# Transmission Control Protocol
# *Philosophy*

Jon Postel, Editor

Information Sciences Institute · University of Southern California

**Preface.** This document describes the DoD Standard Transmission Control Protocol (TCP). There have been eight earlier editions of the ARPA TCP specification on which this standard is based, and the present text draws heavily from them. There have been many contributors to this work both in terms of concepts and in terms of text. This edition incorporates the addition of security, compartmentation, and precedence concepts into the TCP specification.

### 2.1. Elements of the Internetwork System

The internetwork environment consists of hosts connected to networks which are in turn interconnected via gateways. It is assumed here that the networks may be either local networks (e.g., the ETHERNET) or large networks (e.g., the ARPANET), but in any case are based on packet switching technology. The active agents that produce and consume messages are processes. Various levels of protocols in the networks, the gateways, and the hosts support an interprocess communication system that provides two-way data flow on logical connections between process ports.

We specifically assume that data is transmitted from host to host through means of a set of networks. When we say network, we have in mind a packet switched network (PSN). This assumption is probably unnecessary, since a circuit switched network or a hybrid combination of the two could also be used; but for concreteness, we explicitly assume that the hosts are connected to one or more packet switches of a PSN.

The term packet is used generically here to mean the data of one transaction between a host and a packet switch. The format of data blocks exchanged between the packet switches in a network will generally not be of concern to us.

Hosts are computers attached to a network, and from the communication network's point of view, are the sources and destinations of packets. Processes are viewed as the active elements in host computers (in accordance with the fairly common definition of a process as a program in execution). Even terminals and files or other I/O devices are viewed as communicating with each other through the use of processes. Thus, all communication is viewed as inter-process communication.

Since a process may need to distinguish among several communication streams between itself and another process (or processes), we imagine that each process may have a number of ports through which it communicates with the ports of other processes.

### 2.2. Model of Operation

Processes transmit data by calling on the TCP and passing buffers of data as arguments. The TCP packages the data from these buffers into segments and calls on the internet module to transmit each segment to the destination TCP. The receiving TCP places the data from a segment into the receiving user's buffer and notifies the receiving user. The TCPs include control information in the segments which they use to ensure reliable ordered data transmission.

The model of internet communication is that there is an internet protocol module associated with each TCP which provides an interface to the local network. This internet module packages TCP segments inside internet datagrams and routes these datagrams to a destination internet module or intermediate gateway. To transmit the datagram through the local network, it is embedded in a local network packet.

The packet switches may perform further packaging, fragmentation, or other operations to achieve the delivery of the local packet to the destination internet module.

At a gateway between networks, the internet datagram is "unwrapped" from its local packet and examined to determine through which network the internet datagram should travel next. The internet datagram is then "wrapped" in a local packet suitable to the next network and routed to the next gateway, or to the final destination.

A gateway is permitted to break up an internet datagram into smaller internet datagram fragments if this is necessary for transmission through the next network. To do this, the gateway produces a set of internet datagrams; each carrying a fragment. Fragments may be broken into smaller ones at intermediate gateways. The internet datagram fragment format is designed so that the destination internet module can reassemble fragments into internet datagrams.

A destination internet module unwraps the segment from the datagram (after reassembling the datagram, if necessary) and passes it to the destination TCP.

This simple model of the operation glosses over many details. One important feature is the type of service. This provides information to the gateway (or internet module) to guide it in selecting the service parameters to be used in traversing the next network. Included in the type of service information is the precedence of the datagram. Datagrams may also carry security information to permit host and gateways that operate in multilevel secure environments to properly segregate datagrams for security considerations.

### 2.3. The Host Environment

The TCP is assumed to be a module in a time sharing operating system. The users access the TCP much like they would access the file system. The TCP may call on other operating system functions, for example, to manage data structures. The actual interface to the network is assumed to be controlled by a device driver module. The TCP does not call on the network device driver directly, but rather calls on the internet datagram protocol module which may in turn call on the device driver.

Though it is assumed here that processes are supported by the host operating system, the mechanisms of TCP do not preclude implementation of the TCP in a front-end processor. However, in such an implementation, a host-to-front-end protocol must provide the functionality to support the type of TCP-user interface described above.

