a uno de esos enteros utilizando la aritmética de complemento a uno. A efectos de hacer los cálculos se supone que ese campo tiene valor cero. Los ceros añadidos para rellenar, así como la pseudo-cabecera no se cuentan en la longitud del segmento y no son transmitidos.

  La razón de usar la pseudo-encabecera es permitir a la máquina destino comprobar que el segmento ha alcanzado su destino correcto, ya que incluye la dirección internet del "host" destino, así como el número de puerto TCP de la conexión. La máquina destino puede conseguir la información usada en la pseudo-cabecera a partir del datagrama IP que lleva el segmento.

- **Urgent pointer**: Cuando el bit URG está a 1, el campo PUNTERO URGENTE especifica la posición en la secuencia de bytes en la que los datos urgentes acaban. Los datos urgentes deben ser distribuidos lo más rápidamente posible. El protocolo especifica que cuando se encuentran datos urgentes, el software TCP que los recibe debe informar al programa de aplicación asociado a la conexión para que se ponga en modo "urgente". Los detalles de cómo el software TCP debe informar al programa de aplicación dependen del sistema operativo que se esté usando. Los datos urgentes contienen mensajes en vez de datos normales; por ejemplo, señales de interrupción del teclado.

- **Opciones** : El software TCP usa este campo para comunicarse con el software TCP al otro lado de la conexión. En particular, una máquina puede comunicar a otra el máximo tamaño de segmento que está dispuesta a aceptar. Esto es muy importante cuando un computador pequeño va a recibir datos de uno grande. Escoger un tamaño adecuado de segmento es difícil, puesto que el rendimiento de la transmisión empeora para segmentos excesivamente grandes (pues hay que

fragmentarlos en varios datagramas) o excesivamente pequeños (puesto que el tanto por ciento de encabezamientos es muy grande).

- El **PADDING** es relleno para expresar HLEN en palabras de 32bits

El significado de los campos de la pseudo-cabecera es el siguiente:

- DIRECCION IP FUENTE: Es la dirección internet del "host" que envía el segmento.
- DIRECCION IP DESTINO: Es la dirección internet del "host" al que va dirigido el segmento.
- CERO: Campo que contiene el valor cero.
- PROTOCOLO: Especifica protocolo TCP (es decir, 6).
- LONGITUD TCP: Este campo especifica la longitud total del segmento TCP.

Tipos de Segmentos:

- De establecimiento de conexión
- De datos
- ACKs
- Tamaños de ventana
- Cierre de conexión

3.2.- La ventana deslizante del protocolo TCP.

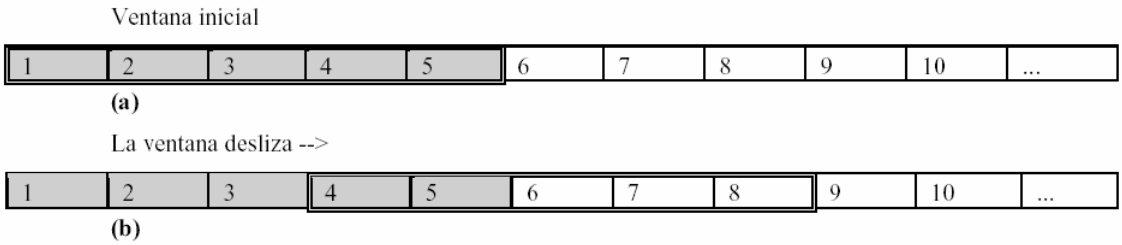
Un protocolo de acuse de recibo simple desperdicia una gran cantidad de ancho de banda de la red puesto que debe retrasar el envío de un nuevo paquete hasta recibir un acuse de recibo para el paquete previo.

La técnica de la *ventana deslizante* utiliza una forma más compleja del algoritmo de acuse de recibo y retransmisión que la anteriormente descrita. La forma de ver este algoritmo es pensar en una secuencia de paquetes que deben ser transmitidos, situar una ventana sobre la secuencia y transmitir los paquetes dentro de la ventana. Cuando se recibe un acuse de recibo para el primer paquete de la ventana, ésta "desliza" una posición y se puede enviar el siguiente paquete, pues ya está dentro de la ventana.

