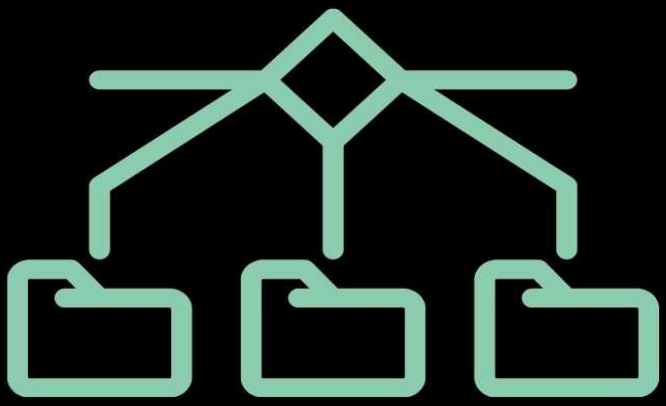
MEM LNC

# TCP / IP

## PROTOCOL FOR BEGINNERS

### The Ultimate Beginner's Guide to Learn TCP / IP protocol Step by Step



Claudia Alves

# Introduction

The Transmission Control Protocol (TCP) is designed to be used as a reliable protocol for communication between host computers in communication computer networks with packet switching, as well as in systems that combine such networks.

This document describes the functions that the transmission management protocol should perform, the program that implements the protocol, and its interface with programs or users who need its services.

## 1.1 Motivation

Computer communications systems are playing an increasingly important role in military, government and civilian applications. This document primarily highlights the requirements for computer communications in the military field, and especially for sustainability in the face of insufficient communication reliability and overload capabilities.

At a time when strategic and tactical networks of computer communications are emerging and disappearing, it is important to provide the means to combine them, as well as standard protocols of communication between

processes that would support a wide range of applications. Anticipating the need for such standards, the Office of the Defence Secretariat for Research and Development presented the Transmission Control Protocol (TCP) protocol described here, based on the standardization of the DoD protocol of communication between processes.

TCP is a direct connectivity reliability protocol designed for a multi-layered hierarchy of protocols that support firewall applications. The TCP protocol ensures the reliability of communications between pairs of processes on host computers included in various computer communication networks, which are combined into a single system.

With regard to the reliability of lower-than-TCP protocols, very modest requests have been made. TCP suggests that it may get a simple, potentially unreliable service for its datagrams from lower-level protocols. In principle, the TCP protocol should be operational on a large set of communication systems, from cable connections to networks with switching packages or electrical circuits.

The TCP protocol is based on concepts first described by the authors Cerf and Kahn in the paper. TCP occupies a multi-level protocol architecture niche directly above the Internet protocol, which allows the TCP protocol to send and receive segments of variable-length information enclosed in the Internet datagram shell. The Internet datagram provides tools to address the sender and recipient of TCP segments on various networks. The Internet Protocol also performs any fragmentation and assembly of TCP segments required for transmission and delivery through multiple networks and intermediate gateways. The Internet Protocol also processes information about priority, security classification, and delineation of TCP segments. So this information can be transmitted directly through multiple networks.

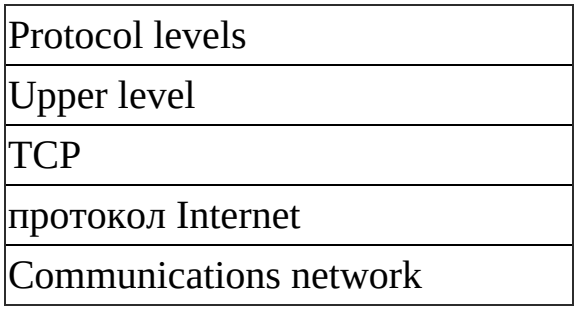



Fig. 1

Most of this document is written in connection with the implementations of

the TCP protocol, which, together with higher-level protocols, are present on the host computer. Some computer systems will be integrated into the network through the main computers containing TCP and Internet levels protocols, as well as specific network software.

The TCP specification describes its interface with higher-level protocols, which were feasible even in the case of the main computer, if the appropriate protocol of communication between the host computer and the main computer is implemented.

## 1.2 Target

The TCP protocol is required to provide a reliable service for inter-process communications in a multi-network system. The TCP protocol should be a general protocol for communication between host computers in multiple networks.

## 1.3 About this document

This document provides a description of the behavior expected of any implementation of the TCP protocol, as well as its interactions with both higher-level protocols and TCP protocols on other computers. The rest of this chapter gives a very brief overview of the protocol and its interfaces. Chapter 2 summarizes the ideological basis for the TCP protocol. Chapter 3 provides both a detailed description of the behavior required of the TCP protocol when different events appear (arrival of new segments, user request, errors, etc.) and descriptions of the details of TCP segment formats.

## 1.4 Interfaces

The TCP protocol interacts on the one hand with the user or application program, and on the other - with a lower-level protocol, such as the Internet protocol.

We explain the interface between the application process and the TCP protocol with acceptable detail. This interface consists of a set of calls that are similar to the operating system calls provided to the file management application process. For example, in this case, there are calls to open and close connections, to send and receive data on established connections. It is also expected that the TCP protocol will be able to interact asynchronously with applications. While TCP protocol developers are given considerable freedom to create interfaces that correspond to the properties of a particular operating system, any acceptable implementation requires some mandatory minimum interface functions between the TCP protocol and the user.

The interface between the TCP protocol and the lower-level protocols is much less given, except that there must be some mechanism by which the two levels can asynchronically communicate with each other. It is generally believed that the lower-level protocol sets this interface. The TCP protocol is designed to work with a very diverse environment of connected computer networks. This document assumes that the lower-level protocol is the Internet.

## 1.5 Action

As mentioned earlier, the primary goal of the TCP protocol is to provide a reliable, secure service for logical chains or connections between pairs of processes. To provide such a service based on less reliable Internet communications, the system must have the capacity to work in the following areas:

- basic data transfer
- Reliability
- Thread management
- Channel separation
- Working with connections
- Priority and safety

The main actions of the TCP protocol in each of these areas are described in the following paragraphs.

Basic data transmission

TCP protocol is able to transmit continuous streams of octets between its customers in both directions, packing a certain number of octets into segments for transmission through Internet systems.

Sometimes users need to make sure that all the data they send to TCP has already been sent. Push is defined for this purpose. To make sure that the data sent to the TCP protocol is indeed transferred, the sender indicates that it should be pushed to the recipient.

Pushing results in TCP protocol programs immediately sending and, accordingly, obtaining the remaining data. Properly pushed can be invisible to the recipient, and the push function itself may not have a record boundary marker.

The

TCP Protocol should have protection against data destruction, This is

achieved by assigning the next number to each transmitted octet, as well as the confirmation requirement (ACK) from the TCP program that receives the data.

Damages are recorded by adding a control amount to each transferable segment, checking it when receiving and then eliminating defective segments.

As long as TCP protocol programs continue to function correctly and the Internet system does not fall completely into components, forwarding errors will not affect the correct receipt of the data. The TCP protocol protects against Internet communications errors.

The
TCP Protocol allows the recipient to manage the amount of data sent to the sender.

Separating channels To allow many processes on a single computer to simultaneously use TCP level communication capabilities, the TCP protocol provides a set of addresses or ports on each host computer.

Each compound is uniquely identified by a pair of sockets. Thus, any socket can be used simultaneously in many compounds.

The port and process correlates are done by each host computer on its own. However, it is useful to link frequently used processes (such as "logger" or time-sharing service) with fixed documented sockets.

This service can then be used through known addresses. Installing and setting up port addresses for other processes can involve more dynamic mechanisms.

The connections of
the Flow and Confidence Management Mechanisms described above require TCP protocol programs to initiate and maintain certain information about the status of each data flow.

If two processes want to share information, the relevant TCP protocol programs must first establish a connection (on each side to initiate status information). At the end of the exchange, the connection must be terminated or closed to free up resources to provide to other users.

Since connections must be installed between unreliable host computers and through an unreliable Internet communication system, a mechanism is used to confirm the connection with timekeeping queue numbers to avoid mistakenly initializing connections.

Priority and security
TCP users can request priority and security for their connection.

## 2. Ideology protocol

### 2.1 Elements of the Merged Network System

The merged network environment consists of host computers included in the network, which in turn connect to each other through the gateways.

It assumes that computer networks can be either local (e.g. ETHERNET) or large networks (such as ARPANET), but in any case they are based on package switching technology. The real agents that create and consume messages that circulate on the network are the processes. Protocols of different levels in networks, gateways, and hosts support a process-related communication system that provides a two-way flow of data on logical connections between process ports.

The term package is used here in general to refer to a portion of data involved in a separate elementary act of interaction between the network and the host computer connected to it. In general, we will not be concerned with the format of the data blocks circulating on the network.

From a communications network perspective, host computers are computers connected to the network and are senders and recipients of packages. Processes are seen as active elements on host computers (according to the most general definition of processes as executable programs).

Because the process can control multiple communication flows leading from it to another process (or other processes), we postulate that each process can have a set of ports through which it communicates with the ports of other processes.

### 2.2 Action Model

The process sends data, calling the TCP protocol program and giving it data buffers as arguments. The TCP protocol packs data from these buffers into segments, and then calls the Internet module to transfer each segment to the target TCP protocol program. This recipient, in turn, places the data from the segment in the recipient's buffers and then notifies his client of the arrival of the data intended for him. TCP protocol programs place control information into segments, which they then use to verify the sequence of data transmission.

The Internet Communications model is that each TCP protocol program is connected to the Internet protocol module, which provides it with a local interface. This Internet module puts TCP segments in Internet datagrams and then sends them to another Internet module or an intermediate gateway. To transfer the datagram over the local network, it in turn is placed in a package of the appropriate type.

Package switches can carry out further packaging, fragmentation, or other operations to transfer packages to the Internet on a local network.

In the gateways between local networks, the Internet datagram is released from the local network package and explored to determine which network it should follow. The Internet datagram is then packaged in a package corresponding to the selected local network and sent to the next gateway or directly to the final recipient.

The gateway has the ability to break the Internet datagram into more small-snippet datagrams, if necessary for transmission on the next local network. To do this, the gateway itself creates a set of Internet datagrams, placing in each one fragment. In the future, the fragments can be broken again by the following gateways into even smaller parts. The format of the Internet datagram fragment is designed so that the recipient - the Internet module was able to collect fragments again in the original Internet datagrams.

The Internet module, which is the recipient, separates the segment from the datagram (after it is assembled if necessary) and then transfers it as intended for the TCP protocol program.

This simple protocol model often obscures many details. One important feature is the type of service. This trait tells the gateway (or The Internet module) to choose the service options that should be used when transferring the datagram to the next local network. The dategram's priority is among the information about the type of service. Datagrams can also carry security information in order to allow host computers and gateways operating in a multi-level security system to subject the relevant datagrams to the test.

## 2.3 Host Computer Software

The TCP protocol program is supposed to be an operating system module. Customers refer to the TCP protocol in much the same way as if they were accessing a file system. The TCP protocol itself can refer to other functions of the operating system, for example, to manage data structures. It is assumed that the actual interface with the local network is carried out by the driver of

the device. The TCP protocol does not directly address network device drivers, but instead makes a call for the Internet Protocol module, which in turn appeals to the device driver.

The TCP protocol mechanism does not preclude its implementation on the input processor. However, in this implementation, the communication protocol between input processors should provide the means to support the user interface described in this document and the TCP protocol.

### 2.4 Interfaces

For user requests to the TCP protocol, the TCP/user interface allows the connection to be opened and closed, sent and received, or the connection status. These queries are similar to other user program requests to the operating system, such as requests to open, read, and close a file.

The interface between TCP and Internet protocols supports requests for parcels and datagrams addressed to TCP modules in host computers anywhere on the Internet. Requests under consideration have arguments for specifying the address, type of service, priority, security, and other managing information.

### 2.5 Link to other protocols

The chart below illustrates the location of the TCP protocol in the protocol hierarchy

Figure 2 Relationship protocols

The TCP protocol is expected to be able to effectively support higher-level protocols. The TCP protocol should easily interact with higher-level protocols such as ARPANET Telnet or AUDIN II THP to the TCP.

### 2.6 Reliable Communications

The flow of data sent to the TCP connection is accepted by the recipient reliably and in the appropriate order.

The transfer is secure thanks to the use of confirmations and queue numbers. Conceptually, each data octet is assigned a queue number. The queue number for the first data octet in the segment is transferred along with this segment and is called the queue number for the segment. Segments also carry a

confirmation number, which is the number for the next expected octet of data being transmitted in the opposite direction. When the TCP protocol transmits a segment with data, it puts a copy of it in the re-transfer queue and starts the timer. When confirmation for this data comes, the relevant segment is removed from the queue. If the confirmation does not come before the expiration date, the segment is re-sent.

Confirming the TCP protocol does not guarantee that the data has reached the final recipient, but only that the TCP protocol program on the recipient's computer takes responsibility for it.

