**RFC 793 (RFC793)**

Internet RFC/STD/FYI/BCP Archives

[ [RFC Index](http://www.faqs.org/rfcs/) | [RFC Search](http://www.faqs.org/rfcs/rfcsearch.html) | [Usenet FAQs](http://www.faqs.org/faqs/) | [Web FAQs](http://www.faqs.org/contrib/) | [Documents](http://www.faqs.org/docs/) | [Cities](http://www.city-data.com/) ]

**Alternate Formats:** [rfc793.txt](http://www.faqs.org/ftp/rfc/rfc793.txt) | [rfc793.txt.pdf](http://www.faqs.org/ftp/rfc/pdf/rfc793.txt.pdf)

[Comment on RFC 793](http://www.faqs.org/rfccomment.php?rfcnum=793)

**RFC 793 - Transmission Control Protocol**

RFC: 793

 TRANSMISSION CONTROL PROTOCOL

 DARPA INTERNET PROGRAM

 PROTOCOL SPECIFICATION

 September 1981

 prepared for

 Defense Advanced Research Projects Agency

 Information Processing Techniques Office

 1400 Wilson Boulevard

 Arlington, Virginia 22209

 by

 Information Sciences Institute

 University of Southern California

 4676 Admiralty Way

 Marina del Rey, California 90291

September 1981

 Transmission Control Protocol

 TABLE OF CONTENTS

 PREFACE ........................................................ iii

1. INTRODUCTION ..................................................... 1

 1.1 Motivation .................................................... 1

 1.2 Scope ......................................................... 2

 1.3 About This Document ........................................... 2

 1.4 Interfaces .................................................... 3

 1.5 Operation ..................................................... 3

2. PHILOSOPHY ....................................................... 7

 2.1 Elements of the Internetwork System ........................... 7

 2.2 Model of Operation ............................................ 7

 2.3 The Host Environment .......................................... 8

 2.4 Interfaces .................................................... 9

 2.5 Relation to Other Protocols ................................... 9

 2.6 Reliable Communication ........................................ 9

 2.7 Connection Establishment and Clearing ........................ 10

 2.8 Data Communication ........................................... 12

 2.9 Precedence and Security ...................................... 13

 2.10 Robustness Principle ......................................... 13

3. FUNCTIONAL SPECIFICATION ........................................ 15

 3.1 Header Format ................................................ 15

 3.2 Terminology .................................................. 19

 3.3 Sequence Numbers ............................................. 24

 3.4 Establishing a connection .................................... 30

 3.5 Closing a Connection ......................................... 37

 3.6 Precedence and Security ...................................... 40

 3.7 Data Communication ........................................... 40

 3.8 Interfaces ................................................... 44

 3.9 Event Processing ............................................. 52

GLOSSARY ............................................................ 79

REFERENCES .......................................................... 85

 [Page i]

 September 1981

Transmission Control Protocol

[Page ii]

September 1981

 Transmission Control Protocol

 PREFACE

This document describes the DoD Standard Transmission Control Protocol

(TCP). There have been nine earlier editions of the ARPA TCP

specification on which this standard is based, and the present text

draws heavily from them. There have been many contributors to this work

both in terms of concepts and in terms of text. This edition clarifies

several details and removes the end-of-letter buffer-size adjustments,

and redescribes the letter mechanism as a push function.

 Jon Postel

 Editor

RFC: 793

Replaces: [RFC 761](http://www.faqs.org/rfcs/rfc761.html)

IENs: 129, 124, 112, 81,

55, 44, 40, 27, 21, 5

 TRANSMISSION CONTROL PROTOCOL

 DARPA INTERNET PROGRAM

 PROTOCOL SPECIFICATION

 1. INTRODUCTION

The Transmission Control Protocol (TCP) is intended for use as a highly

reliable host-to-host protocol between hosts in packet-switched computer

communication networks, and in interconnected systems of such networks.

This document describes the functions to be performed by the

Transmission Control Protocol, the program that implements it, and its

interface to programs or users that require its services.

1.1. Motivation

 Computer communication systems are playing an increasingly important

 role in military, government, and civilian environments. This

 document focuses its attention primarily on military computer

 communication requirements, especially robustness in the presence of

 communication unreliability and availability in the presence of

 congestion, but many of these problems are found in the civilian and

 government sector as well.

 As strategic and tactical computer communication networks are

 developed and deployed, it is essential to provide means of

 interconnecting them and to provide standard interprocess

 communication protocols which can support a broad range of

 applications. In anticipation of the need for such standards, the

 Deputy Undersecretary of Defense for Research and Engineering has

 declared the Transmission Control Protocol (TCP) described herein to

 be a basis for DoD-wide inter-process communication protocol

 standardization.

 TCP is a connection-oriented, end-to-end reliable protocol designed to

 fit into a layered hierarchy of protocols which support multi-network

 applications. The TCP provides for reliable inter-process

 communication between pairs of processes in host computers attached to

 distinct but interconnected computer communication networks. Very few

 assumptions are made as to the reliability of the communication

 protocols below the TCP layer. TCP assumes it can obtain a simple,

 potentially unreliable datagram service from the lower level

 protocols. In principle, the TCP should be able to operate above a

 wide spectrum of communication systems ranging from hard-wired

 connections to packet-switched or circuit-switched networks.

 [Page 1]

 September 1981

Transmission Control Protocol

Introduction

 TCP is based on concepts first described by Cerf and Kahn in [1]. The

 TCP fits into a layered protocol architecture just above a basic

 Internet Protocol [2] which provides a way for the TCP to send and

 receive variable-length segments of information enclosed in internet

 datagram "envelopes". The internet datagram provides a means for

 addressing source and destination TCPs in different networks. The

 internet protocol also deals with any fragmentation or reassembly of

 the TCP segments required to achieve transport and delivery through

 multiple networks and interconnecting gateways. The internet protocol

 also carries information on the precedence, security classification

 and compartmentation of the TCP segments, so this information can be

 communicated end-to-end across multiple networks.

 Protocol Layering

 +---------------------+

 | higher-level |

 +---------------------+

 | TCP |

 +---------------------+

 | internet protocol |

 +---------------------+

 |communication network|

 +---------------------+

 Figure 1

 Much of this document is written in the context of TCP implementations

 which are co-resident with higher level protocols in the host

 computer. Some computer systems will be connected to networks via

 front-end computers which house the TCP and internet protocol layers,

 as well as network specific software. The TCP specification describes

 an interface to the higher level protocols which appears to be

 implementable even for the front-end case, as long as a suitable

 host-to-front end protocol is implemented.

1.2. Scope

 The TCP is intended to provide a reliable process-to-process

 communication service in a multinetwork environment. The TCP is

 intended to be a host-to-host protocol in common use in multiple

 networks.

1.3. About this Document

 This document represents a specification of the behavior required of

 any TCP implementation, both in its interactions with higher level

 protocols and in its interactions with other TCPs. The rest of this

[Page 2]

September 1981

 Transmission Control Protocol

 Introduction

 section offers a very brief view of the protocol interfaces and

 operation. Section 2 summarizes the philosophical basis for the TCP

 design. Section 3 offers both a detailed description of the actions

 required of TCP when various events occur (arrival of new segments,

 user calls, errors, etc.) and the details of the formats of TCP

 segments.

1.4. Interfaces

 The TCP interfaces on one side to user or application processes and on

 the other side to a lower level protocol such as Internet Protocol.

 The interface between an application process and the TCP is

 illustrated in reasonable detail. This interface consists of a set of

 calls much like the calls an operating system provides to an

 application process for manipulating files. For example, there are

 calls to open and close connections and to send and receive data on

 established connections. It is also expected that the TCP can

 asynchronously communicate with application programs. Although

 considerable freedom is permitted to TCP implementors to design

 interfaces which are appropriate to a particular operating system

 environment, a minimum functionality is required at the TCP/user

 interface for any valid implementation.

 The interface between TCP and lower level protocol is essentially

 unspecified except that it is assumed there is a mechanism whereby the

 two levels can asynchronously pass information to each other.

 Typically, one expects the lower level protocol to specify this

 interface. TCP is designed to work in a very general environment of

 interconnected networks. The lower level protocol which is assumed

 throughout this document is the Internet Protocol [2].

1.5. Operation

 As noted above, the primary purpose of the TCP is to provide reliable,

 securable logical circuit or connection service between pairs of

 processes. To provide this service on top of a less reliable internet

 communication system requires facilities in the following areas:

 Basic Data Transfer

 Reliability

 Flow Control

 Multiplexing

 Connections

 Precedence and Security

 The basic operation of the TCP in each of these areas is described in

 the following paragraphs.

 [Page 3]

 September 1981

Transmission Control Protocol

Introduction

 Basic Data Transfer:

 The TCP is able to transfer a continuous stream of octets in each

 direction between its users by packaging some number of octets into

 segments for transmission through the internet system. In general,

 the TCPs decide when to block and forward data at their own

 convenience.

 Sometimes users need to be sure that all the data they have

 submitted to the TCP has been transmitted. For this purpose a push

 function is defined. To assure that data submitted to a TCP is

 actually transmitted the sending user indicates that it should be

 pushed through to the receiving user. A push causes the TCPs to

 promptly forward and deliver data up to that point to the receiver.

 The exact push point might not be visible to the receiving user and

 the push function does not supply a record boundary marker.

 Reliability:

 The TCP must recover from data that is damaged, lost, duplicated, or

 delivered out of order by the internet communication system. This

 is achieved by assigning a sequence number to each octet

 transmitted, and requiring a positive acknowledgment (ACK) from the

 receiving TCP. If the ACK is not received within a timeout

 interval, the data is retransmitted. At the receiver, the sequence

 numbers are used to correctly order segments that may be received

 out of order and to eliminate duplicates. Damage is handled by

 adding a checksum to each segment transmitted, checking it at the

 receiver, and discarding damaged segments.

 As long as the TCPs continue to function properly and the internet

 system does not become completely partitioned, no transmission

 errors will affect the correct delivery of data. TCP recovers from

 internet communication system errors.

 Flow Control:

 TCP provides a means for the receiver to govern the amount of data

 sent by the sender. This is achieved by returning a "window" with

 every ACK indicating a range of acceptable sequence numbers beyond

 the last segment successfully received. The window indicates an

 allowed number of octets that the sender may transmit before

 receiving further permission.

[Page 4]

September 1981

 Transmission Control Protocol

 Introduction

 Multiplexing:

 To allow for many processes within a single Host to use TCP

 communication facilities simultaneously, the TCP provides a set of

 addresses or ports within each host. Concatenated with the network

 and host addresses from the internet communication layer, this forms

 a socket. A pair of sockets uniquely identifies each connection.

 That is, a socket may be simultaneously used in multiple

 connections.

 The binding of ports to processes is handled independently by each

 Host. However, it proves useful to attach frequently used processes

 (e.g., a "logger" or timesharing service) to fixed sockets which are

 made known to the public. These services can then be accessed

 through the known addresses. Establishing and learning the port

 addresses of other processes may involve more dynamic mechanisms.

 Connections:

 The reliability and flow control mechanisms described above require

 that TCPs initialize and maintain certain status information for

 each data stream. The combination of this information, including

 sockets, sequence numbers, and window sizes, is called a connection.

 Each connection is uniquely specified by a pair of sockets

 identifying its two sides.

 When two processes wish to communicate, their TCP's must first

 establish a connection (initialize the status information on each

 side). When their communication is complete, the connection is

 terminated or closed to free the resources for other uses.

 Since connections must be established between unreliable hosts and

 over the unreliable internet communication system, a handshake

 mechanism with clock-based sequence numbers is used to avoid

 erroneous initialization of connections.

 Precedence and Security:

 The users of TCP may indicate the security and precedence of their

 communication. Provision is made for default values to be used when

 these features are not needed.

 [Page 5]

 September 1981

Transmission Control Protocol

[Page 6]

September 1981

 Transmission Control Protocol

 2. PHILOSOPHY

2.1. Elements of the Internetwork System

 The internetwork environment consists of hosts connected to networks

 which are in turn interconnected via gateways. It is assumed here

 that the networks may be either local networks (e.g., the ETHERNET) or

 large networks (e.g., the ARPANET), but in any case are based on

 packet switching technology. The active agents that produce and

 consume messages are processes. Various levels of protocols in the

 networks, the gateways, and the hosts support an interprocess

 communication system that provides two-way data flow on logical

 connections between process ports.

 The term packet is used generically here to mean the data of one

 transaction between a host and its network. The format of data blocks

 exchanged within the a network will generally not be of concern to us.

 Hosts are computers attached to a network, and from the communication

 network's point of view, are the sources and destinations of packets.

 Processes are viewed as the active elements in host computers (in

 accordance with the fairly common definition of a process as a program

 in execution). Even terminals and files or other I/O devices are

 viewed as communicating with each other through the use of processes.

 Thus, all communication is viewed as inter-process communication.

 Since a process may need to distinguish among several communication

 streams between itself and another process (or processes), we imagine

 that each process may have a number of ports through which it

 communicates with the ports of other processes.

2.2. Model of Operation

 Processes transmit data by calling on the TCP and passing buffers of

 data as arguments. The TCP packages the data from these buffers into

 segments and calls on the internet module to transmit each segment to

 the destination TCP. The receiving TCP places the data from a segment

 into the receiving user's buffer and notifies the receiving user. The

 TCPs include control information in the segments which they use to

 ensure reliable ordered data transmission.

 The model of internet communication is that there is an internet

 protocol module associated with each TCP which provides an interface

 to the local network. This internet module packages TCP segments

 inside internet datagrams and routes these datagrams to a destination

 internet module or intermediate gateway. To transmit the datagram

 through the local network, it is embedded in a local network packet.

 The packet switches may perform further packaging, fragmentation, or

 [Page 7]

 September 1981

Transmission Control Protocol

Philosophy

 other operations to achieve the delivery of the local packet to the

 destination internet module.

 At a gateway between networks, the internet datagram is "unwrapped"

 from its local packet and examined to determine through which network

 the internet datagram should travel next. The internet datagram is

 then "wrapped" in a local packet suitable to the next network and

 routed to the next gateway, or to the final destination.

 A gateway is permitted to break up an internet datagram into smaller

 internet datagram fragments if this is necessary for transmission

 through the next network. To do this, the gateway produces a set of

 internet datagrams; each carrying a fragment. Fragments may be

 further broken into smaller fragments at subsequent gateways. The

 internet datagram fragment format is designed so that the destination

 internet module can reassemble fragments into internet datagrams.

 A destination internet module unwraps the segment from the datagram

 (after reassembling the datagram, if necessary) and passes it to the

 destination TCP.

 This simple model of the operation glosses over many details. One

 important feature is the type of service. This provides information

 to the gateway (or internet module) to guide it in selecting the

 service parameters to be used in traversing the next network.

 Included in the type of service information is the precedence of the

 datagram. Datagrams may also carry security information to permit

 host and gateways that operate in multilevel secure environments to

 properly segregate datagrams for security considerations.

2.3. The Host Environment

 The TCP is assumed to be a module in an operating system. The users

 access the TCP much like they would access the file system. The TCP

 may call on other operating system functions, for example, to manage

 data structures. The actual interface to the network is assumed to be

 controlled by a device driver module. The TCP does not call on the

 network device driver directly, but rather calls on the internet

 datagram protocol module which may in turn call on the device driver.