El protocolo TCP utiliza un mecanismo de ventana deslizante para resolver dos problemas importantes: *transmisión eficiente* y *control de flujo extremo a extremo*, permitiendo a la máquina destino ordenar a la fuente restringir el envío de datos hasta que tenga suficiente espacio para tratar más datos. Sin embargo, el protocolo TCP ve a la corriente de datos como una secuencia de octetos o bytes que divide en *segmentos* para la transmisión. Generalmente, cada segmento viaja a través de la red Internet en un solo datagrama IP.

Así, el mecanismo de ventana deslizante TCP opera a nivel de byte, no al de paquete. Los bytes en la secuencia de datos se numeran secuencialmente y el "host" fuente

mantiene tres punteros asociados a cada conexión que define una ventana deslizante. El primer puntero apunta al comienzo de la ventana, separando los bytes que han sido enviados y confirmados de los bytes que se han de enviar. El segundo apunta al final de la ventana, definiendo el byte más alto en la secuencia que puede ser enviado sin que llegue un acuse de recibo. El tercer puntero, dentro de la ventana, separa los bytes que han sido enviados de los bytes que no lo han sido. En la siguiente figura se muestra un ejemplo.



En la figura (a) se muestra la técnica de la ventana deslizante con 5 bytes en la ventana ya enviados y no confirmados. En la figura (b), los bytes hasta el 3 han sido enviados y confirmados; los bytes 4 y 5 han sido enviados pero no confirmados; los bytes del 6 al 8 no han sido enviados, pero lo serán sin esperar a recibir acuse de recibo alguno, y los bytes 9 y más altos no se enviarán hasta que la ventana no se mueva.

Los bytes no confirmados son los que han sido enviados pero no han tenido acuse de recibo. El número de bytes no confirmados en un momento dado está limitado al tamaño de la ventana.

Se ha dicho que la fuente mantiene una ventana para cada conexión y que el destino mantiene otra ventana similar. Sin embargo, como la comunicación TCP es "full-duplex", dos transferencias de datos ocurren simultáneamente en cada conexión, una en cada sentido. Las transmisiones son independientes, puesto que en cualquier momento los datos pueden ir en cualquiera de las dos direcciones, o en ambas. Por tanto, el software TCP mantiene dos ventanas para cada conexión en cada máquina; una desliza sobre la secuencia de bytes a enviar y la otra sobre los bytes que se van recibiendo.

3.3.- Control de flujo.

El método de ventana deslizante del protocolo TCP permite que el tamaño de la ventana varíe. Cada acuse de recibo, que especifica cuantos bytes han sido recibidos, contiene un "informe de ventana" que especifica cuantos bytes adicionales de datos está dispuesto a aceptar el "host" destino. Si este informe indica un tamaño de la ventana mayor que el actual tamaño de la ventana de la máquina fuente, ésta lo incrementa y comienza a mandar más bytes. Si el informe de ventana especifica un tamaño menor, la fuente disminuye el tamaño de su ventana y no enviará datos más allá del límite de la misma. El software TCP no contradice informes de ventana previos y nunca reduce la ventana a posiciones anteriores a las aceptadas anteriormente.

La ventaja de tener un tamaño de ventana variable es que proporciona control de flujo así como transferencia fiable. Si la capacidad de aceptar datos de la máquina destino disminuye, envía un informe de ventana menor. En el caso extremo, el destino puede

informar con un tamaño de ventana cero para detener la transmisión. Más tarde, cuando haya espacio disponible, el destino envía un informe de ventana mayor que cero para comenzar la transmisión de nuevo.

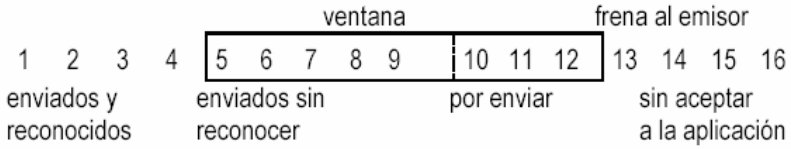
Con el método de las ventanas deslizantes de tamaño variable, el protocolo TCP soluciona el problema de transferencia fiable y control de flujo extremo a extremo; sin embargo, no soluciona el control de congestión en la red Internet, pues ésta tiene conectadas máquinas intermedias de distintas velocidades y tamaños que se comunican a través de redes de distintas características. La misión de ese control corresponde lógicamente al Protocolo de Red IP, que para solucionarlo utiliza el mecanismo menos preciso de los mensajes ICMP de disminución de flujo de la fuente.