A thread management mechanism is used to direct the flow of data between TCP protocol programs. The TCP protocol program tells the "window" to the sending program. This window indicates the number of octets (starting with the confirmation number) that the TCP host program is ready to accept at the moment.

## 2.7 Installing the connection and cancelling it

To identify individual data streams supported by the TCP protocol, the latter identifies port identifiers. Because port identifiers are selected independently by each TCP program, they will not be unique. To ensure that addresses are unique for each TCP protocol program, we combine the Internet address and port ID that identifies the program. As a result, we get a socket, which will be unique in all local networks, united in a single whole.

The connection is completely determined by a pair of sockets at its ends. Local socket can take part in many connections with different other people's sockets. The connection can be used to transmit data in both directions, in other words, it is "completely duplex."

The TCP protocol is free to arbitrarily link ports to processes. However, in any implementation of the protocol, several fundamental concepts must be adhered to. There should be well-known sockets, which the TCP protocol associates exclusively with "appropriate" processes. We imagine how processes can "own" ports and that processes can only initiate connections from the ports they own. (From the point of view of implementing the protocol, "possession" is limited to the host computer, but we can imagine the user command on request of the port (Request Port) or the method of highlighting a group of unique ports to this process, for example by associating senior bytes in the port name with this process).

The connection is set by the OPEN team (open) made from a local port and

has an argument of someone else's socket. In response to such a request, the TCP protocol program provides the name of the local (short) from the union. By this name, the user is addressed to this connection on subsequent calls. There are a few things to keep in mind about connections.

We assume that there is a data structure called the Transmission Control Block (TCB) designed to preserve the information described above. The protocol could be implemented in such a way that the local name for the connection would be a pointer to the TCB structure of the latter. The OPEN query also indicates whether the connection is active or whether there is a passive expectation of an external connection.

Requesting a passive connection opens means that the process waits for external requests for a connection, instead of trying to install it itself. Often, a process that has made a passive discovery request will accept connection requests from any other process. In this case, someone else's socket is indicated as consisting entirely of zeros, which means uncertainty. Uncertain other people's sockets can only be present in passive opening commands.

A service process that wants to serve other processes that is unknown to it could make a request for a passive opening with an unspecified socket. In this case, the connection can be established with any process that requests connections to this local socket. This procedure will be useful if you know that the local socket you you have chosen is associated with a particular service.

Well-known sockets are a convenient mechanism of prior binding of the address of the socket with any standard service. For example, the "server for Telnet" process is tightly linked to a specific socket. Other sockets can be reserved for a file transmitter, Remote Job Entry, a text generator, an echo server, and Sink processes (the last three items are related to text processing). The socket address may be reserved for access to the "view" procedure, which could indicate the socket through which newly-formed services could be obtained. The concept of the well-known socket is part of the TCP specification, but the actual asocialization of sockets with services goes beyond this description of the protocol (see document).

Processes can passively open connections and wait for other processes to receive active opening requests and the TCP protocol will inform them of the connection. Two processes that have made active discovery requests to each other at the same time will get the correct connection. The flexibility of this

approach becomes critical with the support of distributed computing, when the components of the system interact with each other in an asynchronous way.

When sockets are selected for a local passive opening request and someone else's request for active discovery, two cases are of fundamental importance. In the first case, the local passive opening completely determines someone else's socket. At the same time, the selection should be carried out very carefully. In the second case, during the local passive opening, someone else's socket is not specified. Then in principle can be established a connection with any other people's sockets. In all other cases, the selection of sockets has partial limitations.

If the same local socket is made several waiting passive requests for opening (recorded in TCB blocks), and an active request for discovery is made from the outside, then someone else's active socket will be associated with the TCB unit, where it was pointed to this requesting connection socket. And only if such a block TCB does not exist, the choice of partner is made among the blocks TCB with an unspecified foreign socket.

The connection installation procedure uses the Syn sync control flag and exchanges messages three times. This exchange is called a three-variate confirmation.

The connection is initiated when a segment that is carrying the SYN flag is met and is waiting to be recorded in the TCB block. Both the segment and the record are created by requests for discovery from users. The local and other people's sockets are matched when the connection is initiated. The connection is recognized as established when queue numbers are synchronized in both directions between sockets.

The cancellation of the connection also includes the exchange of segments carrying the FIN flag this time.

## 2.8 Data Communication

A set of data transmitted by connection can be seen as a stream of octets. The user who sends the data indicates when the parcel requests whether the data sent in this request should be pushed immediately through the network to the recipient. The indication is carried out by the installation of the PUSH flag (pushing).

The TCP protocol program can collect data from the user and then transmit it to the network as it sees fit in the form of segments. If there is a request for

push, the protocol must transfer all the data not sent earlier. When the TCP program that receives data collides with the push flag, it should not expect to receive new data on the network until the data already available is passed on to the waiting local process.

There is no need to tie push functions to segment boundaries. The data contained in a segment may be the result of one or more parcel requests. Or a single query can generate multiple segments.

The purpose of the PUSH push and flag function is to push the data through the network from sender to recipient. The feature does not process the data itself.

There is a link between the push function and the use of data buffers in the interface between the user and the TCP protocol. Every time the data with the PUSH flag arrives in the recipient's buffer, the contents of that buffer are sent to the user for processing, even if the buffer has not been filled. If the incoming data fills the user's buffer before the push command is received, the user is sent a block of data that matches the size of the buffer. The TCP protocol also has the means to inform the recipient that from some point he is dealing with urgent data. The TCP protocol does not try to determine exactly what the user is doing with the urgent data waiting to be processed. However, it is generally assumed that the process receiving the data will work to process urgent data quickly.

## 2.9 Priority and Safety

The TCP protocol uses the type of service and the Internet protocol security option to ensure priority and security for TCP users at each connection. Not all TCP modules will necessarily operate in a tiered security system. Some modules are limited to conventional, non-specific connections, while others are limited to the first level of security and privacy. Consequently, some implementations of the TCP protocol and services for users can only use a part of the tiered security system.

TCP modules operating in a tiered security system must adequately display security and priority flags in the sent segments. Such TCP modules should also allow their customers or higher protocols, such as Telnet and THP, to specify the required level of security, privacy, and priority for installed connections.

## 2.10 Sustainability Principle

All implementations of the TCP protocol will follow the general principle of

sustainability: be conservative in your actions and grant freedom to others.

## 3 Specification for protocol functions

### 3.1 Headline format

TCP segments are transferred in the form of Internet datagrams. The dategram's headline in the Internet protocol has several information fields, including the addresses of the sender and host. The TCP headline follows the Internet with a headline and complements it with information specific to the TCP protocol.

Title TCP format

<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="16">Source Port</td><td colspan="16">Destination Port</td></tr>
<tr><td colspan="32">Sequence Number</td></tr>
<tr><td colspan="32">Acknowledgment Number</td></tr>
<tr><td colspan="4">Data Offset</td><td colspan="6">Reserved</td><td>URG</td><td>ACK</td><td>PSH</td><td>PST</td><td>SYN</td><td>FIN</td><td colspan="16">Window</td></tr>
<tr><td colspan="16">Checksum</td><td colspan="16">Urgent Pointer</td></tr>
<tr><td colspan="24">Options</td><td colspan="8">Padding</td></tr>
<tr><td colspan="32">Data</td></tr>
</table>

Fig. 3 Title TCP format

Note that each label indicates here the place for the appropriate bit.

Source Port (sender port) 16 bit
sender port number

Destination Port (recipient port) 16 bit
recipient port number

Sequence Number (queue number) 32
bits Queue number for the first data octet in this segment (except when there is a SYN sync flag).

Acknowledgment Number (confirmation number) 32
bits If the ACK control bit is installed, this box contains the next queue number, which the sender of the given dategram wishes to receive in the opposite direction.

Data Offset 4 bits Number of
32-bit words in TCP header. The TCP title always ends at the 32-bit boundary of the word, even if it contains options.

Reserved 6
bits This backup field should be filled with zeros.

Control Bits (control bits) 6
bits of this field from left to right

| | |
|---|---|
| **URG:** | The field of the urgent pointer is involved |
| **ACK:** | Confirmation field involved |
| **PSH:** | Pushing function |
| **RST:** | reboot this connection |
| **Syn:** | synchronizing queue numbers |
| **End:** | No more data to transmit |

Window (window) 16 bit
Number of data octets starting from the octet, whose number is listed in the confirmation box.

Checksum (control amount) 16
bits of the control amount field is a 16-bit supplement of the sum of all 16-bit words of the title and text. If the segment contains an odd number of octets in the title and text to be accounted for in the control amount, the last octet will be supplemented with zeros on the right in order to form a 16-bit word to provide a control amount.

The control amount, among other things, takes into account 96 bits of pseudo-header, which for internal use is put before the TCP header. This pseudo-header contains the sender's address, recipient address, protocol, and length of the TCP segment. This approach protects the TCP protocol from the segments that have been wrong in the route. This information is processed by the Internet protocol. It is transmitted through the TCP/local network interface as arguments or results of requests from the TCP protocol to the IP protocol.

| Sender's address |
|---|
| Recipient's address |

| Zeros | PTCL | TCP length |
|---|---|---|

The length of the TCP segment is the length of the TCP header and the data field measured in the octets. This is not an accurate indication of the number of octet transmitted over the network, it does not take into account 12 pseudo-header octets, but nevertheless the calculation of this parameter is still made.

Urgent Pointer (urgent pointer) 16 bit This box reports the current value of the
urgent pointer.

Options (options) the length of the variable
Option can be located at the end of the title TCP, and their length is multiple 8 bits.

Options can start from any Oct. They can have two formats:

- Single-object type of options;
- option type octet, option length octet and data octets of the option in question.

The octet lengths of the option take into account the option type octet, the length octet itself, and all the octets with the data.

Note that the list of options may be shorter than you can specify in the Data Offset field. The seat in the title left behind the "End-of-Option" option should be filled with zeros. The TCP protocol should be ready to handle all options.

The following options have now been identified:

| **Type** | **Length** | **Value** |
|---|---|---|
| 0 | - | end of the list of options |
| 1 | - | No operations |
| 2 | 4 | Maximum segment size |

Definitions of these options

End of the list of options

00000000 Type 0

This option code determines the end of the list of options. The end of the list may not match the end of the TCP title listed in the Data Offset field.

This option is used after all the options, but not after each one. The option should only be used if there is no other match with the end of the title TCP.

No operations

00000001 Type 1

This type of option can be placed between options. The goal may be to align the next option along the boundary of the word. There is no guarantee that the senders will use this option. Therefore, recipients should be prepared to handle the options even if they do not start at the boundary of the word.

Maximum segment size

| | | |
|---|---|---|
| 00000010 | 00000100 | max.segm. |
| Type 2 | Length 4 | . |

The field of these options is 16 bits. If the option is on the list, it indicates for the TCP protocol program the maximum size of the segment you receive that sent the segment with that option. This option should only be sent at the initial request for a connection (i.e. in segments with the SYN control bit installed). If this option has not been used, there are no size restrictions.

Padding (alignment) the length of the variable title alignment is done to make sure that the TCP header ends and the segment data field starts at the 32-bit boundary. The alignment is zero.

## 3.2 Terminology

Before we can discuss many of the details of the TCP protocol, we need to introduce detailed terminology. To maintain the TCP connection, you need to have several variables. We decided that these variables would be placed in the appropriate entry - transmission control unit (TCB). Among the TCB block variables are local and foreign socket numbers, security and priority flags for this connection, parcel and receipt buffer signs, current segment pointers, and re-parcel queues. In addition, TCB has several variables related to the sender and recipient queues.

Departure

**SND. UNA** parcel unconfirmed

| | |
|---|---|
| **SND. NXT** | send the next segment |
| **SND. WND** | Send a window |
| **SND. UP** | send an urgent pointer |
| **SND. WL1** | Segment queue number used to update the last window |
| **SND. WL2** | Confirmation number in segment used to update last window |
| **ISS** | initial departure queue number |

Getting

| | |
|---|---|
| **RCV. NXT** | - Get the next segment |
| **RCV. WND** | - get a window |
| **RCV. UP** | - Get an urgent pointer |
| **IRS** | - the original number of the receipt queue |

The charts below can help link some of these variables to a place in the queue

Departure queue

1. old queue numbers that have been confirmed
2. queue numbers for data that have not been confirmed
3. queue numbers allowed for the new transfer
4. the next queue numbers whose transmission is not yet allowed

Fig. 4 Departure queue

The departure window is a section of the queue marked with mark 3 in Figure 4.

Receiving queue

1. old queue numbers that have been confirmed
2. queue numbers admitted to the next stage of receipt
3. the next queue numbers that have not yet been approved

Fig. 5 Receiving queue

The receipt window is a section of the queue marked with mark 2 in figure 5.

The discussion also often uses some variables that take their value from the fields of the next segment.

