

2.7. Transmission Control Protocol (TCP)

El *Transmission Control Protocol* (TCP) [[RFC793](#)] es el protocolo de transporte más importante de Internet, y por eso a la pila de protocolos de Internet se le conoce como TCP/IP. De hecho, el control de congestión de Internet (i.e. la razón por la que

las redes no se colapsan cuando se sobrecargan) se basa fundamentalmente en el comportamiento del protocolo TCP, que es el que transporta la inmensa mayoría de la información en Internet.

Tal como se puede observar en la **Figura 2.12**, TCP es un protocolo mucho más complejo que el protocolo UDP que analizamos en el apartado anterior, aunque mantiene algunas características similares como los dos campos de 16 bits para identificar los puertos origen y destino. O el campo *checksum*, que cubre tanto la cabecera TCP como los datos que transporta, así como las direcciones IP y el identificador de protocolo de la cabecera IP. Como estos campos ya se describieron en el capítulo sobre UDP, no se repetirán aquí.

Los campos más importantes de la cabecera TCP son los números de secuencia y de confirmación (de 32 bits cada uno), así como el conjunto de 6 *flags* (bits) de control, y el campo de ventana disponible (*Window*), que nos indican que nos encontramos ante un protocolo orientado a conexión, ordenado, fiable, y con control de flujo. Veremos ahora qué significan todos estos conceptos.

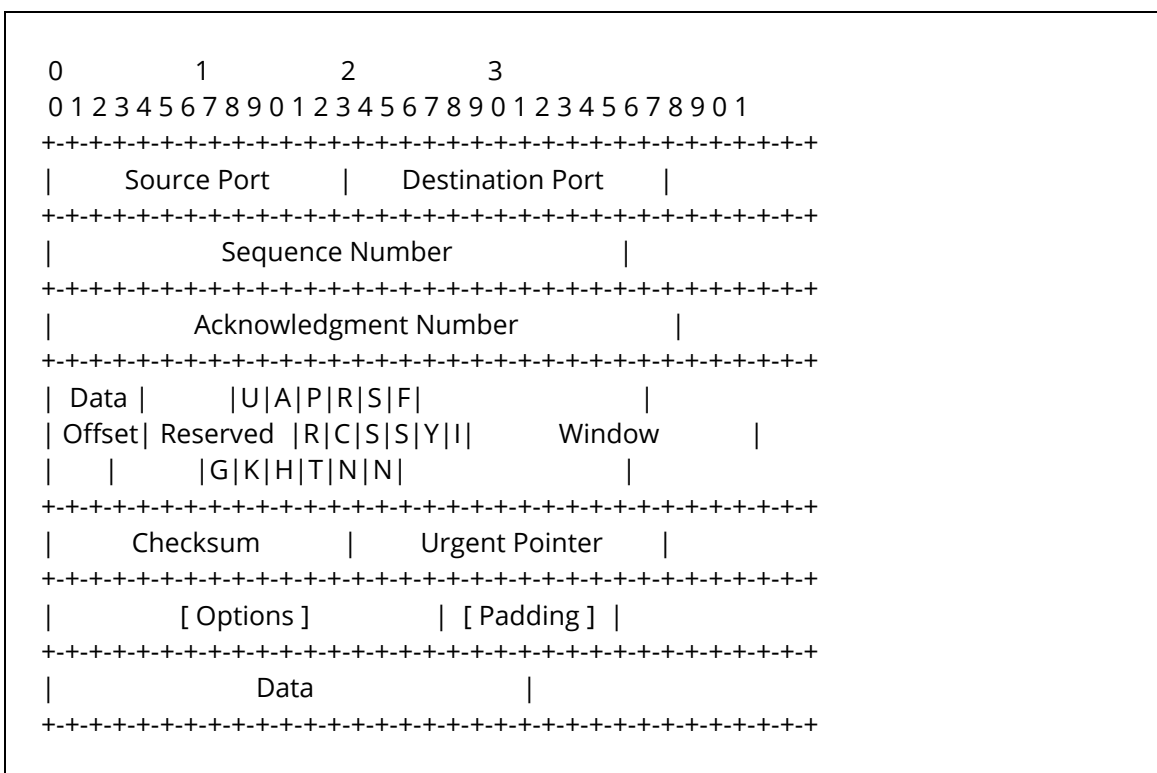


Figura 2.12 – Formato de un segmento TCP [\[RFC793\]](#)