3.3.1.- Protocolo del bit alternante.

El envío consecutivo de mensajes, asentidos uno a uno, necesita de un protocolo que garantice la fiabilidad. El más sencillo de ellos es el **protocolo del bit alternante**. Para aumentar la eficiencia de la transmisión y realizar control del flujo extremo a extremo, se permite que el emisor emita segmentos sin haber obtenido los ACKs de los anteriores. Esto exige la existencia de memoria en el emisor y el receptor.

El emisor guarda en un número acotado de buffers, los segmentos enviados y aún no reconocidos. Cuando los buffers se llenan, el emisor se para. El receptor almacena en sus buffers aquellos segmentos recibidos pero aún no solicitados por la aplicación. Si la aplicación no consume datos, la cola se llena y deja de reconocer nuevos segmentos.

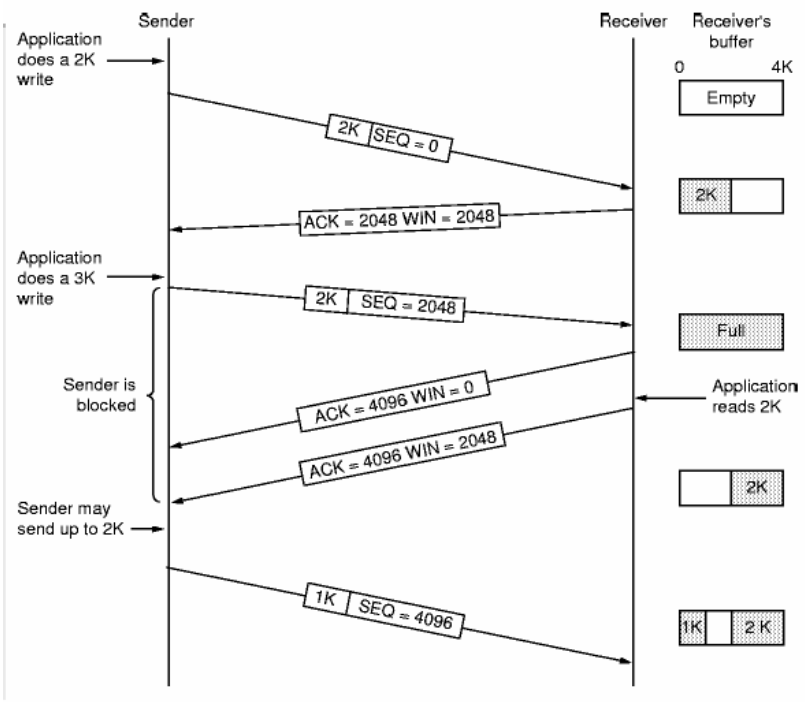
Cada conexión lleva asociadas dos ventanas por sentido, en el emisor y en el receptor.



El tamaño de la ventana en TCP varía con el tiempo. En cada ACK el campo WINDOW especifica la cantidad de datos que el receptor está dispuesto a recibir. Si el tamaño es mayor que el actual, se crean nuevos buffers. Si es menor, reduce el número de buffers (lógicamente no puede perder datos, por lo que la reducción del tamaño sólo se envía cuando se producen ACKs que liberan).

En el caso extremo, el receptor saturado reduce a cero la ventana para detener al emisor (si sigue enviando llena de tráfico la red para nada). Por si luego se perdiesen los mensajes aumentando la ventana, cada cierto tiempo el emisor incrementa el tamaño de la ventana.

Con este tipo de mecanismos, aunque no se puede conocer qué router está congestionado, se pueden tratar problemas de congestión. Si las colas de los routers se llenan, y comienzan a desechar datagramas y a enviar errores con ICMP, una buena implementación de TCP puede detectar el problema y recuperarse. Aunque la ventana sea cero, el emisor puede transmitir datos que tengan puesto el bit de urgente.



3.3.2.- Acuses de recibo y retransmisiones.

