

CC51C Comunicación de Datos Capa Transporte

1 UDP: User Datagram Protocol

Básicamente, UDP permite el envío de datagramas IP en forma casi directa. Lo que se agrega es un checksum sobre el paquete UDP completo (header + datos), y direccionamiento adicional a través de un puerto (*port*) de origen y un puerto de destino. Estos puertos permiten direccionar no sólo a hosts dentro de una red sino además a descriptores de archivos (y por ende a aplicaciones) hacia/desde los cuales se escribe en las aplicaciones. El checksum es opcional en IPv4.

Existen puertos reservados, que son los menores a 1024 y pueden ser usados solamente por usuarios con privilegios. Estos se usan para servicios conocidos y son asignados centralizadamente. En general, se definen en el archivo `/etc/services` en sistemas Unix. El resto de los puertos se asigna en forma más liberal.

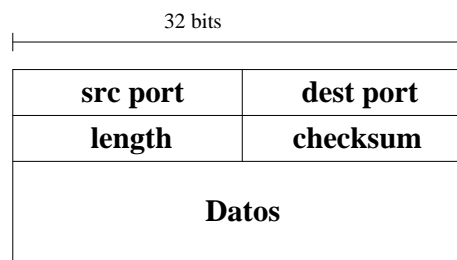


Figura 1: Paquete UDP

Los campos que se ven en la figura 1 corresponden a lo siguiente:

- source port: el puerto de origen
- destination port: el puerto de destino
- length: el largo en bytes del paquete UDP (incluyendo el header)
- checksum: checksum que cubre un pseudo-header IP, un pseudo-header de UDP y los datos.

No existe el concepto de conexión, cada datagrama se envía independientemente y puede incluso ser recibido por una aplicación distinta a pesar de tener el mismo destino y origen.

No existe orden de llegada (un paquete enviado antes que otro puede llegar más tarde), y no se garantiza que llegue el datagrama enviado. Pero sí se garantiza que si llega el paquete, los datos están correctos.

Cabe preguntar qué utilidad presenta UDP, si no permite el envío confiable de datos? Existen básicamente dos razones:

1. eficiencia: no es necesario establecer y terminar una conexión (que es caro) sobre todo si se transmite poca información (ej: DNS) y es posible usar timeouts no estándares para aplicaciones específicas que aumenten el rendimiento (ej: NFS).
2. envío a múltiples destinatarios: el envío de paquetes de multicas y broadcast tiene sentido sólo en un ambiente donde no hay una conexión (que por definición es punto a punto), ni tiene sentido (o no es razonable) enviar ACKs ni retransmitir datos. Por ello, el uso de broadcast y multicast es posible sólo a través de UDP.

2 TCP: Transmission Control Protocol

Provee confiabilidad, control (recuperación) de errores y control de flujo. Todo esto a partir de lo que provee IP. Es necesario detectar errores y retransmitir, ordenar y manejar los tamaños de ventanas apropiados según los estados de receptores y emisores.

Antes de poder transmitir datos, se debe establecer una conexión. Para ello vamos a suponer que existe alguien escuchando en un lado, y se inicia la conexión desde el otro punto. Una vez establecida la conexión, esta se asemeja a un pipe de Unix, bidireccional.

Una conexión TCP se identifica con la tupla [IP origen, port origen, IP destino, port destino]. Con este esquema, es posible que una aplicación maneje simultáneamente varias conexiones que llegan al mismo destino (IP y port), pero que se originan desde una dirección IP o bien desde un port (o ambos) distinto.

En cuanto a direccionamiento, provee lo mismo que UDP: ports de 16 bits. Un número de puerto en UDP no tiene relación con el mismo número de port en TCP, aunque hay protocolos que funcionan sobre ambos.

2.1 Header TCP

- Source port: port de origen
- Destination port: port de destino
- Sequence number: número de secuencia
- ACK number: número de ACK
- hlen o 'Data offset': el tamaño en bloques de 32 bits del header TCP. El largo de los datos se tiene que calcular a partir de esto y del largo indicado en el header IP.
- Reserved: 6 bits no usados inicialmente.
- Flags: 6 bits:
 1. URG: indica que el campo Urgent Ptr tiene información relevante.
 2. ACK: indica que el campo ACK Number tiene un número de secuencia para el cual se incluye el ACK.