 The mechanisms of TCP do not preclude implementation of the TCP in a

 front-end processor. However, in such an implementation, a

 host-to-front-end protocol must provide the functionality to support

 the type of TCP-user interface described in this document.

[Page 8]

September 1981

 Transmission Control Protocol

 Philosophy

2.4. Interfaces

 The TCP/user interface provides for calls made by the user on the TCP

 to OPEN or CLOSE a connection, to SEND or RECEIVE data, or to obtain

 STATUS about a connection. These calls are like other calls from user

 programs on the operating system, for example, the calls to open, read

 from, and close a file.

 The TCP/internet interface provides calls to send and receive

 datagrams addressed to TCP modules in hosts anywhere in the internet

 system. These calls have parameters for passing the address, type of

 service, precedence, security, and other control information.

2.5. Relation to Other Protocols

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

 hierarchy:

 +------+ +-----+ +-----+ +-----+

 |Telnet| | FTP | |Voice| ... | | Application Level

 +------+ +-----+ +-----+ +-----+

 | | | |

 +-----+ +-----+ +-----+

 | TCP | | RTP | ... | | Host Level

 +-----+ +-----+ +-----+

 | | |

 +-------------------------------+

 | Internet Protocol & ICMP | Gateway Level

 +-------------------------------+

 |

 +---------------------------+

 | Local Network Protocol | Network Level

 +---------------------------+

 Protocol Relationships

 Figure 2.

 It is expected that the TCP will be able to support higher level

 protocols efficiently. It should be easy to interface higher level

 protocols like the ARPANET Telnet or AUTODIN II THP to the TCP.

2.6. Reliable Communication

 A stream of data sent on a TCP connection is delivered reliably and in

 order at the destination.

 [Page 9]

 September 1981

Transmission Control Protocol

Philosophy

 Transmission is made reliable via the use of sequence numbers and

 acknowledgments. Conceptually, each octet of data is assigned a

 sequence number. The sequence number of the first octet of data in a

 segment is transmitted with that segment and is called the segment

 sequence number. Segments also carry an acknowledgment number which

 is the sequence number of the next expected data octet of

 transmissions in the reverse direction. When the TCP transmits a

 segment containing data, it puts a copy on a retransmission queue and

 starts a timer; when the acknowledgment for that data is received, the

 segment is deleted from the queue. If the acknowledgment is not

 received before the timer runs out, the segment is retransmitted.

 An acknowledgment by TCP does not guarantee that the data has been

 delivered to the end user, but only that the receiving TCP has taken

 the responsibility to do so.

 To govern the flow of data between TCPs, a flow control mechanism is

 employed. The receiving TCP reports a "window" to the sending TCP.

 This window specifies the number of octets, starting with the

 acknowledgment number, that the receiving TCP is currently prepared to

 receive.

2.7. Connection Establishment and Clearing

 To identify the separate data streams that a TCP may handle, the TCP

 provides a port identifier. Since port identifiers are selected

 independently by each TCP they might not be unique. To provide for

 unique addresses within each TCP, we concatenate an internet address

 identifying the TCP with a port identifier to create a socket which

 will be unique throughout all networks connected together.

 A connection is fully specified by the pair of sockets at the ends. A

 local socket may participate in many connections to different foreign

 sockets. A connection can be used to carry data in both directions,

 that is, it is "full duplex".

 TCPs are free to associate ports with processes however they choose.

 However, several basic concepts are necessary in any implementation.

 There must be well-known sockets which the TCP associates only with

 the "appropriate" processes by some means. We envision that processes

 may "own" ports, and that processes can initiate connections only on

 the ports they own. (Means for implementing ownership is a local

 issue, but we envision a Request Port user command, or a method of

 uniquely allocating a group of ports to a given process, e.g., by

 associating the high order bits of a port name with a given process.)

 A connection is specified in the OPEN call by the local port and

 foreign socket arguments. In return, the TCP supplies a (short) local

[Page 10]

September 1981

 Transmission Control Protocol

 Philosophy

 connection name by which the user refers to the connection in

 subsequent calls. There are several things that must be remembered

 about a connection. To store this information we imagine that there

 is a data structure called a Transmission Control Block (TCB). One

 implementation strategy would have the local connection name be a

 pointer to the TCB for this connection. The OPEN call also specifies

 whether the connection establishment is to be actively pursued, or to

 be passively waited for.

 A passive OPEN request means that the process wants to accept incoming

 connection requests rather than attempting to initiate a connection.

 Often the process requesting a passive OPEN will accept a connection

 request from any caller. In this case a foreign socket of all zeros

 is used to denote an unspecified socket. Unspecified foreign sockets

 are allowed only on passive OPENs.

 A service process that wished to provide services for unknown other

 processes would issue a passive OPEN request with an unspecified

 foreign socket. Then a connection could be made with any process that

 requested a connection to this local socket. It would help if this

 local socket were known to be associated with this service.

 Well-known sockets are a convenient mechanism for a priori associating

 a socket address with a standard service. For instance, the

 "Telnet-Server" process is permanently assigned to a particular

 socket, and other sockets are reserved for File Transfer, Remote Job

 Entry, Text Generator, Echoer, and Sink processes (the last three

 being for test purposes). A socket address might be reserved for

 access to a "Look-Up" service which would return the specific socket

 at which a newly created service would be provided. The concept of a

 well-known socket is part of the TCP specification, but the assignment

 of sockets to services is outside this specification. (See [4].)

 Processes can issue passive OPENs and wait for matching active OPENs

 from other processes and be informed by the TCP when connections have

 been established. Two processes which issue active OPENs to each

 other at the same time will be correctly connected. This flexibility

 is critical for the support of distributed computing in which

 components act asynchronously with respect to each other.

 There are two principal cases for matching the sockets in the local

 passive OPENs and an foreign active OPENs. In the first case, the

 local passive OPENs has fully specified the foreign socket. In this

 case, the match must be exact. In the second case, the local passive

 OPENs has left the foreign socket unspecified. In this case, any

 foreign socket is acceptable as long as the local sockets match.

 Other possibilities include partially restricted matches.

 [Page 11]

 September 1981

Transmission Control Protocol

Philosophy

 If there are several pending passive OPENs (recorded in TCBs) with the

 same local socket, an foreign active OPEN will be matched to a TCB

 with the specific foreign socket in the foreign active OPEN, if such a

 TCB exists, before selecting a TCB with an unspecified foreign socket.

 The procedures to establish connections utilize the synchronize (SYN)

 control flag and involves an exchange of three messages. This

 exchange has been termed a three-way hand shake [3].

 A connection is initiated by the rendezvous of an arriving segment

 containing a SYN and a waiting TCB entry each created by a user OPEN

 command. The matching of local and foreign sockets determines when a

 connection has been initiated. The connection becomes "established"

 when sequence numbers have been synchronized in both directions.

 The clearing of a connection also involves the exchange of segments,

 in this case carrying the FIN control flag.

2.8. Data Communication

 The data that flows on a connection may be thought of as a stream of

 octets. The sending user indicates in each SEND call whether the data

 in that call (and any preceeding calls) should be immediately pushed

 through to the receiving user by the setting of the PUSH flag.

 A sending TCP is allowed to collect data from the sending user and to

 send that data in segments at its own convenience, until the push

 function is signaled, then it must send all unsent data. When a

 receiving TCP sees the PUSH flag, it must not wait for more data from

 the sending TCP before passing the data to the receiving process.

 There is no necessary relationship between push functions and segment

 boundaries. The data in any particular segment may be the result of a

 single SEND call, in whole or part, or of multiple SEND calls.

 The purpose of push function and the PUSH flag is to push data through

 from the sending user to the receiving user. It does not provide a

 record service.

 There is a coupling between the push function and the use of buffers

 of data that cross the TCP/user interface. Each time a PUSH flag is

 associated with data placed into the receiving user's buffer, the

 buffer is returned to the user for processing even if the buffer is

 not filled. If data arrives that fills the user's buffer before a

 PUSH is seen, the data is passed to the user in buffer size units.

 TCP also provides a means to communicate to the receiver of data that

 at some point further along in the data stream than the receiver is

[Page 12]

September 1981

 Transmission Control Protocol

 Philosophy

 currently reading there is urgent data. TCP does not attempt to

 define what the user specifically does upon being notified of pending

 urgent data, but the general notion is that the receiving process will

 take action to process the urgent data quickly.

2.9. Precedence and Security

 The TCP makes use of the internet protocol type of service field and

 security option to provide precedence and security on a per connection

 basis to TCP users. Not all TCP modules will necessarily function in

 a multilevel secure environment; some may be limited to unclassified

 use only, and others may operate at only one security level and

 compartment. Consequently, some TCP implementations and services to

 users may be limited to a subset of the multilevel secure case.

 TCP modules which operate in a multilevel secure environment must

 properly mark outgoing segments with the security, compartment, and

 precedence. Such TCP modules must also provide to their users or

 higher level protocols such as Telnet or THP an interface to allow

 them to specify the desired security level, compartment, and

 precedence of connections.

2.10. Robustness Principle

 TCP implementations will follow a general principle of robustness: be

 conservative in what you do, be liberal in what you accept from

 others.

 [Page 13]

 September 1981

Transmission Control Protocol

[Page 14]

September 1981

 Transmission Control Protocol

 3. FUNCTIONAL SPECIFICATION

3.1. Header Format

 TCP segments are sent as internet datagrams. The Internet Protocol

 header carries several information fields, including the source and

 destination host addresses [2]. A TCP header follows the internet

 header, supplying information specific to the TCP protocol. This

 division allows for the existence of host level protocols other than

 TCP.

 TCP Header Format

 0 1 2 3

 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

 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

 | Source Port | Destination Port |

 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

 | Sequence Number |

 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

 | Acknowledgment Number |

 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

 | Data | |U|A|P|R|S|F| |

 | Offset| Reserved |R|C|S|S|Y|I| Window |

 | | |G|K|H|T|N|N| |

 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

 | Checksum | Urgent Pointer |

 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

 | Options | Padding |

 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

 | data |

 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

 TCP Header Format

 Note that one tick mark represents one bit position.

 Figure 3.

 Source Port: 16 bits

 The source port number.

 Destination Port: 16 bits

 The destination port number.

 [Page 15]

 September 1981

Transmission Control Protocol

Functional Specification

 Sequence Number: 32 bits

 The sequence number of the first data octet in this segment (except

 when SYN is present). If SYN is present the sequence number is the

 initial sequence number (ISN) and the first data octet is ISN+1.

 Acknowledgment Number: 32 bits

 If the ACK control bit is set this field contains the value of the

 next sequence number the sender of the segment is expecting to

 receive. Once a connection is established this is always sent.

 Data Offset: 4 bits

 The number of 32 bit words in the TCP Header. This indicates where

 the data begins. The TCP header (even one including options) is an

 integral number of 32 bits long.

 Reserved: 6 bits

 Reserved for future use. Must be zero.

 Control Bits: 6 bits (from left to right):

 URG: Urgent Pointer field significant

 ACK: Acknowledgment field significant

 PSH: Push Function

 RST: Reset the connection

 SYN: Synchronize sequence numbers

 FIN: No more data from sender

 Window: 16 bits

 The number of data octets beginning with the one indicated in the

 acknowledgment field which the sender of this segment is willing to

 accept.

 Checksum: 16 bits

 The checksum field is the 16 bit one's complement of the one's

 complement sum of all 16 bit words in the header and text. If a

 segment contains an odd number of header and text octets to be

 checksummed, the last octet is padded on the right with zeros to

 form a 16 bit word for checksum purposes. The pad is not

 transmitted as part of the segment. While computing the checksum,

 the checksum field itself is replaced with zeros.

 The checksum also covers a 96 bit pseudo header conceptually

[Page 16]

September 1981

 Transmission Control Protocol

 Functional Specification

 prefixed to the TCP header. This pseudo header contains the Source

 Address, the Destination Address, the Protocol, and TCP length.

 This gives the TCP protection against misrouted segments. This

 information is carried in the Internet Protocol and is transferred

 across the TCP/Network interface in the arguments or results of

 calls by the TCP on the IP.

 +--------+--------+--------+--------+

 | Source Address |

 +--------+--------+--------+--------+

 | Destination Address |

 +--------+--------+--------+--------+

 | zero | PTCL | TCP Length |

 +--------+--------+--------+--------+

 The TCP Length is the TCP header length plus the data length in

 octets (this is not an explicitly transmitted quantity, but is

 computed), and it does not count the 12 octets of the pseudo

 header.

 Urgent Pointer: 16 bits

 This field communicates the current value of the urgent pointer as a

 positive offset from the sequence number in this segment. The

 urgent pointer points to the sequence number of the octet following

 the urgent data. This field is only be interpreted in segments with

 the URG control bit set.

 Options: variable

 Options may occupy space at the end of the TCP header and are a

 multiple of 8 bits in length. All options are included in the

 checksum. An option may begin on any octet boundary. There are two

 cases for the format of an option:

 Case 1: A single octet of option-kind.

 Case 2: An octet of option-kind, an octet of option-length, and

 the actual option-data octets.

 The option-length counts the two octets of option-kind and

 option-length as well as the option-data octets.

 Note that the list of options may be shorter than the data offset

 field might imply. The content of the header beyond the

 End-of-Option option must be header padding (i.e., zero).

 A TCP must implement all options.

 [Page 17]

 September 1981

Transmission Control Protocol

Functional Specification

 Currently defined options include (kind indicated in octal):

 Kind Length Meaning

 ---- ------ -------

 0 - End of option list.

 1 - No-Operation.

 2 4 Maximum Segment Size.

 Specific Option Definitions

 End of Option List

 +--------+

 |00000000|

 +--------+

 Kind=0

 This option code indicates the end of the option list. This

 might not coincide with the end of the TCP header according to

 the Data Offset field. This is used at the end of all options,

 not the end of each option, and need only be used if the end of

 the options would not otherwise coincide with the end of the TCP

 header.

 No-Operation

 +--------+

 |00000001|

 +--------+

 Kind=1

 This option code may be used between options, for example, to

 align the beginning of a subsequent option on a word boundary.

 There is no guarantee that senders will use this option, so

 receivers must be prepared to process options even if they do

 not begin on a word boundary.

 Maximum Segment Size

 +--------+--------+---------+--------+

 |00000010|00000100| max seg size |

 +--------+--------+---------+--------+

 Kind=2 Length=4

[Page 18]

September 1981

 Transmission Control Protocol

 Functional Specification

 Maximum Segment Size Option Data: 16 bits

 If this option is present, then it communicates the maximum

 receive segment size at the TCP which sends this segment.

 This field must only be sent in the initial connection request

 (i.e., in segments with the SYN control bit set). If this

 option is not used, any segment size is allowed.

 Padding: variable

 The TCP header padding is used to ensure that the TCP header ends

 and data begins on a 32 bit boundary. The padding is composed of

 zeros.

3.2. Terminology

 Before we can discuss very much about the operation of the TCP we need

 to introduce some detailed terminology. The maintenance of a TCP

 connection requires the remembering of several variables. We conceive

 of these variables being stored in a connection record called a

 Transmission Control Block or TCB. Among the variables stored in the

 TCB are the local and remote socket numbers, the security and

 precedence of the connection, pointers to the user's send and receive

 buffers, pointers to the retransmit queue and to the current segment.

 In addition several variables relating to the send and receive

 sequence numbers are stored in the TCB.