Todos los datos que envía TCP tienen un número de secuencia asociado (el campo *Sequence Number* es el número de secuencia del primer octeto de datos que transporta), cuya recepción correcta debe confirmar el receptor mediante el campo *Acknowledgment Number* (que realmente indica el siguiente número de secuencia que espera recibir). Por lo tanto, TCP es capaz de identificar si los segmentos llegan

desordenados (las redes IP no garantizan que los datagramas lleguen al destino o que lo hagan en el mismo orden que se enviaron) y reordenarlos (gracias a sus números de secuencia), así como detectar cuando se corrompe (aunque el algoritmo de *checksum* es el mismo que IPv4 y por lo tanto bastante débil) o pierde cualquier segmento de datos (al no recibir confirmación del mismo), y retransmitirlo automáticamente. De hecho, TCP ofrece un servicio de transmisión de datos ordenado, de forma que si tiene algún segmento pendiente no entregará ningún dato posterior a la aplicación. Algunas implementaciones de TCP permitían esto mediante el *flag* URG y el campo de Puntero Urgente, pero ya no se recomienda su uso. Así que en ese caso la recepción de datos se bloqueará hasta que se recupere el segmento perdido/desordenado. Esta es la razón por la que, hasta hace no mucho, todas las aplicaciones multimedia, en las que es más importante recibir datos de manera constante que recibir absolutamente todos los datos (porque es preferible perder un fotograma a congelar la reproducción de un vídeo), usaban UDP en lugar de TCP.

El **control de flujo** de TCP evita saturar al destino, enviando más datos de los que éste puede recibir. Para ello se utiliza el campo Ventana (*Window*), mediante el cual el receptor le indica al emisor el tamaño de *buffer* de recepción que tiene libre, de forma que el emisor nunca puede enviar más datos hasta que el receptor vuelva a “abrir” la ventana, indicando que tiene más *buffer* disponible.

Además de la ventana del receptor, el emisor debe tener una ventana de transmisión (que no tiene representación en ningún campo de la cabecera, al ser un estado interno), que limita la cantidad de información que se envía a la red, y que es la forma en la que se implementa el **control de congestión** en las redes TCP/IP. Este mecanismo es bastante complejo y, como es un factor fundamental en el rendimiento de TCP, han surgido bastantes variantes a lo largo de la historia, pero básicamente consiste en ir inyectando tráfico en la red poco a poco, cada vez más rápido hasta que se detecta que la red empieza a congestionarse (porque se detecta pérdida de paquetes o aumenta el tiempo de llegada de las confirmaciones), en cuyo caso se reduce drásticamente la tasa de envío, para volver a ir subiendo poco a poco e ir repitiendo el proceso continuamente para adaptarse a la capacidad disponible que tenga la red (idealmente todos los flujos TCP comparten equitativamente la capacidad disponible de la red).

Esta combinación de mecanismos hace que la aplicación que hace uso de TCP tenga un control muy limitado sobre cuándo o cómo se envían los datos (teóricamente el *flag* PSH indica a TCP que esos datos deberían entregarse inmediatamente, pero su efecto es limitado en la mayoría de las implementaciones de TCP), puesto que TCP puede retrasar su envío, dividir un mensaje en varios segmentos, esperar a que lleguen datos anteriores antes de entregarlos, etc. De hecho, TCP ofrece un servicio de envío de bytes de manera bidireccional, ordenada y fiable, pero no permite

agrupar bytes en mensajes (a diferencia de UDP), por lo que las aplicaciones sobre TCP deben definir esa semántica si la necesitan.