Puesto que el software TCP envía los datos en segmentos de longitud variable, los acuses de recibo se refieren a la posición en la secuencia de bytes y no a paquetes o segmentos. Cada acuse de recibo especifica la posición inmediatamente superior al byte más alto recibido.

Cada vez que envía un segmento, el software TCP inicia un temporizador y espera por un acuse de recibo. Si éste no llega antes de que el temporizador expire, se supone que el segmento se perdió y, en consecuencia, se retransmite. Esto lo hacen la mayoría de los protocolos. La diferencia del protocolo TCP es que está pensado para usarse en la red Internet, en la que el camino entre dos máquinas puede ser desde una simple red de alta velocidad, hasta un camino a través de muchas redes y nodos intermedios. Por tanto, es imposible conocer *a priori* lo rápido que los acuses de recibo van a llegar desde la fuente.

Además, el retraso depende del tráfico, por lo que éste varía mucho de unos instantes a otros. El protocolo TCP se acomoda a los cambios en los retrasos de la red mediante un algoritmo adaptativo. Para conocer los datos necesarios para el algoritmo adaptativo, el software TCP graba la hora a la que envía el segmento y la hora a la que recibe el acuse de recibo para los datos del segmento. Con estos dos datos el computador calcula el llamado tiempo de retardo. Cada vez que el computador calcula un nuevo tiempo de retardo modifica su noción de tiempo medio de retardo para la conexión. Para calcular este tiempo medio se usa una media ponderada entre el último tiempo de retardo calculado y el tiempo medio anterior; y así el tiempo medio de retardo varía lentamente. Por ejemplo, una técnica de ponderación es usar un factor de peso, T, que varía entre cero y uno, para ponderar el tiempo medio antiguo frente al nuevo tiempo de retardo:

$$T_{\text{medio actual}}=(q*T_{\text{medio anterior}})+((1-q)*\text{Nuevo\_Tiempo\_Retardo})$$

Escogiendo un valor de *q* próximo a 1 hace que la media sea inmune a cambios que duren poco tiempo (por ejemplo, un segmento que sufre un retardo muy grande). Cogiendo un valor de *q* próximo a 0, el tiempo medio de retraso responde a los cambios muy rápidamente.

La fiabilidad se consigue mediante un mecanismo llamado *Positive Acknowledgement with Retransmisión* (PAR). Si no se recibe indicación de que se recibieron correctamente los datos, se retransmiten. Si el checksum es correcto, el receptor envía un ACK positivo. Si no, lo descarta y espera retransmisión.

Los ACKS indican el número de orden del siguiente octeto que se desea recibir (reconociendo que se han recibido correctamente los anteriores). No hace falta reconocer cada uno de los segmentos recibidos. Si se envía un ACK indicando que se han recibido bien 4000 caracteres, se asume que el primer segmento con los 1000 primeros fue correctamente recibido.

Los segmentos TCP envían datos en segmentos de longitud variable. Los segmentos retransmitidos pueden contener más datos que los originales, por lo que los ACKs deben referirse a caracteres, no a segmentos, y el receptor debe reconstruir la secuencia exacta, con independencia del orden de recepción.

La pérdida de ACKs no fuerza una retransmisión. No existen reconocimientos negativos ni rechazo selectivo (no se pueden reconocer los 10000 primeros caracteres a excepción del rango 200-276), lo que puede generar retransmisiones innecesarias. La transmisión siempre es dirigida por la iniciativa del emisor, que retransmite únicamente cuando vencen los plazos y no tiene reconocimiento explícito de que un carácter haya llegado a destino, y retoma desde ahí toda la tarea hasta que reciba un asentimiento que pueda incluir otros caracteres previamente enviados y que ya no necesitará retransmitir.

TCP espera que el destinatario reconozca los segmentos cada cierto tiempo (si tiene datos que enviar, con piggybacking, y si no con un segmento ACK). Cada vez que se envía un segmento, se inicia un temporizador y se espera un plazo, a cuyo vencimiento se asume que se perdió el segmento y se procede a su retransmisión.

Es imposible conocer a priori cuanto tardará el enlace con la otra máquina (el número de routers atravesados y su congestión es imprevisible). TCP debe acomodarse a las diferencias entre los más rápidos y los más lentos, usando un algoritmo adaptativo: mide las prestaciones obtenidas y en función de ellas prevé su futuro.