 Send Sequence Variables

 SND.UNA - send unacknowledged

 SND.NXT - send next

 SND.WND - send window

 SND.UP - send urgent pointer

 SND.WL1 - segment sequence number used for last window update

 SND.WL2 - segment acknowledgment number used for last window

 update

 ISS - initial send sequence number

 Receive Sequence Variables

 RCV.NXT - receive next

 RCV.WND - receive window

 RCV.UP - receive urgent pointer

 IRS - initial receive sequence number

 [Page 19]

 September 1981

Transmission Control Protocol

Functional Specification

 The following diagrams may help to relate some of these variables to

 the sequence space.

 Send Sequence Space

 1 2 3 4

 ----------|----------|----------|----------

 SND.UNA SND.NXT SND.UNA

 +SND.WND

 1 - old sequence numbers which have been acknowledged

 2 - sequence numbers of unacknowledged data

 3 - sequence numbers allowed for new data transmission

 4 - future sequence numbers which are not yet allowed

 Send Sequence Space

 Figure 4.

 The send window is the portion of the sequence space labeled 3 in

 figure 4.

 Receive Sequence Space

 1 2 3

 ----------|----------|----------

 RCV.NXT RCV.NXT

 +RCV.WND

 1 - old sequence numbers which have been acknowledged

 2 - sequence numbers allowed for new reception

 3 - future sequence numbers which are not yet allowed

 Receive Sequence Space

 Figure 5.

 The receive window is the portion of the sequence space labeled 2 in

 figure 5.

 There are also some variables used frequently in the discussion that

 take their values from the fields of the current segment.

[Page 20]

September 1981

 Transmission Control Protocol

 Functional Specification

 Current Segment Variables

 SEG.SEQ - segment sequence number

 SEG.ACK - segment acknowledgment number

 SEG.LEN - segment length

 SEG.WND - segment window

 SEG.UP - segment urgent pointer

 SEG.PRC - segment precedence value

 A connection progresses through a series of states during its

 lifetime. The states are: LISTEN, SYN-SENT, SYN-RECEIVED,

 ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK,

 TIME-WAIT, and the fictional state CLOSED. CLOSED is fictional

 because it represents the state when there is no TCB, and therefore,

 no connection. Briefly the meanings of the states are:

 LISTEN - represents waiting for a connection request from any remote

 TCP and port.

 SYN-SENT - represents waiting for a matching connection request

 after having sent a connection request.

 SYN-RECEIVED - represents waiting for a confirming connection

 request acknowledgment after having both received and sent a

 connection request.

 ESTABLISHED - represents an open connection, data received can be

 delivered to the user. The normal state for the data transfer phase

 of the connection.

 FIN-WAIT-1 - represents waiting for a connection termination request

 from the remote TCP, or an acknowledgment of the connection

 termination request previously sent.

 FIN-WAIT-2 - represents waiting for a connection termination request

 from the remote TCP.

 CLOSE-WAIT - represents waiting for a connection termination request

 from the local user.

 CLOSING - represents waiting for a connection termination request

 acknowledgment from the remote TCP.

 LAST-ACK - represents waiting for an acknowledgment of the

 connection termination request previously sent to the remote TCP

 (which includes an acknowledgment of its connection termination

 request).

 [Page 21]

 September 1981

Transmission Control Protocol

Functional Specification

 TIME-WAIT - represents waiting for enough time to pass to be sure

 the remote TCP received the acknowledgment of its connection

 termination request.

 CLOSED - represents no connection state at all.

 A TCP connection progresses from one state to another in response to

 events. The events are the user calls, OPEN, SEND, RECEIVE, CLOSE,

 ABORT, and STATUS; the incoming segments, particularly those

 containing the SYN, ACK, RST and FIN flags; and timeouts.

 The state diagram in figure 6 illustrates only state changes, together

 with the causing events and resulting actions, but addresses neither

 error conditions nor actions which are not connected with state

 changes. In a later section, more detail is offered with respect to

 the reaction of the TCP to events.

 NOTE BENE: this diagram is only a summary and must not be taken as

 the total specification.

[Page 22]

September 1981

 Transmission Control Protocol

 Functional Specification

 +---------+ ---------\ active OPEN

 | CLOSED | \ -----------

 +---------+<---------\ \ create TCB

 | ^ \ \ snd SYN

 passive OPEN | | CLOSE \ \

 ------------ | | ---------- \ \

 create TCB | | delete TCB \ \

 V | \ \

 +---------+ CLOSE | \

 | LISTEN | ---------- | |

 +---------+ delete TCB | |

 rcv SYN | | SEND | |

 ----------- | | ------- | V

 +---------+ snd SYN,ACK / \ snd SYN +---------+

 | |<----------------- ------------------>| |

 | SYN | rcv SYN | SYN |

 | RCVD |<-----------------------------------------------| SENT |

 | | snd ACK | |

 | |------------------ -------------------| |

 +---------+ rcv ACK of SYN \ / rcv SYN,ACK +---------+

 | -------------- | | -----------

 | x | | snd ACK

 | V V

 | CLOSE +---------+

 | ------- | ESTAB |

 | snd FIN +---------+

 | CLOSE | | rcv FIN

 V ------- | | -------

 +---------+ snd FIN / \ snd ACK +---------+

 | FIN |<----------------- ------------------>| CLOSE |

 | WAIT-1 |------------------ | WAIT |

 +---------+ rcv FIN \ +---------+

 | rcv ACK of FIN ------- | CLOSE |

 | -------------- snd ACK | ------- |

 V x V snd FIN V

 +---------+ +---------+ +---------+

 |FINWAIT-2| | CLOSING | | LAST-ACK|

 +---------+ +---------+ +---------+

 | rcv ACK of FIN | rcv ACK of FIN |

 | rcv FIN -------------- | Timeout=2MSL -------------- |

 | ------- x V ------------ x V

 \ snd ACK +---------+delete TCB +---------+

 ------------------------>|TIME WAIT|------------------>| CLOSED |

 +---------+ +---------+

 TCP Connection State Diagram

 Figure 6.

 [Page 23]

 September 1981

Transmission Control Protocol

Functional Specification

3.3. Sequence Numbers

 A fundamental notion in the design is that every octet of data sent

 over a TCP connection has a sequence number. Since every octet is

 sequenced, each of them can be acknowledged. The acknowledgment

 mechanism employed is cumulative so that an acknowledgment of sequence

 number X indicates that all octets up to but not including X have been

 received. This mechanism allows for straight-forward duplicate

 detection in the presence of retransmission. Numbering of octets

 within a segment is that the first data octet immediately following

 the header is the lowest numbered, and the following octets are

 numbered consecutively.

 It is essential to remember that the actual sequence number space is

 finite, though very large. This space ranges from 0 to 2\*\*32 - 1.

 Since the space is finite, all arithmetic dealing with sequence

 numbers must be performed modulo 2\*\*32. This unsigned arithmetic

 preserves the relationship of sequence numbers as they cycle from

 2\*\*32 - 1 to 0 again. There are some subtleties to computer modulo

 arithmetic, so great care should be taken in programming the

 comparison of such values. The symbol "=<" means "less than or equal"

 (modulo 2\*\*32).

 The typical kinds of sequence number comparisons which the TCP must

 perform include:

 (a) Determining that an acknowledgment refers to some sequence

 number sent but not yet acknowledged.

 (b) Determining that all sequence numbers occupied by a segment

 have been acknowledged (e.g., to remove the segment from a

 retransmission queue).

 (c) Determining that an incoming segment contains sequence numbers

 which are expected (i.e., that the segment "overlaps" the

 receive window).

[Page 24]

September 1981

 Transmission Control Protocol

 Functional Specification

 In response to sending data the TCP will receive acknowledgments. The

 following comparisons are needed to process the acknowledgments.

 SND.UNA = oldest unacknowledged sequence number

 SND.NXT = next sequence number to be sent

 SEG.ACK = acknowledgment from the receiving TCP (next sequence

 number expected by the receiving TCP)

 SEG.SEQ = first sequence number of a segment

 SEG.LEN = the number of octets occupied by the data in the segment

 (counting SYN and FIN)

 SEG.SEQ+SEG.LEN-1 = last sequence number of a segment

 A new acknowledgment (called an "acceptable ack"), is one for which

 the inequality below holds:

 SND.UNA < SEG.ACK =< SND.NXT

 A segment on the retransmission queue is fully acknowledged if the sum

 of its sequence number and length is less or equal than the

 acknowledgment value in the incoming segment.

 When data is received the following comparisons are needed:

 RCV.NXT = next sequence number expected on an incoming segments, and

 is the left or lower edge of the receive window

 RCV.NXT+RCV.WND-1 = last sequence number expected on an incoming

 segment, and is the right or upper edge of the receive window

 SEG.SEQ = first sequence number occupied by the incoming segment

 SEG.SEQ+SEG.LEN-1 = last sequence number occupied by the incoming

 segment

 A segment is judged to occupy a portion of valid receive sequence

 space if

 RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND

 or

 RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

 [Page 25]

 September 1981

Transmission Control Protocol

Functional Specification

 The first part of this test checks to see if the beginning of the

 segment falls in the window, the second part of the test checks to see

 if the end of the segment falls in the window; if the segment passes

 either part of the test it contains data in the window.

 Actually, it is a little more complicated than this. Due to zero

 windows and zero length segments, we have four cases for the

 acceptability of an incoming segment:

 Segment Receive Test

 Length Window

 ------- ------- -------------------------------------------

 0 0 SEG.SEQ = RCV.NXT

 0 >0 RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND

 >0 0 not acceptable

 >0 >0 RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND

 or RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

 Note that when the receive window is zero no segments should be

 acceptable except ACK segments. Thus, it is be possible for a TCP to

 maintain a zero receive window while transmitting data and receiving

 ACKs. However, even when the receive window is zero, a TCP must

 process the RST and URG fields of all incoming segments.

 We have taken advantage of the numbering scheme to protect certain

 control information as well. This is achieved by implicitly including

 some control flags in the sequence space so they can be retransmitted

 and acknowledged without confusion (i.e., one and only one copy of the

 control will be acted upon). Control information is not physically

 carried in the segment data space. Consequently, we must adopt rules

 for implicitly assigning sequence numbers to control. The SYN and FIN

 are the only controls requiring this protection, and these controls

 are used only at connection opening and closing. For sequence number

 purposes, the SYN is considered to occur before the first actual data

 octet of the segment in which it occurs, while the FIN is considered

 to occur after the last actual data octet in a segment in which it

 occurs. The segment length (SEG.LEN) includes both data and sequence

 space occupying controls. When a SYN is present then SEG.SEQ is the

 sequence number of the SYN.

[Page 26]

September 1981

 Transmission Control Protocol

 Functional Specification

 Initial Sequence Number Selection

 The protocol places no restriction on a particular connection being

 used over and over again. A connection is defined by a pair of

 sockets. New instances of a connection will be referred to as

 incarnations of the connection. The problem that arises from this is

 -- "how does the TCP identify duplicate segments from previous

 incarnations of the connection?" This problem becomes apparent if the

 connection is being opened and closed in quick succession, or if the

 connection breaks with loss of memory and is then reestablished.

 To avoid confusion we must prevent segments from one incarnation of a

 connection from being used while the same sequence numbers may still

 be present in the network from an earlier incarnation. We want to

 assure this, even if a TCP crashes and loses all knowledge of the

 sequence numbers it has been using. When new connections are created,

 an initial sequence number (ISN) generator is employed which selects a

 new 32 bit ISN. The generator is bound to a (possibly fictitious) 32

 bit clock whose low order bit is incremented roughly every 4

 microseconds. Thus, the ISN cycles approximately every 4.55 hours.

 Since we assume that segments will stay in the network no more than

 the Maximum Segment Lifetime (MSL) and that the MSL is less than 4.55

 hours we can reasonably assume that ISN's will be unique.

 For each connection there is a send sequence number and a receive

 sequence number. The initial send sequence number (ISS) is chosen by

 the data sending TCP, and the initial receive sequence number (IRS) is

 learned during the connection establishing procedure.

 For a connection to be established or initialized, the two TCPs must

 synchronize on each other's initial sequence numbers. This is done in

 an exchange of connection establishing segments carrying a control bit

 called "SYN" (for synchronize) and the initial sequence numbers. As a

 shorthand, segments carrying the SYN bit are also called "SYNs".

 Hence, the solution requires a suitable mechanism for picking an

 initial sequence number and a slightly involved handshake to exchange

 the ISN's.

 The synchronization requires each side to send it's own initial

 sequence number and to receive a confirmation of it in acknowledgment

 from the other side. Each side must also receive the other side's

 initial sequence number and send a confirming acknowledgment.

 1) A --> B SYN my sequence number is X

 2) A <-- B ACK your sequence number is X

 3) A <-- B SYN my sequence number is Y

 4) A --> B ACK your sequence number is Y

 [Page 27]

 September 1981

Transmission Control Protocol

Functional Specification

 Because steps 2 and 3 can be combined in a single message this is

 called the three way (or three message) handshake.

 A three way handshake is necessary because sequence numbers are not

 tied to a global clock in the network, and TCPs may have different

 mechanisms for picking the ISN's. The receiver of the first SYN has

 no way of knowing whether the segment was an old delayed one or not,

 unless it remembers the last sequence number used on the connection

 (which is not always possible), and so it must ask the sender to

 verify this SYN. The three way handshake and the advantages of a

 clock-driven scheme are discussed in [3].

 Knowing When to Keep Quiet

 To be sure that a TCP does not create a segment that carries a

 sequence number which may be duplicated by an old segment remaining in

 the network, the TCP must keep quiet for a maximum segment lifetime

 (MSL) before assigning any sequence numbers upon starting up or

 recovering from a crash in which memory of sequence numbers in use was

 lost. For this specification the MSL is taken to be 2 minutes. This

 is an engineering choice, and may be changed if experience indicates

 it is desirable to do so. Note that if a TCP is reinitialized in some

 sense, yet retains its memory of sequence numbers in use, then it need

 not wait at all; it must only be sure to use sequence numbers larger

 than those recently used.

 The TCP Quiet Time Concept

 This specification provides that hosts which "crash" without

 retaining any knowledge of the last sequence numbers transmitted on

 each active (i.e., not closed) connection shall delay emitting any

 TCP segments for at least the agreed Maximum Segment Lifetime (MSL)

 in the internet system of which the host is a part. In the

 paragraphs below, an explanation for this specification is given.

 TCP implementors may violate the "quiet time" restriction, but only

 at the risk of causing some old data to be accepted as new or new

 data rejected as old duplicated by some receivers in the internet

 system.

 TCPs consume sequence number space each time a segment is formed and

 entered into the network output queue at a source host. The

 duplicate detection and sequencing algorithm in the TCP protocol

 relies on the unique binding of segment data to sequence space to

 the extent that sequence numbers will not cycle through all 2\*\*32

 values before the segment data bound to those sequence numbers has

 been delivered and acknowledged by the receiver and all duplicate

 copies of the segments have "drained" from the internet. Without

 such an assumption, two distinct TCP segments could conceivably be

[Page 28]

September 1981

 Transmission Control Protocol

 Functional Specification

 assigned the same or overlapping sequence numbers, causing confusion

 at the receiver as to which data is new and which is old. Remember

 that each segment is bound to as many consecutive sequence numbers

 as there are octets of data in the segment.

