DECLARATION OF SANDY GINOZA FOR IETF

RFC 2401: SECURITY ARCHITECTURE FOR THE INTERNET PROTOCOL

RFC: 793: TRANSMISSION CONTROL PROTOCOL DARPA

I, Sandy Ginoza, hereby declare that all statements made herein are of my own knowledge and are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code:

1. I am an employee of Association Management Solutions, LLC (AMS), which acts under contract to the IETF Administration LLC (IETF) as the operator of the RFC Production Center. The RFC Production Center is part of the "RFC Editor" function, which prepares documents for publication and places files in an online repository for the authoritative Request for Comments (RFC) series of documents (RFC Series), and preserves records relating to these documents. The RFC Series includes, among other things, the series of Internet standards developed by the IETF. I hold the position of Director of the RFC Production Center. I began employment with AMS in this capacity on 6 January 2010.

2. Among my responsibilities as Director of the RFC Production Center, I act as the custodian of records relating to the RFC Series, and I am familiar with the record keeping practices relating to the RFC Series, including the creation and maintenance of such records.

3. From June 1999 to 5 January 2010, I was an employee of the Information Sciences Institute at University of Southern California (ISI). I held various position titles with the RFC Editor project at ISI, ending with Senior Editor.

4. The RFC Editor function was conducted by ISI under contract to the United States government prior to 1998. In 1998, ISOC, in furtherance of its IETF activity, entered into the first in a series of contracts with ISI providing for ISI's performance of the RFC Editor function. Beginning in 2010, certain aspects of the RFC Editor function were assumed by the RFC Production Center operation of AMS under contract to ISOC (acting through its IETF function and, in particular, the IETF Administrative Oversight Committee (now the IETF Administration LLC (IETF)). The business records of the RFC Editor function as it was conducted by ISI are currently housed on the computer systems of AMS, as contractor to the IETF.

5. I make this declaration based on my personal knowledge and information contained in the business records of the RFC Editor as they are currently housed at AMS, or confirmation with other responsible RFC Editor personnel with such knowledge.

6. Prior to 1998, the RFC Editor's regular practice was to publish RFCs, making them available from a repository via FTP. When a new RFC was published, an announcement of its publication, with information on how to access the RFC, would be typically sent out within 24 hours of the publication.

7. Any RFC published on the RFC Editor website or via FTP was reasonably accessible to the public and was disseminated or otherwise available to the extent that persons interested and ordinarily skilled in the subject matter or art exercising reasonable diligence could have located it. In particular, the RFCs were indexed and placed in a public repository.

8. The RFCs are kept in an online repository in the course of the RFC Editor's regularly conducted activity and ordinary course of business. The records are made pursuant to established procedures and are relied upon by the RFC Editor in the performance of its functions.

9. It is the regular practice of the RFC Editor to make and keep the RFC records.

10. Based on the business records for the RFC Editor and the RFC Editor's course of conduct in publishing RFCs, I have determined that the publication date of RFC 2401 was no later than November 1998, at which time it was reasonably accessible to the public either on the RFC Editor website or via FTP from a repository. A copy of that RFC is attached to this declaration as an exhibit.

11. Based on the business records for the RFC Editor and the RFC Editor's course of conduct in publishing RFCs, I have determined that the publication date of RFC 793 was no later than October 1992, at which time it was reasonably accessible to the public either on the RFC Editor website or via FTP from a repository. A copy of that RFC is attached to this declaration as an exhibit.

Pursuant to Section 1746 of Title 28 of United States Code, I declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct and that the foregoing is based upon personal knowledge and information and is believed to be true.

Date: 14 February 2019

By: ______________________
Sandy Ginoza

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

TABLE OF CONTENTS

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
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.

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 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.

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.

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.

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 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.

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.

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 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.

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 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.

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.

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 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.

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
```

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

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.

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).

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.

```
                              +---------+ ---------\      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.

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).

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

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 Length	Receive Window	Test
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.

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
```

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 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 10'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 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 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!).

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.

```
    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 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 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.

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.

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.

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.

```
      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.

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 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 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 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 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.

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 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 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 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 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,

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

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.

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).

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.

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".

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".

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.

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.

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.

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".

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).

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".

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.

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.

TIME-WAIT STATE

Return "state = TIME-WAIT", and the TCB pointer.

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>

```
        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
```

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 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.

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 Length	Receive Window	Test
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.

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.

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).

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.

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.

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.

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.

TIME-WAIT STATE

Remain in the TIME-WAIT state. Restart the 2 MSL time-wait timeout.

and return.

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.

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.

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.

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

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

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

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.

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, 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, USC/Information Sciences Institute, September 1981.

Network Working Group
Request for Comments: 2401
Obsoletes: 1825
Category: Standards Track

S. Kent
BBN Corp
R. Atkinson
@Home Network

November 1998

Security Architecture for the Internet Protocol

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Table of Contents

1. Introduction

1.1 Summary of Contents of Document

This memo specifies the base architecture for IPsec compliant systems. The goal of the architecture is to provide various security services for traffic at the IP layer, in both the IPv4 and IPv6 environments. This document describes the goals of such systems, their components and how they fit together with each other and into the IP environment. It also describes the security services offered by the IPsec protocols, and how these services can be employed in the IP environment. This document does not address all aspects of IPsec architecture. Subsequent documents will address additional architectural details of a more advanced nature, e.g., use of IPsec in NAT environments and more complete support for IP multicast. The following fundamental components of the IPsec security architecture are discussed in terms of their underlying, required functionality. Additional RFCs (see Section 1.3 for pointers to other documents) define the protocols in (a), (c), and (d).

a. Security Protocols -- Authentication Header (AH) and Encapsulating Security Payload (ESP)
b. Security Associations -- what they are and how they work, how they are managed, associated processing
c. Key Management -- manual and automatic (The Internet Key Exchange (IKE))
d. Algorithms for authentication and encryption

This document is not an overall Security Architecture for the Internet; it addresses security only at the IP layer, provided through the use of a combination of cryptographic and protocol security mechanisms.

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in RFC 2119 [Bra97].

1.2 Audience

The target audience for this document includes implementers of this IP security technology and others interested in gaining a general background understanding of this system. In particular, prospective users of this technology (end users or system administrators) are part of the target audience. A glossary is provided as an appendix to help fill in gaps in background/vocabulary. This document assumes that the reader is familiar with the Internet Protocol, related networking technology, and general security terms and concepts.

1.3 Related Documents

As mentioned above, other documents provide detailed definitions of some of the components of IPsec and of their inter-relationship. They include RFCs on the following topics:

a. "IP Security Document Roadmap" [TDG97] -- a document providing guidelines for specifications describing encryption and authentication algorithms used in this system.
b. security protocols -- RFCs describing the Authentication Header (AH) [KA98a] and Encapsulating Security Payload (ESP) [KA98b] protocols.
c. algorithms for authentication and encryption -- a separate RFC for each algorithm.
d. automatic key management -- RFCs on "The Internet Key Exchange (IKE)" [HC98], "Internet Security Association and Key Management Protocol (ISAKMP)" [MSST97],"The OAKLEY Key Determination Protocol" [Orm97], and "The Internet IP Security Domain of Interpretation for ISAKMP" [Pip98].

2. Design Objectives

2.1 Goals/Objectives/Requirements/Problem Description

IPsec is designed to provide interoperable, high quality, cryptographically-based security for IPv4 and IPv6. The set of security services offered includes access control, connectionless integrity, data origin authentication, protection against replays (a form of partial sequence integrity), confidentiality (encryption), and limited traffic flow confidentiality. These services are provided at the IP layer, offering protection for IP and/or upper layer protocols.

These objectives are met through the use of two traffic security protocols, the Authentication Header (AH) and the Encapsulating Security Payload (ESP), and through the use of cryptographic key management procedures and protocols. The set of IPsec protocols employed in any context, and the ways in which they are employed, will be determined by the security and system requirements of users, applications, and/or sites/organizations.

When these mechanisms are correctly implemented and deployed, they ought not to adversely affect users, hosts, and other Internet components that do not employ these security mechanisms for protection of their traffic. These mechanisms also are designed to be algorithm-independent. This modularity permits selection of different sets of algorithms without affecting the other parts of the implementation. For example, different user communities may select different sets of algorithms (creating cliques) if required.

A standard set of default algorithms is specified to facilitate interoperability in the global Internet. The use of these algorithms, in conjunction with IPsec traffic protection and key management protocols, is intended to permit system and application developers to deploy high quality, Internet layer, cryptographic security technology.

2.2 Caveats and Assumptions

The suite of IPsec protocols and associated default algorithms are designed to provide high quality security for Internet traffic. However, the security offered by use of these protocols ultimately depends on the quality of the their implementation, which is outside the scope of this set of standards. Moreover, the security of a computer system or network is a function of many factors, including personnel, physical, procedural, compromising emanations, and computer security practices. Thus IPsec is only one part of an overall system security architecture.

Finally, the security afforded by the use of IPsec is critically dependent on many aspects of the operating environment in which the IPsec implementation executes. For example, defects in OS security, poor quality of random number sources, sloppy system management protocols and practices, etc. can all degrade the security provided by IPsec. As above, none of these environmental attributes are within the scope of this or other IPsec standards.

3. System Overview

This section provides a high level description of how IPsec works, the components of the system, and how they fit together to provide the security services noted above. The goal of this description is to enable the reader to "picture" the overall process/system, see how it fits into the IP environment, and to provide context for later sections of this document, which describe each of the components in more detail.

An IPsec implementation operates in a host or a security gateway environment, affording protection to IP traffic. The protection offered is based on requirements defined by a Security Policy Database (SPD) established and maintained by a user or system administrator, or by an application operating within constraints established by either of the above. In general, packets are selected for one of three processing modes based on IP and transport layer header information (Selectors, Section 4.4.2) matched against entries in the database (SPD). Each packet is either afforded IPsec security services, discarded, or allowed to bypass IPsec, based on the applicable database policies identified by the Selectors.

3.1 What IPsec Does

IPsec provides security services at the IP layer by enabling a system to select required security protocols, determine the algorithm(s) to use for the service(s), and put in place any cryptographic keys required to provide the requested services. IPsec can be used to protect one or more "paths" between a pair of hosts, between a pair of security gateways, or between a security gateway and a host. (The term "security gateway" is used throughout the IPsec documents to refer to an intermediate system that implements IPsec protocols. For example, a router or a firewall implementing IPsec is a security gateway.)

The set of security services that IPsec can provide includes access control, connectionless integrity, data origin authentication, rejection of replayed packets (a form of partial sequence integrity), confidentiality (encryption), and limited traffic flow confidentiality. Because these services are provided at the IP layer, they can be used by any higher layer protocol, e.g., TCP, UDP, ICMP, BGP, etc.

The IPsec DOI also supports negotiation of IP compression [SMPT98], motivated in part by the observation that when encryption is employed within IPsec, it prevents effective compression by lower protocol layers.

3.2 How IPsec Works

IPsec uses two protocols to provide traffic security -- Authentication Header (AH) and Encapsulating Security Payload (ESP). Both protocols are described in more detail in their respective RFCs [KA98a, KA98b].

- o The IP Authentication Header (AH) [KA98a] provides connectionless integrity, data origin authentication, and an optional anti-replay service.
- o The Encapsulating Security Payload (ESP) protocol [KA98b] may provide confidentiality (encryption), and limited traffic flow confidentiality. It also may provide connectionless

integrity, data origin authentication, and an anti-replay service. (One or the other set of these security services must be applied whenever ESP is invoked.)

- Both AH and ESP are vehicles for access control, based on the distribution of cryptographic keys and the management of traffic flows relative to these security protocols.

These protocols may be applied alone or in combination with each other to provide a desired set of security services in IPv4 and IPv6. Each protocol supports two modes of use: transport mode and tunnel mode. In transport mode the protocols provide protection primarily for upper layer protocols; in tunnel mode, the protocols are applied to tunneled IP packets. The differences between the two modes are discussed in Section 4.

IPsec allows the user (or system administrator) to control the granularity at which a security service is offered. For example, one can create a single encrypted tunnel to carry all the traffic between two security gateways or a separate encrypted tunnel can be created for each TCP connection between each pair of hosts communicating across these gateways. IPsec management must incorporate facilities for specifying:

- which security services to use and in what combinations
- the granularity at which a given security protection should be applied
- the algorithms used to effect cryptographic-based security

Because these security services use shared secret values (cryptographic keys), IPsec relies on a separate set of mechanisms for putting these keys in place. (The keys are used for authentication/integrity and encryption services.) This document requires support for both manual and automatic distribution of keys. It specifies a specific public-key based approach (IKE -- [MSST97, Orm97, HC98]) for automatic key management, but other automated key distribution techniques MAY be used. For example, KDC-based systems such as Kerberos and other public-key systems such as SKIP could be employed.

3.3 Where IPsec May Be Implemented

There are several ways in which IPsec may be implemented in a host or in conjunction with a router or firewall (to create a security gateway). Several common examples are provided below:

a. Integration of IPsec into the native IP implementation. This requires access to the IP source code and is applicable to both hosts and security gateways.

b. "Bump-in-the-stack" (BITS) implementations, where IPsec is implemented "underneath" an existing implementation of an IP protocol stack, between the native IP and the local network drivers. Source code access for the IP stack is not required in this context, making this implementation approach appropriate for use with legacy systems. This approach, when it is adopted, is usually employed in hosts.

c. The use of an outboard crypto processor is a common design feature of network security systems used by the military, and of some commercial systems as well. It is sometimes referred to as a "Bump-in-the-wire" (BITW) implementation. Such implementations may be designed to serve either a host or a gateway (or both). Usually the BITW device is IP addressable. When supporting a single host, it may be quite analogous to a BITS implementation, but in supporting a router or firewall, it must operate like a security gateway.

4. Security Associations

This section defines Security Association management requirements for all IPv6 implementations and for those IPv4 implementations that implement AH, ESP, or both. The concept of a "Security Association" (SA) is fundamental to IPsec. Both AH and ESP make use of SAs and a major function of IKE is the establishment and maintenance of Security Associations. All implementations of AH or ESP MUST support the concept of a Security Association as described below. The remainder of this section describes various aspects of Security Association management, defining required characteristics for SA policy management, traffic processing, and SA management techniques.

4.1 Definition and Scope

A Security Association (SA) is a simplex "connection" that affords security services to the traffic carried by it. Security services are afforded to an SA by the use of AH, or ESP, but not both. If both AH and ESP protection is applied to a traffic stream, then two (or more) SAs are created to afford protection to the traffic stream. To secure typical, bi-directional communication between two hosts, or between two security gateways, two Security Associations (one in each direction) are required.

A security association is uniquely identified by a triple consisting of a Security Parameter Index (SPI), an IP Destination Address, and a security protocol (AH or ESP) identifier. In principle, the Destination Address may be a unicast address, an IP broadcast address, or a multicast group address. However, IPsec SA management mechanisms currently are defined only for unicast SAs. Hence, in the discussions that follow, SAs will be described in the context of point-to-point communication, even though the concept is applicable in the point-to-multipoint case as well.

As noted above, two types of SAs are defined: transport mode and tunnel mode. A transport mode SA is a security association between two hosts. In IPv4, a transport mode security protocol header appears immediately after the IP header and any options, and before any higher layer protocols (e.g., TCP or UDP). In IPv6, the security protocol header appears after the base IP header and extensions, but may appear before or after destination options, and before higher layer protocols. In the case of ESP, a transport mode SA provides security services only for these higher layer protocols, not for the IP header or any extension headers preceding the ESP header. In the case of AH, the protection is also extended to selected portions of the IP header, selected portions of extension headers, and selected options (contained in the IPv4 header, IPv6 Hop-by-Hop extension header, or IPv6 Destination extension headers). For more details on the coverage afforded by AH, see the AH specification [KA98a].

A tunnel mode SA is essentially an SA applied to an IP tunnel. Whenever either end of a security association is a security gateway, the SA MUST be tunnel mode. Thus an SA between two security gateways is always a tunnel mode SA, as is an SA between a host and a security gateway. Note that for the case where traffic is destined for a security gateway, e.g., SNMP commands, the security gateway is acting as a host and transport mode is allowed. But in that case, the security gateway is not acting as a gateway, i.e., not transiting traffic. Two hosts MAY establish a tunnel mode SA between themselves. The requirement for any (transit traffic) SA involving a security gateway to be a tunnel SA arises due to the need to avoid potential problems with regard to fragmentation and reassembly of IPsec packets, and in circumstances where multiple paths (e.g., via different security gateways) exist to the same destination behind the security gateways.

For a tunnel mode SA, there is an "outer" IP header that specifies the IPsec processing destination, plus an "inner" IP header that specifies the (apparently) ultimate destination for the packet. The security protocol header appears after the outer IP header, and before the inner IP header. If AH is employed in tunnel mode, portions of the outer IP header are afforded protection (as above), as well as all of the tunneled IP packet (i.e., all of the inner IP header is protected, as well as higher layer protocols). If ESP is employed, the protection is afforded only to the tunneled packet, not to the outer header.

In summary,

a) A host MUST support both transport and tunnel mode.

b) A security gateway is required to support only tunnel mode. If it supports transport mode, that should be used only when the security gateway is acting as a host, e.g., for network management.

4.2 Security Association Functionality

The set of security services offered by an SA depends on the security protocol selected, the SA mode, the endpoints of the SA, and on the election of optional services within the protocol. For example, AH provides data origin authentication and connectionless integrity for IP datagrams (hereafter referred to as just "authentication"). The "precision" of the authentication service is a function of the granularity of the security association with which AH is employed, as discussed in Section 4.4.2, "Selectors".

AH also offers an anti-replay (partial sequence integrity) service at the discretion of the receiver, to help counter denial of service attacks. AH is an appropriate protocol to employ when confidentiality is not required (or is not permitted, e.g , due to government restrictions on use of encryption). AH also provides authentication for selected portions of the IP header, which may be necessary in some contexts. For example, if the integrity of an IPv4 option or IPv6 extension header must be protected en route between sender and receiver, AH can provide this service (except for the non-predictable but mutable parts of the IP header.)

ESP optionally provides confidentiality for traffic. (The strength of the confidentiality service depends in part, on the encryption algorithm employed.) ESP also may optionally provide authentication (as defined above). If authentication is negotiated for an ESP SA, the receiver also may elect to enforce an anti-replay service with the same features as the AH anti-replay service. The scope of the authentication offered by ESP is narrower than for AH, i.e., the IP header(s) "outside" the ESP header is(are) not protected. If only the upper layer protocols need to be authenticated, then ESP authentication is an appropriate choice and is more space efficient than use of AH encapsulating ESP. Note that although both confidentiality and authentication are optional, they cannot both be omitted. At least one of them MUST be selected.

If confidentiality service is selected, then an ESP (tunnel mode) SA between two security gateways can offer partial traffic flow confidentiality. The use of tunnel mode allows the inner IP headers to be encrypted, concealing the identities of the (ultimate) traffic source and destination. Moreover, ESP payload padding also can be invoked to hide the size of the packets, further concealing the external characteristics of the traffic. Similar traffic flow confidentiality services may be offered when a mobile user is assigned a dynamic IP address in a dialup context, and establishes a (tunnel mode) ESP SA to a corporate firewall (acting as a security gateway). Note that fine granularity SAs generally are more vulnerable to traffic analysis than coarse granularity ones which are carrying traffic from many subscribers.

4.3 Combining Security Associations

The IP datagrams transmitted over an individual SA are afforded protection by exactly one security protocol, either AH or ESP, but not both. Sometimes a security policy may call for a combination of services for a particular traffic flow that is not achievable with a single SA. In such instances it will be necessary to employ multiple SAs to implement the required security policy. The term "security association bundle" or "SA bundle" is applied to a sequence of SAs through which traffic must be processed to satisfy a security policy. The order of the sequence is defined by the policy. (Note that the SAs that comprise a bundle may terminate at different endpoints. For example, one SA may extend between a mobile host and a security gateway and a second, nested SA may extend to a host behind the gateway.)

Security associations may be combined into bundles in two ways: transport adjacency and iterated tunneling.

- Transport adjacency refers to applying more than one security protocol to the same IP datagram, without invoking tunneling. This approach to combining AH and ESP allows for only one level of combination; further nesting yields no added benefit (assuming use of adequately strong algorithms in each protocol) since the processing is performed at one IPsec instance at the (ultimate) destination.