Variables for the next segment

| | |
|---|---|
| **SEG. Seq** | queue number for segment |
| **SEG. Ack** | Confirmation number for segment |
| **SEG. Len** | Segment length |
| **SEG. Wnd** | window for segment |
| **SEG. UP** | Term pointer for segment |
| **SEG. Prc** | Priority for the segment |

The connection during operation passes through a series of intermediate states. These are LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT, and the fictitious state of CLOSED. The CLOSED state is fictitious because it represents a state where there is no TCP block and therefore no connection. Summary of states:

| | |
|---|---|
| **LISTEN** | Waiting for a connection request from other people's ports and TCP programs |
| **SYN-SENT** | Waiting for a doubles connection request. On our part, the request has already been made. |
| **SYN-RECEIVED** | Waiting for confirmation after the connection request has already been accepted and sent. |
| | The state of the open connection, the data taken can be |

| | |
|---|---|
| **ESTABLISHED** | presented to the user. This is the normal state of the connection in the data transfer phase. |
| **FIN-WAIT-1** | Waiting for a request from someone else's TCP program, or confirming a previously sent connection closing request. |
| **FIN-WAIT-2** | Waiting for a request to close the connection from someone else's TCP program. |
| **CLOSE-WAIT** | Waiting for a request to close the connection on the part of your client. |
| **CLOSING** | Waiting for confirmation from someone else's TCP program to request the connection to close. |
| **LAST-ACK** | Waiting for a request to close a connection previously sent to someone else's TCP program (the request also included confirmation of someone else's request to close the connection). |
| **TIME-WAIT** | Waiting for enough time to expire and you can be sure that someone else's TCP program has received confirmation of its request to close the connection. |
| **CLOSED** | The state of complete absence of connection. |

The TCP connection goes from one state to another in response to events. An event is a customer's request (opening, sending, receiving, closing, failure, receiving a connection state), receiving segments, and especially those that contain SYN, ACK, RST and FIN flags, and expiring the allotted time.

The state chart in figure 6 only illustrates the change of states, and the events that caused it, the actions that are produced, but not the addresses, the terms of the errors, and the actions not directly related to the state change.

For more information on the TCP protocol program's response to events, see the latest chapter.

Remark. This chart is only a summary, but should not be perceived as a complete specification.

## 3.3 Queue number

The fundamental idea in designing a protocol is that each octet of data sent to a TCP connection has a queue number. Since each octet is numbered, each of them can be identified. The acceptable identification mechanism is cumulative, so the identification of number X means that all octets with previous numbers have already been received. This mechanism allows you to record the appearance of duplicates in the conditions of re-transfer. The numbering of octets within the segment is carried out so that the first octet of data immediately after the title has the lowest number, and the following octets have numbers on the increasing.

It is important to remember that the number of rooms for the queue, although large, but limited. The room ranges from 0 to 2-32-1. Because the set is limited, all arithmetic operations with queue numbers must be carried out on the module 2.32. This does not mean that any pre-arithmetic check of the queue numbers for the hit in the range of 2-32-1 to 0. In the work with modular arithmetic there are some subtleties, so you need to carefully program the comparison of such large values. This means "less or equal" (on the module of 2/32).

The TCP protocol should provide the following types of comparisons for queue numbers:

(a) whether the confirmation number is the queue number for octets that have already been sent but have not yet been confirmed;

(b) whether all octets in the segment have received confirmation of their numbers (i.e. whether this segment should be removed from the queue for a second parcel);

(c) whether the segment that came contains the numbers we expect (i.e. "override" whether this segment is the recipient's window).

In response to the data, the TCP protocol will receive confirmation. To work with the evidence you have received, you need to be able to make a comparison for

**SND. UNA** the oldest of the unconfirmed rooms,

**SND. NXT** the next number in line waiting for the parcel,

| | |
|---|---|
| **SEG. Ack** | confirmation numbers received from someone else's host of the TCP program (the next queue number expected by someone else's TCP program) |
| **SEG. Seq** | First Octet queue numbers in the segment, |
| **SEG. Len** | number of octets in the segment data field (given SYN and FIN), |
| **SEG. SEQ+SEG. LEN-1** | the last octet queue numbers from the segment. |

The new confirmation (called "confirmation of acceptability") is a confirmation of the feasibility of
SND inequalities. UNA < SEG. ACK =< SND. NXT

A segment from the re-parcel queue receives full confirmation if the amount of its number in the queue and the length of the data field is less, or equal to the confirmation number from the selected segment.

When you receive the data, you need to make comparison operations for the following values:

| | |
|---|---|
| **RCV. NXT** | the next number from the queue of incoming segments, as well as the left or bottom boundary of the receipt window, |
| **RCV. NXT+RCV. WND-1** | The queue number of the last segment expected in the incoming segment, as well as the right or upper boundary of the receipt window, |
| **SEG. Seq** | the first number in the queue brought by the incoming segment, |
| **SEG. SEQ+SEG. LEN-1** | the last number in the queue brought by the incoming segment. |

It is believed that the segment overlaps part of the permitted numbers in the receiving queue if
the RCV. NXT =< SEG. SEQ < RCV. NXT+RCV. WND or
RCV. NXT =< SEG. SEQ+SEG. LEN-1 < RCV. NXT+RCV. WND

The first part of this text looks at whether the beginning of the segment hits the window, and the second part - whether the back of the segment falls into the window. If any part of the test is running, the segment ends up in the

window.

The reality is a little more complicated. By selecting a zero-length window or zero-length segment, we get four eligibility options for incoming segments

| Segment length | receipt window | Test |
|---|---|---|
| 0 | 0 | SEG. SEQ = RCV. Nxt |
| 0 | >0 | RCV. NXT =< SEG. SEQ < RCV. NXT+RCV. WND |
| >0 | 0 | Unacceptable |
| >0 | >0 | RCV. NXT =< SEG. SEQ < RCV. NXT+RCV. WND<br>или<br>RCV. NXT =< SEG. SEQ+SEG. LEN-1 < RCV. NXT+RCV. WND |
| | | |

Note that when the window is zero, no segments will be accepted except for the ACK segments. Thus, the TCP protocol can set a zero receipt window when data is transmitted and confirmed. However, even when the receipt window is zero, the TCP protocol program is required to process the RST and URG fields of all incoming segments.

We have benefited from this numbering scheme in that it also allows protection for certain managing information. This is achieved indirectly by inserting some control flags into the queue, so that they can be re-sent and confirmed without failure (i.e. one or more copies will be used).

The information management is not really in the segment data field. Therefore, we must adopt the rules for indirectly assigning queue numbers to management segments. SYN and FIN are the only control signals acceptable for such protection, and they are only used when the connection is opened and closed. For the purpose of maintaining a priority, the SYN signal is seen as standing in front of the first valid data octet in the segment where both were placed. At the same time, FIN is considered to be standing after the last real data octet in the segment. Segment length (SEG. LEN) takes into account both the data and the queue numbers set aside for management. When SYN is present, THE SEG value. The SE's corresponds to the queue number for the SYN signal.

## Choosing the original number for the queue

The protocol does not restrict the reuse of a particular connection. The connection is set by selecting a pair of segments. New requests for a connection will be seen as re-implementation of this connection. This approach raises the following problem: "How does the TCP protocol distinguish segment duplicates left over from the previous implementation of that connection?"

To avoid a failure, we should avoid using segments of this connection implementation when the network still has the same queue numbers left over from the previous connection implementation. We want to insure against this, even if the TCP protocol program fails and loses all information about the queue numbers it uses. When creating new connections, the original queue numbers generator (ISN) is used, which selects the new 32 bit ISN values. The generator is tied to a 32-bit watch (probably fictitious), whose value changes every 4 microseconds. Thus, the full cycle of ISN hours is approximately 4.55 hours. Since we believe that segments will exist online for no more than the maximum lifetime of the segment (Maximum Segment Lifetime - MSL), and that MSL is less than 4.55 hours, we can reasonably believe that THE numbers will be unique.

For each connection, there is a number in the departure queue and a number in the receiving queue. The initial number in the departure queue (ISS) is selected by the TCP program that sends data in that queue, and the initial number in the receiving queue (IRS) is determined at the time the connection is established.

When or when a connection is established or initiated, both TCP programs must synchronize the original queue numbers with each other. This is done by exchanging segments that set connections that carry a control bit called "SYN" (for synchronize) that carry the source numbers for queues. For short, the segments that carry the SYN bit are also called SYN segments. Therefore, solving the problem requires an acceptable mechanism for selecting the original queue number and the few confirmation signals when exchanging ISN numbers.

Synchronization requires each party participating in the connection to send its own original queue number, and also receive confirmation from a partner. Each party must also receive the initial queue number from the partner and send a confirmation.

1) A --> B сигнал SYN: мой номер очереди X

2) A <-- B сигнал ACK: ваш номер очереди X

3) A <-- B сигнал SYN: мой номер очереди Y

4) A --> B сигнал ACK: ваш номер очереди Y

Because steps 2 and 3 can be combined into one message, the latter is called confirmation of three paths (three messages).

Confirmation of three paths is necessary because queue numbers are not tied to some global clock of this computer network, and TCP programs may have different mechanisms for selecting ISN numbers. The recipient of the first SYN signal cannot know if the signal has lingered and is already out of date, or if it is not, even if the recipient does not remember the last queue number used by this connection (which is also not always possible). So he should ask the sender to check this SYN signal. Confirmation of the three paths and the benefits of the timekeeping scheme are discussed in the article.

A Period of Silence

To make sure that TCP does not create a queue-carrying segment that is already used by the old segment that is still "walking" on the network, TCP must remain silent for at least the duration of the segment 'MSL' until it assigns any queue numbers when it starts or recovers from a failure, when memory records for the previous queue numbers have been lost. This MSL specification takes 2 minutes.

This value has been selected by the developers and can be changed if the practice shows the need for it. Note that if the protocol program is in some ways re-initiated, but the memory left the previous queue numbers, there is no need in anticipation; you should only make sure that the new queue numbers are larger than they used before.

The concept of a period of silence in the TCP protocol

This specification stipulates that computers that fail with the loss of all information about the latest queue numbers transmitted through open (i.e. not closed special teams) connections will refrain from sending any TCP segments for at least the maximum life of the segment (Maximum Segment Lifetime - MSL) in the Internet, whose part is the host. Subsequent paragraphs provide an explanation for this specification. Some implementations of the TCP protocol may violate the Silence Agreement, at

the risk that some recipients in the Internet will perceive old data as new, or new data will be discarded as duplicates of actually outdated segments.

Protocol programs use new queue numbers whenever a segment is formed and placed on the host in the network departure queue. The duplicate-fixing procedure and the queue algorithm in the TCP protocol rely on the unique link of segment data to the queue. The queue numbers do not have time to pass the entire range in 2-32 values, before the associated data from the sent segment will receive confirmation from the recipient, and all copies of the duplicates of the segment in this segment will leave the Internet network. Without this condition, it can be assumed that two separate TCP segments may be assigned the same or overlapping numbers, which will cause a problem for the recipient in determining which data is new and which are out of date. Recall that each segment is tied to both the many queues following each other, and to the data octets available in this segment.

Under normal conditions, TCP monitors the current queue number to be sent, as well as the oldest of the expected confirmations, avoiding the misuse of the queue number before confirmation is received from earlier use of the same number. This alone does not guarantee that the old data - duplicates will be removed from the network, so the queue numbers are made very large to reduce the likelihood that the duplicates wandering around the network will cause a failure on arrival. At the exchange rate of 2 megabytes/sec, the queue in the 2/32 octet is 4.5 hours long. Because the maximum life time of a segment on the network is unlikely to exceed a few dozen seconds, this is considered sufficient protection for future networks, even if data speeds increase to tens of megabits/sec.

At 100 megabits/sec, one cycle of use of all queue numbers is 5.4 minutes, which may be quite small but still acceptable.

However, the basic mechanism for registering takes and maintaining queues can be reversed if the TCP program that sends data does not have a place in memory to store the numbers in the queue that it last used for a specific connection. For example, if the TCP program starts with a queue number of 0 when creating all connections, then when the TCP crashes and re-starts, the TCP program can re-form the previous connection (perhaps after half-open connection analysis) and send packages on it whose queue numbers are completely identical or only partially overlap with the numbers that are still present on the network and sent by the previous implementation of the same

connection. In the absence of information about queue numbers previously used to transmit information on this particular connection, the TCP protocol specification recommends that the sender refrain from sending segments on this with a connection, allowing segments running into the network of the old connection to leave the system.

Even host computers that can track current time and use it when selecting the original queue numbers (i.e. even if time is used to select the original number when implementing each new connection) are not protected from this problem.

As an example, let's say that the connection is open with the starting number of the S queue. The original queue number will be S1'ISN (t), and this is the last queue number in the old connection implementation! If the recovery happens quickly enough, the old duplicates created with a queue number close to the S1 number can be retrieved by their recipient and processed as if they were new packages in the new connection implementation.

The problem is that the host- the recipient may not know how long it has been in a glitch and whether there are still old duplicates left over from previous connections implementations.