 Under normal conditions, TCPs keep track of the next sequence number

 to emit and the oldest awaiting acknowledgment so as to avoid

 mistakenly using a sequence number over before its first use has

 been acknowledged. This alone does not guarantee that old duplicate

 data is drained from the net, so the sequence space has been made

 very large to reduce the probability that a wandering duplicate will

 cause trouble upon arrival. At 2 megabits/sec. it takes 4.5 hours

 to use up 2\*\*32 octets of sequence space. Since the maximum segment

 lifetime in the net is not likely to exceed a few tens of seconds,

 this is deemed ample protection for foreseeable nets, even if data

 rates escalate to l0's of megabits/sec. At 100 megabits/sec, the

 cycle time is 5.4 minutes which may be a little short, but still

 within reason.

 The basic duplicate detection and sequencing algorithm in TCP can be

 defeated, however, if a source TCP does not have any memory of the

 sequence numbers it last used on a given connection. For example, if

 the TCP were to start all connections with sequence number 0, then

 upon crashing and restarting, a TCP might re-form an earlier

 connection (possibly after half-open connection resolution) and emit

 packets with sequence numbers identical to or overlapping with

 packets still in the network which were emitted on an earlier

 incarnation of the same connection. In the absence of knowledge

 about the sequence numbers used on a particular connection, the TCP

 specification recommends that the source delay for MSL seconds

 before emitting segments on the connection, to allow time for

 segments from the earlier connection incarnation to drain from the

 system.

 Even hosts which can remember the time of day and used it to select

 initial sequence number values are not immune from this problem

 (i.e., even if time of day is used to select an initial sequence

 number for each new connection incarnation).

 Suppose, for example, that a connection is opened starting with

 sequence number S. Suppose that this connection is not used much

 and that eventually the initial sequence number function (ISN(t))

 takes on a value equal to the sequence number, say S1, of the last

 segment sent by this TCP on a particular connection. Now suppose,

 at this instant, the host crashes, recovers, and establishes a new

 incarnation of the connection. The initial sequence number chosen is

 S1 = ISN(t) -- last used sequence number on old incarnation of

 connection! If the recovery occurs quickly enough, any old

 [Page 29]

 September 1981

Transmission Control Protocol

Functional Specification

 duplicates in the net bearing sequence numbers in the neighborhood

 of S1 may arrive and be treated as new packets by the receiver of

 the new incarnation of the connection.

 The problem is that the recovering host may not know for how long it

 crashed nor does it know whether there are still old duplicates in

 the system from earlier connection incarnations.

 One way to deal with this problem is to deliberately delay emitting

 segments for one MSL after recovery from a crash- this is the "quite

 time" specification. Hosts which prefer to avoid waiting are

 willing to risk possible confusion of old and new packets at a given

 destination may choose not to wait for the "quite time".

 Implementors may provide TCP users with the ability to select on a

 connection by connection basis whether to wait after a crash, or may

 informally implement the "quite time" for all connections.

 Obviously, even where a user selects to "wait," this is not

 necessary after the host has been "up" for at least MSL seconds.

 To summarize: every segment emitted occupies one or more sequence

 numbers in the sequence space, the numbers occupied by a segment are

 "busy" or "in use" until MSL seconds have passed, upon crashing a

 block of space-time is occupied by the octets of the last emitted

 segment, if a new connection is started too soon and uses any of the

 sequence numbers in the space-time footprint of the last segment of

 the previous connection incarnation, there is a potential sequence

 number overlap area which could cause confusion at the receiver.

3.4. Establishing a connection

 The "three-way handshake" is the procedure used to establish a

 connection. This procedure normally is initiated by one TCP and

 responded to by another TCP. The procedure also works if two TCP

 simultaneously initiate the procedure. When simultaneous attempt

 occurs, each TCP receives a "SYN" segment which carries no

 acknowledgment after it has sent a "SYN". Of course, the arrival of

 an old duplicate "SYN" segment can potentially make it appear, to the

 recipient, that a simultaneous connection initiation is in progress.

 Proper use of "reset" segments can disambiguate these cases.

 Several examples of connection initiation follow. Although these

 examples do not show connection synchronization using data-carrying

 segments, this is perfectly legitimate, so long as the receiving TCP

 doesn't deliver the data to the user until it is clear the data is

 valid (i.e., the data must be buffered at the receiver until the

 connection reaches the ESTABLISHED state). The three-way handshake

 reduces the possibility of false connections. It is the

[Page 30]

September 1981

 Transmission Control Protocol

 Functional Specification

 implementation of a trade-off between memory and messages to provide

 information for this checking.

 The simplest three-way handshake is shown in figure 7 below. The

 figures should be interpreted in the following way. Each line is

 numbered for reference purposes. Right arrows (-->) indicate

 departure of a TCP segment from TCP A to TCP B, or arrival of a

 segment at B from A. Left arrows (<--), indicate the reverse.

 Ellipsis (...) indicates a segment which is still in the network

 (delayed). An "XXX" indicates a segment which is lost or rejected.

 Comments appear in parentheses. TCP states represent the state AFTER

 the departure or arrival of the segment (whose contents are shown in

 the center of each line). Segment contents are shown in abbreviated

 form, with sequence number, control flags, and ACK field. Other

 fields such as window, addresses, lengths, and text have been left out

 in the interest of clarity.

 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><CTL=ACK><DATA> --> ESTABLISHED

 Basic 3-Way Handshake for Connection Synchronization

 Figure 7.

 In line 2 of figure 7, TCP A begins by sending a SYN segment

 indicating that it will use sequence numbers starting with sequence

 number 100. In line 3, TCP B sends a SYN and acknowledges the SYN it

 received from TCP A. Note that the acknowledgment field indicates TCP

 B is now expecting to hear sequence 101, acknowledging the SYN which

 occupied sequence 100.

 At line 4, TCP A responds with an empty segment containing an ACK for

 TCP B's SYN; and in line 5, TCP A sends some data. Note that the

 sequence number of the segment in line 5 is the same as in line 4

 because the ACK does not occupy sequence number space (if it did, we

 would wind up ACKing ACK's!).

 [Page 31]

 September 1981

Transmission Control Protocol

Functional Specification

 Simultaneous initiation is only slightly more complex, as is shown in

 figure 8. Each TCP cycles from CLOSED to SYN-SENT to SYN-RECEIVED to

 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><CTL=SYN,ACK> ...

 6. ESTABLISHED <-- <SEQ=300><ACK=101><CTL=SYN,ACK> <-- SYN-RECEIVED

 7. ... <SEQ=101><ACK=301><CTL=ACK> --> ESTABLISHED

 Simultaneous Connection Synchronization

 Figure 8.

 The principle reason for the three-way handshake is to prevent old

 duplicate connection initiations from causing confusion. To deal with

 this, a special control message, reset, has been devised. If the

 receiving TCP is in a non-synchronized state (i.e., SYN-SENT,

 SYN-RECEIVED), it returns to LISTEN on receiving an acceptable reset.

 If the TCP is in one of the synchronized states (ESTABLISHED,

 FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), it

 aborts the connection and informs its user. We discuss this latter

 case under "half-open" connections below.

[Page 32]

September 1981

 Transmission Control Protocol

 Functional Specification

 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

 Recovery from Old Duplicate SYN

 Figure 9.

 As a simple example of recovery from old duplicates, consider

 figure 9. At line 3, an old duplicate SYN arrives at TCP B. TCP B

 cannot tell that this is an old duplicate, so it responds normally

 (line 4). TCP A detects that the ACK field is incorrect and returns a

 RST (reset) with its SEQ field selected to make the segment

 believable. TCP B, on receiving the RST, returns to the LISTEN state.

 When the original SYN (pun intended) finally arrives at line 6, the

 synchronization proceeds normally. If the SYN at line 6 had arrived

 before the RST, a more complex exchange might have occurred with RST's

 sent in both directions.

 Half-Open Connections and Other Anomalies

 An established connection is said to be "half-open" if one of the

 TCPs has closed or aborted the connection at its end without the

 knowledge of the other, or if the two ends of the connection have

 become desynchronized owing to a crash that resulted in loss of

 memory. Such connections will automatically become reset if an

 attempt is made to send data in either direction. However, half-open

 connections are expected to be unusual, and the recovery procedure is

 mildly involved.

 If at site A the connection no longer exists, then an attempt by the

 [Page 33]

 September 1981

Transmission Control Protocol

Functional Specification

 user at site B to send any data on it will result in the site B TCP

 receiving a reset control message. Such a message indicates to the

 site B TCP that something is wrong, and it is expected to abort the

 connection.

 Assume that two user processes A and B are communicating with one

 another when a crash occurs causing loss of memory to A's TCP.

 Depending on the operating system supporting A's TCP, it is likely

 that some error recovery mechanism exists. When the TCP is up again,

 A is likely to start again from the beginning or from a recovery

 point. As a result, A will probably try to OPEN the connection again

 or try to SEND on the connection it believes open. In the latter

 case, it receives the error message "connection not open" from the

 local (A's) TCP. In an attempt to establish the connection, A's TCP

 will send a segment containing SYN. This scenario leads to the

 example shown in figure 10. After TCP A crashes, the user attempts to

 re-open the connection. TCP B, in the meantime, thinks the connection

 is open.

 TCP A TCP B

 1. (CRASH) (send 300,receive 100)

 2. CLOSED ESTABLISHED

 3. SYN-SENT --> <SEQ=400><CTL=SYN> --> (??)

 4. (!!) <-- <SEQ=300><ACK=100><CTL=ACK> <-- ESTABLISHED

 5. SYN-SENT --> <SEQ=100><CTL=RST> --> (Abort!!)

 6. SYN-SENT CLOSED

 7. SYN-SENT --> <SEQ=400><CTL=SYN> -->

 Half-Open Connection Discovery

 Figure 10.

 When the SYN arrives at line 3, TCP B, being in a synchronized state,

 and the incoming segment outside the window, responds with an

 acknowledgment indicating what sequence it next expects to hear (ACK

 100). TCP A sees that this segment does not acknowledge anything it

 sent and, being unsynchronized, sends a reset (RST) because it has

 detected a half-open connection. TCP B aborts at line 5. TCP A will

[Page 34]

September 1981

 Transmission Control Protocol

 Functional Specification

 continue to try to establish the connection; the problem is now

 reduced to the basic 3-way handshake of figure 7.

 An interesting alternative case occurs when TCP A crashes and TCP B

 tries to send data on what it thinks is a synchronized connection.

 This is illustrated in figure 11. In this case, the data arriving at

 TCP A from TCP B (line 2) is unacceptable because no such connection

 exists, so TCP A sends a RST. The RST is acceptable so TCP B

 processes it and aborts the connection.

 TCP A TCP B

 1. (CRASH) (send 300,receive 100)

 2. (??) <-- <SEQ=300><ACK=100><DATA=10><CTL=ACK> <-- ESTABLISHED

 3. --> <SEQ=100><CTL=RST> --> (ABORT!!)

 Active Side Causes Half-Open Connection Discovery

 Figure 11.

 In figure 12, we find the two TCPs A and B with passive connections

 waiting for SYN. An old duplicate arriving at TCP B (line 2) stirs B

 into action. A SYN-ACK is returned (line 3) and causes TCP A to

 generate a RST (the ACK in line 3 is not acceptable). TCP B accepts

 the reset and returns to its passive LISTEN state.

 TCP A TCP B

 1. LISTEN LISTEN

 2. ... <SEQ=Z><CTL=SYN> --> SYN-RECEIVED

 3. (??) <-- <SEQ=X><ACK=Z+1><CTL=SYN,ACK> <-- SYN-RECEIVED

 4. --> <SEQ=Z+1><CTL=RST> --> (return to LISTEN!)

 5. LISTEN LISTEN

 Old Duplicate SYN Initiates a Reset on two Passive Sockets

 Figure 12.

 [Page 35]

 September 1981

Transmission Control Protocol

Functional Specification

 A variety of other cases are possible, all of which are accounted for

 by the following rules for RST generation and processing.

 Reset Generation

 As a general rule, reset (RST) must be sent whenever a segment arrives

 which apparently is not intended for the current connection. A reset

 must not be sent if it is not clear that this is the case.

 There are three groups of states:

 1. If the connection does not exist (CLOSED) then a reset is sent

 in response to any incoming segment except another reset. In

 particular, SYNs addressed to a non-existent connection are rejected

 by this means.

 If the incoming segment has an ACK field, the reset takes its

 sequence number from the ACK field of the segment, otherwise the

 reset has sequence number zero and the ACK field is set to the sum

 of the sequence number and segment length of the incoming segment.

 The connection remains in the CLOSED state.

 2. If the connection is in any non-synchronized state (LISTEN,

 SYN-SENT, SYN-RECEIVED), and the incoming segment acknowledges

 something not yet sent (the segment carries an unacceptable ACK), or

 if an incoming segment has a security level or compartment which

 does not exactly match the level and compartment requested for the

 connection, a reset is sent.

 If our SYN has not been acknowledged and the precedence level of the

 incoming segment is higher than the precedence level requested then

 either raise the local precedence level (if allowed by the user and

 the system) or send a reset; or if the precedence level of the

 incoming segment is lower than the precedence level requested then

 continue as if the precedence matched exactly (if the remote TCP

 cannot raise the precedence level to match ours this will be

 detected in the next segment it sends, and the connection will be

 terminated then). If our SYN has been acknowledged (perhaps in this

 incoming segment) the precedence level of the incoming segment must

 match the local precedence level exactly, if it does not a reset

 must be sent.

 If the incoming segment has an ACK field, the reset takes its

 sequence number from the ACK field of the segment, otherwise the

 reset has sequence number zero and the ACK field is set to the sum

 of the sequence number and segment length of the incoming segment.

 The connection remains in the same state.

[Page 36]

September 1981

 Transmission Control Protocol

 Functional Specification

 3. If the connection is in a synchronized state (ESTABLISHED,

 FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT),

 any unacceptable segment (out of window sequence number or

 unacceptible acknowledgment number) must elicit only an empty

 acknowledgment segment containing the current send-sequence number

 and an acknowledgment indicating the next sequence number expected

 to be received, and the connection remains in the same state.

 If an incoming segment has a security level, or compartment, or

 precedence which does not exactly match the level, and compartment,

 and precedence requested for the connection,a reset is sent and

 connection goes to the CLOSED state. The reset takes its sequence

 number from the ACK field of the incoming segment.

 Reset Processing

 In all states except SYN-SENT, all reset (RST) segments are validated

 by checking their SEQ-fields. A reset is valid if its sequence number

 is in the window. In the SYN-SENT state (a RST received in response

 to an initial SYN), the RST is acceptable if the ACK field

 acknowledges the SYN.

 The receiver of a RST first validates it, then changes state. If the

 receiver was in the LISTEN state, it ignores it. If the receiver was

 in SYN-RECEIVED state and had previously been in the LISTEN state,

 then the receiver returns to the LISTEN state, otherwise the receiver

 aborts the connection and goes to the CLOSED state. If the receiver

 was in any other state, it aborts the connection and advises the user

 and goes to the CLOSED state.