```
Host 1 --- Security ---- Internet -- Security --- Host 2
 | |        Gwy 1                     Gwy 2        | |
 | |                                               | |
 | -----Security Association 1 (ESP transport)------- |
 |                                                    |
 -------Security Association 2 (AH transport)----------
```

- Iterated tunneling refers to the application of multiple layers of security protocols effected through IP tunneling. This approach allows for multiple levels of nesting, since each tunnel can originate or terminate at a different IPsec site along the path. No special treatment is expected for ISAKMP traffic at intermediate security gateways other than what can be specified through appropriate SPD entries (See Case 3 in Section 4.5)

There are 3 basic cases of iterated tunneling -- support is required only for cases 2 and 3.:

1. both endpoints for the SAs are the same -- The inner and outer tunnels could each be either AH or ESP, though it is unlikely that Host 1 would specify both to be the same, i.e., AH inside of AH or ESP inside of ESP.

```
Host 1 --- Security ---- Internet -- Security --- Host 2
 | |        Gwy 1                      Gwy 2        | |
 | |                                                | |
 | -------Security Association 1 (tunnel)---------- | |
 |                                                    |
 ---------Security Association 2 (tunnel)--------------
```

2. one endpoint of the SAs is the same -- The inner and uter tunnels could each be either AH or ESP.

```
Host 1 --- Security ---- Internet -- Security --- Host 2
 | |        Gwy 1                      Gwy 2         |
 | |                                     |           |
 | ----Security Association 1 (tunnel)----           |
 |                                                   |
 ---------Security Association 2 (tunnel)-------------
```

3. neither endpoint is the same -- The inner and outer tunnels could each be either AH or ESP.