One way to solve this problem is to have a deliberate delay in sending segments over the course of MSL time after recovery in the wake of the crash - a period of silence specification. Host computers that prefer to avoid a pause run the risk of getting the problem of old and new packages colliding on a recipient who wishes not to resort to a period of silence.

Protocol implementations can offer TCP users the option of choosing between waiting in connections after a failure and not following a period of silence for any connection. Obviously, for those programs where the user has chosen the standby mode, in fact, this is not necessary after the computer has been turned off for at least MSL seconds.

So, each sent segment takes one or more numbers in the waiting queue. The numbers assigned to the segment are "busy" or "in the works" until the MSL seconds expire. If you crash, a certain place in the queue for a certain time continues to be occupied by octets from the last sent segment. If a new connection is created soon and it uses any numbers from the queue when they are still using a segment from the previous implementation of the connection, then this area of overlapping queue numbers may cause problems for the recipient.

## 3.4 Connecting

"Three-way confirmation" is a procedure used to connect. This procedure is usually initiated by the TCP protocol program in response to a request from another TCP program. This procedure also works if two TCP programs initiate it at the same time. When an initialization attempt is made at both ends at the same time, each TCP protocol program receives a "SYN" segment that is not validated for syn already sent. Of course, the arrival of old SYN duplicates can impress the recipient as if the connection is being opened at the same time. Correct use of "reset" segments can prevent the ambiguity of such situations.

Here are a few examples of the initialization of the connections: While these examples do not show the synchronization of the connection with the segments carrying the data, this is perfectly valid because the TCP program receiving the segments will not transmit the data to its customer until the correctness of the data becomes apparent (i.e., the data must be "stored" by the user until the connection is established).

The simplest three-way confirmation procedure is shown below in Figure 7. The drawings should be interpreted as follows. "XXX" indicates a segment that is lost or rejected. Comments appear in brackets.

Here, the "states" of the TCP protocol program correspond to the moment immediately after the segment is sent or received (the contents of this segment are shown in the middle column of each line). The content of the segment is in abbreviated form and is a queue number, control flags and an ACK field. Other segment fields, such as the window, length, and data field, remain outside our interest.

| . | TCP A | . | | | TCP B |
|---|---|---|---|---|---|
| 1. | CLOSED | . | | | LISTEN |
| 2. | SYN-SENT | --> | <SEQ=100><CTL=SYN> | --> | SYN-RECEIVED |
| 3. | ESTABLISHED | <-- | <SEQ=300><ACK=101><CTL=SYN,ACK> | <-- | SYN-RECEIVED |
| 4. | ESTABLISHED | --> | <SEQ=101><ACK=301><CTL=ACK> | --> | ESTABLISHED |
| | | | | | |

| | | | | | |
|---|---|---|---|---|---|
| 5. | ESTABLISHED | --> | <SEQ=101><ACK=301><br><CTL=ACK><DATA> | --> | ESTABLISHED |

The basic procedure of confirming three ways to synchronize the Figure 7 connection.

On Line 2 of Figure 7, the TCP A program starts by sending a SYN signal, showing that it will use queue numbers starting at number 100. On line 3, the TCB B program sends a SYN signal, as well as confirmation that the SYN signal from the TCP A program has been received. Note that the confirmation box informs that the TCP B program is currently waiting to receive the number 101. The latter also confirms that the SYN signal has already taken a place in the queue at number 100.

On line 4 for TCP B sent in line 3 of the SYN signal, TCP A responds with an empty segment containing the ACK signal. In line 5, TCP A transmits a portion of data over the network. Note that the segment in line 5 has the same queue number as the segment in line 4, because the ACK signal in the queue does not take up space (if it was not, we should have obtained confirmation - ACK - for the confirmation itself!).

Figure 8 shows the same initialization with minor complications. Each TCP program takes turns with closed, SYN-SENT, SYN-RECIEVED and finally, ESTABLISHED.

| . | TCP A | . | | | TCP B |
|---|---|---|---|---|---|
| 1. | CLOSED | . | | | CLOSED |
| 2. | SYN-SENT | --> | <SEQ=100><CTL=SYN> | ... | |
| 3. | SYN-RECEIVED | <-- | <SEQ=300><CTL=SYN> | <-- | SYN-SENT |
| 4. | . | ... | <SEQ=100><CTL=SYN> | --> | SYN-RECEIVED |
| 5. | SYN-RECEIVED | --> | <SEQ=100><ACK=301><br><CTL=SYN,ACK> | ... | . |
| 6. | ESTABLISHED | <-- | <SEQ=300><ACK=101><br><CTL=SYN,ACK> | <-- | SYN-RECEIVED |
| 7. | . | ... | <SEQ=101><ACK=301><br><CTL=ACK> | --> | ESTABLISHED |

Simultaneous Connection Synchronization
Figure 8

The main reason for using the confirmation of the three ways is to try to prevent the occurrence of failures when receiving old duplicates that initiate the connection. To work with the confirmation of three ways, a special control message - reboot (reset) was invented.

If the TCP program receiving the signal is out of sync (i.e. in SYN-SENT, SYN-RECEIVED), it returns the LISTEN signal, indicating that it has received an acceptable restart signal. If the TCP program is in one of the synchronized states (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), it will eliminate the connection and inform its client about it. We will discuss this situation below when considering "half-open" connections.

| . | TCP A | . | | | TCP B |
|---|---|---|---|---|---|
| 1. | CLOSED | . | | | LISTEN |
| 2. | SYN-SENT | --> | <SEQ=100><CTL=SYN> | ... | . |
| 3. | (duplicate) | ... | <SEQ=90><CTL=SYN> | --> | SYN-RECEIVED |
| 4. | SYN-SENT | <-- | <SEQ=300><ACK=91><CTL=SYN,ACK> | <-- | SYN-RECEIVED |
| 5. | SYN-SENT | --> | <SEQ=91><CTL=RST> | --> | LISTEN |
| 6. | . | ... | <SEQ=100><CTL=SYN> | --> | SYN-RECEIVED |
| 7. | SYN-SENT | <-- | <SEQ=400><ACK=101><CTL=SYN,ACK> | <-- | SYN-RECEIVED |
| 8. | ESTABLISHED | --> | <SEQ=101><ACK=401><CTL=ACK> | --> | ESTABLISHED |

Getting an old SYN signal duplicate
Figure 9

As a simple example, let's look at the situation with the old duplicates in Figure 9. On line 3, an old SYN duplicate reaches the TCP B program.

TCP A detects an erroneous value in the ACK field and therefore returns the RST (reboot) signal. At the same time, the value of the SE's field is chosen in

such a way as to make the segment believable. About a gram of TCP B when the RST signal is received goes to LISTEN. When the SYN signal, valid rather than out-of date, reaches TCP B on line 6, the synchronization process is normal. If the SYN signal on line 6 reaches the TCP B program before the RST signal, there may be a more complex exchange combination with the RST in both directions.

Half-open connections and other anomalies

An established connection is called "half open" if one of the TCP programs shuts down or abandons the connection. And she did it on her end without warning her partner. This can also occur if synchronization at the ends of the line is disrupted due to a failure that resulted in memory loss. If these connections attempt to send data in a direction, the connection is automatically restarted. However, it is assumed that half-open connections are rare and the recovery procedure is used in the network very moderately.

If the connection at the end of A is considered to be non-existent and the customer at the end of B tries to send the data, it will result in the TCP program at the end of B receiving a reset notification. This message shows the TCP program at the end of B that something is wrong and is being asked to eliminate that connection.

Suppose two customers at points A and B communicate with each other, and at that point there is a collapse that leads to memory loss in the TCP program at the end of A. Depending on the operating system that serves the TCP A program, there is likely to be some mechanism to correct the error. When the TCP A program is re-launched, it will probably re-start from the beginning or with the instructions to overcome the failure. As a result, Program A will probably try to open (OPEN) a connection or send information (SEND) through a connection that it believes is open. Once the TCP A program fails, the user will try to re-open the connection.

| . | TCP A | . | | | TCP B |
|---|---|---|---|---|---|
| 1. | Failed | . | (package number 300, receipt - 100) | | |
| 2. | CLOSED | . | | | ESTABLISHED |
| 3. | SYN-SENT | --> | <SEQ=400><CTL=SYN> | --> | (??) |
| | | | | | |

| | | | | | |
|---|---|---|---|---|---|
| 4. | (!!) | <-- | <SEQ=300><ACK=100><br><CTL=SYN> | <-- | ESTABLISHED |
| 5. | SYN-SENT | --> | <SEQ=100><CTL=RST> | --> | (elimination!!) |
| 6. | SYN-SENT | . | | | CLOSED |
| 7. | SYN-SENT | --> | <SEQ=400><CTL=SYN> | --> | . |

Detection of a half-open link
Figure 10

When the SYN signal reaches the synchronized TCP B program on line 3, and the segment that has arrived is outside the window, TCP B responds with confirmation, showing the queue number it wants to receive (ACK-100). The TCP A program, seeing that the segment on line 4 did not confirm the information it sent, records the lack of synchronization and sends a reset signal (RST) because it is found that the connection is half open. On line 5, TCP B eliminates the connection. The TCP A program will continue to attempt to establish a connection.

The problem is now solved by simply confirming three paths (Figure 7).

Another interesting story is if the TCP A program fails and TCP B, believing it is in a state of synchronization, is trying to send data. This situation is shown in Figure 11.

In this case, the data sent by TCP B and that came to TCP A (line 2). will be rejected because the connection they use does not exist. Based on this, TCP A sends an RST signal. Once the RST signal is accepted by TCP B, it will be reviewed and the connection used before will be eliminated.

| . | TCP A | . | | | TCP B |
|---|---|---|---|---|---|
| 1. | Failed | . | (package number 300, receipt - 100) | | |
| 2. | (??) | <-- | <SEQ=300><ACK=100><br><DATA=10><CTL=SYN> | <-- | ESTABLISHED |
| 3. | . | --> | <SEQ=100><CTL=RST> | --> | (elimination!!) |

The active side leads to the discovery of a
half-open link
Figure 11

Figure 12 shows that two TCP programs - A and B - are waiting for the SYN signal.

| . | TCP A | . | | | TCP B |
|---|---|---|---|---|---|
| 1. | LISTEN | . | | | LISTEN |
| 2. | . | ... | <SEQ=Z><CTL=SYN> | --> | SYN-RECEIVED |
| 3. | (??) | <-- | <SEQ=X><ACK=Z+1><br><CTL=SYN,ACK> | <-- | SYN-RECEIVED |
| 4. | . | --> | <SEQ=Z+1><CTL=RST> | --> | Return to LISTEN |
| 5. | LISTEN | . | | | LISTEN |

Old SYN signal duplicate initiates reboot
on two passive sockets
Figure 12

There can be many other variations that can be explained by the following rules for creating and processing RST signals.

Create a reboot signal

According to the main rule, the restart signal (RST) must be sent whenever a segment arrives that is obviously not intended for that connection. If it is not clear whether this case is the case, you should refrain from rebooting.

You can identify three groups of states to connect:

1. If the connection does not exist (CLOSED), the restart signal is sent in response to any segment that has come, except for the oncoming restart signal. In particular, SYN signals addressed to a non-existent connection are rejected in this way.

   If the incoming segment has a flag in the ACK field, the segment with the reboot signal receives a number for the queue from the first segment ACK field. Otherwise, the segment with the restart signal has

a zero queue number and a value in the ACK field equal to the amount of the queue number of the queue and its same length. The connection remains closed.

2. If the connection is in any non-synchronized state (LISTEN, SYN-SENT, SYN-RECEIVED), if any confirmation of the incoming segment has not yet been sent (the segment carries an unacceptable value in the ACK field) or the incoming segment has a level of security/closure that does not correspond to the level and protection of that connection, a reset signal is sent.

   If our SYN signal has not been confirmed and the priority level of the upcoming segment is higher than the requested level, either the local priority level will be increased (if this is acceptable to the user and the system), or a reboot signal will be sent. Or, if the priority level of the coming segment is less than requested, the processing will continue further, as if the level was the same (if someone else's TCP program can not raise the level of priority to ours, it will be noted in the next segment sent by it, then the connection will be closed). If our SYN signal has received confirmation (perhaps in the segment that came to us), then the level of priority of the coming segment should exactly correspond to the places at a different level. If the last condition is not met, a restart signal is sent.

   If the incoming segment carries an ACK signal, the restart signal will have a queue number corresponding to the ACK signal number in the incoming segment. Otherwise, the restart signal will have a zero queue number, and the ACK signal will have a number equal to the number of the segment that has come and its length. The connection doesn't change its state.

3. If the connection is in a synchronized state (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST ACK, TIME-WAIT), then any inappropriate segment (not entering the queue number window, carrying the wrong confirmation number) should result in a segment with an empty confirmation box containing the current number in the parcel queue, as well as confirmation indicating the next number. The connection remains in its former state.