3.5. Closing a Connection

 CLOSE is an operation meaning "I have no more data to send." The

 notion of closing a full-duplex connection is subject to ambiguous

 interpretation, of course, since it may not be obvious how to treat

 the receiving side of the connection. We have chosen to treat CLOSE

 in a simplex fashion. The user who CLOSEs may continue to RECEIVE

 until he is told that the other side has CLOSED also. Thus, a program

 could initiate several SENDs followed by a CLOSE, and then continue to

 RECEIVE until signaled that a RECEIVE failed because the other side

 has CLOSED. We assume that the TCP will signal a user, even if no

 RECEIVEs are outstanding, that the other side has closed, so the user

 can terminate his side gracefully. A TCP will reliably deliver all

 buffers SENT before the connection was CLOSED so a user who expects no

 data in return need only wait to hear the connection was CLOSED

 successfully to know that all his data was received at the destination

 TCP. Users must keep reading connections they close for sending until

 the TCP says no more data.

 [Page 37]

 September 1981

Transmission Control Protocol

Functional Specification

 There are essentially three cases:

 1) The user initiates by telling the TCP to CLOSE the connection

 2) The remote TCP initiates by sending a FIN control signal

 3) Both users CLOSE simultaneously

 Case 1: Local user initiates the close

 In this case, a FIN segment can be constructed and placed on the

 outgoing segment queue. No further SENDs from the user will be

 accepted by the TCP, and it enters the FIN-WAIT-1 state. RECEIVEs

 are allowed in this state. All segments preceding and including FIN

 will be retransmitted until acknowledged. When the other TCP has

 both acknowledged the FIN and sent a FIN of its own, the first TCP

 can ACK this FIN. Note that a TCP receiving a FIN will ACK but not

 send its own FIN until its user has CLOSED the connection also.

 Case 2: TCP receives a FIN from the network

 If an unsolicited FIN arrives from the network, the receiving TCP

 can ACK it and tell the user that the connection is closing. The

 user will respond with a CLOSE, upon which the TCP can send a FIN to

 the other TCP after sending any remaining data. The TCP then waits

 until its own FIN is acknowledged whereupon it deletes the

 connection. If an ACK is not forthcoming, after the user timeout

 the connection is aborted and the user is told.

 Case 3: both users close simultaneously

 A simultaneous CLOSE by users at both ends of a connection causes

 FIN segments to be exchanged. When all segments preceding the FINs

 have been processed and acknowledged, each TCP can ACK the FIN it

 has received. Both will, upon receiving these ACKs, delete the

 connection.

[Page 38]

September 1981

 Transmission Control Protocol

 Functional Specification

 TCP A TCP B

 1. ESTABLISHED ESTABLISHED

 2. (Close)

 FIN-WAIT-1 --> <SEQ=100><ACK=300><CTL=FIN,ACK> --> CLOSE-WAIT

 3. FIN-WAIT-2 <-- <SEQ=300><ACK=101><CTL=ACK> <-- CLOSE-WAIT

 4. (Close)

 TIME-WAIT <-- <SEQ=300><ACK=101><CTL=FIN,ACK> <-- LAST-ACK

 5. TIME-WAIT --> <SEQ=101><ACK=301><CTL=ACK> --> CLOSED

 6. (2 MSL)

 CLOSED

 Normal Close Sequence

 Figure 13.

 TCP A TCP B

 1. ESTABLISHED ESTABLISHED

 2. (Close) (Close)

 FIN-WAIT-1 --> <SEQ=100><ACK=300><CTL=FIN,ACK> ... FIN-WAIT-1

 <-- <SEQ=300><ACK=100><CTL=FIN,ACK> <--

 ... <SEQ=100><ACK=300><CTL=FIN,ACK> -->

 3. CLOSING --> <SEQ=101><ACK=301><CTL=ACK> ... CLOSING

 <-- <SEQ=301><ACK=101><CTL=ACK> <--

 ... <SEQ=101><ACK=301><CTL=ACK> -->

 4. TIME-WAIT TIME-WAIT

 (2 MSL) (2 MSL)

 CLOSED CLOSED

 Simultaneous Close Sequence

 Figure 14.

 [Page 39]

 September 1981

Transmission Control Protocol

Functional Specification

3.6. Precedence and Security

 The intent is that connection be allowed only between ports operating

 with exactly the same security and compartment values and at the

 higher of the precedence level requested by the two ports.

 The precedence and security parameters used in TCP are exactly those

 defined in the Internet Protocol (IP) [2]. Throughout this TCP

 specification the term "security/compartment" is intended to indicate

 the security parameters used in IP including security, compartment,

 user group, and handling restriction.

 A connection attempt with mismatched security/compartment values or a

 lower precedence value must be rejected by sending a reset. Rejecting

 a connection due to too low a precedence only occurs after an

 acknowledgment of the SYN has been received.

 Note that TCP modules which operate only at the default value of

 precedence will still have to check the precedence of incoming

 segments and possibly raise the precedence level they use on the

 connection.

 The security paramaters may be used even in a non-secure environment

 (the values would indicate unclassified data), thus hosts in

 non-secure environments must be prepared to receive the security

 parameters, though they need not send them.

3.7. Data Communication

 Once the connection is established data is communicated by the

 exchange of segments. Because segments may be lost due to errors

 (checksum test failure), or network congestion, TCP uses

 retransmission (after a timeout) to ensure delivery of every segment.

 Duplicate segments may arrive due to network or TCP retransmission.

 As discussed in the section on sequence numbers the TCP performs

 certain tests on the sequence and acknowledgment numbers in the

 segments to verify their acceptability.

 The sender of data keeps track of the next sequence number to use in

 the variable SND.NXT. The receiver of data keeps track of the next

 sequence number to expect in the variable RCV.NXT. The sender of data

 keeps track of the oldest unacknowledged sequence number in the

 variable SND.UNA. If the data flow is momentarily idle and all data

 sent has been acknowledged then the three variables will be equal.

 When the sender creates a segment and transmits it the sender advances

 SND.NXT. When the receiver accepts a segment it advances RCV.NXT and

 sends an acknowledgment. When the data sender receives an

[Page 40]

September 1981

 Transmission Control Protocol

 Functional Specification

 acknowledgment it advances SND.UNA. The extent to which the values of

 these variables differ is a measure of the delay in the communication.

 The amount by which the variables are advanced is the length of the

 data in the segment. Note that once in the ESTABLISHED state all

 segments must carry current acknowledgment information.

 The CLOSE user call implies a push function, as does the FIN control

 flag in an incoming segment.

 Retransmission Timeout

 Because of the variability of the networks that compose an

 internetwork system and the wide range of uses of TCP connections the

 retransmission timeout must be dynamically determined. One procedure

 for determining a retransmission time out is given here as an

 illustration.

 An Example Retransmission Timeout Procedure

 Measure the elapsed time between sending a data octet with a

 particular sequence number and receiving an acknowledgment that

 covers that sequence number (segments sent do not have to match

 segments received). This measured elapsed time is the Round Trip

 Time (RTT). Next compute a Smoothed Round Trip Time (SRTT) as:

 SRTT = ( ALPHA \* SRTT ) + ((1-ALPHA) \* RTT)

 and based on this, compute the retransmission timeout (RTO) as:

 RTO = min[UBOUND,max[LBOUND,(BETA\*SRTT)]]

 where UBOUND is an upper bound on the timeout (e.g., 1 minute),

 LBOUND is a lower bound on the timeout (e.g., 1 second), ALPHA is

 a smoothing factor (e.g., .8 to .9), and BETA is a delay variance

 factor (e.g., 1.3 to 2.0).

 The Communication of Urgent Information

 The objective of the TCP urgent mechanism is to allow the sending user

 to stimulate the receiving user to accept some urgent data and to

 permit the receiving TCP to indicate to the receiving user when all

 the currently known urgent data has been received by the user.

 This mechanism permits a point in the data stream to be designated as

 the end of urgent information. Whenever this point is in advance of

 the receive sequence number (RCV.NXT) at the receiving TCP, that TCP

 must tell the user to go into "urgent mode"; when the receive sequence

 number catches up to the urgent pointer, the TCP must tell user to go

 [Page 41]

 September 1981

Transmission Control Protocol

Functional Specification

 into "normal mode". If the urgent pointer is updated while the user

 is in "urgent mode", the update will be invisible to the user.

 The method employs a urgent field which is carried in all segments

 transmitted. The URG control flag indicates that the urgent field is

 meaningful and must be added to the segment sequence number to yield

 the urgent pointer. The absence of this flag indicates that there is

 no urgent data outstanding.

 To send an urgent indication the user must also send at least one data

 octet. If the sending user also indicates a push, timely delivery of

 the urgent information to the destination process is enhanced.

 Managing the Window

 The window sent in each segment indicates the range of sequence

 numbers the sender of the window (the data receiver) is currently

 prepared to accept. There is an assumption that this is related to

 the currently available data buffer space available for this

 connection.

 Indicating a large window encourages transmissions. If more data

 arrives than can be accepted, it will be discarded. This will result

 in excessive retransmissions, adding unnecessarily to the load on the

 network and the TCPs. Indicating a small window may restrict the

 transmission of data to the point of introducing a round trip delay

 between each new segment transmitted.

 The mechanisms provided allow a TCP to advertise a large window and to

 subsequently advertise a much smaller window without having accepted

 that much data. This, so called "shrinking the window," is strongly

 discouraged. The robustness principle dictates that TCPs will not

 shrink the window themselves, but will be prepared for such behavior

 on the part of other TCPs.

 The sending TCP must be prepared to accept from the user and send at

 least one octet of new data even if the send window is zero. The

 sending TCP must regularly retransmit to the receiving TCP even when

 the window is zero. Two minutes is recommended for the retransmission

 interval when the window is zero. This retransmission is essential to

 guarantee that when either TCP has a zero window the re-opening of the

 window will be reliably reported to the other.

 When the receiving TCP has a zero window and a segment arrives it must

 still send an acknowledgment showing its next expected sequence number

 and current window (zero).

 The sending TCP packages the data to be transmitted into segments

[Page 42]

September 1981

 Transmission Control Protocol

 Functional Specification

 which fit the current window, and may repackage segments on the

 retransmission queue. Such repackaging is not required, but may be

 helpful.

 In a connection with a one-way data flow, the window information will

 be carried in acknowledgment segments that all have the same sequence

 number so there will be no way to reorder them if they arrive out of

 order. This is not a serious problem, but it will allow the window

 information to be on occasion temporarily based on old reports from

 the data receiver. A refinement to avoid this problem is to act on

 the window information from segments that carry the highest

 acknowledgment number (that is segments with acknowledgment number

 equal or greater than the highest previously received).

 The window management procedure has significant influence on the

 communication performance. The following comments are suggestions to

 implementers.

 Window Management Suggestions

 Allocating a very small window causes data to be transmitted in

 many small segments when better performance is achieved using

 fewer large segments.

 One suggestion for avoiding small windows is for the receiver to

 defer updating a window until the additional allocation is at

 least X percent of the maximum allocation possible for the

 connection (where X might be 20 to 40).

 Another suggestion is for the sender to avoid sending small

 segments by waiting until the window is large enough before

 sending data. If the the user signals a push function then the

 data must be sent even if it is a small segment.

 Note that the acknowledgments should not be delayed or unnecessary

 retransmissions will result. One strategy would be to send an

 acknowledgment when a small segment arrives (with out updating the

 window information), and then to send another acknowledgment with

 new window information when the window is larger.

 The segment sent to probe a zero window may also begin a break up

 of transmitted data into smaller and smaller segments. If a

 segment containing a single data octet sent to probe a zero window

 is accepted, it consumes one octet of the window now available.

 If the sending TCP simply sends as much as it can whenever the

 window is non zero, the transmitted data will be broken into

 alternating big and small segments. As time goes on, occasional

 pauses in the receiver making window allocation available will

 [Page 43]

 September 1981

Transmission Control Protocol

Functional Specification

 result in breaking the big segments into a small and not quite so

 big pair. And after a while the data transmission will be in

 mostly small segments.

 The suggestion here is that the TCP implementations need to

 actively attempt to combine small window allocations into larger

 windows, since the mechanisms for managing the window tend to lead

 to many small windows in the simplest minded implementations.

3.8. Interfaces

 There are of course two interfaces of concern: the user/TCP interface

 and the TCP/lower-level interface. We have a fairly elaborate model

 of the user/TCP interface, but the interface to the lower level

 protocol module is left unspecified here, since it will be specified

 in detail by the specification of the lowel level protocol. For the

 case that the lower level is IP we note some of the parameter values

 that TCPs might use.

 User/TCP Interface

 The following functional description of user commands to the TCP is,

 at best, fictional, since every operating system will have different

 facilities. Consequently, we must warn readers that different TCP

 implementations may have different user interfaces. However, all

 TCPs must provide a certain minimum set of services to guarantee

 that all TCP implementations can support the same protocol

 hierarchy. This section specifies the functional interfaces

 required of all TCP implementations.

 TCP User Commands

 The following sections functionally characterize a USER/TCP

 interface. The notation used is similar to most procedure or

 function calls in high level languages, but this usage is not

 meant to rule out trap type service calls (e.g., SVCs, UUOs,

 EMTs).

 The user commands described below specify the basic functions the

 TCP must perform to support interprocess communication.

 Individual implementations must define their own exact format, and

 may provide combinations or subsets of the basic functions in

 single calls. In particular, some implementations may wish to

 automatically OPEN a connection on the first SEND or RECEIVE

 issued by the user for a given connection.

[Page 44]

September 1981

 Transmission Control Protocol

 Functional Specification

 In providing interprocess communication facilities, the TCP must

 not only accept commands, but must also return information to the

 processes it serves. The latter consists of:

 (a) general information about a connection (e.g., interrupts,

 remote close, binding of unspecified foreign socket).

 (b) replies to specific user commands indicating success or

 various types of failure.

 Open

 Format: OPEN (local port, foreign socket, active/passive

 [, timeout] [, precedence] [, security/compartment] [, options])

 -> local connection name

 We assume that the local TCP is aware of the identity of the

 processes it serves and will check the authority of the process

 to use the connection specified. Depending upon the

 implementation of the TCP, the local network and TCP identifiers

 for the source address will either be supplied by the TCP or the

 lower level protocol (e.g., IP). These considerations are the

 result of concern about security, to the extent that no TCP be

 able to masquerade as another one, and so on. Similarly, no

 process can masquerade as another without the collusion of the

 TCP.

 If the active/passive flag is set to passive, then this is a

 call to LISTEN for an incoming connection. A passive open may

 have either a fully specified foreign socket to wait for a

 particular connection or an unspecified foreign socket to wait

 for any call. A fully specified passive call can be made active

 by the subsequent execution of a SEND.

 A transmission control block (TCB) is created and partially

 filled in with data from the OPEN command parameters.

 On an active OPEN command, the TCP will begin the procedure to

 synchronize (i.e., establish) the connection at once.

 The timeout, if present, permits the caller to set up a timeout

 for all data submitted to TCP. If data is not successfully

 delivered to the destination within the timeout period, the TCP

 will abort the connection. The present global default is five

 minutes.

 The TCP or some component of the operating system will verify

 the users authority to open a connection with the specified

 [Page 45]

 September 1981

Transmission Control Protocol

Functional Specification

 precedence or security/compartment. The absence of precedence

 or security/compartment specification in the OPEN call indicates

 the default values must be used.

 TCP will accept incoming requests as matching only if the

 security/compartment information is exactly the same and only if

 the precedence is equal to or higher than the precedence

 requested in the OPEN call.