```
Host 1 --- Security ---- Internet -- Security --- Host 2
 |          Gwy 1                      Gwy 2         |
 |            |                          |           |
 |            --Security Assoc 1 (tunnel)-           |
 |                                                   |
 -----------Security Association 2 (tunnel)-----------
```

These two approaches also can be combined, e.g., an SA bundle could be constructed from one tunnel mode SA and one or two transport mode SAs, applied in sequence. (See Section 4.5 "Basic Combinations of Security Associations.") Note that nested tunnels can also occur where neither the source nor the destination endpoints of any of the tunnels are the same. In that case, there would be no host or security gateway with a bundle corresponding to the nested tunnels.

For transport mode SAs, only one ordering of security protocols seems appropriate. AH is applied to both the upper layer protocols and (parts of) the IP header. Thus if AH is used in a transport mode, in conjunction with ESP, AH SHOULD appear as the first header after IP, prior to the appearance of ESP. In that context, AH is applied to the ciphertext output of ESP. In contrast, for tunnel mode SAs, one can imagine uses for various orderings of AH and ESP. The required set of SA bundle types that MUST be supported by a compliant IPsec implementation is described in Section 4.5.

4.4 Security Association Databases

Many of the details associated with processing IP traffic in an IPsec implementation are largely a local matter, not subject to standardization. However, some external aspects of the processing must be standardized, to ensure interoperability and to provide a minimum management capability that is essential for productive use of IPsec. This section describes a general model for processing IP traffic relative to security associations, in support of these interoperability and functionality goals. The model described below is nominal; compliant implementations need not match details of this model as presented, but the external behavior of such implementations must be mappable to the externally observable characteristics of this model.

There are two nominal databases in this model: the Security Policy Database and the Security Association Database. The former specifies the policies that determine the disposition of all IP traffic inbound or outbound from a host, security gateway, or BITS or BITW IPsec implementation. The latter database contains parameters that are associated with each (active) security association. This section also defines the concept of a Selector, a set of IP and upper layer protocol field values that is used by the Security Policy Database to map traffic to a policy, i.e., an SA (or SA bundle).

Each interface for which IPsec is enabled requires nominally separate inbound vs. outbound databases (SAD and SPD), because of the directionality of many of the fields that are used as selectors. Typically there is just one such interface, for a host or security gateway (SG). Note that an SG would always have at least 2 interfaces, but the "internal" one to the corporate net, usually would not have IPsec enabled and so only one pair of SADs and one pair of SPDs would be needed. On the other hand, if a host had multiple interfaces or an SG had multiple external interfaces, it might be necessary to have separate SAD and SPD pairs for each interface.

4.4.1 The Security Policy Database (SPD)

Ultimately, a security association is a management construct used to enforce a security policy in the IPsec environment. Thus an essential element of SA processing is an underlying Security Policy Database (SPD) that specifies what services are to be offered to IP datagrams and in what fashion. The form of the database and its interface are outside the scope of this specification. However, this section does specify certain minimum management functionality that must be provided, to allow a user or system administrator to control how IPsec is applied to traffic transmitted or received by a host or transiting a security gateway.

The SPD must be consulted during the processing of all traffic (INBOUND and OUTBOUND), including non-IPsec traffic. In order to support this, the SPD requires distinct entries for inbound and outbound traffic. One can think of this as separate SPDs (inbound vs. outbound). In addition, a nominally separate SPD must be provided for each IPsec-enabled interface.

An SPD must discriminate among traffic that is afforded IPsec protection and traffic that is allowed to bypass IPsec. This applies to the IPsec protection to be applied by a sender and to the IPsec protection that must be present at the receiver. For any outbound or inbound datagram, three processing choices are possible: discard, bypass IPsec, or apply IPsec. The first choice refers to traffic that is not allowed to exit the host, traverse the security gateway, or be delivered to an application at all. The second choice refers to traffic that is allowed to pass without additional IPsec protection. The third choice refers to traffic that is afforded IPsec protection, and for such traffic the SPD must specify the security services to be provided, protocols to be employed, algorithms to be used, etc.

For every IPsec implementation, there MUST be an administrative interface that allows a user or system administrator to manage the SPD. Specifically, every inbound or outbound packet is subject to processing by IPsec and the SPD must specify what action will be taken in each case. Thus the administrative interface must allow the user (or system administrator) to specify the security processing to be applied to any packet entering or exiting the system, on a packet by packet basis. (In a host IPsec implementation making use of a socket interface, the SPD may not need to be consulted on a per packet basis, but the effect is still the same.) The management interface for the SPD MUST allow creation of entries consistent with the selectors defined in Section 4.4.2, and MUST support (total) ordering of these entries. It is expected that through the use of wildcards in various selector fields, and because all packets on a single UDP or TCP connection will tend to match a single SPD entry, this requirement will not impose an unreasonably detailed level of SPD specification. The selectors are analogous to what are found in a stateless firewall or filtering router and which are currently manageable this way.

In host systems, applications MAY be allowed to select what security processing is to be applied to the traffic they generate and consume. (Means of signalling such requests to the IPsec implementation are outside the scope of this standard.) However, the system administrator MUST be able to specify whether or not a user or application can override (default) system policies. Note that application specified policies may satisfy system requirements, so that the system may not need to do additional IPsec processing beyond that needed to meet an application's requirements. The form of the management interface is not specified by this document and may differ for hosts vs. security gateways, and within hosts the interface may differ for socket-based vs. BITS implementations. However, this document does specify a standard set of SPD elements that all IPsec implementations MUST support.

The SPD contains an ordered list of policy entries. Each policy entry is keyed by one or more selectors that define the set of IP traffic encompassed by this policy entry. (The required selector types are defined in Section 4.4.2.) These define the granularity of policies or SAs. Each entry includes an indication of whether traffic matching this policy will be bypassed, discarded, or subject to IPsec processing. If IPsec processing is to be applied, the entry includes an SA (or SA bundle) specification, listing the IPsec protocols, modes, and algorithms to be employed, including any nesting requirements. For example, an entry may call for all matching traffic to be protected by ESP in transport mode using 3DES-CBC with an explicit IV, nested inside of AH in tunnel mode using HMAC/SHA-1. For each selector, the policy entry specifies how to derive the corresponding values for a new Security Association Database (SAD, see Section 4.4.3) entry from those in the SPD and the packet (Note that at present, ranges are only supported for IP addresses; but wildcarding can be expressed for all selectors):

a. use the value in the packet itself -- This will limit use of the SA to those packets which have this packet's value for the selector even if the selector for the policy entry has a range of allowed values or a wildcard for this selector.
b. use the value associated with the policy entry -- If this were to be just a single value, then there would be no difference between (b) and (a). However, if the allowed values for the selector are a range (for IP addresses) or

wildcard, then in the case of a range,(b) would enable use of the SA by any packet with a selector value within the range not just by packets with the selector value of the packet that triggered the creation of the SA. In the case of a wildcard, (b) would allow use of the SA by packets with any value for this selector.

For example, suppose there is an SPD entry where the allowed value for source address is any of a range of hosts (192.168.2.1 to 192.168.2.10). And suppose that a packet is to be sent that has a source address of 192.168.2.3. The value to be used for the SA could be any of the sample values below depending on what the policy entry for this selector says is the source of the selector value:

source for the	example of
value to be	new SAD
used in the SA	selector value
a. packet	192.168.2.3 (one host)
b. SPD entry	192.168.2.1 to 192.168.2.10 (range of hosts)

Note that if the SPD entry had an allowed value of wildcard for the source address, then the SAD selector value could be wildcard (any host). Case (a) can be used to prohibit sharing, even among packets that match the same SPD entry.

As described below in Section 4.4.3, selectors may include "wildcard" entries and hence the selectors for two entries may overlap. (This is analogous to the overlap that arises with ACLs or filter entries in routers or packet filtering firewalls.) Thus, to ensure consistent, predictable processing, SPD entries MUST be ordered and the SPD MUST always be searched in the same order, so that the first matching entry is consistently selected. (This requirement is necessary as the effect of processing traffic against SPD entries must be deterministic, but there is no way to canonicalize SPD entries given the use of wildcards for some selectors.) More detail on matching of packets against SPD entries is provided in Section 5.

Note that if ESP is specified, either (but not both) authentication or encryption can be omitted. So it MUST be possible to configure the SPD value for the authentication or encryption algorithms to be "NULL". However, at least one of these services MUST be selected, i.e., it MUST NOT be possible to configure both of them as "NULL".

The SPD can be used to map traffic to specific SAs or SA bundles. Thus it can function both as the reference database for security policy and as the map to existing SAs (or SA bundles). (To accommodate the bypass and discard policies cited above, the SPD also

MUST provide a means of mapping traffic to these functions, even though they are not, per se, IPsec processing.) The way in which the SPD operates is different for inbound vs. outbound traffic and it also may differ for host vs. security gateway, BITS, and BITW implementations. Sections 5.1 and 5.2 describe the use of the SPD for outbound and inbound processing, respectively.

Because a security policy may require that more than one SA be applied to a specified set of traffic, in a specific order, the policy entry in the SPD must preserve these ordering requirements, when present. Thus, it must be possible for an IPsec implementation to determine that an outbound or inbound packet must be processed thorough a sequence of SAs. Conceptually, for outbound processing, one might imagine links (to the SAD) from an SPD entry for which there are active SAs, and each entry would consist of either a single SA or an ordered list of SAs that comprise an SA bundle. When a packet is matched against an SPD entry and there is an existing SA or SA bundle that can be used to carry the traffic, the processing of the packet is controlled by the SA or SA bundle entry on the list. For an inbound IPsec packet for which multiple IPsec SAs are to be applied, the lookup based on destination address, IPsec protocol, and SPI should identify a single SA.

The SPD is used to control the flow of ALL traffic through an IPsec system, including security and key management traffic (e.g., ISAKMP) from/to entities behind a security gateway. This means that ISAKMP traffic must be explicitly accounted for in the SPD, else it will be discarded. Note that a security gateway could prohibit traversal of encrypted packets in various ways, e.g., having a DISCARD entry in the SPD for ESP packets or providing proxy key exchange. In the latter case, the traffic would be internally routed to the key management module in the security gateway.

4.4.2 Selectors

An SA (or SA bundle) may be fine-grained or coarse-grained, depending on the selectors used to define the set of traffic for the SA. For example, all traffic between two hosts may be carried via a single SA, and afforded a uniform set of security services. Alternatively, traffic between a pair of hosts might be spread over multiple SAs, depending on the applications being used (as defined by the Next Protocol and Port fields), with different security services offered by different SAs. Similarly, all traffic between a pair of security gateways could be carried on a single SA, or one SA could be assigned for each communicating host pair. The following selector parameters MUST be supported for SA management to facilitate control of SA granularity. Note that in the case of receipt of a packet with an ESP header, e.g., at an encapsulating security gateway or BITW implementation, the transport layer protocol, source/destination ports, and Name (if present) may be "OPAQUE", i.e., inaccessible because of encryption or fragmentation. Note also that both Source and Destination addresses should either be IPv4 or IPv6.

- Destination IP Address (IPv4 or IPv6): this may be a single IP address (unicast, anycast, broadcast (IPv4 only), or multicast group), a range of addresses (high and low values (inclusive), address + mask, or a wildcard address. The last three are used to support more than one destination system sharing the same SA (e.g., behind a security gateway). Note that this selector is conceptually different from the "Destination IP Address" field in the <Destination IP Address, IPsec Protocol, SPI> tuple used to uniquely identify an SA. When a tunneled packet arrives at the tunnel endpoint, its SPI/Destination address/Protocol are used to look up the SA for this packet in the SAD. This destination address comes from the encapsulating IP header. Once the packet has been processed according to the tunnel SA and has come out of the tunnel, its selectors are "looked up" in the Inbound SPD. The Inbound SPD has a selector called destination address. This IP destination address is the one in the inner (encapsulated) IP header. In the case of a transport'd packet, there will be only one IP header and this ambiguity does not exist. [REQUIRED for all implementations]

- Source IP Address(es) (IPv4 or IPv6): this may be a single IP address (unicast, anycast, broadcast (IPv4 only), or multicast group), range of addresses (high and low values inclusive), address + mask, or a wildcard address. The last three are used to support more than one source system sharing the same SA (e.g., behind a security gateway or in a multihomed host). [REQUIRED for all implementations]

- Name: There are 2 cases (Note that these name forms are supported in the IPsec DOI.)
    1. User ID
        a. a fully qualified user name string (DNS), e.g., mozart@foo.bar.com
        b. X.500 distinguished name, e.g., C = US, SP = MA, O = GTE Internetworking, CN = Stephen T. Kent.
    2. System name (host, security gateway, etc.)
        a. a fully qualified DNS name, e.g., foo.bar.com
        b. X.500 distinguished name
        c. X.500 general name

  NOTE: One of the possible values of this selector is "OPAQUE".

[REQUIRED for the following cases. Note that support for name forms other than addresses is not required for manually keyed SAs.

- o User ID
  - native host implementations
  - BITW and BITS implementations acting as HOSTS with only one user
  - security gateway implementations for INBOUND processing.
- o System names -- all implementations]

- Data sensitivity level: (IPSO/CIPSO labels) [REQUIRED for all systems providing information flow security as per Section 8, OPTIONAL for all other systems.]

- Transport Layer Protocol: Obtained from the IPv4 "Protocol" or the IPv6 "Next Header" fields. This may be an individual protocol number. These packet fields may not contain the Transport Protocol due to the presence of IP extension headers, e.g., a Routing Header, AH, ESP, Fragmentation Header, Destination Options, Hop-by-hop options, etc. Note that the Transport Protocol may not be available in the case of receipt of a packet with an ESP header, thus a value of "OPAQUE" SHOULD be supported.
  [REQUIRED for all implementations]

  NOTE: To locate the transport protocol, a system has to chain through the packet headers checking the "Protocol" or "Next Header" field until it encounters either one it recognizes as a transport protocol, or until it reaches one that isn't on its list of extension headers, or until it encounters an ESP header that renders the transport protocol opaque.

- Source and Destination (e.g., TCP/UDP) Ports: These may be individual UDP or TCP port values or a wildcard port. (The use of the Next Protocol field and the Source and/or Destination Port fields (in conjunction with the Source and/or Destination Address fields), as an SA selector is sometimes referred to as "session-oriented keying."). Note that the source and destination ports may not be available in the case of receipt of a packet with an ESP header, thus a value of "OPAQUE" SHOULD be supported.

  The following table summarizes the relationship between the "Next Header" value in the packet and SPD and the derived Port Selector value for the SPD and SAD.

Next Hdr in Packet	Transport Layer Protocol in SPD	Derived Port Selector Field Value in SPD and SAD
ESP	ESP or ANY	ANY (i.e., don't look at it)
-don't care-	ANY	ANY (i.e., don't look at it)
specific value fragment	specific value	NOT ANY (i.e., drop packet)
specific value not fragment	specific value	actual port selector field

If the packet has been fragmented, then the port information may not be available in the current fragment. If so, discard the fragment. An ICMP PMTU should be sent for the first fragment, which will have the port information. [MAY be supported]

The IPsec implementation context determines how selectors are used. For example, a host implementation integrated into the stack may make use of a socket interface. When a new connection is established the SPD can be consulted and an SA (or SA bundle) bound to the socket. Thus traffic sent via that socket need not result in additional lookups to the SPD/SAD. In contrast, a BITS, BITW, or security gateway implementation needs to look at each packet and perform an SPD/SAD lookup based on the selectors. The allowable values for the selector fields differ between the traffic flow, the security association, and the security policy.

The following table summarizes the kinds of entries that one needs to be able to express in the SPD and SAD. It shows how they relate to the fields in data traffic being subjected to IPsec screening. (Note: the "wild" or "wildcard" entry for src and dst addresses includes a mask, range, etc.)

Field	Traffic Value	SAD Entry	SPD Entry
src addr	single IP addr	single,range,wild	single,range,wildcard
dst addr	single IP addr	single,range,wild	single,range,wildcard
xpt protocol*	xpt protocol	single,wildcard	single,wildcard
src port*	single src port	single,wildcard	single,wildcard
dst port*	single dst port	single,wildcard	single,wildcard
user id*	single user id	single,wildcard	single,wildcard
sec. labels	single value	single,wildcard	single,wildcard

* The SAD and SPD entries for these fields could be "OPAQUE" because the traffic value is encrypted.

NOTE: In principle, one could have selectors and/or selector values in the SPD which cannot be negotiated for an SA or SA bundle. Examples might include selector values used to select traffic for discarding or enumerated lists which cause a separate SA to be created for each item on the list. For now, this is left for future versions of this document and the list of required selectors and selector values is the same for the SPD and the SAD. However, it is acceptable to have an administrative interface that supports use of selector values which cannot be negotiated provided that it does not mislead the user into believing it is creating an SA with these selector values. For example, the interface may allow the user to specify an enumerated list of values but would result in the creation of a separate policy and SA for each item on the list. A vendor might support such an interface to make it easier for its customers to specify clear and concise policy specifications.

4.4.3 Security Association Database (SAD)

In each IPsec implementation there is a nominal Security Association Database, in which each entry defines the parameters associated with one SA. Each SA has an entry in the SAD. For outbound processing, entries are pointed to by entries in the SPD. Note that if an SPD entry does not currently point to an SA that is appropriate for the packet, the implementation creates an appropriate SA (or SA Bundle) and links the SPD entry to the SAD entry (see Section 5.1.1). For inbound processing, each entry in the SAD is indexed by a destination IP address, IPsec protocol type, and SPI. The following parameters are associated with each entry in the SAD. This description does not purport to be a MIB, but only a specification of the minimal data items required to support an SA in an IPsec implementation.

For inbound processing: The following packet fields are used to look up the SA in the SAD:

o Outer Header's Destination IP address: the IPv4 or IPv6 Destination address.
  [REQUIRED for all implementations]
o IPsec Protocol: AH or ESP, used as an index for SA lookup in this database. Specifies the IPsec protocol to be applied to the traffic on this SA.
  [REQUIRED for all implementations]
o SPI: the 32-bit value used to distinguish among different SAs terminating at the same destination and using the same IPsec protocol.
  [REQUIRED for all implementations]

For each of the selectors defined in Section 4.4.2, the SA entry in the SAD MUST contain the value or values which were negotiated at the time the SA was created. For the sender, these values are used to decide whether a given SA is appropriate for use with an outbound packet. This is part of checking to see if there is an existing SA that can be used. For the receiver, these values are used to check that the selector values in an inbound packet match those for the SA (and thus indirectly those for the matching policy). For the receiver, this is part of verifying that the SA was appropriate for this packet. (See Section 6 for rules for ICMP messages.) These fields can have the form of specific values, ranges, wildcards, or "OPAQUE" as described in section 4.4.2, "Selectors". Note that for an ESP SA, the encryption algorithm or the authentication algorithm could be "NULL". However they MUST not both be "NULL".

The following SAD fields are used in doing IPsec processing:

- Sequence Number Counter: a 32-bit value used to generate the Sequence Number field in AH or ESP headers.
  [REQUIRED for all implementations, but used only for outbound traffic.]
- Sequence Counter Overflow: a flag indicating whether overflow of the Sequence Number Counter should generate an auditable event and prevent transmission of additional packets on the SA.
  [REQUIRED for all implementations, but used only for outbound traffic.]
- Anti-Replay Window: a 32-bit counter and a bit-map (or equivalent) used to determine whether an inbound AH or ESP packet is a replay.
  [REQUIRED for all implementations but used only for inbound traffic. NOTE: If anti-replay has been disabled by the receiver, e.g., in the case of a manually keyed SA, then the Anti-Replay Window is not used.]
- AH Authentication algorithm, keys, etc.
  [REQUIRED for AH implementations]
- ESP Encryption algorithm, keys, IV mode, IV, etc.
  [REQUIRED for ESP implementations]
- ESP authentication algorithm, keys, etc. If the authentication service is not selected, this field will be null.
  [REQUIRED for ESP implementations]
- Lifetime of this Security Association: a time interval after which an SA must be replaced with a new SA (and new SPI) or terminated, plus an indication of which of these actions should occur. This may be expressed as a time or byte count, or a simultaneous use of both, the first lifetime to expire taking precedence. A compliant implementation MUST support both types of lifetimes, and must support a simultaneous use of both. If time is employed, and if IKE employs X.509 certificates for SA establishment, the SA lifetime must be constrained by the validity intervals of the certificates, and the NextIssueDate of the CRLs used in the IKE exchange

for the SA. Both initiator and responder are responsible for constraining SA lifetime in this fashion.
[REQUIRED for all implementations]

NOTE: The details of how to handle the refreshing of keys when SAs expire is a local matter. However, one reasonable approach is:

(a) If byte count is used, then the implementation SHOULD count the number of bytes to which the IPsec algorithm is applied. For ESP, this is the encryption algorithm (including Null encryption) and for AH, this is the authentication algorithm. This includes pad bytes, etc. Note that implementations SHOULD be able to handle having the counters at the ends of an SA get out of synch, e.g., because of packet loss or because the implementations at each end of the SA aren't doing things the same way.
(b) There SHOULD be two kinds of lifetime -- a soft lifetime which warns the implementation to initiate action such as setting up a replacement SA and a hard lifetime when the current SA ends.
(c) If the entire packet does not get delivered during the SAs lifetime, the packet SHOULD be discarded.

o IPsec protocol mode: tunnel, transport or wildcard. Indicates which mode of AH or ESP is applied to traffic on this SA. Note that if this field is "wildcard" at the sending end of the SA, then the application has to specify the mode to the IPsec implementation. This use of wildcard allows the same SA to be used for either tunnel or transport mode traffic on a per packet basis, e.g., by different sockets. The receiver does not need to know the mode in order to properly process the packet's IPsec headers.

[REQUIRED as follows, unless implicitly defined by context:
- host implementations must support all modes
- gateway implementations must support tunnel mode]

NOTE: The use of wildcard for the protocol mode of an inbound SA may add complexity to the situation in the receiver (host only). Since the packets on such an SA could be delivered in either tunnel or transport mode, the security of an incoming packet could depend in part on which mode had been used to deliver it. If, as a result, an application cared about the SA mode of a given packet, then the application would need a mechanism to obtain this mode information.

- o Path MTU: any observed path MTU and aging variables. See Section 6.1.2.4
  [REQUIRED for all implementations but used only for outbound traffic]

4.5 Basic Combinations of Security Associations

This section describes four examples of combinations of security associations that MUST be supported by compliant IPsec hosts or security gateways. Additional combinations of AH and/or ESP in tunnel and/or transport modes MAY be supported at the discretion of the implementor. Compliant implementations MUST be capable of generating these four combinations and on receipt, of processing them, but SHOULD be able to receive and process any combination. The diagrams and text below describe the basic cases. The legend for the diagrams is:

```
==== = one or more security associations (AH or ESP, transport
       or tunnel)
---- = connectivity (or if so labelled, administrative boundary)
Hx   = host x
SGx  = security gateway x
X*   = X supports IPsec
```

NOTE: The security associations below can be either AH or ESP. The mode (tunnel vs transport) is determined by the nature of the endpoints. For host-to-host SAs, the mode can be either transport or tunnel.

Case 1. The case of providing end-to-end security between 2 hosts across the Internet (or an Intranet).

```
 ====================================
 |                                  |
H1* ------ (Inter/Intranet) ------ H2*
```

Note that either transport or tunnel mode can be selected by the hosts. So the headers in a packet between H1 and H2 could look like any of the following:

```
     Transport                 Tunnel
-----------------        ---------------------
1. [IP1][AH][upper]      4. [IP2][AH][IP1][upper]
2. [IP1][ESP][upper]     5. [IP2][ESP][IP1][upper]
3. [IP1][AH][ESP][upper]
```

Note that there is no requirement to support general nesting, but in transport mode, both AH and ESP can be applied to the packet. In this event, the SA establishment procedure MUST ensure that first ESP, then AH are applied to the packet.

Case 2. This case illustrates simple virtual private networks support.

```
                       ===========================
                       |                         |
  ---------------------|----                  ---|-----------------------
  |                    |   |                  |  |                      |
  |  H1 -- (Local --- SG1* |--- (Internet) ---| SG2* --- (Local --- H2  |
  |        Intranet)       |                  |          Intranet)      |
  --------------------------                  ---------------------------
      admin. boundary                               admin. boundary
```

Only tunnel mode is required here. So the headers in a packet between SG1 and SG2 could look like either of the following:

```
                        Tunnel
                ---------------------
                4. [IP2][AH][IP1][upper]
                5. [IP2][ESP][IP1][upper]
```

Case 3. This case combines cases 1 and 2, adding end-to-end security between the sending and receiving hosts. It imposes no new requirements on the hosts or security gateways, other than a requirement for a security gateway to be configurable to pass IPsec traffic (including ISAKMP traffic) for hosts behind it.

```
     ===============================================================
     |                                                             |
     |                 =========================                   |
     |                 |                       |                   |
  ---|-----------------|----                ---|-------------------|---
  |  |                 |   |                |  |                   |  |
  | H1* -- (Local --- SG1* |-- (Internet) --| SG2* --- (Local --- H2* |
  |        Intranet)       |                |          Intranet)      |
  --------------------------                ---------------------------
       admin. boundary                            admin. boundary
```

Case 4. This covers the situation where a remote host (H1) uses the Internet to reach an organization's firewall (SG2) and to then gain access to some server or other machine (H2). The remote host could be a mobile host (H1) dialing up to a local PPP/ARA server (not shown) on the Internet and then crossing the Internet to the home organization's firewall (SG2), etc. The details of support for this case, (how H1 locates SG2, authenticates it, and verifies its authorization to represent H2) are discussed in Section 4.6.3, "Locating a Security Gateway".