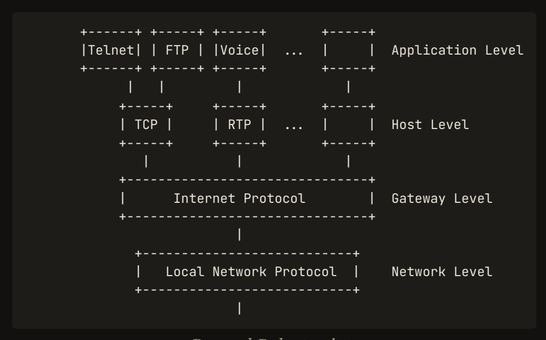
### 2.4. Interfaces

The TCP/user interface provides for calls made by the user on the TCP to OPEN or CLOSE a connection, to SEND or RECEIVE data, or to obtain STATUS about a connection. These calls are like other calls from user programs on the operating system, for example, the calls to open, read from, and close a file.

The TCP/internet interface provides calls to send and receive datagrams addressed to TCP modules in hosts anywhere in the internet system. These calls have parameters for passing the address, type of service, precedence, security, and other control information.

### 2.5. Relation to Other Protocols

The following diagram illustrates the place of the TCP in the protocol hierarchy:




*Protocol Relationships*
*Figure 2.*

It is expected that the TCP will be able to support higher level protocols efficiently. It should be easy to interface higher level protocols like the ARPANET Telnet [3] or AUTODIN II THP to the TCP.

### 2.6. Reliable Communication

A stream of data sent on a TCP connection is delivered reliably and in order at the destination.

Transmission is made reliable via the use of sequence numbers and acknowledgments. Conceptually, each octet of data is assigned a sequence number. The sequence number of the first octet of data in a segment is the sequence number transmitted with that segment and is called the segment sequence number. Segments also carry an acknowledgment number which is the sequence number of the next expected data octet of transmissions in the reverse direction. When the TCP transmits a segment, it puts a copy on a retransmission queue and starts a timer; when the acknowledgment for that data is received, the segment is deleted from the queue. If the acknowledgment is not received before the timer runs out, the segment is retransmitted.

An acknowledgment by TCP does not guarantee that the data has been delivered to the end user, but only that the receiving TCP has taken the responsibility to do so.

To govern the flow of data into a TCP, a flow control mechanism is employed. The the data receiving TCP reports a window to the sending TCP. This window specifies the number of octets, starting with the acknowledgment number that the data receiving TCP is currently prepared to receive.

### 2.7. Connection Establishment and Clearing

To identify the separate data streams that a TCP may handle, the TCP provides a port identifier. Since port identifiers are selected independently by each operating system, TCP, or user, they might not be unique. To provide for unique addresses at each TCP, we concatenate an internet address identifying the TCP with a port identifier to create a socket which will be unique throughout all networks connected together.

A connection is fully specified by the pair of sockets at the ends. A local socket may participate in many connections to different foreign sockets. A connection can be used to carry data in both directions, that is, it is "full duplex".

TCPs are free to associate ports with processes however they choose. However, several basic concepts seem necessary in any implementation.

There must be well-known sockets which the TCP associates only with the "appropriate" processes by some means. We envision that processes may "own" ports, and that processes can only initiate connections on the ports they own. (Means for implementing ownership is a local issue, but we envision a Request Port user command, or a method of uniquely allocating a group of ports to a given process, e.g., by associating the high order bits of a port name with a given process.)

A connection is specified in the OPEN call by the local port and foreign socket arguments. In return, the TCP supplies a (short) local connection name by which the user refers to the connection in subsequent calls. There are several things that must be remembered about a connection. To store this information we imagine that there is a data structure called a Transmission Control Block (TCB). One implementation strategy would have the local connection name be a pointer to the TCB for this connection. The OPEN call also specifies whether the connection establishment is to be actively pursued, or to be passively waited for.

A passive OPEN request means that the process wants to accept incoming connection requests rather than attempting to initiate a connection. Often the process requesting a passive OPEN will accept a connection request from any caller. In this case a foreign socket of all zeros is used to denote an unspecified socket. Unspecified foreign sockets are allowed only on passive OPENs.

A service process that wished to provide services for unknown other processes could issue a passive OPEN request with an unspecified foreign socket. Then a connection could be made with any process that requested a connection to this local socket. It would help if this local socket were known to be associated with this service.