 The precedence for the connection is the higher of the values

 requested in the OPEN call and received from the incoming

 request, and fixed at that value for the life of the

 connection.Implementers may want to give the user control of

 this precedence negotiation. For example, the user might be

 allowed to specify that the precedence must be exactly matched,

 or that any attempt to raise the precedence be confirmed by the

 user.

 A local connection name will be returned to the user by the TCP.

 The local connection name can then be used as a short hand term

 for the connection defined by the <local socket, foreign socket>

 pair.

 Send

 Format: SEND (local connection name, buffer address, byte

 count, PUSH flag, URGENT flag [,timeout])

 This call causes the data contained in the indicated user buffer

 to be sent on the indicated connection. If the connection has

 not been opened, the SEND is considered an error. Some

 implementations may allow users to SEND first; in which case, an

 automatic OPEN would be done. If the calling process is not

 authorized to use this connection, an error is returned.

 If the PUSH flag is set, the data must be transmitted promptly

 to the receiver, and the PUSH bit will be set in the last TCP

 segment created from the buffer. If the PUSH flag is not set,

 the data may be combined with data from subsequent SENDs for

 transmission efficiency.

 If the URGENT flag is set, segments sent to the destination TCP

 will have the urgent pointer set. The receiving TCP will signal

 the urgent condition to the receiving process if the urgent

 pointer indicates that data preceding the urgent pointer has not

 been consumed by the receiving process. The purpose of urgent

 is to stimulate the receiver to process the urgent data and to

 indicate to the receiver when all the currently known urgent

[Page 46]

September 1981

 Transmission Control Protocol

 Functional Specification

 data has been received. The number of times the sending user's

 TCP signals urgent will not necessarily be equal to the number

 of times the receiving user will be notified of the presence of

 urgent data.

 If no foreign socket was specified in the OPEN, but the

 connection is established (e.g., because a LISTENing connection

 has become specific due to a foreign segment arriving for the

 local socket), then the designated buffer is sent to the implied

 foreign socket. Users who make use of OPEN with an unspecified

 foreign socket can make use of SEND without ever explicitly

 knowing the foreign socket address.

 However, if a SEND is attempted before the foreign socket

 becomes specified, an error will be returned. Users can use the

 STATUS call to determine the status of the connection. In some

 implementations the TCP may notify the user when an unspecified

 socket is bound.

 If a timeout is specified, the current user timeout for this

 connection is changed to the new one.

 In the simplest implementation, SEND would not return control to

 the sending process until either the transmission was complete

 or the timeout had been exceeded. However, this simple method

 is both subject to deadlocks (for example, both sides of the

 connection might try to do SENDs before doing any RECEIVEs) and

 offers poor performance, so it is not recommended. A more

 sophisticated implementation would return immediately to allow

 the process to run concurrently with network I/O, and,

 furthermore, to allow multiple SENDs to be in progress.

 Multiple SENDs are served in first come, first served order, so

 the TCP will queue those it cannot service immediately.

 We have implicitly assumed an asynchronous user interface in

 which a SEND later elicits some kind of SIGNAL or

 pseudo-interrupt from the serving TCP. An alternative is to

 return a response immediately. For instance, SENDs might return

 immediate local acknowledgment, even if the segment sent had not

 been acknowledged by the distant TCP. We could optimistically

 assume eventual success. If we are wrong, the connection will

 close anyway due to the timeout. In implementations of this

 kind (synchronous), there will still be some asynchronous

 signals, but these will deal with the connection itself, and not

 with specific segments or buffers.

 In order for the process to distinguish among error or success

 indications for different SENDs, it might be appropriate for the

 [Page 47]

 September 1981

Transmission Control Protocol

Functional Specification

 buffer address to be returned along with the coded response to

 the SEND request. TCP-to-user signals are discussed below,

 indicating the information which should be returned to the

 calling process.

 Receive

 Format: RECEIVE (local connection name, buffer address, byte

 count) -> byte count, urgent flag, push flag

 This command allocates a receiving buffer associated with the

 specified connection. If no OPEN precedes this command or the

 calling process is not authorized to use this connection, an

 error is returned.

 In the simplest implementation, control would not return to the

 calling program until either the buffer was filled, or some

 error occurred, but this scheme is highly subject to deadlocks.

 A more sophisticated implementation would permit several

 RECEIVEs to be outstanding at once. These would be filled as

 segments arrive. This strategy permits increased throughput at

 the cost of a more elaborate scheme (possibly asynchronous) to

 notify the calling program that a PUSH has been seen or a buffer

 filled.

 If enough data arrive to fill the buffer before a PUSH is seen,

 the PUSH flag will not be set in the response to the RECEIVE.

 The buffer will be filled with as much data as it can hold. If

 a PUSH is seen before the buffer is filled the buffer will be

 returned partially filled and PUSH indicated.

 If there is urgent data the user will have been informed as soon

 as it arrived via a TCP-to-user signal. The receiving user

 should thus be in "urgent mode". If the URGENT flag is on,

 additional urgent data remains. If the URGENT flag is off, this

 call to RECEIVE has returned all the urgent data, and the user

 may now leave "urgent mode". Note that data following the

 urgent pointer (non-urgent data) cannot be delivered to the user

 in the same buffer with preceeding urgent data unless the

 boundary is clearly marked for the user.

 To distinguish among several outstanding RECEIVEs and to take

 care of the case that a buffer is not completely filled, the

 return code is accompanied by both a buffer pointer and a byte

 count indicating the actual length of the data received.

 Alternative implementations of RECEIVE might have the TCP

[Page 48]

September 1981

 Transmission Control Protocol

 Functional Specification

 allocate buffer storage, or the TCP might share a ring buffer

 with the user.

 Close

 Format: CLOSE (local connection name)

 This command causes the connection specified to be closed. If

 the connection is not open or the calling process is not

 authorized to use this connection, an error is returned.

 Closing connections is intended to be a graceful operation in

 the sense that outstanding SENDs will be transmitted (and

 retransmitted), as flow control permits, until all have been

 serviced. Thus, it should be acceptable to make several SEND

 calls, followed by a CLOSE, and expect all the data to be sent

 to the destination. It should also be clear that users should

 continue to RECEIVE on CLOSING connections, since the other side

 may be trying to transmit the last of its data. Thus, CLOSE

 means "I have no more to send" but does not mean "I will not

 receive any more." It may happen (if the user level protocol is

 not well thought out) that the closing side is unable to get rid

 of all its data before timing out. In this event, CLOSE turns

 into ABORT, and the closing TCP gives up.

 The user may CLOSE the connection at any time on his own

 initiative, or in response to various prompts from the TCP

 (e.g., remote close executed, transmission timeout exceeded,

 destination inaccessible).

 Because closing a connection requires communication with the

 foreign TCP, connections may remain in the closing state for a

 short time. Attempts to reopen the connection before the TCP

 replies to the CLOSE command will result in error responses.

 Close also implies push function.

 Status

 Format: STATUS (local connection name) -> status data

 This is an implementation dependent user command and could be

 excluded without adverse effect. Information returned would

 typically come from the TCB associated with the connection.

 This command returns a data block containing the following

 information:

 local socket,

 [Page 49]

 September 1981

Transmission Control Protocol

Functional Specification

 foreign socket,

 local connection name,

 receive window,

 send window,

 connection state,

 number of buffers awaiting acknowledgment,

 number of buffers pending receipt,

 urgent state,

 precedence,

 security/compartment,

 and transmission timeout.

 Depending on the state of the connection, or on the

 implementation itself, some of this information may not be

 available or meaningful. If the calling process is not

 authorized to use this connection, an error is returned. This

 prevents unauthorized processes from gaining information about a

 connection.

 Abort

 Format: ABORT (local connection name)

 This command causes all pending SENDs and RECEIVES to be

 aborted, the TCB to be removed, and a special RESET message to

 be sent to the TCP on the other side of the connection.

 Depending on the implementation, users may receive abort

 indications for each outstanding SEND or RECEIVE, or may simply

 receive an ABORT-acknowledgment.

 TCP-to-User Messages

 It is assumed that the operating system environment provides a

 means for the TCP to asynchronously signal the user program. When

 the TCP does signal a user program, certain information is passed

 to the user. Often in the specification the information will be

 an error message. In other cases there will be information

 relating to the completion of processing a SEND or RECEIVE or

 other user call.

 The following information is provided:

 Local Connection Name Always

 Response String Always

 Buffer Address Send & Receive

 Byte count (counts bytes received) Receive

 Push flag Receive

 Urgent flag Receive

[Page 50]

September 1981

 Transmission Control Protocol

 Functional Specification

 TCP/Lower-Level Interface

 The TCP calls on a lower level protocol module to actually send and

 receive information over a network. One case is that of the ARPA

 internetwork system where the lower level module is the Internet

 Protocol (IP) [2].

 If the lower level protocol is IP it provides arguments for a type

 of service and for a time to live. TCP uses the following settings

 for these parameters:

 Type of Service = Precedence: routine, Delay: normal, Throughput:

 normal, Reliability: normal; or 00000000.

 Time to Live = one minute, or 00111100.

 Note that the assumed maximum segment lifetime is two minutes.

 Here we explicitly ask that a segment be destroyed if it cannot

 be delivered by the internet system within one minute.

 If the lower level is IP (or other protocol that provides this

 feature) and source routing is used, the interface must allow the

 route information to be communicated. This is especially important

 so that the source and destination addresses used in the TCP

 checksum be the originating source and ultimate destination. It is

 also important to preserve the return route to answer connection

 requests.

 Any lower level protocol will have to provide the source address,

 destination address, and protocol fields, and some way to determine

 the "TCP length", both to provide the functional equivlent service

 of IP and to be used in the TCP checksum.

 [Page 51]

 September 1981

Transmission Control Protocol

Functional Specification

3.9. Event Processing

 The processing depicted in this section is an example of one possible

 implementation. Other implementations may have slightly different

 processing sequences, but they should differ from those in this

 section only in detail, not in substance.

 The activity of the TCP can be characterized as responding to events.

 The events that occur can be cast into three categories: user calls,

 arriving segments, and timeouts. This section describes the

 processing the TCP does in response to each of the events. In many

 cases the processing required depends on the state of the connection.

 Events that occur:

 User Calls

 OPEN

 SEND

 RECEIVE

 CLOSE

 ABORT

 STATUS

 Arriving Segments

 SEGMENT ARRIVES

 Timeouts

 USER TIMEOUT

 RETRANSMISSION TIMEOUT

 TIME-WAIT TIMEOUT

 The model of the TCP/user interface is that user commands receive an

 immediate return and possibly a delayed response via an event or

 pseudo interrupt. In the following descriptions, the term "signal"

 means cause a delayed response.

 Error responses are given as character strings. For example, user

 commands referencing connections that do not exist receive "error:

 connection not open".

 Please note in the following that all arithmetic on sequence numbers,

 acknowledgment numbers, windows, et cetera, is modulo 2\*\*32 the size

 of the sequence number space. Also note that "=<" means less than or

 equal to (modulo 2\*\*32).

[Page 52]

September 1981

 Transmission Control Protocol

 Functional Specification

 A natural way to think about processing incoming segments is to

 imagine that they are first tested for proper sequence number (i.e.,

 that their contents lie in the range of the expected "receive window"

 in the sequence number space) and then that they are generally queued

 and processed in sequence number order.

 When a segment overlaps other already received segments we reconstruct

 the segment to contain just the new data, and adjust the header fields

 to be consistent.

 Note that if no state change is mentioned the TCP stays in the same

 state.

 [Page 53]

 September 1981

Transmission Control Protocol

Functional Specification

 OPEN Call

 OPEN Call

 CLOSED STATE (i.e., TCB does not exist)

 Create a new transmission control block (TCB) to hold connection

 state information. Fill in local socket identifier, foreign

 socket, precedence, security/compartment, and user timeout

 information. Note that some parts of the foreign socket may be

 unspecified in a passive OPEN and are to be filled in by the

 parameters of the incoming SYN segment. Verify the security and

 precedence requested are allowed for this user, if not return

 "error: precedence not allowed" or "error: security/compartment

 not allowed." If passive enter the LISTEN state and return. If

 active and the foreign socket is unspecified, return "error:

 foreign socket unspecified"; if active and the foreign socket is

 specified, issue a SYN segment. An initial send sequence number

 (ISS) is selected. A SYN segment of the form <SEQ=ISS><CTL=SYN>

 is sent. Set SND.UNA to ISS, SND.NXT to ISS+1, enter SYN-SENT

 state, and return.

 If the caller does not have access to the local socket specified,

 return "error: connection illegal for this process". If there is

 no room to create a new connection, return "error: insufficient

 resources".

 LISTEN STATE

 If active and the foreign socket is specified, then change the

 connection from passive to active, select an ISS. Send a SYN

 segment, set SND.UNA to ISS, SND.NXT to ISS+1. Enter SYN-SENT

 state. Data associated with SEND may be sent with SYN segment or

 queued for transmission after entering ESTABLISHED state. The

 urgent bit if requested in the command must be sent with the data

 segments sent as a result of this command. If there is no room to

 queue the request, respond with "error: insufficient resources".

 If Foreign socket was not specified, then return "error: foreign

 socket unspecified".

[Page 54]

September 1981

 Transmission Control Protocol

 Functional Specification

OPEN Call

 SYN-SENT STATE

 SYN-RECEIVED STATE

 ESTABLISHED STATE

 FIN-WAIT-1 STATE

 FIN-WAIT-2 STATE

 CLOSE-WAIT STATE

 CLOSING STATE

 LAST-ACK STATE

 TIME-WAIT STATE

 Return "error: connection already exists".

 [Page 55]

 September 1981

Transmission Control Protocol

Functional Specification

 SEND Call

 SEND Call

 CLOSED STATE (i.e., TCB does not exist)

 If the user does not have access to such a connection, then return

 "error: connection illegal for this process".

 Otherwise, return "error: connection does not exist".

 LISTEN STATE

 If the foreign socket is specified, then change the connection

 from passive to active, select an ISS. Send a SYN segment, set

 SND.UNA to ISS, SND.NXT to ISS+1. Enter SYN-SENT state. Data

 associated with SEND may be sent with SYN segment or queued for

 transmission after entering ESTABLISHED state. The urgent bit if

 requested in the command must be sent with the data segments sent

 as a result of this command. If there is no room to queue the

 request, respond with "error: insufficient resources". If

 Foreign socket was not specified, then return "error: foreign

 socket unspecified".

 SYN-SENT STATE

 SYN-RECEIVED STATE

 Queue the data for transmission after entering ESTABLISHED state.

 If no space to queue, respond with "error: insufficient

 resources".

 ESTABLISHED STATE

 CLOSE-WAIT STATE

 Segmentize the buffer and send it with a piggybacked

 acknowledgment (acknowledgment value = RCV.NXT). If there is

 insufficient space to remember this buffer, simply return "error:

 insufficient resources".

 If the urgent flag is set, then SND.UP <- SND.NXT-1 and set the

 urgent pointer in the outgoing segments.

[Page 56]

September 1981

 Transmission Control Protocol

 Functional Specification

SEND Call

 FIN-WAIT-1 STATE

 FIN-WAIT-2 STATE

 CLOSING STATE

 LAST-ACK STATE

 TIME-WAIT STATE

 Return "error: connection closing" and do not service request.