```
        ======================================================
        |                                                    |
        |==============================                      |
        ||                            |                      |
        ||                         ---|----------------------|---
        ||                         |  |                      |  |
        H1* ----- (Internet) ------| SG2* ---- (Local ----- H2* |
              ^                    |           Intranet)        |
              |                    ------------------------------
        could be dialup              admin. boundary (optional)
        to PPP/ARA server
```

Only tunnel mode is required between H1 and SG2. So the choices for the SA between H1 and SG2 would be one of the ones in case 2. The choices for the SA between H1 and H2 would be one of the ones in case 1.

Note that in this case, the sender MUST apply the transport header before the tunnel header. Therefore the management interface to the IPsec implementation MUST support configuration of the SPD and SAD to ensure this ordering of IPsec header application.

As noted above, support for additional combinations of AH and ESP is optional. Use of other, optional combinations may adversely affect interoperability.

4.6 SA and Key Management

IPsec mandates support for both manual and automated SA and cryptographic key management. The IPsec protocols, AH and ESP, are largely independent of the associated SA management techniques, although the techniques involved do affect some of the security services offered by the protocols. For example, the optional anti-replay services available for AH and ESP require automated SA management. Moreover, the granularity of key distribution employed with IPsec determines the granularity of authentication provided. (See also a discussion of this issue in Section 4.7.) In general, data origin authentication in AH and ESP is limited by the extent to which secrets used with the authentication algorithm (or with a key management protocol that creates such secrets) are shared among multiple possible sources.

The following text describes the minimum requirements for both types of SA management.

4.6.1 Manual Techniques

The simplest form of management is manual management, in which a person manually configures each system with keying material and security association management data relevant to secure communication with other systems. Manual techniques are practical in small, static environments but they do not scale well. For example, a company could create a Virtual Private Network (VPN) using IPsec in security gateways at several sites. If the number of sites is small, and since all the sites come under the purview of a single administrative domain, this is likely to be a feasible context for manual management techniques. In this case, the security gateway might selectively protect traffic to and from other sites within the organization using a manually configured key, while not protecting traffic for other destinations. It also might be appropriate when only selected communications need to be secured. A similar argument might apply to use of IPsec entirely within an organization for a small number of hosts and/or gateways. Manual management techniques often employ statically configured, symmetric keys, though other options also exist.

4.6.2 Automated SA and Key Management

Widespread deployment and use of IPsec requires an Internet-standard, scalable, automated, SA management protocol. Such support is required to facilitate use of the anti-replay features of AH and ESP, and to accommodate on-demand creation of SAs, e.g., for user- and session-oriented keying. (Note that the notion of "rekeying" an SA actually implies creation of a new SA with a new SPI, a process that generally implies use of an automated SA/key management protocol.)

The default automated key management protocol selected for use with IPsec is IKE [MSST97, Orm97, HC98] under the IPsec domain of interpretation [Pip98]. Other automated SA management protocols MAY be employed.

When an automated SA/key management protocol is employed, the output from this protocol may be used to generate multiple keys, e.g., for a single ESP SA. This may arise because:

- o the encryption algorithm uses multiple keys (e.g., triple DES)
- o the authentication algorithm uses multiple keys
- o both encryption and authentication algorithms are employed

The Key Management System may provide a separate string of bits for each key or it may generate one string of bits from which all of them are extracted. If a single string of bits is provided, care needs to be taken to ensure that the parts of the system that map the string of bits to the required keys do so in the same fashion at both ends of the SA. To ensure that the IPsec implementations at each end of the SA use the same bits for the same keys, and irrespective of which part of the system divides the string of bits into individual keys, the encryption key(s) MUST be taken from the first (left-most, high-order) bits and the authentication key(s) MUST be taken from the remaining bits. The number of bits for each key is defined in the relevant algorithm specification RFC. In the case of multiple encryption keys or multiple authentication keys, the specification for the algorithm must specify the order in which they are to be selected from a single string of bits provided to the algorithm.

4.6.3 Locating a Security Gateway

This section discusses issues relating to how a host learns about the existence of relevant security gateways and once a host has contacted these security gateways, how it knows that these are the correct security gateways. The details of where the required information is stored is a local matter.

Consider a situation in which a remote host (H1) is using the Internet to gain access to a server or other machine (H2) and there is a security gateway (SG2), e.g., a firewall, through which H1's traffic must pass. An example of this situation would be a mobile host (Road Warrior) crossing the Internet to the home organization's firewall (SG2). (See Case 4 in the section 4.5 Basic Combinations of Security Associations.) This situation raises several issues:

1. How does H1 know/learn about the existence of the security gateway SG2?
2. How does it authenticate SG2, and once it has authenticated SG2, how does it confirm that SG2 has been authorized to represent H2?
3. How does SG2 authenticate H1 and verify that H1 is authorized to contact H2?
4. How does H1 know/learn about backup gateways which provide alternate paths to H2?

To address these problems, a host or security gateway MUST have an administrative interface that allows the user/administrator to configure the address of a security gateway for any sets of destination addresses that require its use. This includes the ability to configure:

- o the requisite information for locating and authenticating the security gateway and verifying its authorization to represent the destination host.
- o the requisite information for locating and authenticating any backup gateways and verifying their authorization to represent the destination host.

It is assumed that the SPD is also configured with policy information that covers any other IPsec requirements for the path to the security gateway and the destination host.

This document does not address the issue of how to automate the discovery/verification of security gateways.

4.7 Security Associations and Multicast

The receiver-orientation of the Security Association implies that, in the case of unicast traffic, the destination system will normally select the SPI value. By having the destination select the SPI value, there is no potential for manually configured Security Associations to conflict with automatically configured (e.g., via a key management protocol) Security Associations or for Security Associations from multiple sources to conflict with each other. For multicast traffic, there are multiple destination systems per multicast group. So some system or person will need to coordinate among all multicast groups to select an SPI or SPIs on behalf of each multicast group and then communicate the group's IPsec information to all of the legitimate members of that multicast group via mechanisms not defined here.

Multiple senders to a multicast group SHOULD use a single Security Association (and hence Security Parameter Index) for all traffic to that group when a symmetric key encryption or authentication algorithm is employed. In such circumstances, the receiver knows only that the message came from a system possessing the key for that multicast group. In such circumstances, a receiver generally will not be able to authenticate which system sent the multicast traffic. Specifications for other, more general multicast cases are deferred to later IPsec documents.

At the time this specification was published, automated protocols for multicast key distribution were not considered adequately mature for standardization. For multicast groups having relatively few members, manual key distribution or multiple use of existing unicast key distribution algorithms such as modified Diffie-Hellman appears feasible. For very large groups, new scalable techniques will be needed. An example of current work in this area is the Group Key Management Protocol (GKMP) [HM97].

5. IP Traffic Processing

As mentioned in Section 4.4.1 "The Security Policy Database (SPD)", the SPD must be consulted during the processing of all traffic (INBOUND and OUTBOUND), including non-IPsec traffic. If no policy is found in the SPD that matches the packet (for either inbound or outbound traffic), the packet MUST be discarded.

NOTE: All of the cryptographic algorithms used in IPsec expect their input in canonical network byte order (see Appendix in RFC 791) and generate their output in canonical network byte order. IP packets are also transmitted in network byte order.

5.1 Outbound IP Traffic Processing

5.1.1 Selecting and Using an SA or SA Bundle

In a security gateway or BITW implementation (and in many BITS implementations), each outbound packet is compared against the SPD to determine what processing is required for the packet. If the packet is to be discarded, this is an auditable event. If the traffic is allowed to bypass IPsec processing, the packet continues through "normal" processing for the environment in which the IPsec processing is taking place. If IPsec processing is required, the packet is either mapped to an existing SA (or SA bundle), or a new SA (or SA bundle) is created for the packet. Since a packet's selectors might match multiple policies or multiple extant SAs and since the SPD is ordered, but the SAD is not, IPsec MUST:

1. Match the packet's selector fields against the outbound policies in the SPD to locate the first appropriate policy, which will point to zero or more SA bundles in the SAD.

2. Match the packet's selector fields against those in the SA bundles found in (1) to locate the first SA bundle that matches. If no SAs were found or none match, create an appropriate SA bundle and link the SPD entry to the SAD entry. If no key management entity is found, drop the packet.

3. Use the SA bundle found/created in (2) to do the required IPsec processing, e.g., authenticate and encrypt.

In a host IPsec implementation based on sockets, the SPD will be consulted whenever a new socket is created, to determine what, if any, IPsec processing will be applied to the traffic that will flow on that socket.

NOTE: A compliant implementation MUST not allow instantiation of an ESP SA that employs both a NULL encryption and a NULL authentication algorithm. An attempt to negotiate such an SA is an auditable event.

5.1.2 Header Construction for Tunnel Mode

This section describes the handling of the inner and outer IP headers, extension headers, and options for AH and ESP tunnels. This includes how to construct the encapsulating (outer) IP header, how to handle fields in the inner IP header, and what other actions should be taken. The general idea is modeled after the one used in RFC 2003, "IP Encapsulation with IP":

- o The outer IP header Source Address and Destination Address identify the "endpoints" of the tunnel (the encapsulator and decapsulator). The inner IP header Source Address and Destination Addresses identify the original sender and recipient of the datagram, (from the perspective of this tunnel), respectively. (see footnote 3 after the table in 5.1.2.1 for more details on the encapsulating source IP address.)
- o The inner IP header is not changed except to decrement the TTL as noted below, and remains unchanged during its delivery to the tunnel exit point.
- o No change to IP options or extension headers in the inner header occurs during delivery of the encapsulated datagram through the tunnel.
- o If need be, other protocol headers such as the IP Authentication header may be inserted between the outer IP header and the inner IP header.

The tables in the following sub-sections show the handling for the different header/option fields (constructed = the value in the outer field is constructed independently of the value in the inner).

5.1.2.1 IPv4 -- Header Construction for Tunnel Mode

	<-- How Outer Hdr Relates to Inner Hdr -->	
	Outer Hdr at	Inner Hdr at
IPv4	Encapsulator	Decapsulator
Header fields:	--------------------	------------
version	4 (1)	no change
header length	constructed	no change
TOS	copied from inner hdr (5)	no change
total length	constructed	no change
ID	constructed	no change
flags (DF,MF)	constructed, DF (4)	no change
fragmt offset	constructed	no change

TTL	constructed (2)	decrement (2)
protocol	AH, ESP, routing hdr	no change
checksum	constructed	constructed (2)
src address	constructed (3)	no change
dest address	constructed (3)	no change
Options	never copied	no change

1. The IP version in the encapsulating header can be different from the value in the inner header.

2. The TTL in the inner header is decremented by the encapsulator prior to forwarding and by the decapsulator if it forwards the packet. (The checksum changes when the TTL changes.)

   Note: The decrementing of the TTL is one of the usual actions that takes place when forwarding a packet. Packets originating from the same node as the encapsulator do not have their TTL's decremented, as the sending node is originating the packet rather than forwarding it.

3. src and dest addresses depend on the SA, which is used to determine the dest address which in turn determines which src address (net interface) is used to forward the packet.

   NOTE: In principle, the encapsulating IP source address can be any of the encapsulator's interface addresses or even an address different from any of the encapsulator's IP addresses, (e.g., if it's acting as a NAT box) so long as the address is reachable through the encapsulator from the environment into which the packet is sent. This does not cause a problem because IPsec does not currently have any INBOUND processing requirement that involves the Source Address of the encapsulating IP header. So while the receiving tunnel endpoint looks at the Destination Address in the encapsulating IP header, it only looks at the Source Address in the inner (encapsulated) IP header.

4. configuration determines whether to copy from the inner header (IPv4 only), clear or set the DF.

5. If Inner Hdr is IPv4 (Protocol = 4), copy the TOS. If Inner Hdr is IPv6 (Protocol = 41), map the Class to TOS.

5.1.2.2 IPv6 -- Header Construction for Tunnel Mode

See previous section 5.1.2 for notes 1-5 indicated by (footnote number).

	<-- How Outer Hdr Relates Inner Hdr --->	
	Outer Hdr at	Inner Hdr at
IPv6	Encapsulator	Decapsulator
Header fields:		
version	6 (1)	no change
class	copied or configured (6)	no change
flow id	copied or configured	no change
len	constructed	no change
next header	AH,ESP,routing hdr	no change
hop limit	constructed (2)	decrement (2)
src address	constructed (3)	no change
dest address	constructed (3)	no change
Extension headers	never copied	no change

6. If Inner Hdr is IPv6 (Next Header = 41), copy the Class. If Inner Hdr is IPv4 (Next Header = 4), map the TOS to Class.

5.2 Processing Inbound IP Traffic

Prior to performing AH or ESP processing, any IP fragments are reassembled. Each inbound IP datagram to which IPsec processing will be applied is identified by the appearance of the AH or ESP values in the IP Next Protocol field (or of AH or ESP as an extension header in the IPv6 context).

Note: Appendix C contains sample code for a bitmask check for a 32 packet window that can be used for implementing anti-replay service.

5.2.1 Selecting and Using an SA or SA Bundle

Mapping the IP datagram to the appropriate SA is simplified because of the presence of the SPI in the AH or ESP header. Note that the selector checks are made on the inner headers not the outer (tunnel) headers. The steps followed are:

1. Use the packet's destination address (outer IP header), IPsec protocol, and SPI to look up the SA in the SAD. If the SA lookup fails, drop the packet and log/report the error.

2. Use the SA found in (1) to do the IPsec processing, e.g., authenticate and decrypt. This step includes matching the packet's (Inner Header if tunneled) selectors to the selectors in the SA. Local policy determines the specificity of the SA selectors (single value, list, range, wildcard). In general, a packet's source address MUST match the SA selector value. However, an ICMP packet received on a tunnel mode SA may have a source address

other than that bound to the SA and thus such packets should be permitted as exceptions to this check. For an ICMP packet, the selectors from the enclosed problem packet (the source and destination addresses and ports should be swapped) should be checked against the selectors for the SA. Note that some or all of these selectors may be inaccessible because of limitations on how many bits of the problem packet the ICMP packet is allowed to carry or due to encryption. See Section 6.

Do (1) and (2) for every IPsec header until a Transport Protocol Header or an IP header that is NOT for this system is encountered. Keep track of what SAs have been used and their order of application.

3. Find an incoming policy in the SPD that matches the packet. This could be done, for example, by use of backpointers from the SAs to the SPD or by matching the packet's selectors (Inner Header if tunneled) against those of the policy entries in the SPD.

4. Check whether the required IPsec processing has been applied, i.e., verify that the SA's found in (1) and (2) match the kind and order of SAs required by the policy found in (3).

   NOTE: The correct "matching" policy will not necessarily be the first inbound policy found. If the check in (4) fails, steps (3) and (4) are repeated until all policy entries have been checked or until the check succeeds.

At the end of these steps, pass the resulting packet to the Transport Layer or forward the packet. Note that any IPsec headers processed in these steps may have been removed, but that this information, i.e., what SAs were used and the order of their application, may be needed for subsequent IPsec or firewall processing.

Note that in the case of a security gateway, if forwarding causes a packet to exit via an IPsec-enabled interface, then additional IPsec processing may be applied.

5.2.2 Handling of AH and ESP tunnels

The handling of the inner and outer IP headers, extension headers, and options for AH and ESP tunnels should be performed as described in the tables in Section 5.1.

6. ICMP Processing (relevant to IPsec)

The focus of this section is on the handling of ICMP error messages. Other ICMP traffic, e.g., Echo/Reply, should be treated like other traffic and can be protected on an end-to-end basis using SAs in the usual fashion.

An ICMP error message protected by AH or ESP and generated by a router SHOULD be processed and forwarded in a tunnel mode SA. Local policy determines whether or not it is subjected to source address checks by the router at the destination end of the tunnel. Note that if the router at the originating end of the tunnel is forwarding an ICMP error message from another router, the source address check would fail. An ICMP message protected by AH or ESP and generated by a router MUST NOT be forwarded on a transport mode SA (unless the SA has been established to the router acting as a host, e.g., a Telnet connection used to manage a router). An ICMP message generated by a host SHOULD be checked against the source IP address selectors bound to the SA in which the message arrives. Note that even if the source of an ICMP error message is authenticated, the returned IP header could be invalid. Accordingly, the selector values in the IP header SHOULD also be checked to be sure that they are consistent with the selectors for the SA over which the ICMP message was received.

The table in Appendix D characterize ICMP messages as being either host generated, router generated, both, unknown/unassigned. ICMP messages falling into the last two categories should be handled as determined by the receiver's policy.

An ICMP message not protected by AH or ESP is unauthenticated and its processing and/or forwarding may result in denial of service. This suggests that, in general, it would be desirable to ignore such messages. However, it is expected that many routers (vs. security gateways) will not implement IPsec for transit traffic and thus strict adherence to this rule would cause many ICMP messages to be discarded. The result is that some critical IP functions would be lost, e.g., redirection and PMTU processing. Thus it MUST be possible to configure an IPsec implementation to accept or reject (router) ICMP traffic as per local security policy.

The remainder of this section addresses how PMTU processing MUST be performed at hosts and security gateways. It addresses processing of both authenticated and unauthenticated ICMP PMTU messages. However, as noted above, unauthenticated ICMP messages MAY be discarded based on local policy.

6.1 PMTU/DF Processing

6.1.1 DF Bit

In cases where a system (host or gateway) adds an encapsulating header (ESP tunnel or AH tunnel), it MUST support the option of copying the DF bit from the original packet to the encapsulating header (and processing ICMP PMTU messages). This means that it MUST be possible to configure the system's treatment of the DF bit (set, clear, copy from encapsulated header) for each interface. (See Appendix B for rationale.)

6.1.2 Path MTU Discovery (PMTU)

This section discusses IPsec handling for Path MTU Discovery messages. ICMP PMTU is used here to refer to an ICMP message for:

```
        IPv4 (RFC 792):
                - Type = 3 (Destination Unreachable)
                - Code = 4 (Fragmentation needed and DF set)
                - Next-Hop MTU in the low-order 16 bits of the second
                  word of the ICMP header (labelled "unused" in RFC
                  792), with high-order 16 bits set to zero