Well-known sockets are a convenient mechanism for a priori associating a socket address with a standard service. For instance, the "Telnet-Server" process might be permanently assigned to a particular socket, and other sockets might be reserved for File Transfer, Remote Job Entry, Text Generator, Echoer, and Sink processes (the last three being for test purposes). A socket address might be reserved for access to a "Look-Up" service which would return the specific socket at which a newly created service would be provided. The concept of a well-known socket is part of the TCP specification, but the assignment of sockets to services is outside this specification.

Processes can issue passive OPENs and wait for matching calls from other processes and be informed by the TCP when connections have been established. Two processes which issue calls to each other at the same time are correctly connected. This flexibility is critical for the support of distributed computing in which components act asynchronously with respect to each other.

There are two cases for matching the sockets in the local request and an incoming segment. In the first case, the local request has fully specified the foreign socket. In this case, the match must be exact. In the second case, the local request has left the foreign socket unspecified. In this case, any foreign socket is acceptable as long as the local sockets match.

If there are several pending passive OPENs (recorded in TCBs) with the same local socket, an incoming segment should be matched to a request with the specific foreign socket in the segment, if such a request exists, before selecting a request with an unspecified foreign socket.

The procedures to establish and clear connections utilize synchronize (SYN) and finis (FIN) control flags and involve an exchange of three messages. This exchange has been termed a three-way hand shake [4].

A connection is initiated by the rendezvous of an arriving segment containing a SYN and a waiting TCB entry created by a user OPEN command. The matching of local and foreign sockets determines when a connection has been initiated. The connection becomes "established" when sequence numbers have been synchronized in both directions.

The clearing of a connection also involves the exchange of segments, in this case carrying the FIN control flag.

### 2.8. Data Communication

The data that flows on a connection may be thought of as a stream of octets, or as a sequence of records. In TCP the records are called letters and are of variable length. The sending user indicates in each SEND call whether the data in that call completes a letter by the setting of the end-of-letter parameter.

The length of a letter may be such that it must be broken into segments before it can be transmitted to its destination. We assume that the segments will normally be reassembled into a letter before being passed to the receiving process. A segment may contain all or a part of a letter, but a segment never contains parts of more than one letter. The end of a letter is marked by the appearance of an EOL control flag in a segment. A sending TCP is allowed to collect data from the sending user and to send that data in segments at its own convenience, until the end of letter is signaled then it must send all unsent data. When a receiving TCP has a complete letter, it must not wait for more data from the sending TCP before passing the letter to the receiving process.

There is a coupling between letters as sent and the use of buffers of data that cross the TCP/user interface. Each time an end-of-letter (EOL) flag is associated with data placed into the receiving user's buffer, the buffer is returned to the user for processing even if the buffer is not filled. If a letter is longer than the user's buffer, the letter is passed to the user in buffer size units, the last of which may be only partly full. The receiving TCP's buffer size may be communicated to the sending TCP when the connection is being established.

The TCP is responsible for regulating the flow of segments on the connections, as a way of preventing itself from becoming saturated or overloaded with traffic. This is done using a window flow control mechanism. The data receiving TCP reports to the data sending TCP a window which is the range of sequence numbers of data octets that data receiving TCP is currently prepared to accept.

TCP also provides a means to communicate to the receiver of data that at some point further along in the data stream than the receiver is currently reading there is urgent data. TCP does not attempt to define what the user specifically does upon being notified of pending urgent data, but the general notion is that the receiving process should take action to read through the end urgent data quickly.

### 2.9. Precedence and Security

The TCP makes use of the internet protocol type of service field and security option to provide precedence and security on a per connection basis to TCP users. Not all TCP modules will necessarily function in a multilevel secure environment, some may be limited to unclassified use only, and others may operate at only one security level and compartment. Consequently, some TCP implementations and services to users may be limited to a subset of the multilevel secure case.

TCP modules which operate in a multilevel secure environment should properly mark outgoing segments with the security, compartment, and precedence. Such TCP modules should also provide to their users or higher level protocols such as Telnet or THP an interface to allow them to specify the desired security level, compartment, and precedence of connections.

### 2.10. Robustness Principle

*TCP implementations should follow a general principle of robustness: be conservative in what you do, be liberal in what you accept from others.*

## References

[3] Feinler, E. and J. Postel, ARPANET Protocol Handbook, Network Information Center, SRI International, Menlo Park, CA, January 1978.

[4] Dalal, Y. and C. Sunshine, "Connection Management in Transport Protocols," Computer Networks, Vol. 2, No. 6, pp. 454-473, December 1978.



doi:10.17487/RFC0761