TCP computa el tiempo entre que se envió un segmento y se recibió su ACK, y va calculando la media en el tiempo.

3.3.3.- Algoritmo de Jacobson.

Por cada conexión TCP mantiene una variable RTT, que es la mejor estimación del tiempo de ida y vuelta del destino en cuestión. Al enviarse un segmento se inicia un temporizador, si llega el ACK antes de que expire, el TCP mide ese tiempo ("nuevo") y actualiza RTT:

$$RTT = (a*viejo\_RTT) + ((1-a)*nuevo\_RTT) \qquad (0<\mathbf{a}<1)$$

a: factor de amortiguamiento
Y asume que el temporizador debe valer **b*RTT**.
Si **b** es pequeño, puedes reenviar muchos paquetes innecesariamente (sólo estaban algo retrasados). Si **b** es grande, puedes esperar mucho antes de notar que se perdió. Recomendación: b=2.

La medida del RTT debe considerar si el ACK se produce por la llegada del paquete original (que tardó) o por el retransmitido (que fue rápido). Considerar lo primero retrasará el rendimiento, considerar lo segundo puede conducir a exigir un RTT demasiado pequeño, lo que también generará retransmisiones innecesarias.

3.3.4.- Algoritmo de Karn.

Según Karn, los segmentos para los que se ha realizado retransmisión no deben considerarse a la hora de actualizar el RTT. En cambio, siempre que se retransmite un segmento, TCP dobla el temporizador (se multiplica por 2), hasta que nos encontremos un segmento que llegue sin necesidad de retransmisión y que reduzca el RTT.

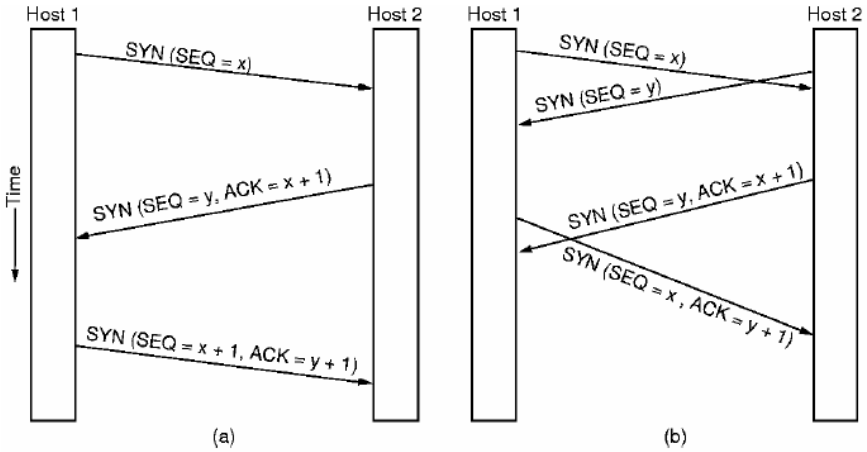
3.4.- Establecimiento y liberación de una conexión TCP.

Para establecer una conexión, el protocolo TCP utiliza el *three-way handshake*. El primer segmento transmitido se puede identificar puesto que tiene el bit SYN a 1 en el campo CODIGO. El segundo segmento, respuesta al anterior, tiene los bits SYN y ACK a 1, indicando que reconoce el primer SYN y continúa el proceso de conexión. El último caso es simplemente un acuse de recibo para informar al destino de que ambas máquinas están de acuerdo en que la conexión ha sido establecida. Normalmente, el TCP en una máquina espera pasivamente por una conexión y la otra la inicia; sin embargo, el *three-way handshake* está diseñado para que la conexión se abra incluso si las dos partes la intentan iniciar simultáneamente. Una vez que la conexión está abierta, los datos pueden ir en ambas direcciones.

Los servicios orientados a conexión constan de tres fases:

- En la fase de establecimiento de la conexión se reservan los recursos para asegurar un grado de coherencia del servicio.

- Durante la fase de transferencia, los datos son transmitidos secuencialmente por la ruta establecida, llegando a su destino en el mismo orden en que fueron enviados.
- La fase de terminación de la conexión consiste en finalizar la conexión entre origen y destino cuando deja de ser necesaria.