        IPv6 (RFC 1885):
                - Type = 2 (Packet Too Big)
                - Code = 0 (Fragmentation needed)
                - Next-Hop MTU in the 32 bit MTU field of the ICMP6
                  message
```

6.1.2.1 Propagation of PMTU

The amount of information returned with the ICMP PMTU message (IPv4 or IPv6) is limited and this affects what selectors are available for use in further propagating the PMTU information. (See Appendix B for more detailed discussion of this topic.)

o PMTU message with 64 bits of IPsec header -- If the ICMP PMTU message contains only 64 bits of the IPsec header (minimum for IPv4), then a security gateway MUST support the following options on a per SPI/SA basis:

a. if the originating host can be determined (or the possible sources narrowed down to a manageable number), send the PM information to all the possible originating hosts.
b. if the originating host cannot be determined, store the PMTU with the SA and wait until the next packet(s) arrive from the originating host for the relevant security association. If

the packet(s) are bigger than the PMTU, drop the packet(s), and compose ICMP PMTU message(s) with the new packet(s) and the updated PMTU, and send the ICMP message(s) about the problem to the originating host. Retain the PMTU information for any message that might arrive subsequently (see Section 6.1.2.4, "PMTU Aging").

- PMTU message with >64 bits of IPsec header -- If the ICMP message contains more information from the original packet then there may be enough non-opaque information to immediately determine to which host to propagate the ICMP/PMTU message and to provide that system with the 5 fields (source address, destination address, source port, destination port, transport protocol) needed to determine where to store/update the PMTU. Under such circumstances, a security gateway MUST generate an ICMP PMTU message immediately upon receipt of an ICMP PMTU from further down the path.

- Distributing the PMTU to the Transport Layer -- The host mechanism for getting the updated PMTU to the transport layer is unchanged, as specified in RFC 1191 (Path MTU Discovery).

6.1.2.2 Calculation of PMTU

The calculation of PMTU from an ICMP PMTU MUST take into account the addition of any IPsec header -- AH transport, ESP transport, AH/ESP transport, ESP tunnel, AH tunnel. (See Appendix B for discussion of implementation issues.)

Note: In some situations the addition of IPsec headers could result in an effective PMTU (as seen by the host or application) that is unacceptably small. To avoid this problem, the implementation may establish a threshold below which it will not report a reduced PMTU. In such cases, the implementation would apply IPsec and then fragment the resulting packet according to the PMTU. This would result in a more efficient use of the available bandwidth.

6.1.2.3 Granularity of PMTU Processing

In hosts, the granularity with which ICMP PMTU processing can be done differs depending on the implementation situation. Looking at a host, there are 3 situations that are of interest with respect to PMTU issues (See Appendix B for additional details on this topic.):

a. Integration of IPsec into the native IP implementation
b. Bump-in-the-stack implementations, where IPsec is implemented "underneath" an existing implementation of a TCP/IP protocol stack, between the native IP and the local network drivers

c. No IPsec implementation -- This case is included because it is relevant in cases where a security gateway is sending PMTU information back to a host.

Only in case (a) can the PMTU data be maintained at the same granularity as communication associations. In (b) and (c), the IP layer will only be able to maintain PMTU data at the granularity of source and destination IP addresses (and optionally TOS), as described in RFC 1191. This is an important difference, because more than one communication association may map to the same source and destination IP addresses, and each communication association may have a different amount of IPsec header overhead (e.g., due to use of different transforms or different algorithms).

Implementation of the calculation of PMTU and support for PMTUs at the granularity of individual communication associations is a local matter. However, a socket-based implementation of IPsec in a host SHOULD maintain the information on a per socket basis. Bump in the stack systems MUST pass an ICMP PMTU to the host IP implementation, after adjusting it for any IPsec header overhead added by these systems. The calculation of the overhead SHOULD be determined by analysis of the SPI and any other selector information present in a returned ICMP PMTU message.

6.1.2.4 PMTU Aging

In all systems (host or gateway) implementing IPsec and maintaining PMTU information, the PMTU associated with a security association (transport or tunnel) MUST be "aged" and some mechanism put in place for updating the PMTU in a timely manner, especially for discovering if the PMTU is smaller than it needs to be. A given PMTU has to remain in place long enough for a packet to get from the source end of the security association to the system at the other end of the security association and propagate back an ICMP error message if the current PMTU is too big. Note that if there are nested tunnels, multiple packets and round trip times might be required to get an ICMP message back to an encapsulator or originating host.

Systems SHOULD use the approach described in the Path MTU Discovery document (RFC 1191, Section 6.3), which suggests periodically resetting the PMTU to the first-hop data-link MTU and then letting the normal PMTU Discovery processes update the PMTU as necessary. The period SHOULD be configurable.

7. Auditing

Not all systems that implement IPsec will implement auditing. For the most part, the granularity of auditing is a local matter. However, several auditable events are identified in the AH and ESP specifications and for each of these events a minimum set of information that SHOULD be included in an audit log is defined. Additional information also MAY be included in the audit log for each of these events, and additional events, not explicitly called out in this specification, also MAY result in audit log entries. There is no requirement for the receiver to transmit any message to the purported transmitter in response to the detection of an auditable event, because of the potential to induce denial of service via such action.

8. Use in Systems Supporting Information Flow Security

Information of various sensitivity levels may be carried over a single network. Information labels (e.g., Unclassified, Company Proprietary, Secret) [DoD85, DoD87] are often employed to distinguish such information. The use of labels facilitates segregation of information, in support of information flow security models, e.g., the Bell-LaPadula model [BL73]. Such models, and corresponding supporting technology, are designed to prevent the unauthorized flow of sensitive information, even in the face of Trojan Horse attacks. Conventional, discretionary access control (DAC) mechanisms, e.g., based on access control lists, generally are not sufficient to support such policies, and thus facilities such as the SPD do not suffice in such environments.

In the military context, technology that supports such models is often referred to as multi-level security (MLS). Computers and networks often are designated "multi-level secure" if they support the separation of labelled data in conjunction with information flow security policies. Although such technology is more broadly applicable than just military applications, this document uses the acronym "MLS" to designate the technology, consistent with much extant literature.

IPsec mechanisms can easily support MLS networking. MLS networking requires the use of strong Mandatory Access Controls (MAC), which unprivileged users or unprivileged processes are incapable of controlling or violating. This section pertains only to the use of these IP security mechanisms in MLS (information flow security policy) environments. Nothing in this section applies to systems not claiming to provide MLS.

As used in this section, "sensitivity information" might include implementation-defined hierarchic levels, categories, and/or releasability information.

AH can be used to provide strong authentication in support of mandatory access control decisions in MLS environments. If explicit IP sensitivity information (e.g., IPSO [Ken91]) is used and confidentiality is not considered necessary within the particular operational environment, AH can be used to authenticate the binding between sensitivity labels in the IP header and the IP payload (including user data). This is a significant improvement over labeled IPv4 networks where the sensitivity information is trusted even though there is no authentication or cryptographic binding of the information to the IP header and user data. IPv4 networks might or might not use explicit labelling. IPv6 will normally use implicit sensitivity information that is part of the IPsec Security Association but not transmitted with each packet instead of using explicit sensitivity information. All explicit IP sensitivity information MUST be authenticated using either ESP, AH, or both.

Encryption is useful and can be desirable even when all of the hosts are within a protected environment, for example, behind a firewall or disjoint from any external connectivity. ESP can be used, in conjunction with appropriate key management and encryption algorithms, in support of both DAC and MAC. (The choice of encryption and authentication algorithms, and the assurance level of an IPsec implementation will determine the environments in which an implementation may be deemed sufficient to satisfy MLS requirements.) Key management can make use of sensitivity information to provide MAC. IPsec implementations on systems claiming to provide MLS SHOULD be capable of using IPsec to provide MAC for IP-based communications.

8.1 Relationship Between Security Associations and Data Sensitivity

Both the Encapsulating Security Payload and the Authentication Header can be combined with appropriate Security Association policies to provide multi-level secure networking. In this case each SA (or SA bundle) is normally used for only a single instance of sensitivity information. For example, "PROPRIETARY - Internet Engineering" must be associated with a different SA (or SA bundle) from "PROPRIETARY - Finance".

8.2 Sensitivity Consistency Checking

An MLS implementation (both host and router) MAY associate sensitivity information, or a range of sensitivity information with an interface, or a configured IP address with its associated prefix (the latter is sometimes referred to as a logical interface, or an interface alias). If such properties exist, an implementation SHOULD compare the sensitivity information associated with the packet against the sensitivity information associated with the interface or address/prefix from which the packet arrived, or through which the packet will depart. This check will either verify that the sensitivities match, or that the packet's sensitivity falls within the range of the interface or address/prefix.

The checking SHOULD be done on both inbound and outbound processing.

8.3 Additional MLS Attributes for Security Association Databases

Section 4.4 discussed two Security Association databases (the Security Policy Database (SPD) and the Security Association Database (SAD)) and the associated policy selectors and SA attributes. MLS networking introduces an additional selector/attribute:

- Sensitivity information.

The Sensitivity information aids in selecting the appropriate algorithms and key strength, so that the traffic gets a level of protection appropriate to its importance or sensitivity as described in section 8.1. The exact syntax of the sensitivity information is implementation defined.

8.4 Additional Inbound Processing Steps for MLS Networking

After an inbound packet has passed through IPsec processing, an MLS implementation SHOULD first check the packet's sensitivity (as defined by the SA (or SA bundle) used for the packet) with the interface or address/prefix as described in section 8.2 before delivering the datagram to an upper-layer protocol or forwarding it.

The MLS system MUST retain the binding between the data received in an IPsec protected packet and the sensitivity information in the SA or SAs used for processing, so appropriate policy decisions can be made when delivering the datagram to an application or forwarding engine. The means for maintaining this binding are implementation specific.

8.5 Additional Outbound Processing Steps for MLS Networking

An MLS implementation of IPsec MUST perform two additional checks besides the normal steps detailed in section 5.1.1. When consulting the SPD or the SAD to find an outbound security association, the MLS implementation MUST use the sensitivity of the data to select an appropriate outbound SA or SA bundle. The second check comes before forwarding the packet out to its destination, and is the sensitivity consistency checking described in section 8.2.

8.6 Additional MLS Processing for Security Gateways

An MLS security gateway MUST follow the previously mentioned inbound and outbound processing rules as well as perform some additional processing specific to the intermediate protection of packets in an MLS environment.

A security gateway MAY act as an outbound proxy, creating SAs for MLS systems that originate packets forwarded by the gateway. These MLS systems may explicitly label the packets to be forwarded, or the whole originating network may have sensitivity characteristics associated with it. The security gateway MUST create and use appropriate SAs for AH, ESP, or both, to protect such traffic it forwards.

Similarly such a gateway SHOULD accept and process inbound AH and/or ESP packets and forward appropriately, using explicit packet labeling, or relying on the sensitivity characteristics of the destination network.

9. Performance Issues

The use of IPsec imposes computational performance costs on the hosts or security gateways that implement these protocols. These costs are associated with the memory needed for IPsec code and data structures, and the computation of integrity check values, encryption and decryption, and added per-packet handling. The per-packet computational costs will be manifested by increased latency and, possibly, reduced throughout. Use of SA/key management protocols, especially ones that employ public key cryptography, also adds computational performance costs to use of IPsec. These per-association computational costs will be manifested in terms of increased latency in association establishment. For many hosts, it is anticipated that software-based cryptography will not appreciably reduce throughput, but hardware may be required for security gateways (since they represent aggregation points), and for some hosts.

The use of IPsec also imposes bandwidth utilization costs on transmission, switching, and routing components of the Internet infrastructure, components not implementing IPsec. This is due to the increase in the packet size resulting from the addition of AH and/or ESP headers, AH and ESP tunneling (which adds a second IP header), and the increased packet traffic associated with key management protocols. It is anticipated that, in most instances, this increased bandwidth demand will not noticeably affect the Internet infrastructure. However, in some instances, the effects may be significant, e.g., transmission of ESP encrypted traffic over a dialup link that otherwise would have compressed the traffic.

Note: The initial SA establishment overhead will be felt in the first packet. This delay could impact the transport layer and application. For example, it could cause TCP to retransmit the SYN before the ISAKMP exchange is done. The effect of the delay would be different on UDP than TCP because TCP shouldn't transmit anything other than the SYN until the connection is set up whereas UDP will go ahead and transmit data beyond the first packet.

Note: As discussed earlier, compression can still be employed at layers above IP. There is an IETF working group (IP Payload Compression Protocol (ippcp)) working on "protocol specifications that make it possible to perform lossless compression on individual payloads before the payload is processed by a protocol that encrypts it. These specifications will allow for compression operations to be performed prior to the encryption of a payload by IPsec protocols."

10. Conformance Requirements

All IPv4 systems that claim to implement IPsec MUST comply with all requirements of the Security Architecture document. All IPv6 systems MUST comply with all requirements of the Security Architecture document.

11. Security Considerations

The focus of this document is security; hence security considerations permeate this specification.

12. Differences from RFC 1825

This architecture document differs substantially from RFC 1825 in detail and in organization, but the fundamental notions are unchanged. This document provides considerable additional detail in terms of compliance specifications. It introduces the SPD and SAD, and the notion of SA selectors. It is aligned with the new versions of AH and ESP, which also differ from their predecessors. Specific requirements for supported combinations of AH and ESP are newly added, as are details of PMTU management.

Acknowledgements

Many of the concepts embodied in this specification were derived from or influenced by the US Government's SP3 security protocol, ISO/IEC's NLSP, the proposed swIPe security protocol [SDNS, ISO, IB93, IBK93], and the work done for SNMP Security and SNMPv2 Security.

For over 3 years (although it sometimes seems *much* longer), this document has evolved through multiple versions and iterations. During this time, many people have contributed significant ideas and energy to the process and the documents themselves. The authors would like to thank Karen Seo for providing extensive help in the review, editing, background research, and coordination for this version of the specification. The authors would also like to thank the members of the IPsec and IPng working groups, with special mention of the efforts of (in alphabetic order): Steve Bellovin, Steve Deering, James Hughes, Phil Karn, Frank Kastenholz, Perry Metzger, David Mihelcic, Hilarie Orman, Norman Shulman, William Simpson, Harry Varnis, and Nina Yuan.

Appendix A -- Glossary

This section provides definitions for several key terms that are employed in this document. Other documents provide additional definitions and background information relevant to this technology, e.g., [VK83, HA94]. Included in this glossary are generic security service and security mechanism terms, plus IPsec-specific terms.

Access Control

Access control is a security service that prevents unauthorized use of a resource, including the prevention of use of a resource in an unauthorized manner. In the IPsec context, the resource to which access is being controlled is often:

- o for a host, computing cycles or data
- o for a security gateway, a network behind the gateway

or

bandwidth on that network.

Anti-replay

[See "Integrity" below]

Authentication

This term is used informally to refer to the combination of two nominally distinct security services, data origin authentication and connectionless integrity. See the definitions below for each of these services.

Availability

Availability, when viewed as a security service, addresses the security concerns engendered by attacks against networks that deny or degrade service. For example, in the IPsec context, the use of anti-replay mechanisms in AH and ESP support availability.

Confidentiality

Confidentiality is the security service that protects data from unauthorized disclosure. The primary confidentiality concern in most instances is unauthorized disclosure of application level data, but disclosure of the external characteristics of communication also can be a concern in some circumstances. Traffic flow confidentiality is the service that addresses this latter concern by concealing source and destination addresses, message length, or frequency of communication. In the IPsec context, using ESP in tunnel mode, especially at a security gateway, can provide some level of traffic flow confidentiality. (See also traffic analysis, below.)

Encryption

Encryption is a security mechanism used to transform data from an intelligible form (plaintext) into an unintelligible form (ciphertext), to provide confidentiality. The inverse transformation process is designated "decryption". Oftimes the term "encryption" is used to generically refer to both processes.

Data Origin Authentication

Data origin authentication is a security service that verifies the identity of the claimed source of data. This service is usually bundled with connectionless integrity service.

Integrity

Integrity is a security service that ensures that modifications to data are detectable. Integrity comes in various flavors to match application requirements. IPsec supports two forms of integrity: connectionless and a form of partial sequence integrity. Connectionless integrity is a service that detects modification of an individual IP datagram, without regard to the ordering of the datagram in a stream of traffic. The form of partial sequence integrity offered in IPsec is referred to as anti-replay integrity, and it detects arrival of duplicate IP datagrams (within a constrained window). This is in contrast to connection-oriented integrity, which imposes more stringent sequencing requirements on traffic, e.g., to be able to detect lost or re-ordered messages. Although authentication and integrity services often are cited separately, in practice they are intimately connected and almost always offered in tandem.

Security Association (SA)

A simplex (uni-directional) logical connection, created for security purposes. All traffic traversing an SA is provided the same security processing. In IPsec, an SA is an internet layer abstraction implemented through the use of AH or ESP.

Security Gateway

A security gateway is an intermediate system that acts as the communications interface between two networks. The set of hosts (and networks) on the external side of the security gateway is viewed as untrusted (or less trusted), while the networks and hosts and on the internal side are viewed as trusted (or more trusted). The internal subnets and hosts served by a security gateway are presumed to be trusted by virtue of sharing a common, local, security administration. (See "Trusted Subnetwork" below.) In the IPsec context, a security gateway is a point at which AH and/or ESP is implemented in order to serve a set of internal hosts, providing security services for these hosts when they communicate with external hosts also employing IPsec (either directly or via another security gateway).

SPI

Acronym for "Security Parameters Index". The combination of a destination address, a security protocol, and an SPI uniquely identifies a security association (SA, see above). The SPI is carried in AH and ESP protocols to enable the receiving system to select the SA under which a received packet will be processed. An SPI has only local significance, as defined by the creator of the SA (usually the receiver of the packet carrying the SPI); thus an SPI is generally viewed as an opaque bit string. However, the creator of an SA may choose to interpret the bits in an SPI to facilitate local processing.

Traffic Analysis

The analysis of network traffic flow for the purpose of deducing information that is useful to an adversary. Examples of such information are frequency of transmission, the identities of the conversing parties, sizes of packets, flow identifiers, etc. [Sch94]

Trusted Subnetwork

A subnetwork containing hosts and routers that trust each other not to engage in active or passive attacks. There also is an assumption that the underlying communications channel (e.g., a LAN or CAN) isn't being attacked by other means.

Appendix B -- Analysis/Discussion of PMTU/DF/Fragmentation Issues

B.1 DF bit

In cases where a system (host or gateway) adds an encapsulating header (e.g., ESP tunnel), should/must the DF bit in the original packet be copied to the encapsulating header?

Fragmenting seems correct for some situations, e.g., it might be appropriate to fragment packets over a network with a very small MTU, e.g., a packet radio network, or a cellular phone hop to mobile node, rather than propagate back a very small PMTU for use over the rest of the path. In other situations, it might be appropriate to set the DF bit in order to get feedback from later routers about PMTU constraints which require fragmentation. The existence of both of these situations argues for enabling a system to decide whether or not to fragment over a particular network "link", i.e., for requiring an implementation to be able to copy the DF bit (and to process ICMP PMTU messages), but making it an option to be selected on a per interface basis. In other words, an administrator should be able to configure the router's treatment of the DF bit (set, clear, copy from encapsulated header) for each interface.

Note: If a bump-in-the-stack implementation of IPsec attempts to apply different IPsec algorithms based on source/destination ports, it will be difficult to apply Path MTU adjustments.

B.2 Fragmentation

If required, IP fragmentation occurs after IPsec processing within an IPsec implementation. Thus, transport mode AH or ESP is applied only to whole IP datagrams (not to IP fragments). An IP packet to which AH or ESP has been applied may itself be fragmented by routers en route, and such fragments MUST be reassembled prior to IPsec processing at a receiver. In tunnel mode, AH or ESP is applied to an IP packet, the payload of which may be a fragmented IP packet. For example, a security gateway, "bump-in-the-stack" (BITS), or "bump-in-the-wire" (BITW) IPsec implementation may apply tunnel mode AH to such fragments. Note that BITS or BITW implementations are examples of where a host IPsec implementation might receive fragments to which tunnel mode is to be applied. However, if transport mode is to be applied, then these implementations MUST reassemble the fragments prior to applying IPsec.

NOTE: IPsec always has to figure out what the encapsulating IP header fields are. This is independent of where you insert IPsec and is intrinsic to the definition of IPsec. Therefore any IPsec implementation that is not integrated into an IP implementation must include code to construct the necessary IP headers (e.g., IP2):

o AH-tunnel --> IP2-AH-IP1-Transport-Data
o ESP-tunnel --> IP2-ESP_hdr-IP1-Transport-Data-ESP_trailer

****

Overall, the fragmentation/reassembly approach described above works for all cases examined.

	AH Xport		AH Tunnel		ESP Xport		ESP Tunnel	
Implementation approach	IPv4	IPv6	IPv4	IPv6	IPv4	IPv6	IPv4	IPv6
Hosts (integr w/ IP stack)	Y	Y	Y	Y	Y	Y	Y	Y
Hosts (betw/ IP and drivers)	Y	Y	Y	Y	Y	Y	Y	Y
S. Gwy (integr w/ IP stack)			Y	Y			Y	Y
Outboard crypto processor *								

* If the crypto processor system has its own IP address, then it is covered by the security gateway case. This box receives the packet from the host and performs IPsec processing. It has to be able to handle the same AH, ESP, and related IPv4/IPv6 tunnel processing that a security gateway would have to handle. If it doesn't have it's own address, then it is similar to the bump-in-the stack implementation between IP and the network drivers.

The following analysis assumes that:

1. There is only one IPsec module in a given system's stack. There isn't an IPsec module A (adding ESP/encryption and thus) hiding the transport protocol, SRC port, and DEST port from IPsec module B.
2. There are several places where IPsec could be implemented (as shown in the table above).
    a. Hosts with integration of IPsec into the native IP implementation. Implementer has access to the source for the stack.
    b. Hosts with bump-in-the-stack implementations, where IPsec is implemented between IP and the local network drivers. Source access for stack is not available; but there are well-defined interfaces that allows the IPsec code to be incorporated into the system.

c. Security gateways and outboard crypto processors with integration of IPsec into the stack.

3. Not all of the above approaches are feasible in all hosts. But it was assumed that for each approach, there are some hosts for whom the approach is feasible.

For each of the above 3 categories, there are IPv4 and IPv6, AH transport and tunnel modes, and ESP transport and tunnel modes -- for a total of 24 cases (3 x 2 x 4).

Some header fields and interface fields are listed here for ease of reference -- they're not in the header order, but instead listed to allow comparison between the columns. (* = not covered by AH authentication. ESP authentication doesn't cover any headers that precede it.)

		IP/Transport Interface
IPv4	IPv6	(RFC 1122 -- Sec 3.4)
Version = 4	Version = 6	
Header Len		
*TOS	Class,Flow Lbl	TOS
Packet Len	Payload Len	Len
ID		ID (optional)
*Flags		DF
*Offset		
*TTL	*Hop Limit	TTL
Protocol	Next Header	
*Checksum		
Src Address	Src Address	Src Address
Dst Address	Dst Address	Dst Address
Options?	Options?	Opt

? = AH covers Option-Type and Option-Length, but might not cover Option-Data.

The results for each of the 20 cases is shown below ("works" = will work if system fragments after outbound IPsec processing, reassembles before inbound IPsec processing). Notes indicate implementation issues.

a. Hosts (integrated into IP stack)
   - o AH-transport --> (IP1-AH-Transport-Data)
     - - IPv4 -- works
     - - IPv6 -- works
   - o AH-tunnel --> (IP2-AH-IP1-Transport-Data)
     - - IPv4 -- works
     - - IPv6 -- works

- ESP-transport --> (IP1-ESP_hdr-Transport-Data-ESP_trailer)
  - IPv4 -- works
  - IPv6 -- works
- ESP-tunnel --> (IP2-ESP_hdr-IP1-Transport-Data-ESP_trailer)
  - IPv4 -- works
  - IPv6 -- works

b. Hosts (Bump-in-the-stack) -- put IPsec between IP layer and network drivers. In this case, the IPsec module would have to do something like one of the following for fragmentation and reassembly.

   - do the fragmentation/reassembly work itself and send/receive the packet directly to/from the network layer. In AH or ESP transport mode, this is fine. In AH or ESP tunnel mode where the tunnel end is at the ultimate destination, this is fine. But in AH or ESP tunnel modes where the tunnel end is different from the ultimate destination and where the source host is multi-homed, this approach could result in sub-optimal routing because the IPsec module may be unable to obtain the information needed (LAN interface and next-hop gateway) to direct the packet to the appropriate network interface. This is not a problem if the interface and next-hop gateway are the same for the ultimate destination and for the tunnel end. But if they are different, then IPsec would need to know the LAN interface and the next-hop gateway for the tunnel end. (Note: The tunnel end (security gateway) is highly likely to be on the regular path to the ultimate destination. But there could also be more than one path to the destination, e.g., the host could be at an organization with 2 firewalls. And the path being used could involve the less commonly chosen firewall.) OR
   - pass the IPsec'd packet back to the IP layer where an extra IP header would end up being pre-pended and the IPsec module would have to check and let IPsec'd fragments go by.

     OR
   - pass the packet contents to the IP layer in a form such that the IP layer recreates an appropriate IP header

At the network layer, the IPsec module will have access to the following selectors from the packet -- SRC address, DST address, Next Protocol, and if there's a transport layer header --> SRC port and DST port. One cannot assume IPsec has access to the Name. It is assumed that the available selector information is sufficient to figure out the relevant Security Policy entry and Security Association(s).