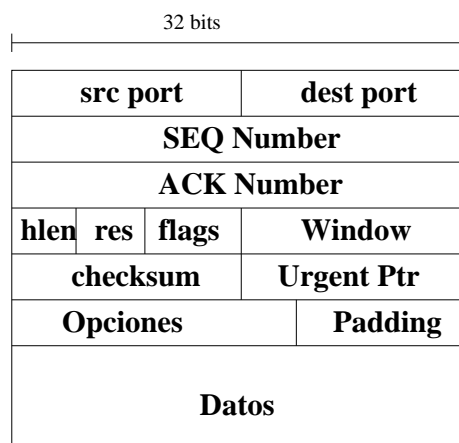


Figura 2: Paquete TCP

3. PSH: Push: es la forma en que se indica que no se debiera seguir juntando bytes para pasarselos juntos al proceso.
 4. RST: Reset: se envía siempre cuando se recibe un paquete que no parece estar destinado a la conexión para la cual llegó. Sirve para volver a sincronizar ambos participantes.
 5. SYN: Sincronización de números de secuencia. Usado al iniciar una conexión.
 6. FIN: Usado para finalizar una conexión.
- Window: se usa sólo si hay un ACK, indica el tamaño del buffer que tiene disponible el receptor. No se debe enviar más que esa cantidad de bytes a partir del número de secuencia indicado en el ACK Number.
 - checksum: checksum sobre un pseudo-header de IP, TCP y los datos (tal como en UDP, pero es obligatorio siempre).
 - Urgent Pointer: indica el primer byte después de los datos urgentes. Sólo es relevante si el bit URG está seteado.
 - opciones: datos opcionales como por ejemplo el MSS (Maximum Segment Size), basado en el mínimo MTU del camino, el uso de *selective acknowledgement* (SACK), uso de tamaños de ventanas más grandes, entre otros.
 - padding: relleno para que el largo del header sea un múltiplo de 32 bits.

2.2 Ventanas en TCP

TCP usa un esquema de ventana corredera parecido a Go-Back-N, con algunas optimizaciones y diferencias. Los números de secuencia no cuentan paquetes, sino bytes, al

igual que los tamaños de ventana. Los datos a enviar se ven como un stream, que se divide en segmentos TCP, que es una subsecuencia de bytes que se envía en un paquete TCP. El número de secuencia y ACK que se usa dentro del header TCP se refiere al primer byte de datos. Por ejemplo, si llega un paquete que contiene datos hasta el byte n , el ACK se envía con el número $n + 1$, indicando que se ha recibido todo lo anterior a ese byte.

La implementación además permite control de flujo, variando el tamaño de las ventanas según el estado de los buffers. Esta información se comparte con la contraparte a través del campo *window* con cada ACK.

2.3 Establecer y cerrar una conexión TCP

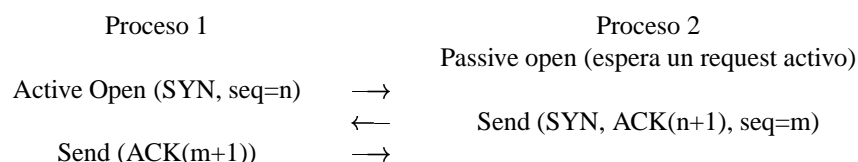
Al establecer una conexión TCP, se negocian varios parámetros entre las partes:

- MSS: se calcula usando un algoritmo llamado *Path MTU Discovery* (antes era fijo: 536 bytes si destino era remoto, MTU si era local).
- SEQ Number: para no confundir dos conexiones se elige un número de secuencia al azar para cada dirección. Son conocidos los problemas de seguridad ocasionados al no implementar un buen algoritmo de generación de números aleatorios (constant positive increment es peor que random positive increment por ejemplo).
- Tamaños iniciales de las ventanas del receptor según cada dirección. Dependen de los buffers que se destinan inicialmente a la conexión.

Como se puede ver en el diagrama de la figura 3, una conexión tiene varios estados posibles. En cada instante, cada una de las partes tiene determinado estado (no necesariamente el mismo que su contraparte). Una de las tareas más difíciles para TCP es garantizar que, si bien los estados pueden ser distintos, no existan inconsistencias en los estados, no importa qué suceda con los paquetes que se envían y reciben. El TCB al que se hace referencia es una estructura imaginaria llamada *Transmission Control Block*, que mapea una conexión a un nombre local.