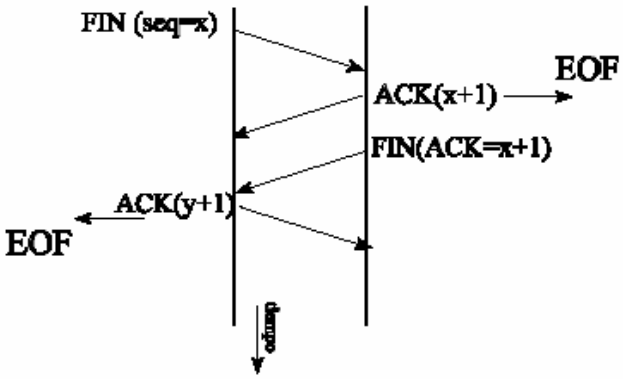
Este intercambio de tres mensajes es necesario para una correcta sincronización entre los dos lados de la conexión. Esto es debido a que, dado que se trabaja en una red no fiable, el software TCP debe usar retransmisiones para las peticiones de conexión. El problema surge cuando se reciben peticiones retransmitidas de conexión cuando ésta está estableciéndose, o cuando la conexión ya ha sido abierta, usada y terminada. El *three-way handshake*, junto con la regla de que TCP ignora peticiones de retransmisión una vez que la conexión está establecida, soluciona el problema.

El *three-way handshake* sirve además para que los dos lados de la conexión se pongan de acuerdo en los números de secuencia iniciales. Estos números de secuencia son transmitidos y confirmados durante el establecimiento de la conexión. Los números de secuencia no tienen que empezar en 1 obligatoriamente. De echo, no deberían. Al empezar, una máquina (por ejemplo la A) pasa su número de secuencia inicial, *X*, en el primer paquete, que tiene el bit SYN a 1. La segunda máquina, B, recibe el SYN, graba el número de secuencia inicial de A y responde indicando su número de secuencia inicial, *Y*, así como un número de acuse de recibo que especifica que B está esperando el byte *X+1*. En el último paquete, la máquina A reconoce el número de secuencia inicial de B enviando como número acuse de recibo *Y+1*.

Para entender el proceso de liberación de una conexión hay que tener en cuenta que la comunicación es "full-dúplex", y que se transfieren dos secuencias de bytes independientes, una en cada dirección. Cuando un programa de aplicación indica al software TCP que no tiene más datos que transmitir, éste libera la conexión en esa dirección. Para liberar su mitad de conexión, el software TCP envía el último paquete de datos con el bit FIN a 1.

En el otro extremo, el software TCP reconoce el FIN e indica al programa de aplicación que no se van a recibir más datos (por ejemplo, usando el mecanismo de final de fichero del sistema operativo). El bit FIN, al igual que se había visto con el SYN, ocupa una posición en la secuencia de bytes de la cabecera.

La transmisión del ACK correspondiente al FIN evita que A retransmita, ya que el receptor puede tardar en enviar su FIN.



Una vez que la conexión se ha liberado en una dirección, no se aceptan más datos en esa dirección. Mientras tanto, los datos pueden seguir transmitiéndose en la dirección opuesta hasta que ésta también se libere. Cuando ambas direcciones han sido liberadas, la conexión se borra y los recursos que ésta utiliza (buffers, etc.) se liberan.

Un programa de aplicación libera una conexión cuando deja de usarla. Por tanto, liberar la conexión es algo normal. Sin embargo, a veces, aparecen condiciones anormales que obligan a una aplicación a abandonar la conexión. El protocolo TCP proporciona un mecanismo de "reset" para estas desconexiones anormales. Un lado de la conexión inicia el "reset" enviando un segmento con el bit RST a 1. El otro lado de la conexión responde abortando la misma. También informa al programa de aplicación que el "reset" ha ocurrido. El "reseteo" de una conexión significa que la transferencia en ambas direcciones cesa inmediatamente, y los recursos de la misma (buffers,etc.) se liberan.

3.7.- Envio forzado de datos o fuera de banda.

El protocolo TCP permite dividir la secuencia de bytes a transmitir en segmentos. La ventaja es la eficiencia. Se pueden acumular suficientes bytes en un "buffer" para crear segmentos razonablemente largos, con lo que se reduce el alto porcentaje de encabezamientos de los segmentos cortos.