```
      o AH-transport  --> (IP1-AH-Transport-Data)
                - IPv4 -- works
                - IPv6 -- works
      o AH-tunnel --> (IP2-AH-IP1-Transport-Data)
                - IPv4 -- works
                - IPv6 -- works
      o ESP-transport --> (IP1-ESP_hdr-Transport-Data-ESP_trailer)
                - IPv4 -- works
                - IPv6 -- works
      o ESP-tunnel -->  (IP2-ESP_hdr-IP1-Transport-Data-ESP_trailer)
                - IPv4 -- works
                - IPv6 -- works
```

c. Security gateways -- integrate IPsec into the IP stack

NOTE: The IPsec module will have access to the following selectors from the packet -- SRC address, DST address, Next Protocol, and if there's a transport layer header --> SRC port and DST port. It won't have access to the User ID (only Hosts have access to User ID information.) Unlike some Bump-in-the-stack implementations, security gateways may be able to look up the Source Address in the DNS to provide a System Name, e.g., in situations involving use of dynamically assigned IP addresses in conjunction with dynamically updated DNS entries. It also won't have access to the transport layer information if there is an ESP header, or if it's not the first fragment of a fragmented message. It is assumed that the available selector information is sufficient to figure out the relevant Security Policy entry and Security Association(s).