Para poder establecer una conexión TCP, se usa el *Three way handshake*, con el cual se asegura que ambos extremos estén de acuerdo en el estado de la conexión y se intercambian los parámetros necesarios.

2.3.1 Three way handshake



El término de una conexión TCP se hace en forma implícita, enviando un paquete con el bit FIN seteado. A partir de ese momento, la conexión se encuentra cerrada para ese sentido, y se espera que el otro lado envíe un paquete análogo, cerrando el otro

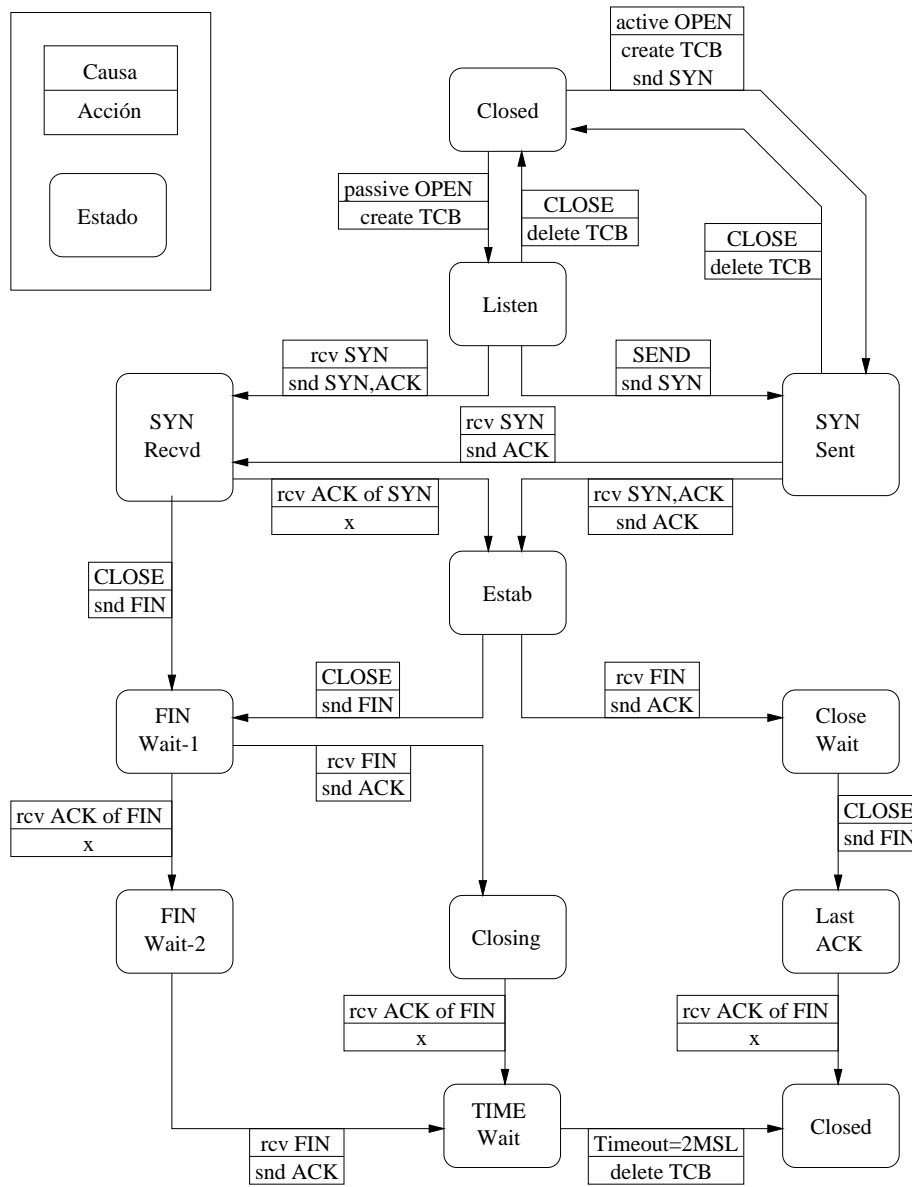


Figura 3: Diagrama de estados para una conexión TCP

sentido de la conexión. Una vez que esto sucede, se puede eliminar la conexión en ambos lados (esperando un tiempo prudente primero).