Aunque el uso de "buffers" incrementa la densidad de transmisión de la red, puede no ser conveniente para algunos tipos de aplicación; por ejemplo, en el caso de una conexión TCP utilizada para enviar caracteres desde un terminal interactivo a una máquina remota. El usuario espera una respuesta instantánea a cada pulsación de una tecla. Si el software TCP "bufferea" los datos, la respuesta se puede retrasar.

Para satisfacer a usuarios interactivos, el TCP proporciona una operación de *push* que un programa de aplicación puede utilizar para obligar al software TCP a transmitir los bytes de la secuencia sin que se llene el "buffer". Además, esta operación hace que el bit PSH del campo CODIGO esté a 1, con lo que los datos serán entregados al programa de aplicación en la máquina destino inmediatamente. Por lo tanto, cuando se envían datos desde una terminal interactiva, se usa la función de *push* después de cada pulsación de tecla.

Además de la función de *push*, el protocolo TCP proporciona una utilidad de *puntero* que permite a la fuente informar al destino que los datos de la secuencia, hasta los que señala este puntero, son urgentes, y, por tanto, deben ser procesados lo más rápidamente posible. Por ejemplo, en una conexión TCP usada para intercambiar datos entre un terminal interactivo y un computador, los caracteres que paran y arrancan la salida por pantalla (por ejemplo control-s y control-q) deberían ser considerados urgentes, ya que el destino debe procesarlos inmediatamente.

Otra técnica diferente para el envío de caracteres con carácter de "urgentes" se realiza gracias al puntero de urgencias, habilitado por el bit de código URG. Esto es especialmente interesante en el envío de interrupciones (^C). Debe notificarse (según cada sistema operativo) a la aplicación receptora de la llegada de caracteres enviados de esta manera, sin importar su posición en el stream.

3.5.- Control de la congestión

La congestión se manifiesta en grandes retardos causados por el exceso de tráfico en los servidores: encolan lo que no pueden retransmitir hasta que se llenan las colas y pierden lo que les llega. Los retrasos aumentan las retransmisiones, y por tanto supone más sobrecarga, hasta colapsar la red.

Lo adecuado es bajar la tasa de emisión. El estándar TCP recomienda actualmente 2 técnicas: **slow start** y **multiplicative decrease**, que permiten mejorar las prestaciones entre 2 y 10 veces. Para ello se mantiene una ventana de congestión:

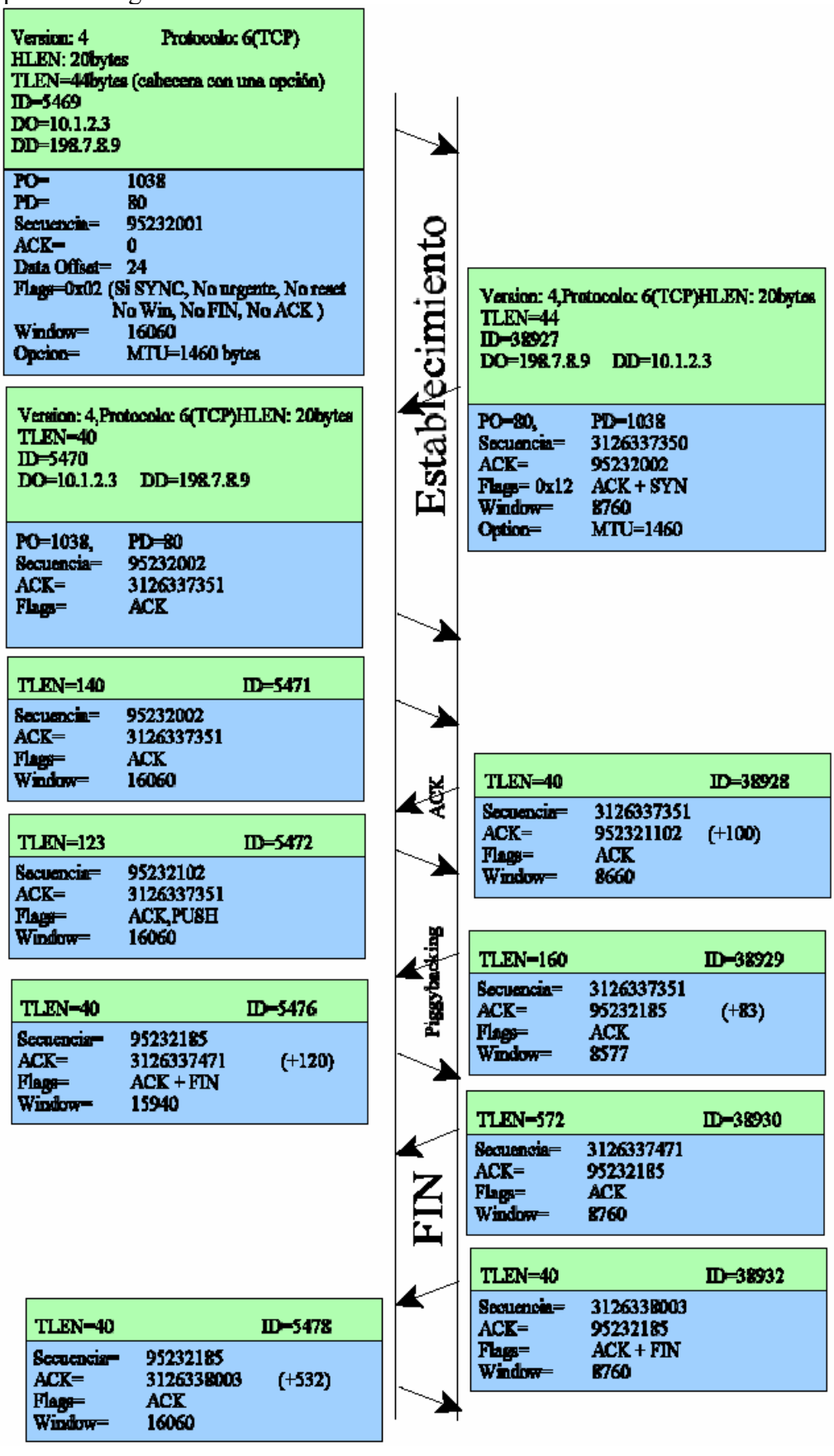
ventana_permitida = min(ventana_anunciada, ventana_congestión)

Cuando no hay congestión, ambas ventanas son iguales. Cuando hay congestión, la segunda se reduce:

- **Multiplicative decrease**: cuando se pierde un segmento, se reduce la ventana de congestión a la mitad (hasta un mínimo), y se aumentan los timers de los segmentos que quedan. Como la ventana decrece exponencialmente, se reduce el tráfico exponencialmente, para bajar pronto la congestión. Cuando acaba la congestión se dobla la ventana, se llegaría pronto a otra congestión:

- **Slow start**: Cuando acaba la congestión la ventana de congestión se aumenta en un segmento cada vez que se recibe un ACK. Aún así el tráfico crece rápidamente de nuevo: envía 1, recibe su ACK, aumenta la ventana a 2 y recibe 2 ACKs, aumenta 1 por cada ACK y puede enviar 4, luego 8... Cuando la ventana de congestión llega a la mitad de su valor original, se frena la tasa de aumento de la ventana.

Ejemplo de dialogo TCP:



3.6.- Puertos TCP.

El protocolo TCP está encima del IP en el esquema de capas de la red Internet. TCP permite a varios programas de aplicación en una misma máquina comunicarse simultáneamente y demultiplexar el tráfico TCP que se recibe entre las distintas aplicaciones. El protocolo TCP incorpora los denominados puertos, que identifican el último destino dentro de una máquina, el programa de aplicación que está haciendo uso de ese puerto. Cada puerto tiene asociado un entero para identificarlo. Dado que el número de puerto es único dentro de una máquina, el destino último para el tráfico TCP queda perfectamente determinado por la dirección internet del "host" y el número de puerto TCP en ese "host".

3.7.- Números reservados para puertos TCP.

El protocolo TCP utiliza combina una adjudicación de números de puertos TCP dinámica y estática, usando una serie de asignaciones de puertos conocida para un conjunto de programas que se utilizan comúnmente (por ejemplo, correo electrónico). Sin embargo, la mayoría de los números de puerto están disponibles para que el sistema operativo los utilice a medida que los programas de aplicación los necesiten.