   If the incoming segment has a level of protection, isolation, or priority that does not correspond to the local connection level, a restart signal is

sent and the connection goes to CLOSED. The reboot signal has a queue number corresponding to the ACK signal number in the queue.

Processing a signal to restart

For all states except SYN-SENT, all segments with the Reset Signal (RST) are tested by se. A restart signal is recognized if its queue number hits the window. In the SYN SENT (RST signal received in response to the sending of the SYN trigger signal), the RST signal is recognized if the ACK field confirms the previously made sending of the SYN signal.

The recipient of the RST signal first checks it, and only then changes its condition. If the recipient was in LISTEN, they ignore the signal. If the recipient was in SYN-RECEIVED, it returns to LISTEN. In other cases, the recipient will eliminate the connection and go into CLOSED state. If the recipient is in a different state, he will liquidate the connection and notify his client before moving to CLOSED.

## 3.5 Closing connection

The closed state means "I don't have data to transmit." Of course, closing a full duplex connection is the subject of many interpretations, as it is not obvious how to interpret the side receiving the information in the connection. We decided to interpret CLOSE in a simplified manner. A CUSTOMER in CLOSE can still receive information (RECEIVE) until the partner also tells them that they are moving into CLOSE. In this way, the client can gracefully complete the work at their end of the connection. The TCP protocol program is guaranteed to receive all buffers with information sent before the connection was closed. Therefore, a customer who does not expect information from the connection should only wait for a report of the successful closure of this connection, which means that all the data is received by the TCP program that receives the information. Customers should keep the connection information they already have closed for reading until the TCP protocol program informs them that such information is no longer available.

Three cases are of particular importance:

1) the customer initiates the closure of the connection by giving the command of their TCP protocol program.

2) closing the connection begins with the partner sending a control signal to the FIN. .

3) both customers command to close at the same time.

Case 1 A local customer initiates a closure
In this case, a segment with a FIN signal is created and placed in a segment queue, After that, the TCP program will no longer accept any commands from this client to send data on a closed connection, and it goes into FIN-WAIT-1.

When the partner's TCP program confirms that the FIN signal will be received and sends its FIN signal here, the local program can confirm that the latter is receiving it. Note that the TCP program receiving the FIN signal will confirm it, but will not send its own FIN signal until its client also closes the connections.

Case 2 The TCP program receives a FIN signal from the network.
If an unclaimed FIN signal comes from the network, the TCP program that receives it can confirm that the connection is closed and notify its customer that the connection is closed.

Case 3 Both customers close the connection simultaneously. The simultaneous closure of the
connection by customers at both ends results in the exchange of segments with the FIN signal.

| . | TCP A | . | | | TCP B |
|---|---|---|---|---|---|
| 1. | ESTABLISHED | . | | | ESTABLISHED |
| 2. | (Close)<br>FIN-WAIT-1 | --> | <SEQ=100><ACK=300><br><CTL=FIN,ACK> | --> | CLOSE-WAIT |
| 3. | FIN-WAIT-2 | <-- | <SEQ=300><ACK=101><br><CTL=ACK> | <-- | CLOSE-WAIT |
| 4. | (Close)<br>TIME-WAIT | <-- | <SEQ=300><ACK=101><br><CTL=FIN,ACK> | <-- | LAST-ACK |
| 5. | TIME-WAIT | --> | <SEQ=101><ACK=301><br><CTL=ACK> | --> | CLOSED |
| 6. | (2 MSL)<br>CLOSED | . | | | |

Normal Closing Procedure
Figure 13

| . | TCP A | . | | | TCP B |
|---|---|---|---|---|---|
| 1. | ESTABLISHED | . | | | ESTABLISHED |
| 2. | (Close) | . | | | (Close) |
| . | FIN-WAIT-1 | --> | <SEQ=100><ACK=300><br><CTL=FIN,ACK> | ... | FIN-WAIT-1 |
| . | | <-- | <SEQ=300><ACK=100><br><CTL=FIN,ACK> | <-- | . |
| . | | ... | <SEQ=100><ACK=300><br><CTL=FIN,ACK> | --> | . |
| 3. | CLOSING | --> | <SEQ=101><ACK=301><br><CTL=ACK> | ... | CLOSING |
| . | | <-- | <SEQ=301><ACK=101><br><CTL=ACK> | <-- | . |
| . | | ... | <SEQ=101><ACK=301><br><CTL=ACK> | --> | . |
| 4. | TIME-WAIT | . | | | TIME-WAIT |
| . | (2 MSL) | . | | | (2 MSL) |
| . | CLOSED | . | | | CLOSED |

The simultaneous closing of the connection at both ends
of Figure 14

### 3.6 Priority and Safety

The idea of this part of the protocol is that the connection can only exist between ports of equal importance without danger and closure, and the priority level should correspond to the highest of the values requested by the two ports.

The priority and security parameters used in the TCP protocol are exactly in line with the Internet (IP) protocol. In the TCP protocol specification, the concept of "security" should include the parameters used in IP, which include actual security, privacy, group definition for users, and support for various restrictions.

Attempting to connect to unacceptable security/privacy values, or with a low priority, should be rejected by sending a restart signal. Refusal to connect due to insufficient priority can only be granted after receiving confirmation on the

SYN signal.

Note that TCP modules that only work with the default priority will nevertheless check the priority of the incoming segments and, if necessary, raise the priority level applicable to that particular connection.

Security options can even be used by TCP programs that do not support the security system (the security field values would apply to non-classified data). Thus, hosts that do not support the security system should be prepared to accept security settings, although they do not need to send such parameters.

## 3.7 Data transfer

If the connection is established, the data is transmitted through segment exchange. Because segments may be lost as a result of errors (such as control errors) or network overload, the TCP program uses a re-parcel mechanism (after a certain amount of time) to make sure each segment is received. The chapter on queue numbers discussed how the TCP program in segments checks the number of queues and confirmation numbers for their correctness.

The sender of the data using the SND variable value. NXT tracks the next number in the queue to be sent. Recipient of the data using the RCV variable. NXT tracks the next number it expects to arrive. In the SND variable. The data sender puts the value of the oldest number that was sent but has not yet been confirmed. If the data flow had instantly dried up and all the data sent had been confirmed, then all of these variables would have contained the same value.

When an information sender creates and sends a segment, it increases the value of the SND variable. Nxt. The addressee increases the value of the RCV variable when the segment is received. NXT and sends confirmation. When the TCP program that sends the data is confirmed, it increases the SND value. Una. The difference in the values of these variables is a measure that characterizes the latency of segments in the network. The amount to be used to increment the value of these variables every time is the length of the data field in the segment. Note that because the connections are in the ESTABLISHED state, all segments, in addition to the data itself, must carry some information about the confirmation of previously sent segments.

The user's request to close the connection (CLOSE) involves using the push function, which is carried out using the CONTROL flag OF FIN to the incoming segment.

Re-sending time

Due to the fickleness of the networks that make up the unified system and the large number of customers using TCP connections, there is a need for a dynamic definition of the re-delivery time control time. One of the procedures for determining this time is given as an illustration.

An example of the time-measuring procedure for re-sending segments First, measure the time that has passed between sending a data octet with a certain number in the queue, and receiving confirmation indicating, among others, to that number (the segments sent are not required to correspond to the resulting segments).

Измеренная временная задержка - это время обращения (Round Trip Time - RTT). Следующий шаг - вычисление усредненного времени обращения (Smoothed Round Trip Time - SRTT):
SRTT = (ALPHA * RSTT) + ( (1-ALPHA) * RTT)

The re-parcel time (Retransmission Timeout - RTO) is then determined by using the found value:
RTO, min, LBOUND, (BETA)

UBOUND - Upper limit of time limit (e.g. 1 min),

LBOUND - lower limit (e.g. 1 sec).

ALPHA - smoothing factor (e.g. 0.8 to 0.9),

BETA - delay change factor (e.g. 1.3 to 2.0).

Transfer of urgent information

The TCP Protocol Urgent Transfer Mechanism is designed to enable the customer sending the data to accept some urgent information, as well as to allow the TCP program, which receives the data, to inform its client when all the information is available to date.

This mechanism allows you to mark a point in the data stream as the end of the emergency information block. When in the TCP program, accepting the data, this point will be ahead of the number indicator in the receiving queue (RCV). NXT), this TCP program should give the command to its client to go into "urgent mode." When the number in the queue for receiving catches up with an urgent pointer in the data stream, the TCP program should give the customer command to come to "normal mode." If the urgent pointer changes

its position when the client is in "urgent mode," the latter will not know about it.

This method uses the flag field of urgency, which is present in all transferable segments. The unit in the URG control flag field means that the field of urgency is involved. To get a pointer to this field in the data stream, you need to supplement it with the number of the segment in question in the queue. The absence of the URG flag means that the sender does not have urgent data that has not been sent.

When specifying urgency, the customer must also send at least one octet of data. If the customer who puts the data additionally orders the push function, the transmission of urgent information to the waiting process will be accelerated many times.

## Window control

The window sent to each segment indicates the range of queue numbers that the window sender (a.s. the recipient of the data) is ready to accept at this time. It is assumed that this mechanism is related to the current location in the data buffer.

The large window's indication stimulates the transmission. But if you have more data than TCP can accept, the data will be discarded. This will result in unnecessary information forwarding and unnecessary increased workload on the TCP network and program. Indicating a small window can limit data transfer to the speed at which it takes to travel on the network after each segment is sent.

Such protocol mechanisms allow the TCP program to declare a large window, but subsequently declare a window of much smaller size and not accept such a large amount of data. Such so-called window cut looks rather disconcerting. The sustainability principle dictates that the TCP protocol program does not shorten the window itself, but is prepared for such action by another TCP program.

The TCP program that sends data should be prepared to accept at least one octet of new data from the customer and transmit at least one octet of new data over the network, even if the departure window is zero. The TCP program must periodically re-send data to another TCP program, even if the window is zero. In the case of a zero window, it is recommended that you use a two-minute re-parcel interval. Such repeated parcels are important to ensure that if one of the two TCP programs has a zero window, the opening of this

window will be guaranteed to be communicated to the partner.

When the TCP program that receives the data has a zero window, but a segment comes to it, the program must send confirmation and specify what number in the queue it is waiting for and what window it currently has (window zero).

The TCP program packs the data intended for forwarding to segments that fill the current window. However, it cannot repackage existing segments in the queue for a second parcel. Such repackage, although not necessary, but can be useful.

In a one-way data connection, the window information will be transmitted to the confirmation segments, and they will all have the same queue number. Therefore, there will be no way to restore their order when receiving. This is not a serious problem, but it may inadvertently lead to information about the window from an out-of-date message. To avoid such a problem, the window drop-out should be taken from the segments with the largest number in the queue (these are segments whose confirmation number is larger or equal to the largest number previously received).

The window management procedure has a significant impact on communications characteristics. Further comments include suggestions to the developers.

### Wishes for window management

A very small window results in very small segments of data being transmitted, while the best mode is when using larger segments.

To avoid using small windows, the data recipient is asked to delay changing the window until the available space is X percent of the maximum possible memory for this connection (where X can be between 20 and 40).

Another tip not to send small segments is to keep the sender from rushing to send the data until the window is big enough. But if a customer commands to use a push feature, the data should be sent immediately, even if it's done by a small segment.

Note that in order to avoid unnecessary re-shipments, there is no need to delay the delivery of confirmations. The strategy may be to send confirmation when you receive a small segment (without updating the window information), then send another confirmation with new information about the window if the latter expands.

A segment sent to check a zero window can trigger a breakdown of the transmitted data into more and more small segments. If a segment containing a single data octet and sent to check the zero window is received, it will take one octet in the current window. If TCP simply sends as much data as it can, whenever the window is not zero, the data transmitted will be broken down into large and small segments. After some time, the occasional pause in the recipient's release of the window will break down large segments into numerous and not-so-large pairs. So, after some time, data transmission will be mostly small segments.

The proposal is for the implementation of the TCP protocol to actively try to combine small windows into larger windows, as window controls seek to introduce many small windows into the simplest implementations of the protocol.

## 3.8 Interfaces

Of course, two interfaces are affected here. The interface is the TCP client/protocol and the interface of the TCP/lower level protocol. We have a more carefully designed model for the first one. But this does not address the interface with the lower-level protocol module, as it is done in the specification of the latter. For a case where the bottom level in the interface is the IP protocol, we will only note some of the acceptable parameters used by the TCP protocol.

Customer/TCP interface

The following functional description of the client commands sent to the TCP protocol program is, at best, speculative, as each operating system will have its own characteristics. Therefore, we need to warn readers that different implementations of the TCP protocol may have a different interface with the customer. However, all implementations of the TCP protocol must provide some minimal service to ensure that they all adhere to a single hierarchy of protocol. This chapter describes the functional interface that is required for all implementations of the TCP protocol.

TCP Customer Team

The following paragraphs functionally characterize the TCP client interface/protocol. Call notation is like noting most function call butting or notation in high-level languages, but that doesn't mean trap calls are wrong (e.g. SVC, UUO, EMT).