2.4 Timeout

La primera variable que se puede controlar para manejar la congestión es el timeout. Este necesariamente debe ser variable, para adaptarse a las distintas condiciones que se tienen en distintas partes y/o momentos de la red. Si el timeout es demasiado corto, se asumirá pérdida de paquetes cuando en realidad sólo hay una demora. En cambio, si el timeout es demasiado largo, nos demoramos mucho en detectar una pérdida y se desperdicia ancho de banda esperando ACKs que no van a llegar. Es por esto que el Timeout se debe adaptar a las condiciones actuales de cada conexión específica.

Se va manteniendo una estimación del RTT (Round Trip Time) actual, la cual se actualiza para cada ACK que llega. Esta estimación se calcula basado en la estimación anterior y el último cálculo:

$$RTT = (\alpha * RTT) + (1 - \alpha) * (RTT_Sample)$$

Actualmente, se utiliza típicamente un valor de $\frac{\alpha=7}{8}$ de modo de reaccionar lentamente a los cambios. Teniendo una estimación del tiempo promedio, normalmente se definía un timeout del doble de ese valor. Sin embargo, frente a fuerte congestión, la varianza del RTT aumenta, pudiendo incluso abarcar dentro de lo razonable el doble del RTT medio (carga sobre el 30% puede producir esto). Para evitar esto, TCP calcula también la desviación estándar del RTT (afortunadamente esto no requiere mucho cálculo) y usa la varianza multiplicada por el RTT como timeout. El código es:

$$DIFF = RTT_sample - RTT$$

$$RTT = RTT - \alpha * DIFF$$

$$MDEV = MDEV + \alpha * (abs(DIFF) - MDEV)$$

Un problema pendiente es cómo calcular RTT frente a retransmisiones, puesto que un ack puede corresponder al original o al retransmitido. El algoritmo de Karn, implementado en TCP, no actualiza RTT cuando se recibe el ack de un segmento retransmitido. Al mismo tiempo, el emisor multiplica por dos su timeout cada vez que retransmite un segmento. En general, se acepta un valor máximo de 120 segundos (lo que implica que será imposible usar las implementaciones actuales de TCP para comunicarse con Marte).

2.5 Algoritmos de control de congestión

- Slow start

En TCP el emisor maneja dos tamaños para la ventana: el de congestión y el que le informa el receptor. El tamaño usado siempre es el menor de ambos valores. La ventana de congestión parte con tamaño 1, y sólo al recibir ACKs se aumenta ese tamaño. El tamaño se aumenta en forma exponencial si se van

recibiendo todos los ACKs, pero el receptor puede manejar la forma en que crece el tamaño por ejemplo enviando un ACK por cada 2 segmentos recibidos (sin que haya que retransmitir). Cuando se alcanza un cierto nivel (que se supone es cercano a la capacidad de la red), se va incrementando el tamaño de la ventana en forma lineal. Siempre el tope de este crecimiento es el tamaño de los buffers del receptor, así que si se alcanza ese tamaño de ventana, no se sigue agrandando.

La forma en que se detecta (o más bien se adivina) la capacidad de la red es bastante simple: si se comienzan a descartar paquetes se asume que el tamaño era más grande que la capacidad de la red.

- Congestion avoidance

Se hace una suposición importante: que la pérdida de segmentos causada por errores en los datos es mínima (mucho menor al 1%). Así que se asume que si hay pérdida de segmentos, es por causa de congestión en alguna parte de la red.

Una pérdida se detecta cuando se produce alguna de las siguientes situaciones:

1. ocurre un timeout
2. se recibe un ACK duplicado

En tales casos, el tamaño de congestión se divide por la mitad. A medida que se van recibiendo nuevamente los ACKs, se vuelve a incrementar el tamaño de congestión, pero sólo de a 1.

- Fast retransmit

Se espera un tiempo antes de volver a enviar todos los paquetes de la ventana, porque es probable que se haya perdido un sólo paquete. Esto evita muchos casos en que se duplicarían envíos innecesariamente.