 [Page 57]

 September 1981

Transmission Control Protocol

Functional Specification

 RECEIVE Call

 RECEIVE Call

 CLOSED STATE (i.e., TCB does not exist)

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

 "error: connection illegal for this process".

 Otherwise return "error: connection does not exist".

 LISTEN STATE

 SYN-SENT STATE

 SYN-RECEIVED STATE

 Queue for processing after entering ESTABLISHED state. If there

 is no room to queue this request, respond with "error:

 insufficient resources".

 ESTABLISHED STATE

 FIN-WAIT-1 STATE

 FIN-WAIT-2 STATE

 If insufficient incoming segments are queued to satisfy the

 request, queue the request. If there is no queue space to

 remember the RECEIVE, respond with "error: insufficient

 resources".

 Reassemble queued incoming segments into receive buffer and return

 to user. Mark "push seen" (PUSH) if this is the case.

 If RCV.UP is in advance of the data currently being passed to the

 user notify the user of the presence of urgent data.

 When the TCP takes responsibility for delivering data to the user

 that fact must be communicated to the sender via an

 acknowledgment. The formation of such an acknowledgment is

 described below in the discussion of processing an incoming

 segment.

[Page 58]

September 1981

 Transmission Control Protocol

 Functional Specification

RECEIVE Call

 CLOSE-WAIT STATE

 Since the remote side has already sent FIN, RECEIVEs must be

 satisfied by text already on hand, but not yet delivered to the

 user. If no text is awaiting delivery, the RECEIVE will get a

 "error: connection closing" response. Otherwise, any remaining

 text can be used to satisfy the RECEIVE.

 CLOSING STATE

 LAST-ACK STATE

 TIME-WAIT STATE

 Return "error: connection closing".

 [Page 59]

 September 1981

Transmission Control Protocol

Functional Specification

 CLOSE Call

 CLOSE Call

 CLOSED STATE (i.e., TCB does not exist)

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

 "error: connection illegal for this process".

 Otherwise, return "error: connection does not exist".

 LISTEN STATE

 Any outstanding RECEIVEs are returned with "error: closing"

 responses. Delete TCB, enter CLOSED state, and return.

 SYN-SENT STATE

 Delete the TCB and return "error: closing" responses to any

 queued SENDs, or RECEIVEs.

 SYN-RECEIVED STATE

 If no SENDs have been issued and there is no pending data to send,

 then form a FIN segment and send it, and enter FIN-WAIT-1 state;

 otherwise queue for processing after entering ESTABLISHED state.

 ESTABLISHED STATE

 Queue this until all preceding SENDs have been segmentized, then

 form a FIN segment and send it. In any case, enter FIN-WAIT-1

 state.

 FIN-WAIT-1 STATE

 FIN-WAIT-2 STATE

 Strictly speaking, this is an error and should receive a "error:

 connection closing" response. An "ok" response would be

 acceptable, too, as long as a second FIN is not emitted (the first

 FIN may be retransmitted though).

[Page 60]

September 1981

 Transmission Control Protocol

 Functional Specification

CLOSE Call

 CLOSE-WAIT STATE

 Queue this request until all preceding SENDs have been

 segmentized; then send a FIN segment, enter CLOSING state.

 CLOSING STATE

 LAST-ACK STATE

 TIME-WAIT STATE

 Respond with "error: connection closing".

 [Page 61]

 September 1981

Transmission Control Protocol

Functional Specification

 ABORT Call

 ABORT Call

 CLOSED STATE (i.e., TCB does not exist)

 If the user should not have access to such a connection, return

 "error: connection illegal for this process".

 Otherwise return "error: connection does not exist".

 LISTEN STATE

 Any outstanding RECEIVEs should be returned with "error:

 connection reset" responses. Delete TCB, enter CLOSED state, and

 return.

 SYN-SENT STATE

 All queued SENDs and RECEIVEs should be given "connection reset"

 notification, delete the TCB, enter CLOSED state, and return.

 SYN-RECEIVED STATE

 ESTABLISHED STATE

 FIN-WAIT-1 STATE

 FIN-WAIT-2 STATE

 CLOSE-WAIT STATE

 Send a reset segment:

 <SEQ=SND.NXT><CTL=RST>

 All queued SENDs and RECEIVEs should be given "connection reset"

 notification; all segments queued for transmission (except for the

 RST formed above) or retransmission should be flushed, delete the

 TCB, enter CLOSED state, and return.

 CLOSING STATE

 LAST-ACK STATE

 TIME-WAIT STATE

 Respond with "ok" and delete the TCB, enter CLOSED state, and

 return.

[Page 62]

September 1981

 Transmission Control Protocol

 Functional Specification

STATUS Call

 STATUS Call

 CLOSED STATE (i.e., TCB does not exist)

 If the user should not have access to such a connection, return

 "error: connection illegal for this process".

 Otherwise return "error: connection does not exist".

 LISTEN STATE

 Return "state = LISTEN", and the TCB pointer.

 SYN-SENT STATE

 Return "state = SYN-SENT", and the TCB pointer.

 SYN-RECEIVED STATE

 Return "state = SYN-RECEIVED", and the TCB pointer.

 ESTABLISHED STATE

 Return "state = ESTABLISHED", and the TCB pointer.

 FIN-WAIT-1 STATE

 Return "state = FIN-WAIT-1", and the TCB pointer.

 FIN-WAIT-2 STATE

 Return "state = FIN-WAIT-2", and the TCB pointer.

 CLOSE-WAIT STATE

 Return "state = CLOSE-WAIT", and the TCB pointer.

 CLOSING STATE

 Return "state = CLOSING", and the TCB pointer.

 LAST-ACK STATE

 Return "state = LAST-ACK", and the TCB pointer.

 [Page 63]

 September 1981

Transmission Control Protocol

Functional Specification

 STATUS Call

 TIME-WAIT STATE

 Return "state = TIME-WAIT", and the TCB pointer.

[Page 64]

September 1981

 Transmission Control Protocol

 Functional Specification

SEGMENT ARRIVES

 SEGMENT ARRIVES

 If the state is CLOSED (i.e., TCB does not exist) then

 all data in the incoming segment is discarded. An incoming

 segment containing a RST is discarded. An incoming segment not

 containing a RST causes a RST to be sent in response. The

 acknowledgment and sequence field values are selected to make the

 reset sequence acceptable to the TCP that sent the offending

 segment.

 If the ACK bit is off, sequence number zero is used,

 <SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

 If the ACK bit is on,

 <SEQ=SEG.ACK><CTL=RST>

 Return.

 If the state is LISTEN then

 first check for an RST

 An incoming RST should be ignored. Return.

 second check for an ACK

 Any acknowledgment is bad if it arrives on a connection still in

 the LISTEN state. An acceptable reset segment should be formed

 for any arriving ACK-bearing segment. The RST should be

 formatted as follows:

 <SEQ=SEG.ACK><CTL=RST>

 Return.

 third check for a SYN

 If the SYN bit is set, check the security. If the

 security/compartment on the incoming segment does not exactly

 match the security/compartment in the TCB then send a reset and

 return.

 <SEQ=SEG.ACK><CTL=RST>

 [Page 65]

 September 1981

Transmission Control Protocol

Functional Specification

 SEGMENT ARRIVES

 If the SEG.PRC is greater than the TCB.PRC then if allowed by

 the user and the system set TCB.PRC<-SEG.PRC, if not allowed

 send a reset and return.

 <SEQ=SEG.ACK><CTL=RST>

 If the SEG.PRC is less than the TCB.PRC then continue.

 Set RCV.NXT to SEG.SEQ+1, IRS is set to SEG.SEQ and any other

 control or text should be queued for processing later. ISS

 should be selected and a SYN segment sent of the form:

 <SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>

 SND.NXT is set to ISS+1 and SND.UNA to ISS. The connection

 state should be changed to SYN-RECEIVED. Note that any other

 incoming control or data (combined with SYN) will be processed

 in the SYN-RECEIVED state, but processing of SYN and ACK should

 not be repeated. If the listen was not fully specified (i.e.,

 the foreign socket was not fully specified), then the

 unspecified fields should be filled in now.

 fourth other text or control

 Any other control or text-bearing segment (not containing SYN)

 must have an ACK and thus would be discarded by the ACK

 processing. An incoming RST segment could not be valid, since

 it could not have been sent in response to anything sent by this

 incarnation of the connection. So you are unlikely to get here,

 but if you do, drop the segment, and return.

 If the state is SYN-SENT then

 first check the ACK bit

 If the ACK bit is set

 If SEG.ACK =< ISS, or SEG.ACK > SND.NXT, send a reset (unless

 the RST bit is set, if so drop the segment and return)

 <SEQ=SEG.ACK><CTL=RST>

 and discard the segment. Return.

 If SND.UNA =< SEG.ACK =< SND.NXT then the ACK is acceptable.

 second check the RST bit

[Page 66]

September 1981

 Transmission Control Protocol

 Functional Specification

SEGMENT ARRIVES

 If the RST bit is set

 If the ACK was acceptable then signal the user "error:

 connection reset", drop the segment, enter CLOSED state,

 delete TCB, and return. Otherwise (no ACK) drop the segment

 and return.

 third check the security and precedence

 If the security/compartment in the segment does not exactly

 match the security/compartment in the TCB, send a reset

 If there is an ACK

 <SEQ=SEG.ACK><CTL=RST>

 Otherwise

 <SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

 If there is an ACK

 The precedence in the segment must match the precedence in the

 TCB, if not, send a reset

 <SEQ=SEG.ACK><CTL=RST>

 If there is no ACK

 If the precedence in the segment is higher than the precedence

 in the TCB then if allowed by the user and the system raise

 the precedence in the TCB to that in the segment, if not

 allowed to raise the prec then send a reset.

 <SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

 If the precedence in the segment is lower than the precedence

 in the TCB continue.

 If a reset was sent, discard the segment and return.

 fourth check the SYN bit

 This step should be reached only if the ACK is ok, or there is

 no ACK, and it the segment did not contain a RST.

 If the SYN bit is on and the security/compartment and precedence

 [Page 67]

 September 1981

Transmission Control Protocol

Functional Specification

 SEGMENT ARRIVES

 are acceptable then, RCV.NXT is set to SEG.SEQ+1, IRS is set to

 SEG.SEQ. SND.UNA should be advanced to equal SEG.ACK (if there

 is an ACK), and any segments on the retransmission queue which

 are thereby acknowledged should be removed.

 If SND.UNA > ISS (our SYN has been ACKed), change the connection

 state to ESTABLISHED, form an ACK segment

 <SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

 and send it. Data or controls which were queued for

 transmission may be included. If there are other controls or

 text in the segment then continue processing at the sixth step

 below where the URG bit is checked, otherwise return.

 Otherwise enter SYN-RECEIVED, form a SYN,ACK segment

 <SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>

 and send it. If there are other controls or text in the

 segment, queue them for processing after the ESTABLISHED state

 has been reached, return.

 fifth, if neither of the SYN or RST bits is set then drop the

 segment and return.

[Page 68]

September 1981

 Transmission Control Protocol

 Functional Specification

SEGMENT ARRIVES

 Otherwise,

 first check sequence number

 SYN-RECEIVED STATE

 ESTABLISHED STATE

 FIN-WAIT-1 STATE

 FIN-WAIT-2 STATE

 CLOSE-WAIT STATE

 CLOSING STATE

 LAST-ACK STATE

 TIME-WAIT STATE

 Segments are processed in sequence. Initial tests on arrival

 are used to discard old duplicates, but further processing is

 done in SEG.SEQ order. If a segment's contents straddle the

 boundary between old and new, only the new parts should be

 processed.

 There are four cases for the acceptability test for an incoming

 segment:

 Segment Receive Test

 Length Window

 ------- ------- -------------------------------------------

 0 0 SEG.SEQ = RCV.NXT

 0 >0 RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND

 >0 0 not acceptable

 >0 >0 RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND

 or RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

 If the RCV.WND is zero, no segments will be acceptable, but

 special allowance should be made to accept valid ACKs, URGs and

 RSTs.

 If an incoming segment is not acceptable, an acknowledgment

 should be sent in reply (unless the RST bit is set, if so drop

 the segment and return):

 <SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

 After sending the acknowledgment, drop the unacceptable segment

 and return.

 [Page 69]

 September 1981

Transmission Control Protocol

Functional Specification

 SEGMENT ARRIVES

 In the following it is assumed that the segment is the idealized

 segment that begins at RCV.NXT and does not exceed the window.

 One could tailor actual segments to fit this assumption by

 trimming off any portions that lie outside the window (including

 SYN and FIN), and only processing further if the segment then

 begins at RCV.NXT. Segments with higher begining sequence

 numbers may be held for later processing.

 second check the RST bit,

 SYN-RECEIVED STATE

 If the RST bit is set

 If this connection was initiated with a passive OPEN (i.e.,

 came from the LISTEN state), then return this connection to

 LISTEN state and return. The user need not be informed. If

 this connection was initiated with an active OPEN (i.e., came

 from SYN-SENT state) then the connection was refused, signal

 the user "connection refused". In either case, all segments

 on the retransmission queue should be removed. And in the

 active OPEN case, enter the CLOSED state and delete the TCB,

 and return.

 ESTABLISHED

 FIN-WAIT-1

 FIN-WAIT-2

 CLOSE-WAIT

 If the RST bit is set then, any outstanding RECEIVEs and SEND

 should receive "reset" responses. All segment queues should be

 flushed. Users should also receive an unsolicited general

 "connection reset" signal. Enter the CLOSED state, delete the

 TCB, and return.

 CLOSING STATE

 LAST-ACK STATE

 TIME-WAIT

 If the RST bit is set then, enter the CLOSED state, delete the

 TCB, and return.

[Page 70]

September 1981

 Transmission Control Protocol

 Functional Specification

SEGMENT ARRIVES

 third check security and precedence

 SYN-RECEIVED

 If the security/compartment and precedence in the segment do not

 exactly match the security/compartment and precedence in the TCB

 then send a reset, and return.

 ESTABLISHED STATE

 If the security/compartment and precedence in the segment do not

 exactly match the security/compartment and precedence in the TCB

 then send a reset, any outstanding RECEIVEs and SEND should

 receive "reset" responses. All segment queues should be

 flushed. Users should also receive an unsolicited general

 "connection reset" signal. Enter the CLOSED state, delete the

 TCB, and return.

 Note this check is placed following the sequence check to prevent

 a segment from an old connection between these ports with a

 different security or precedence from causing an abort of the

 current connection.

 fourth, check the SYN bit,

 SYN-RECEIVED

 ESTABLISHED STATE

 FIN-WAIT STATE-1

 FIN-WAIT STATE-2

 CLOSE-WAIT STATE

 CLOSING STATE

 LAST-ACK STATE

 TIME-WAIT STATE

 If the SYN is in the window it is an error, send a reset, any

 outstanding RECEIVEs and SEND should receive "reset" responses,

 all segment queues should be flushed, the user should also

 receive an unsolicited general "connection reset" signal, enter

 the CLOSED state, delete the TCB, and return.

 If the SYN is not in the window this step would not be reached

 and an ack would have been sent in the first step (sequence

 number check).

 [Page 71]

 September 1981

Transmission Control Protocol

Functional Specification

 SEGMENT ARRIVES

 fifth check the ACK field,

 if the ACK bit is off drop the segment and return

 if the ACK bit is on

 SYN-RECEIVED STATE

 If SND.UNA =< SEG.ACK =< SND.NXT then enter ESTABLISHED state

 and continue processing.