The client commands below identify the main operations that the TCP

protocol program must perform to support interflow communications. Individual protocol implementations must define their own specific format and can provide combinations or sets of basic functions for single calls. In particular, some implementations can automatically open the connection (OPEN) as soon as the customer gives the first parcel command (SEND) or receipt (RECEIVE).

In order to maintain the interface between processes, TCP must not only accept commands, but also return some information to serviced processes. This information consists of:

a) general information about the connection (i.e. interruptions, partner connection closing, communication management with a non-predetermined alien socket).

b) responding to specific customer commands indicating success or different types of errors.

Opening Team
OPEN Format
(their port, foreign port, active/passive, control time, priority)

We believe that the local TCP program is responsible for identifying the processes you serve and will verify that processes that wish to access this connection are relevant. Depending on the implementation of the TCP protocol, either the TCP program or the lower-level protocol (such as IP) will create the sender's address, or rather the identifiers for the local network and the TCP interface. Such forethought is the result of security consideration, namely that none of the TCP programs can disguise itself as any other, etc.

If the flag is active/passive, it means that the "listening" (LISTEN) of the external connection signal has been requested. Passive discovery can either give a full description of someone else's socket, with which this connection should be established, or not give any indication about someone else's socket, which should give a signal. Passive opening of a connection with a well-defined foreign socket can become an active open connection at any time if a command is given to send the data (SEND). A transmission control unit (TCB) is created and partially filled with information from the OPEN team settings.

In the case of the Active Opening Team (OPEN), the TCP protocol will immediately start the connection synchronization procedure (more precisely,

installing).

The time control time, if present among the function parameters, allows the client to set a elimination time control for all data sent on behalf of the TCP protocol. If any data does not reach its target during this time of control, TCP will eliminate the connection. Currently, the generally accepted control time is five minutes.

The TCP protocol program or any component of the operating system will verify the customer's rights to open a connection that has priority and security prioritization and security. If priority or security/privacy is not specified, a CALL call using the default settings should be used.

The TCP protocol program will only accept incoming requests if they have the same level of security/closure. The priority of requests should be equal to or exceed the priority specified in the opening team (OPEN).

If the connection's priority is greater than the value specified in the CALL request, priority is taken from the query that has come and becomes a permanent feature of the connection. Developers can re-instruct customers to negotiate a connection priority. For example, a customer might specify a priority that should be assigned to the connection. In another example, any attempt to increase the priority of the connection should receive customer approval.

At the end of the operation, the TCP protocol returns the local name to the customer for an open connection. Subsequently, the name of the compound can be used as a short designation for the connection, identifiable by the pair of the local socket, someone else's socket.

The team on the send of data
FORMAT
SEND (local connection name, buffer address, number of bytes with data, push flag, flag of urgency (control time)

This command results in the data contained in the customer's buffer being transferred to the specified connection. If the connection was not open by this time, the SEND command is erroneous. Some implementations of the TCP protocol may allow customers to start communicating immediately with the SEND command. At the same time, the OPEN team should be carried out automatically. If the process that gave the command to the parcel is not authorized to use the connection, the team returns the error to the customer.

If the push flag (PUSH) is installed, the data must be transmitted to the

intended destination with the appropriate message, and the PUSH bit must be placed on the last of the segments created in the TCP buffer.

If the host who builds the gram date fragments cannot complete its work at the allotted time due to the error of the fragment, it throws out the datagram and can send a message that exceeded the time. This message should not be sent at all unless a zero snippet of the datagram has been received.

From the gateway comes a message with the code 0, and from the host computer - a message with code 1.

| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Error message format with settings | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Type | | | | | | | | Code | | | | | | | | Checksum | | | | | | | | | | | | | | | |
| Pointer | | | | | | | | Not used | | | | | | | | | | | | | | | | | | | | | | | |
| Internet headline q 64 bits of data from the original dategram | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

IP protocol fields:

The

recipient's address and the sender's address are entered into the gram returning the error.

If the time target was specified, the current value set by the client will be changed in this connection.

In simplest implementations, the SEND command does not betray control of the program that caused it until the transmission is completed or until the control time is exceeded. However, this simple method can result in a lock (for example, when both sides at the ends of the connection try to execute the SEND command first, and only then - RECEIVE) and poor performance. Therefore, this approach is not recommended. In a more complex implementation, the return from SEND function is immediate, allowing the process to run in parallel with the input/inference into the computer network. And even more than that - allows you to perform several SEND commands at the same time. Multiple SEND commands are processed on a "first come first served" basis, so the TCP protocol will have a queue that cannot be serviced instantly.

Indirectly, we have assumed the asynchronousness of the interface with the client, in which the SEND command causes the appearance of a special kind

of signal or pseudo-interruption from the TCP maintenance program. The alternative is to return immediately from the team. For example, SEND commands can return immediate confirmation from the local system, even if the sent segment has not received confirmation from someone else's TCP program. We are optimistic about the possible success of this operation. If we make a mistake, the connection will be severed anyway after the control time. Some asynchronous signals will still remain in the implementations of this type of TCP (synchronous) protocol, but they will refer to the connection itself, not to specific segments or buffers.

In order for the process to distinguish between error messages and successful reports from different SEND commands, it may be convenient to return a buffer address in response to a SEND request along with a coded response. The signals between the client and the TCP protocol are discussed below and determine the information that should be returned to the process that sent the team.

Receiving team

RECEIVE format
(local connection name, buffer address, byte counter) - byte counter, flag of urgency, push flag

This command places the information in a buffer associated with a specific connection. If the team is not preceded by an OPEN command, or if the call-making process is not authorized to use the connection, the error is returned.

In the simplest implementation of the protocol, management should not be transferred to the challenged program until either the buffer is filled or there is an error. However, this scheme is largely subject to blocking.

A more complex implementation could allow several RECEIVE commands to be put forward at a time. These requests will be executed as data segments become available. This strategy allows you to increase bandwidth by using a more advanced scheme (possibly asynchronous), as well as alerting the program that a PUSH push signal has been received or a buffer is filled.

If you have enough data to fill the buffer before you receive a PUSH push push signal, the PUSH flag will not be installed in response to RECEIVE. The buffer will hold as much data as its capacity allows. If the PUSH signal is detected before the buffer is filled, the buffer will be returned partially filled and with the PUSH signal.

If urgent data is found, the customer will be alerted by the TCP protocol

program as soon as they arrive. The customer receiving the data must go to "urgent mode" on this signal. If the URGENT urgency flag is installed, additional urgent data will remain unreceived. If the URGENT flag is dropped, the receipt request will refund all urgent data and the client may be exempt from the "urgent mode." Note that the data following the urgency index (non-urgent data) cannot be delivered to the customer in the same buffer with the previous urgent data, unless the client has clearly defined the boundary.

To distinguish between multiple RECEIVED commands made and to monitor buffers, the code returned to the customer is accompanied by both a buffer pointer and the amount of data actually received.

Other implements in the RECEIVE team may themselves set aside a buffer to accommodate the data they receive, or the TCP protocol program can use a cyclical buffer at the same time as the customer.

Connection closing team

CLOSE format
(local connection name)

This command leads to the closure of the connection. If the connection is not open or the process that has given the command is not authorized to use this connection, it is returned with an error communication. It is assumed that closing the connection will be a slow operation in the sense that the remaining SEND commands will transmit data for some time (and even, if necessary, reapply) as far as thread management will allow, and the ordered work will not be done. In this way, you can make multiple SEND parcel commands and then close the connection with the CLOSE command, making sure that the data sent will reach the recipient. Obviously, customers should continue to give commands to receive data from already closed connections, because someone else's program will still try to send the remaining data to it.

So the CLOSE command should be interpreted as "I don't have any more data to send" rather than "I don't want to get anything anymore." It may happen (if the protocol at the customer level is not fully thought out) that the party closing the connection will not be able to get rid of all their data before the control time expires. In this case, the CLOSE command is broadcast to ABORT, and the TCP program refuses to connect.

The customer can command CLOSE at any time at his own discretion, as well as in response to various messages from the TCP protocol (for example,

when the connection is closed from the other side, the transfer control time is exceeded, the recipient is not available).

Because closing a connection requires communication with someone else's TCP program, connections can be closed for a short time. Trying to re-open the connection before TCP responds to the CLOSE command will result in a return to the error message call.

The closing team involves the operation of pushing the data through the connection.

Connection status team

Status format

(local connection name) -This is a client team that is based on a specific implementation.

- local juice
- someone else's juice
- Local link name
- receipt box
- Departure window
- Connection status
- number of buffers waiting to be confirmed
- number of buffers waiting for data
- urgency status
- Priorities
- security/closure
- time control

Depending on the state of the connection or the specifics of the protocol implementation, some of this information may not be available or meaningless. If the process that you call does not have the right to use this connection, the error message is rotated. This approach does not allow un empowered processes to obtain information about the connection.

Unit elimination team

ABORT format

(local connection name)

This command eliminates all unfinished parcel operations and data. The TCB

unit is eliminated and a special RESET message must be sent to the TCP program at the other end of the connection. Depending on the implementation of the protocol, customers can receive a liquidation message in response to each outstanding request for a parcel or receipt of data. Alternatively, customers can simply get confirmation from the ABORT team instead.

Messages to the customer from the TCP program

It is assumed that the operating system's service functions provide the TCP program with the means to send asynchronous signal. When the TCP program sends a signal to the client's program, a certain amount of information is also transmitted to the customer himself. This is often done in the form of error messages. In other cases, information related to the completion of the parcel and receipt of the data, as well as the execution of other client commands, will be provided.

The following information is provided:

| | |
|---|---|
| local connection name | Always |
| report line | Always |
| Buffer address | sending and receiving data |
| number of bytes (the meter received bytes) | Getting data |
| flag pushing | Getting data |
| flag of urgency | Getting data |

TCP interface with lower-level protocol

The TCP protocol program for the actual delivery of information over the network, as well as to receive it makes requests to the module of the protocol of the lower level. One example of this implementation is the ARPA Internetwork system, where the lower-level module is the Internet Protocol (IP).

If the lower-level protocol is IP, it asks for the required type of service and the life of the data on the network as call arguments. The TCP protocol program uses the following values for these parameters:

Type of service - priority: normal, delayed: normal,
bandwidth: normal, reliable: normal,
i.e. 000,00000

Lifetime- One Minute, or 00111100

Note that the maximum life time of the segment is two minutes. Here we clearly determine that the segment should be eliminated if it does not reach the recipient in the Internet within one minute.

If the IP protocol (or any other protocol with the same functions) is located below and the routing procedure is applied, the interface must allow the transmission of route information. This is especially important because the recipient and recipient addresses that are included in the TCP protocol control amount will correspond to the actual sender of the data and the most recent recipient. It is also important to maintain the return route to respond to state queries.

Any lower-level protocol will be required to provide the sender's address, recipient address, protocol field, and some TCP message length procedure required for both IP protocol service functions and the TCP protocol itself.

## 3.9 Event Processing

The processing described in this chapter is just an example of one possible implementation of the protocol. Other implementations may have slightly different processing procedures, but they should differ from those described in this chapter only in detail, but not in substance.

The TCP protocol program can be seen as a response to events. These events can be broken down into three categories: customer requests, segment arrival, expiration of control time. This chapter describes the activities in the TCP protocol in response to each of these events. In many cases, the processing required depends on the state of the connection. Events that can occur:

Customer teams

- to open the connection
- to send data
- to get the data
- closing the connection
- to eliminate the connection
- to determine the status of the connection

Receiving
Segments Expiration Of Control Time

- For customer action
- to re-send
- Waiting

The TCP and customer interface model is that the client commands are executed immediately, and the likely deferred report is provided through an event or pseudo-break mechanism. In the further description, the concept of "signal" may indicate some basis for sending a deferred report.

Error messages are provided in the form of text lines. For example, customer commands addressed to non-existent connections will receive the message "error: connection not open."

Please note that in the future all the arithmetic for queue numbers, confirmation numbers, windows, etc. is carried out on the module 2-32, which corresponds to the size of many queue numbers.

Note, too, that "zlt;" means "less or equal" (on the module 2/32).

To understand the meaning of processing incoming segments, it would be natural to imagine that they are first checked for the correctness of the queue number (i.e. that their information falls within the "receiving window" range among the expected queue numbers) and that they will, in general, be queued and processed accordingly by their numbers.

If some segments overlap with other previously received segments, we design a segment that contains only truly new data, and then adjust the header fields accordingly.

Note that the state of the TCP protocol program remains unchanged when processing events unless the opposite is specified specifically.

Open request

Closed (i.e. there is no TCB)
Create a new transmission control unit (TCB) to store connection status information. Customers may be given the opportunity to check security and priority settings if they do not receive an "error: precedence not allowed" or "error: security/compartment not allowed" in the event of a passive opening.