Todos estos mecanismos, y la necesidad de almacenar estado por cada flujo TCP en el que se participa, hace que TCP sea un **protocolo orientado a conexión**, esto es, que sea necesario realizar una negociación antes de intercambiar datos. A esa negociación inicial de TCP se le conoce como **3-way handshake**, por los tres pasos que realiza (**Figura 2.13**).

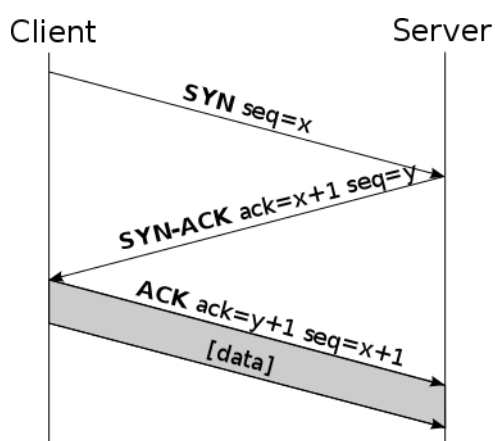


Figura 2.13 – TCP 3-way handshake (Fuente: [Wikipedia](https://es.wikipedia.org/wiki/Handshake))

En particular, el cliente es el que inicia la comunicación, enviando un segmento TCP sin datos, únicamente con el *flag* SYN activo y un número de secuencia aleatorio (x) al servidor. Si éste está dispuesto a recibir conexiones, enviará de vuelta otro segmento TCP sin datos, pero con los *flags* SYN y ACK, confirmando el número de secuencia del cliente ($x+1$) y enviando su propio número de secuencia aleatorio (y). Finalmente, el cliente debe enviar un segmento TCP con el *flag* ACK activo (ya sin el *flag* SYN), confirmando el número de secuencia del servidor ($y+1$). El último segmento del 3-way handshake ya podría enviar datos, aunque normalmente estos se envían en un segmento posterior. Si el servidor no desea establecer una nueva conexión, responderá al primer segmento de SYN con otro segmento TCP sin datos y con los *flags* RST y ACK activos, confirmando el número de secuencia del cliente ($x+1$). Aunque en la mayoría de los protocolos es el cliente el que envía datos de nivel de aplicación primero, y luego el servidor responde, TCP permite enviar datos en ambos sentidos en cuanto se completa el 3-way handshake.

En cuanto a la finalización de la conexión TCP, aunque es bastante habitual que los extremos cierren la conexión consecutivamente, realmente cada sentido de la conexión puede cerrarse de manera independiente. Para ello basta con enviar un segmento TCP con el *flag* FIN activo, que indica que el emisor no enviará más datos después de ese, y al que el otro extremo responderá con un *flag* ACK confirmando

su número de secuencia. Aunque éste podrá seguir enviando datos hasta que envíe su propio *flag* FIN para cerrar también su sentido de la conexión, y completar una desconexión ordenada. En caso de error (e.g. si la aplicación por encima de TCP se cierra inesperadamente), cualquiera de los extremos de la comunicación puede enviar un segmento con el *flag* de RST activo, que indica que la conexión debe cerrarse inmediatamente.

Por último, la cabecera TCP también permite incluir opciones, codificadas con una estructura TLV (Tipo-Longitud-Valor) y que permiten definir más parámetros de la conexión: como el tamaño máximo de los segmentos (*Maximum Segment Size*) que se pueden recibir, ampliar el tamaño máximo de la ventana de recepción (*Windows Scale*), incluir marcas de tiempo (*Timestamp*) en los segmentos para calcular el RTT (*Round Trip Time*) de la red, o implementar el mecanismo de confirmaciones selectivas (SACK). El tamaño de la cabecera TCP, incluidas sus opciones, se indica en el campo *Data Offset*, lo que también permite encontrar rápidamente dónde empiezan los datos de la aplicación.