 If the segment acknowledgment is not acceptable, form a

 reset segment,

 <SEQ=SEG.ACK><CTL=RST>

 and send it.

 ESTABLISHED STATE

 If SND.UNA < SEG.ACK =< SND.NXT then, set SND.UNA <- SEG.ACK.

 Any segments on the retransmission queue which are thereby

 entirely acknowledged are removed. Users should receive

 positive acknowledgments for buffers which have been SENT and

 fully acknowledged (i.e., SEND buffer should be returned with

 "ok" response). If the ACK is a duplicate

 (SEG.ACK < SND.UNA), it can be ignored. If the ACK acks

 something not yet sent (SEG.ACK > SND.NXT) then send an ACK,

 drop the segment, and return.

 If SND.UNA < SEG.ACK =< SND.NXT, the send window should be

 updated. If (SND.WL1 < SEG.SEQ or (SND.WL1 = SEG.SEQ and

 SND.WL2 =< SEG.ACK)), set SND.WND <- SEG.WND, set

 SND.WL1 <- SEG.SEQ, and set SND.WL2 <- SEG.ACK.

 Note that SND.WND is an offset from SND.UNA, that SND.WL1

 records the sequence number of the last segment used to update

 SND.WND, and that SND.WL2 records the acknowledgment number of

 the last segment used to update SND.WND. The check here

 prevents using old segments to update the window.

[Page 72]

September 1981

 Transmission Control Protocol

 Functional Specification

SEGMENT ARRIVES

 FIN-WAIT-1 STATE

 In addition to the processing for the ESTABLISHED state, if

 our FIN is now acknowledged then enter FIN-WAIT-2 and continue

 processing in that state.

 FIN-WAIT-2 STATE

 In addition to the processing for the ESTABLISHED state, if

 the retransmission queue is empty, the user's CLOSE can be

 acknowledged ("ok") but do not delete the TCB.

 CLOSE-WAIT STATE

 Do the same processing as for the ESTABLISHED state.

 CLOSING STATE

 In addition to the processing for the ESTABLISHED state, if

 the ACK acknowledges our FIN then enter the TIME-WAIT state,

 otherwise ignore the segment.

 LAST-ACK STATE

 The only thing that can arrive in this state is an

 acknowledgment of our FIN. If our FIN is now acknowledged,

 delete the TCB, enter the CLOSED state, and return.

 TIME-WAIT STATE

 The only thing that can arrive in this state is a

 retransmission of the remote FIN. Acknowledge it, and restart

 the 2 MSL timeout.

 sixth, check the URG bit,

 ESTABLISHED STATE

 FIN-WAIT-1 STATE

 FIN-WAIT-2 STATE

 If the URG bit is set, RCV.UP <- max(RCV.UP,SEG.UP), and signal

 the user that the remote side has urgent data if the urgent

 pointer (RCV.UP) is in advance of the data consumed. If the

 user has already been signaled (or is still in the "urgent

 mode") for this continuous sequence of urgent data, do not

 signal the user again.

 [Page 73]

 September 1981

Transmission Control Protocol

Functional Specification

 SEGMENT ARRIVES

 CLOSE-WAIT STATE

 CLOSING STATE

 LAST-ACK STATE

 TIME-WAIT

 This should not occur, since a FIN has been received from the

 remote side. Ignore the URG.

 seventh, process the segment text,

 ESTABLISHED STATE

 FIN-WAIT-1 STATE

 FIN-WAIT-2 STATE

 Once in the ESTABLISHED state, it is possible to deliver segment

 text to user RECEIVE buffers. Text from segments can be moved

 into buffers until either the buffer is full or the segment is

 empty. If the segment empties and carries an PUSH flag, then

 the user is informed, when the buffer is returned, that a PUSH

 has been received.

 When the TCP takes responsibility for delivering the data to the

 user it must also acknowledge the receipt of the data.

 Once the TCP takes responsibility for the data it advances

 RCV.NXT over the data accepted, and adjusts RCV.WND as

 apporopriate to the current buffer availability. The total of

 RCV.NXT and RCV.WND should not be reduced.

 Please note the window management suggestions in section 3.7.

 Send an acknowledgment of the form:

 <SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

 This acknowledgment should be piggybacked on a segment being

 transmitted if possible without incurring undue delay.

[Page 74]

September 1981

 Transmission Control Protocol

 Functional Specification

SEGMENT ARRIVES

 CLOSE-WAIT STATE

 CLOSING STATE

 LAST-ACK STATE

 TIME-WAIT STATE

 This should not occur, since a FIN has been received from the

 remote side. Ignore the segment text.

 eighth, check the FIN bit,

 Do not process the FIN if the state is CLOSED, LISTEN or SYN-SENT

 since the SEG.SEQ cannot be validated; drop the segment and

 return.

 If the FIN bit is set, signal the user "connection closing" and

 return any pending RECEIVEs with same message, advance RCV.NXT

 over the FIN, and send an acknowledgment for the FIN. Note that

 FIN implies PUSH for any segment text not yet delivered to the

 user.

 SYN-RECEIVED STATE

 ESTABLISHED STATE

 Enter the CLOSE-WAIT state.

 FIN-WAIT-1 STATE

 If our FIN has been ACKed (perhaps in this segment), then

 enter TIME-WAIT, start the time-wait timer, turn off the other

 timers; otherwise enter the CLOSING state.

 FIN-WAIT-2 STATE

 Enter the TIME-WAIT state. Start the time-wait timer, turn

 off the other timers.

 CLOSE-WAIT STATE

 Remain in the CLOSE-WAIT state.

 CLOSING STATE

 Remain in the CLOSING state.

 LAST-ACK STATE

 Remain in the LAST-ACK state.

 [Page 75]

 September 1981

Transmission Control Protocol

Functional Specification

 SEGMENT ARRIVES

 TIME-WAIT STATE

 Remain in the TIME-WAIT state. Restart the 2 MSL time-wait

 timeout.

 and return.

[Page 76]

September 1981

 Transmission Control Protocol

 Functional Specification

USER TIMEOUT

 USER TIMEOUT

 For any state if the user timeout expires, flush all queues, signal

 the user "error: connection aborted due to user timeout" in general

 and for any outstanding calls, delete the TCB, enter the CLOSED

 state and return.

 RETRANSMISSION TIMEOUT

 For any state if the retransmission timeout expires on a segment in

 the retransmission queue, send the segment at the front of the

 retransmission queue again, reinitialize the retransmission timer,

 and return.

 TIME-WAIT TIMEOUT

 If the time-wait timeout expires on a connection delete the TCB,

 enter the CLOSED state and return.

 [Page 77]

 September 1981

Transmission Control Protocol

[Page 78]

September 1981

 Transmission Control Protocol

 GLOSSARY

1822

 BBN Report 1822, "The Specification of the Interconnection of

 a Host and an IMP". The specification of interface between a

 host and the ARPANET.

ACK

 A control bit (acknowledge) occupying no sequence space, which

 indicates that the acknowledgment field of this segment

 specifies the next sequence number the sender of this segment

 is expecting to receive, hence acknowledging receipt of all

 previous sequence numbers.

ARPANET message

 The unit of transmission between a host and an IMP in the

 ARPANET. The maximum size is about 1012 octets (8096 bits).

ARPANET packet

 A unit of transmission used internally in the ARPANET between

 IMPs. The maximum size is about 126 octets (1008 bits).

connection

 A logical communication path identified by a pair of sockets.

datagram

 A message sent in a packet switched computer communications

 network.

Destination Address

 The destination address, usually the network and host

 identifiers.

FIN

 A control bit (finis) occupying one sequence number, which

 indicates that the sender will send no more data or control

 occupying sequence space.

fragment

 A portion of a logical unit of data, in particular an internet

 fragment is a portion of an internet datagram.

FTP

 A file transfer protocol.

 [Page 79]

 September 1981

Transmission Control Protocol

Glossary

header

 Control information at the beginning of a message, segment,

 fragment, packet or block of data.

host

 A computer. In particular a source or destination of messages

 from the point of view of the communication network.

Identification

 An Internet Protocol field. This identifying value assigned

 by the sender aids in assembling the fragments of a datagram.

IMP

 The Interface Message Processor, the packet switch of the

 ARPANET.

internet address

 A source or destination address specific to the host level.

internet datagram

 The unit of data exchanged between an internet module and the

 higher level protocol together with the internet header.

internet fragment

 A portion of the data of an internet datagram with an internet

 header.

IP

 Internet Protocol.

IRS

 The Initial Receive Sequence number. The first sequence

 number used by the sender on a connection.

ISN

 The Initial Sequence Number. The first sequence number used

 on a connection, (either ISS or IRS). Selected on a clock

 based procedure.

ISS

 The Initial Send Sequence number. The first sequence number

 used by the sender on a connection.

leader

 Control information at the beginning of a message or block of

 data. In particular, in the ARPANET, the control information

 on an ARPANET message at the host-IMP interface.

[Page 80]

September 1981

 Transmission Control Protocol

 Glossary

left sequence

 This is the next sequence number to be acknowledged by the

 data receiving TCP (or the lowest currently unacknowledged

 sequence number) and is sometimes referred to as the left edge

 of the send window.

local packet

 The unit of transmission within a local network.

module

 An implementation, usually in software, of a protocol or other

 procedure.

MSL

 Maximum Segment Lifetime, the time a TCP segment can exist in

 the internetwork system. Arbitrarily defined to be 2 minutes.

octet

 An eight bit byte.

Options

 An Option field may contain several options, and each option

 may be several octets in length. The options are used

 primarily in testing situations; for example, to carry

 timestamps. Both the Internet Protocol and TCP provide for

 options fields.

packet

 A package of data with a header which may or may not be

 logically complete. More often a physical packaging than a

 logical packaging of data.

port

 The portion of a socket that specifies which logical input or

 output channel of a process is associated with the data.

process

 A program in execution. A source or destination of data from

 the point of view of the TCP or other host-to-host protocol.

PUSH

 A control bit occupying no sequence space, indicating that

 this segment contains data that must be pushed through to the

 receiving user.

RCV.NXT

 receive next sequence number

 [Page 81]

 September 1981

Transmission Control Protocol

Glossary

RCV.UP

 receive urgent pointer

RCV.WND

 receive window

receive next sequence number

 This is the next sequence number the local TCP is expecting to

 receive.

receive window

 This represents the sequence numbers the local (receiving) TCP

 is willing to receive. Thus, the local TCP considers that

 segments overlapping the range RCV.NXT to

 RCV.NXT + RCV.WND - 1 carry acceptable data or control.

 Segments containing sequence numbers entirely outside of this

 range are considered duplicates and discarded.

RST

 A control bit (reset), occupying no sequence space, indicating

 that the receiver should delete the connection without further

 interaction. The receiver can determine, based on the

 sequence number and acknowledgment fields of the incoming

 segment, whether it should honor the reset command or ignore

 it. In no case does receipt of a segment containing RST give

 rise to a RST in response.

RTP

 Real Time Protocol: A host-to-host protocol for communication

 of time critical information.

SEG.ACK

 segment acknowledgment

SEG.LEN

 segment length

SEG.PRC

 segment precedence value

SEG.SEQ

 segment sequence

SEG.UP

 segment urgent pointer field

[Page 82]

September 1981

 Transmission Control Protocol

 Glossary

SEG.WND

 segment window field

segment

 A logical unit of data, in particular a TCP segment is the

 unit of data transfered between a pair of TCP modules.

segment acknowledgment

 The sequence number in the acknowledgment field of the

 arriving segment.

segment length

 The amount of sequence number space occupied by a segment,

 including any controls which occupy sequence space.

segment sequence

 The number in the sequence field of the arriving segment.

send sequence

 This is the next sequence number the local (sending) TCP will

 use on the connection. It is initially selected from an

 initial sequence number curve (ISN) and is incremented for

 each octet of data or sequenced control transmitted.

send window

 This represents the sequence numbers which the remote

 (receiving) TCP is willing to receive. It is the value of the

 window field specified in segments from the remote (data

 receiving) TCP. The range of new sequence numbers which may

 be emitted by a TCP lies between SND.NXT and

 SND.UNA + SND.WND - 1. (Retransmissions of sequence numbers

 between SND.UNA and SND.NXT are expected, of course.)

SND.NXT

 send sequence

SND.UNA

 left sequence

SND.UP

 send urgent pointer

SND.WL1

 segment sequence number at last window update

SND.WL2

 segment acknowledgment number at last window update

 [Page 83]

 September 1981

Transmission Control Protocol

Glossary

SND.WND

 send window

socket

 An address which specifically includes a port identifier, that

 is, the concatenation of an Internet Address with a TCP port.

Source Address

 The source address, usually the network and host identifiers.

SYN

 A control bit in the incoming segment, occupying one sequence

 number, used at the initiation of a connection, to indicate

 where the sequence numbering will start.

TCB

 Transmission control block, the data structure that records

 the state of a connection.

TCB.PRC

 The precedence of the connection.

TCP

 Transmission Control Protocol: A host-to-host protocol for

 reliable communication in internetwork environments.

TOS

 Type of Service, an Internet Protocol field.

Type of Service

 An Internet Protocol field which indicates the type of service

 for this internet fragment.

URG

 A control bit (urgent), occupying no sequence space, used to

 indicate that the receiving user should be notified to do

 urgent processing as long as there is data to be consumed with

 sequence numbers less than the value indicated in the urgent

 pointer.

urgent pointer

 A control field meaningful only when the URG bit is on. This

 field communicates the value of the urgent pointer which

 indicates the data octet associated with the sending user's

 urgent call.

[Page 84]

September 1981

 Transmission Control Protocol

 REFERENCES

[1] Cerf, V., and R. Kahn, "A Protocol for Packet Network

 Intercommunication", IEEE Transactions on Communications,

 Vol. COM-22, No. 5, pp 637-648, May 1974.

[2] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program

 Protocol Specification", [RFC 791](http://www.faqs.org/rfcs/rfc791.html), USC/Information Sciences

 Institute, September 1981.

[3] Dalal, Y. and C. Sunshine, "Connection Management in Transport

 Protocols", Computer Networks, Vol. 2, No. 6, pp. 454-473,

 December 1978.

[4] Postel, J., "Assigned Numbers", [RFC 790](http://www.faqs.org/rfcs/rfc790.html), USC/Information Sciences

 Institute, September 1981.

[Comment on RFC 793](http://www.faqs.org/rfccomment.php?rfcnum=793)

|  |
| --- |
| Comments about this RFC:* [RFC 793: what do you mean by RFC ?](http://www.faqs.org/qa/rfcc-1892.html) by hari (4/13/2005)
 |

|  |  |  |
| --- | --- | --- |
| Previous: [RFC 0792 - Internet Control Message Protocol](http://www.faqs.org/rfcs/rfc792.html)  |   | Next: [RFC 0794 - Pre-emption](http://www.faqs.org/rfcs/rfc794.html)  |

[ [RFC Index](http://www.faqs.org/rfcs/) | [RFC Search](http://www.faqs.org/rfcs/rfcsearch.html) | [Usenet FAQs](http://www.faqs.org/faqs/) | [Web FAQs](http://www.faqs.org/contrib/) | [Documents](http://www.faqs.org/docs/) | [Cities](http://www.city-data.com/) ]