If the opening is active and indicates someone else's socket, then send a segment with a SYN signal. The starting number for the departure queue is chosen. The SYN segment and signal sent are in the form of a zlt.-ISS." Set the SND variable. UNA's ISS, and SND. NXT in ISS.1. Go to SYN-SENT.

Return control of the process that caused the team in question.

If the customer who made the request did not gain access to the socket specified in the request, return the message "error: connection illegal for this process." If there is no place in your computer's memory to create a new connection, return the message "error: insufficient resources."

LISTEN State

If there is activation and is indicated by someone else's socket, then change the state of the connection from passive to active, choose ISS. Send a segment with the SYN signal, bring in the SND. UNA is the value of ISS, and in SND. NXT ISS+1. Go to THE SYN-SEND state.

The data listed in the SEND command can be sent in the same segment with the SYN signal, or may be placed in a transfer queue that can be made after the transition to an ESTABLISHED state. If the team is asked to use a bit of urgency, the data segments should be sent as a result of the execution. If there is no place in the queue of orders for shipment, the result will be the answer "error: insufficient resources." If someone else's socket is not specified, then return the message "error: foreign socket unspecified"

State

SYN-SENT
SYN-RECEIVED
ESTABLISHED
FIN-WAIT-1
FIN-WAIT-2
CLOSE-WAIT
CLOSING
LAST-ACK
TIME-WAIT

return the message "error: connection already exist" in response to the opening team

SEND request

CLOSED (e.g. no TCB block)

If a customer does not have access to such a connection, return the message

"error: connection illegal for this process."

LISTEN State

If the socket is specified, then change the state of the connection from passive to active, choose the ISS number. Send a segment with the SYN signal, set SND. UNA's ISS, and SND. NXT in ISS.1. Set a new state OF SYN-SENT. Data from the SEND call can be sent along with the SYN signal, or can be placed in a queue and sent after the established state has been established.

If the team is asked to use a bit of urgency, it must be transferred along with the segment of data that occurs when that command is being performed. If there is no room in the queue for the request, return the message "error: insufficient resources." If someone else's socket is not specified, then return "error: foreign socket unspecified."

SYN-SENT

State SYN-RECEIVING

Put the data in line to send after the established state is established.

ESTABLISHED

STATE CLOSE-WAIT

Segment the data buffer and send it back (confirmation value q RCV. NXT). If you don't have enough memory space to place this buffer, just return the message "error: insufficient resources."

If the flag of urgency is installed, it will be in the SND. UP value SND. NXT-1 and set an urgency index to the outgoing segments.

State

FIN-WAIT-1

FIN-WAIT-2

CLOSING

LAST-ACK

TIME-WAIT

Return the message "error: connection closing" and do not comply with the customer's request.

RECEIVE request

CLOSED (for example, there is no TCB block)

If a customer does not have access to such a connection, return the message "error: connection illegal for this process."

State

LISTEN

SYN-SENT

SYN-RECEIVED

Place the request in the service queue after the ESTABISHED state has been established. If there is no place in the queue for this, return the message "error: sufficient resources."

State

ESTABLISHED

FIN-WAIT-1

FIN-WAIT-2

If there is not enough data in the incoming segments to fulfill this request, place the latter in the service queue. If there is no place in the queue to place the RECEIVE request, return the message "error: insufficient resources."

Collect data from the incoming segments in the receiving buffer and then pass it on to the customer. Install the "detected push" (PUSH) flag if it occurs.

If the data currently being transmitted to the customer was preceded by RCV. UP, then notify the customer of the presence of urgent data. When the TCP protocol assumes responsibility for the customer's receipt of the data, it actually means sharing information with the sender in the form of confirmations. The formation of such confirmation is discussed below when considering the algorithm for processing the incoming segment.

CLOSE-WAIT State

Because the partner at the other end of the connection has already sent a FIN signal, THE RECEIVE commands should receive data already available in the system, not just those that have already been passed on to the customer.

State

CLOSING

LAST-ACK
TIME-WAIT

Вернуть сообщение "error connection closing".

CLOSE request

CLOSED (e.g. no TCB block)
If a customer does not have access to such a connection, return the message "error: connection illegal for this process."

LISTEN
Any remaining unsatisfied RECEIVE requests will be completed with the message "error: closing."

SYN-SENT
to erase the TCB block and return the "error closing" message to any remaining SEND or RECEIVE requests.

If no SEND requests are made and there is no data waiting to be sent, then form a FIN segment and send it, then go to the FIN-WAIT-1 state. Otherwise, put the data in the queue for review after the ESTABLISHED state has been established.

THE state of ESTABLISHED
Put the request in a queue waiting for all the data of the previous commands to be segmented.

Fin-WAIT-1 Condition
FIN-WAIT-2
Strictly speaking, this situation is erroneous and should result in the customer receiving an "error: connection closing" message.

CLOSE-WAIT State
Put this query in a queue until all previous SEND requests are placed in segments.

State

CLOSING
LAST-ACK
TIME-WAIT

Возвратить сообщение "error: connection closing".

ABORT Request

CLOSED (e.g. no TCB block)
If a customer does not have access to such a connection, return the message "error: connection illegal for this process."

LISTEN
Any remaining RECEIVED requests must end with the return of the message "error: connection reset."

SYN-SENT
All in-line SEND and RECEIVE requests must receive a "connection reset" message, erase the TCB block, go to CLOSED, and return control to the customer.

State

SYN-RECEIVED

ESTABLISHED

FIN-WAIT-1

FIN-WAIT-2

CLOSE-WAIT

Послать сегмент перезагрузки
<SEQ=SND.NXT><CTL=RST>

All in-line SEND and RECEIVED requests should receive a "connection reset" message. All segments in the transmission queue (with the exception of the newly formed RST signal) and the re-mailing queue must be eliminated. Erase the TCB block, go to a CLOSED state, regain customer control.

State

CLOSING

LAST-ACK

TIME-WAIT

Return the "ok" message and erase the TCB block, go to CLOSED, return control to the client.

STATUS request

CLOSED (e.g. no TCB block)
If a customer does not have access to such a connection, return the message "error: connection illegal for this process."

LISTEN
Return the message "state'LISTEN" and the pointer to the TCB block.

STATE SYN-SENT
Return the message "state'SYN-SENT" and the pointer to the TCB block.

SYN-RECEIVED
Return the message "state'SYNRECEIVED" and the pointer to the TCB block.

ESTABLISHED
Return the message "state'ESTABLISHED" and the pointer to the TCB block.

State FIN-WAIT-1
Return the message "state-FIN-WAIT-1" and a pointer to the TCB block.

State FIN-WAIT-2
Return the message "state-FIN-WAIT-2" and a pointer to the TCB block.

CLOSE-WAIT Return
the message "state-CLOSE-WAIT" and the pointer to the TCB block.

CLOSE
Return the message "state'CLOSING" and the pointer to the TCB block.

STATE LAST-ACK
Return the message "state'LAST-ACK" and pointer to the TCB block.

TIME-WAIT Return
the message "state'TIME-WAIT" and the pointer to the TCB block.

The arrival of segments

If the closed connection state (for example, there is no TCB block), all data from the specified segment will be thrown away. The segment that came with the RST signal will be eliminated. A segment that does not contain an RST signal will trigger an RST signal in response. Confirmation and queue number will be chosen in such a way as to make the reboot sequence acceptable to the TCP program that sent the segment that caused the reaction.

Если бит ACK сброшен, то используется номер очереди нуль:
<SEQ=0><ACK=SEG. SEQ+SEG. LEN><CTL=RST,ACK>

If the ACK is installed, it is
zlt;SE'SEG. Return control of the
aborted program.

If you have a LISTEN connection state,

check the presence of the RST signal first.
The RST signal that came along with the segment should be ignored, and management should be returned to the interrupted program.

Second, check for the presence of ACK. Any confirmation is a mistake if it has come to the end of a
connection still in listen.
Return control of the
aborted program.

Third, check for syn signal presence If a
SYN bit is installed, then check security.
If the value of
se. PRC is smaller than TCB. PRC, then move on to the next point.

Install RCV. NXT in SEG. THE IRS install in SEG. And the rest of the texts and control functions are put in a queue for later processing. Select the value for ISS and send a confirmation segment in the form
of zlt;ISS." NXT'lt'SYN, ACK'gt;
Variable SND. NXT is set in ISS'1, and SND. UNA at ISS. Set a new SYN-RECEIVED state to connect. Note that SYN-RECEIVED will process all incoming data and control teams (along with SYN), but will no longer process SYN and ACK signals as before. If the LISTEN state is not fully formulated (for example, it is not specified exhaustively someone else's socket, then at this point should be determined the fields of the block TCB, which remain unfilled.

Fourth, look for the rest of the management commands in the coming segment, as well as the data itself. Any segments with other control commands or completed text (but not SYN signal) must receive confirmation from the local TCP program, and thus be discarded while working with confirmation. An incoming segment with an RST signal may not be correct because it cannot be a response to the information given by this connection implementation. So you're unlikely to get this signal, but if it does, throw away the incoming segment and return control of the interrupted program.

If the STATE of the SYN-SENT connection

First, check out the ACK bit If the ACK bit is on display, then
in case the SEG. ACK is or SEG. ACK > SND. NXT,
send a reboot signal tothe
zlt's SEG. If you don't exhibit a bit of
RST. If it's all on display, then doing nothing to throw out the incoming segment and regain control of the aborted program.) Eliminate the segment, return the controls.
If SND. UNA =< SEG. ACK =< SND. NXT, the confirmation received in the segment becomes acceptable.

Second, check out the bit of RST.
In the event that the RST bit is exposed
If you expected to receive a segment with confirmation, then give the customer an ad "error: connection reset", return the segment, go to CLOSED, remove the TCB block, and finally return the management of the aborted program.

Thirdly, check safety and priority levels
If the security/privacy values in the upcoming segment do not match exactly those in the TCB block, then send a reboot signal, or rather if there is an ACK, then send a
zlt;SE'SEG. Otherwise,
senda
zlt.s.a. SEQ+SEG. LEN><CTL=RST,ACK>

If there is an ACK, the priority in the next segment should coincide with the priority specified in the TCB block.
ACK><CTL=RST>

If there is no ACK, you can do this
if the segment prioritizes above the priority in the TCB block, then if the client and the system allow it, increase the priority in the TCB block to the value specified in the segment.
If you don't allow to increase priority, send a reboot signal:
SEQ+SEG. If the segment has a
smaller priority than the TCB block, just move on to the next point of analysis.

Fourth, check the installation of the SYN bit. This step should only be done if

the ACK bit is not causing problems, or if it is not installed, the segment also does not contain an RST signal.

If the SYN bit is set and the priority security/privacy settings are acceptable, then the SEG variable. NXT write down the seg value. The IRS set the SEG equal. Seq.

Snd. UNA should be upgraded to SEG. ACK (if there is ACK), and any segments in the queue for a second parcel that have thus received confirmation must be removed.

If SND. UNA iss (our SYN signal will receive confirmation), then install to connect the state of ECTABLISHED and form the ACK segment of the zlt;SE-SND. NXT><ACK=RCV. NXT'lt'lt;CTL'ACK'gt; and send it. If there are other commands or even some text in the data field in this segment, then continue processing from the sixth stage below, where the URG bit is checked.

If you have commands or text in the data box in the intended segment, put them in the queue for processing after the ESTABLISH status is established.

Fifth, if the SYN or RST bits are installed, you will throw out the incoming segment and return control of the interrupted program. If the connection was in a state not described above at the time of the segment's arrival,

First, check the
Status queue number

SYN-RECEIVED
ESTABLISHED
FIN-WAIT-1
FIN-WAIT-2
CLOSE-WAIT
CLOSING
LAST-ACK
TIME-WAIT

Segments are processed one by one. When the segment is received, the test is

first performed to remove old duplicates, but further processing is done in order of SEG numbers. Seq. If the content of the segment overlaps the boundary between old and yet new information, only new data should be processed.

The eligibility test of the incoming segment considers four options:

| Segment length | Receiving window | Text |
|---|---|---|
| 0 | 0 | SEG. SEQ = RCV. Nxt |
| 0 | >0 | RCV. NXT =< SEG. SEQ < RCV.NXT+RCV.WND |
| >0 | 0 | Segment is unacceptable |
| >0 | >0 | RCV. NXT =< SEG. SEQ < RCV.NXT+RCV.WND или RCV. NXT =< SEG. SEQ+SEG. LEN-1 < RCV.NXT+RCV.WND |

If RCV. WND zero, no segments will be acceptable, but the acceptability of getting the correct signals of ACK, URG and RST should be specifically specified.

If the incoming segment is not acceptable, then send its confirmation in return. NXT><ACK=RCV. Ifthe RST bit is not installed (if it is still on display, then throw out the incoming segment and return control of the aborted program.)

After sending confirmation, eliminate the unaccepted segment and return control of the interrupted program.

In the future, the recommendations are based on the assumption that the incoming segment is idealized and begins with the RSV. NXT that doesn't go out the window. Real segments can be adjusted to this assumption by discarding any parts of them that go beyond the window (including even SYN and FIN signals), and further processing them only if the segment then starts with the RCV number. Nxt. Segments with large numbers in the queue are saved for processing in the future.

Second, check out the bit of RST.
SYN-RECEIVED If
the RST bit is set, then do the following:
If the connection was initiated by the OPEN passive opening team (for

example, the transition from LISTEN state was made), then return the connection to LISTEN, There is no need to inform the user about this connection.

State

ESTABLISHED

FIN-WAIT-1

FIN-WAIT-2

CLOSE-WAIT

If you have a bit of RST installed, all receiving RECEIVE and SEND requests waiting to be processed should receive a "reset" response.

Remove all segments from the queues. Customers must receive a non-binding general message called "connection reset." Go to CLOSED, erase the TCB block and return control of the aborted program.

State

CLOSING

LAST-ACK

TIME-WAIT

If you put up a bit of RST, then go to CLOSED, remove the TCB block and return control of the interrupted program.

Third, check the importance of security and priority in the upcoming segment

SYN-RECIEVED If
security/closure and priority in the next segment do not match the security/privacy and priority specified in the TCB block, send a reboot signal and return control to the aborted program.

ESTABLISHED If
security/closure and priority in the queue do not match exactly with the safety/privacy and priority specified in the TCB block, then send a reset signal that still remains unserved RECEIVED and SEND requests must receive a "reset" response.

Note that this check is put after checking the number in the queue in order to prevent the rupture of this connection, initiated by obtaining a segment that

remained from the previous connection with other security and priority that once existed between these ports.

Fourth, check out a bit of SYN
States

SYN-RECEIVED

ESTABLISHED

FIN-WAIT-1

FIN-WAIT-2

CLOSE-WAIT

CLOSING

LAST-ACK

TIME-WAIT

If SYN is within the window, send a reboot signal. Any RECEIVE and SEND teams waiting to be processed should get a "reset" response, remove all segments from the queues, and the customer must receive a non-binding general message "reset connection." Go to CLOSED, remove the TCB block, return control of the interrupted program.

If SYN is outside the window, it should not reach this point. In the first stage (check of the queue number) confirmation had to be sent. Fifth, check the ACK box If the ACK bit is not installed, then the segment is eliminated, and the controls transfer the interrupted program.

If a bit of ACK is set

by THE SYN-RECEIVED State.
If SND. UNA =< SEG. ACK =< SND. NXT, then go to the ESTABLISHED state and continue processing.

If the confirmation in the segment was not acceptable, then to form a segment with a reboot signal

of the SEG. ACK and
send it.

Established
State If SND. UNA < SEG. ACK =< SND. NXT, then set in SND. UNA value from SEG. Ack. Any segments from the queue for a second parcel that have received confirmation are removed. Customers should receive positive

feedback on buffers that have been sent by the SEND team and have now been fully confirmed (for example, a team to send a buffer with data should end with the message "ok").

If the confirmation is a duplicate (SEG. ACK < SND. UNA), it can be ignored.

If the ACK message confirms something that has not yet been sent (SEG. ACK > SND. NXT), then send ACK, eliminate the segment and return control of the aborted program.

If SND. UNA < SEG. ACK =< SND. NXT, you should update the box for the parcel.

If (SND. WL1 < SEG. or (SND. WL1 = SEG. SND and SND. WL2 =< SEG. ACK) then install SND. WND according to THE SEG. WND, SND. WL1 in SEG. SEQ, SND. WL2 in SEG. Ack.

Note that in SND. WND recorded a bias relative to the SND. UNA, xnj d SND. WL1 recorded a queue number for the last segment used to update the SND. WND as well as that in SND. WL2 recorded a confirmation number from the last segment used to update the SND. Wnd. At the same time, the check protects against the use of outdated segments to update the window.

The FIN-WAIT-1 condition
is the same as when processing in the case of established state, but if our FIN signal has now received confirmation, then go to FIN-WAIT-2 and continue processing in this state.

The FIN-WAIT-2 state
is the same as when processing for an ESTABLISHED state case, but if the re-parcel queue is empty, the customer's command to close the CLOSE connection can get confirmation ("ok"), but does not remove the TCB block.

THE CLOSE-WAIT condition
is the same treatment as it was for the ESTABLISHED standing case.

THE state of CLOSING
is the same as in the case of ESTABLISHED state processing, but if ACK in the coming segment confirms our FIN signal, then go to TIME-WAIT.

The last-ACK state
is the only thing that can happen in this state is getting confirmation on the FIN signal.

THE state of TIME-WAIT

The only thing that can happen in this state is the re-transmission of someone else's FIN signal.

Confirm the signal and re-start after the 2MSL control time.

Sixth, check out a bit of URG
Status

ESTABLISHED

FIN-WAIT-1

FIN-WAIT-2

If a bit of URG is installed in the RCV. UP bring max (RCV. UP, SEG. UP), and let the customer know that there is urgent data at the other end of the connection if the urgent pointer (RCV. UP) stands in front of the data. If the customer has already been notified of this chain of urgent data (or if it is still in "urgency mode"), you should not remind him of it again.

State

CLOSE-WAIT

CLOSING

LAST-ACK

TIME-WAIT

This should not happen because it was obtained from the other end of the connection. Ignore the bit of URG.

Seventh, process data from their
State segment

ESTABLISHED

FIN-WAIT-1

FIN-WAIT-2

Once we were able to ESTABLISHED, it became possible to accept the text to be placed in the receipt buffers specified by the client. Text from segments can be rolled into buffers until either the buffer is filled or the segment becomes empty. If the segment is empty and carries the PUSH push flag, when the buffer is returned, notify the customer that a PUSH signal has been

received.

When the TCP protocol assumes responsibility for the customer's receipt of the data, it must also confirm that the data has been received.

Once the TCP program takes over the management of the data flow, it puts the value of the RCV. NXT in front of the data spectrum, and RCV. WND sets the buffer capacity accordingly at the moment. In general, the value of RCV. NXT and RCV. WND should not be adjusted to a smaller side.

Please note that window management is based on the assumptions in Chapter 3.7.

Послать подтверждение в виде
<SEQ=SND. NXT><ACK=RCV. NXT><CTL=ACK>

This confirmation should be added to the segment leaving the other end of the connection and, if possible, without unnecessary delay.

State

CLOSE-WAIT

CLOSING

LAST-ACK

TIME-WAIT

This should not happen because it was obtained from the other end of the connection. Ignore the text in the segment.

Eighth, check the FIN bit
not to handle the FIN signal if the state is CLOSED, LISTEN or SYN-SENT because there is no way to check the SEG. THROW out the incoming segment and regain control of the interrupted program.

If the FIN bit is installed, then give the customer a "connection closing" signal, with the same message to complete all pending RECEIVED requests, to install the RCV. NXT in front of the place in the FIN signal queue, send for the last confirmation. Note that the FIN signal involves pushing (PUSH) texts in all segments not yet received by the customer.

State

SYN-RECEIVED

ESTABLISHED

Go to CLOSE-WAIT

State FIN-WAIT-1
If our FIN signal has received confirmation (possibly in the same segment), then go to TIME-WAIT, to run a control timer, disable all other timers.

State FIN-WAIT-2
Go to TIME-WAIT.

CLOSE-WAIT State
To Stay close-WAIT

State OF CLOSING
Stay in the State of CLOSING

State of LAST-ACK
Stay in Last-ACK State

TIME-WAIT State
Stay in TIME-WAIT.

Expiration of the control time for the client
If the control time has expired, no matter what state the program is in, remove all queues, give the customer a general signal "error: connection aborted due to user timeout", the same signal to give all waiting requests, eliminate the TCB block, go to THE closed state, return control of the interrupted program.

Expiration of the control time for re-sending
In whatever state the program is, if for the segment in the queue for the second parcel expired the control time, send this segment again, but out of turn, to re-initiate the timer of the re-parcel, to return control of the interrupted program.

Expiration of the time-WAIT time control period If the
time-WAIT status has expired, then eliminate the connection and the TCB block, go to CLOSED, return control of the aborted program.

## Dictionary

1822
BBN Report 1822, "The Specification of the Interconnection of a Host and an IMP". Спецификация интерфейса между хост-компьютером и сетью ARPANET.

ACK

Control Bit (confirmation) that does not occupy any place in the queue.

ARPANET message
parcel unit between host computer and IMP network ARPANET. Maximum size of such a unit is about 1012 octets (8096 bits).

ARPANET package
Package Parcel Unit between different IMP within the ARPANET network. The maximum size of such a unit is about 126 octets (1008 bits).

Connecting
the Logical Way for Communications, defined by a pair of sockets.

Datagram
Message sent through a computer communication system with package communication.

Recipient Address
Address is usually network and host computer identifiers

FIN
Control bit (ultimate) occupying one place in the queue and indicating that the TCP protocol program will no longer send data or any commands to the queue to take place.

Fragment
Part of the logical data unit. In particular, a fragment of the Internet are part of the Internet datagram.

FTP
File Transfer Protocol

Headline
Control information at the beginning of a message, segment, snippet, package, or data block

In particular, it is the sender and recipient of messages in terms of communication network.

Identification
of the Internet Protocol Field.

IMP
Interface Messaging Processor, Package Switch in APRANET Network.

Internet
address of the sender or recipient at the host level

Internet datagram
Block, transmitted between the Internet protocol module and the program of higher protocol, equipped with the Internet header.

Internet snippets
Are Part of the Data from the Internet Datagram, which has acquired its own Internet headline.

IP
протокол Internet

THE IRS
Initial Number is in the receiving queue.

ISN
The original queue number.

ISS
Initial number in the parcel queue.

In particular, the ARPANET network reports some control information in the host-IMP interface format.

The remaining queue
is the next number in the queue to be confirmed by the TCP program receiving the data (or otherwise the smallest number in the queue that has not yet received its confirmation).

Local
data block package transmitted to local network

The implementation module
is usually a program, of some kind of protocol or other procedure.

MSL
Maximum segment lifespan.

Octet
Byte, consisting of eight bits

Options
field options may contain multiple options, each option may have a length of several octets.

A
data package with a title that in turn may or may not be logically completed.

Port
Part of the Socket, indicating a logical input or output channel for the process dealing with the data.

The process
is some program.

PUSH
Is a control bit that does not require a place in the queue and indicates that this segment contains data that should be "pushed" to the customer-recipient.

RCV.NXT
Next number in the receiving queue

Rcv. UP
Urgent pointer to receive

RCV.WND
Window Receipt

Next number in the receiving queue
Is the next number in the queue that wants to get the local TCP protocol program

Receiving Window
This concept characterizes the numbers in the queue that should be beamed by the local TCP protocol program. NXT to RCV. NXT+RCV. WND-1, carry data and management commands that should be taken into account. Segments whose numbers in the queue in no way fall into this range are perceived as duplicates and eliminated.

RST
Checkpoint (reboot bit) that does not take a place in the queue and indicates that the recipient of this bit must eliminate the connection without any additional action.

RTP
Real Time Protocol. Protocol for the transmission of critical information between host computers.

Seg. ACK
Segment Confirmation

Seg. LEN

Segment Length

Seg. PRC
Priority Value in The Segment

Seg. Se'
queue number for the segment

Seg. UP
Field Urgent Pointer for Segment

Seg. WND
Box Box in Segment

In particular, the TCP segment is a block of data that is transmitted between a pair of TCP modules.

Confirmation of the
No.

The length of
the segment Place in the queue, which is occupied by the data of this segment (taking into account also all the commands, which also give a place in the queue).

Segment number in the
queue Value in the no-room field at the next segment

The number in the departure queue number is the next queue number for the local TCP protocol program that sends the data and uses those numbers to control the connection.

The Window parcel window
is a set of numbers from the queue that someone else's TCP protocol program wants to receive. NXT to SND. UNA+SND. WND-1 (of course, this implies re-sending those data whose numbers lie between the SND. UNA and SND. NXT).

SND.NXT
Queue for parcel

Snd. UNA
queue of data not yet sent

Snd. UP
Urgent pointer in line for parcel

SND.WL1
Segment queue number in the latest updated window

SND.WL2
Segment Confirmation Number in the latest updated window

SND.WND
Package Window

Socket
Address, which specifically includes a port ID.

The sender's
address is an address of departure, usually consisting of network identifiers and a host computer.

SYN
Control bit in the incoming segment, which occupies one place in the queue and is used to initiate the connection, to indicate where the queue numbers begin.

TCB
Control Block for Transmission, a certain data structure where the status of the connection is recorded.

TCB.PRC
Priority of this connection

TCP
Transfer Control Protocol, protocol for reliable transmission of information between host computers in the system of merged networks.

TOS
Type of Service, headline field in Internet protocol

The type
of header field service in the Internet protocol that determines the type of service for a given piece in the Internet standard.

This
bit requires a customer to be ordered to use expedited processing as long as there is data whose numbers are in line less than indicated in the urgent pointer.

The Urgent Pointer is
only important if a bit of URG is set.