```
      o AH-tunnel --> (IP2-AH-IP1-Transport-Data)
                - IPv4 -- works
                - IPv6 -- works
      o ESP-tunnel -->  (IP2-ESP_hdr-IP1-Transport-Data-ESP_trailer)
                - IPv4 -- works
                - IPv6 -- works
```

****

B.3 Path MTU Discovery

As mentioned earlier, "ICMP PMTU" refers to an ICMP message used for Path MTU Discovery.

The legend for the diagrams below in B.3.1 and B.3.3 (but not B.3.2) is:

```
     ==== = security association (AH or ESP, transport or tunnel)
```

---- = connectivity (or if so labelled, administrative boundary)
.... = ICMP message (hereafter referred to as ICMP PMTU) for

IPv4:
- Type = 3 (Destination Unreachable)
- Code = 4 (Fragmentation needed and DF set)
- Next-Hop MTU in the low-order 16 bits of the second word of the ICMP header (labelled unused in RFC 792), with high-order 16 bits set to zero

IPv6 (RFC 1885):
- Type = 2 (Packet Too Big)
- Code = 0 (Fragmentation needed and DF set)
- Next-Hop MTU in the 32 bit MTU field of the ICMP6

Hx = host x
Rx = router x
SGx = security gateway x
X* = X supports IPsec

B.3.1 Identifying the Originating Host(s)

The amount of information returned with the ICMP message is limited and this affects what selectors are available to identify security associations, originating hosts, etc. for use in further propagating the PMTU information.

In brief... An ICMP message must contain the following information from the "offending" packet:
- IPv4 (RFC 792) -- IP header plus a minimum of 64 bits

Accordingly, in the IPv4 context, an ICMP PMTU may identify only the first (outermost) security association. This is because the ICMP PMTU may contain only 64 bits of the "offending" packet beyond the IP header, which would capture only the first SPI from AH or ESP. In the IPv6 context, an ICMP PMTU will probably provide all the SPIs and the selectors in the IP header, but maybe not the SRC/DST ports (in the transport header) or the encapsulated (TCP, UDP, etc.) protocol. Moreover, if ESP is used, the transport ports and protocol selectors may be encrypted.

Looking at the diagram below of a security gateway tunnel (as mentioned elsewhere, security gateways do not use transport mode)...

```
    H1   ===================           H3
      \  |                 |          /
   H0 -- SG1* ---- R1 ---- SG2* ---- R2 -- H5
      /  ^        |                   \
    H2   |........|                    H4
```

Suppose that the security policy for SG1 is to use a single SA to SG2 for all the traffic between hosts H0, H1, and H2 and hosts H3, H4, and H5. And suppose H0 sends a data packet to H5 which causes R1 to send an ICMP PMTU message to SG1. If the PMTU message has only the SPI, SG1 will be able to look up the SA and find the list of possible hosts (H0, H1, H2, wildcard); but SG1 will have no way to figure out that H0 sent the traffic that triggered the ICMP PMTU message.

original packet	after IPsec processing	ICMP packet
		IP-3 header (S = R1, D = SG1)
		ICMP header (includes PMTU)
	IP-2 header	IP-2 header (S = SG1, D = SG2)
	ESP header	minimum of 64 bits of ESP hdr (*)
IP-1 header	IP-1 header	
TCP header	TCP header	
TCP data	TCP data	
	ESP trailer	

(*) The 64 bits will include enough of the ESP (or AH) header to include the SPI.
- ESP -- SPI (32 bits), Seq number (32 bits)
- AH -- Next header (8 bits), Payload Len (8 bits), Reserved (16 bits), SPI (32 bits)

This limitation on the amount of information returned with an ICMP message creates a problem in identifying the originating hosts for the packet (so as to know where to further propagate the ICMP PMTU information). If the ICMP message contains only 64 bits of the IPsec header (minimum for IPv4), then the IPsec selectors (e.g., Source and Destination addresses, Next Protocol, Source and Destination ports, etc.) will have been lost. But the ICMP error message will still provide SG1 with the SPI, the PMTU information and the source and destination gateways for the relevant security association.

The destination security gateway and SPI uniquely define a security association which in turn defines a set of possible originating hosts. At this point, SG1 could:

a. send the PMTU information to all the possible originating hosts. This would not work well if the host list is a wild card or if many/most of the hosts weren't sending to SG1; but it might work if the SPI/destination/etc mapped to just one or a small number of hosts.
b. store the PMTU with the SPI/etc and wait until the next packet(s) arrive from the originating host(s) for the relevant security association. If it/they are bigger than the PMTU, drop the packet(s), and compose ICMP PMTU message(s) with the new packet(s) and the updated PMTU, and send the originating host(s) the ICMP message(s) about the problem. This involves a delay in notifying the originating host(s), but avoids the problems of (a).

Since only the latter approach is feasible in all instances, a security gateway MUST provide such support, as an option. However, if the ICMP message contains more information from the original packet, then there may be enough information to immediately determine to which host to propagate the ICMP/PMTU message and to provide that system with the 5 fields (source address, destination address, source port, destination port, and transport protocol) needed to determine where to store/update the PMTU. Under such circumstances, a security gateway MUST generate an ICMP PMTU message immediately upon receipt of an ICMP PMTU from further down the path. NOTE: The Next Protocol field may not be contained in the ICMP message and the use of ESP encryption may hide the selector fields that have been encrypted.

B.3.2 Calculation of PMTU

The calculation of PMTU from an ICMP PMTU has to take into account the addition of any IPsec header by H1 -- AH and/or ESP transport, or ESP or AH tunnel. Within a single host, multiple applications may share an SPI and nesting of security associations may occur. (See Section 4.5 Basic Combinations of Security Associations for description of the combinations that MUST be supported). The diagram below illustrates an example of security associations between a pair of hosts (as viewed from the perspective of one of the hosts.) (ESPx or AHx = transport mode)

```
Socket 1 -------------------------|
                                  |
Socket 2 (ESPx/SPI-A) ---------- AHx (SPI-B) -- Internet
```

In order to figure out the PMTU for each socket that maps to SPI-B, it will be necessary to have backpointers from SPI-B to each of the 2 paths that lead to it -- Socket 1 and Socket 2/SPI-A.

B.3.3 Granularity of Maintaining PMTU Data

In hosts, the granularity with which PMTU ICMP processing can be done differs depending on the implementation situation. Looking at a host, there are three situations that are of interest with respect to PMTU issues:

a. Integration of IPsec into the native IP implementation
b. Bump-in-the-stack implementations, where IPsec is implemented "underneath" an existing implementation of a TCP/IP protocol stack, between the native IP and the local network drivers
c. No IPsec implementation -- This case is included because it is relevant in cases where a security gateway is sending PMTU information back to a host.

Only in case (a) can the PMTU data be maintained at the same granularity as communication associations. In the other cases, the IP layer will maintain PMTU data at the granularity of Source and Destination IP addresses (and optionally TOS/Class), as described in RFC 1191. This is an important difference, because more than one communication association may map to the same source and destination IP addresses, and each communication association may have a different amount of IPsec header overhead (e.g., due to use of different transforms or different algorithms). The examples below illustrate this.

In cases (a) and (b)... Suppose you have the following situation. H1 is sending to H2 and the packet to be sent from R1 to R2 exceeds the PMTU of the network hop between them.

```
                 ==================================
                 |                                |
                H1* --- R1 ----- R2 ---- R3 ---- H2*
                 ^       |
                 |.......|
```

If R1 is configured to not fragment subscriber traffic, then R1 sends an ICMP PMTU message with the appropriate PMTU to H1. H1's processing would vary with the nature of the implementation. In case (a) (native IP), the security services are bound to sockets or the equivalent. Here the IP/IPsec implementation in H1 can store/update the PMTU for the associated socket. In case (b), the IP layer in H1 can store/update the PMTU but only at the granularity of Source and Destination addresses and possibly TOS/Class, as noted above. So the result may be sub-optimal, since the PMTU for a given SRC/DST/TOS/Class will be the subtraction of the largest amount of IPsec header used for any communication association between a given source and destination.

In case (c), there has to be a security gateway to have any IPsec processing. So suppose you have the following situation. H1 is sending to H2 and the packet to be sent from SG1 to R exceeds the PMTU of the network hop between them.

```
                      ================
                      |              |
             H1 ---- SG1* --- R --- SG2* ---- H2
              ^       |
              |.......|
```

As described above for case (b), the IP layer in H1 can store/update the PMTU but only at the granularity of Source and Destination addresses, and possibly TOS/Class. So the result may be sub-optimal, since the PMTU for a given SRC/DST/TOS/Class will be the subtraction of the largest amount of IPsec header used for any communication association between a given source and destination.

B.3.4 Per Socket Maintenance of PMTU Data

Implementation of the calculation of PMTU (Section B.3.2) and support for PMTUs at the granularity of individual "communication associations" (Section B.3.3) is a local matter. However, a socket-based implementation of IPsec in a host SHOULD maintain the information on a per socket basis. Bump in the stack systems MUST pass an ICMP PMTU to the host IP implementation, after adjusting it for any IPsec header overhead added by these systems. The determination of the overhead SHOULD be determined by analysis of the SPI and any other selector information present in a returned ICMP PMTU message.

B.3.5 Delivery of PMTU Data to the Transport Layer

The host mechanism for getting the updated PMTU to the transport layer is unchanged, as specified in RFC 1191 (Path MTU Discovery).

B.3.6 Aging of PMTU Data

This topic is covered in Section 6.1.2.4.

Appendix C -- Sequence Space Window Code Example

This appendix contains a routine that implements a bitmask check for a 32 packet window. It was provided by James Hughes (jim_hughes@stortek.com) and Harry Varnis (hgv@anubis.network.com) and is intended as an implementation example. Note that this code both checks for a replay and updates the window. Thus the algorithm, as shown, should only be called AFTER the packet has been authenticated. Implementers might wish to consider splitting the code to do the check for replays before computing the ICV. If the packet is not a replay, the code would then compute the ICV, (discard any bad packets), and if the packet is OK, update the window.

```
#include <stdio.h>
#include <stdlib.h>
typedef unsigned long u_long;

enum {
    ReplayWindowSize = 32
};

u_long bitmap = 0;                 /* session state - must be 32 bits */
u_long lastSeq = 0;                     /* session state */

/* Returns 0 if packet disallowed, 1 if packet permitted */
int ChkReplayWindow(u_long seq);

int ChkReplayWindow(u_long seq) {
    u_long diff;

    if (seq == 0) return 0;             /* first == 0 or wrapped */
    if (seq > lastSeq) {                /* new larger sequence number */
        diff = seq - lastSeq;
        if (diff < ReplayWindowSize) {  /* In window */
            bitmap <<= diff;
            bitmap |= 1;                /* set bit for this packet */
        } else bitmap = 1;          /* This packet has a "way larger" */
        lastSeq = seq;
        return 1;                       /* larger is good */
    }
    diff = lastSeq - seq;
    if (diff >= ReplayWindowSize) return 0; /* too old or wrapped */
    if (bitmap & ((u_long)1 << diff)) return 0; /* already seen */
    bitmap |= ((u_long)1 << diff);              /* mark as seen */
    return 1;                           /* out of order but good */
}

char string_buffer[512];
```

```
#define STRING_BUFFER_SIZE sizeof(string_buffer)

int main() {
    int result;
    u_long last, current, bits;

    printf("Input initial state (bits in hex, last msgnum):\n");
    if (!fgets(string_buffer, STRING_BUFFER_SIZE, stdin)) exit(0);
    sscanf(string_buffer, "%lx %lu", &bits, &last);
    if (last != 0)
    bits |= 1;
    bitmap = bits;
    lastSeq = last;
    printf("bits:%08lx last:%lu\n", bitmap, lastSeq);
    printf("Input value to test (current):\n");

    while (1) {
        if (!fgets(string_buffer, STRING_BUFFER_SIZE, stdin)) break;
        sscanf(string_buffer, "%lu", &current);
        result = ChkReplayWindow(current);
        printf("%-3s", result ? "OK" : "BAD");
        printf(" bits:%08lx last:%lu\n", bitmap, lastSeq);
    }
    return 0;
}
```

Appendix D -- Categorization of ICMP messages

The tables below characterize ICMP messages as being either host generated, router generated, both, unassigned/unknown. The first set are IPv4. The second set are IPv6.

IPv4

Type	Name/Codes	Reference
HOST GENERATED:		
3	Destination Unreachable	
	2 Protocol Unreachable	[RFC792]
	3 Port Unreachable	[RFC792]
	8 Source Host Isolated	[RFC792]
	14 Host Precedence Violation	[RFC1812]
10	Router Selection	[RFC1256]

Type	Name/Codes	Reference
ROUTER GENERATED:		
3	Destination Unreachable	
	0 Net Unreachable	[RFC792]
	4 Fragmentation Needed, Don’t Fragment was Set	[RFC792]
	5 Source Route Failed	[RFC792]
	6 Destination Network Unknown	[RFC792]
	7 Destination Host Unknown	[RFC792]
	9 Comm. w/Dest. Net. is Administratively Prohibited	[RFC792]
	11 Destination Network Unreachable for Type of Service	[RFC792]
5	Redirect	
	0 Redirect Datagram for the Network (or subnet)	[RFC792]
	2 Redirect Datagram for the Type of Service & Network	[RFC792]
9	Router Advertisement	[RFC1256]
18	Address Mask Reply	[RFC950]

IPv4

Type	Name/Codes	Reference
BOTH ROUTER AND HOST GENERATED:		
0	Echo Reply	[RFC792]
3	Destination Unreachable	
	1 Host Unreachable	[RFC792]
	10 Comm. w/Dest. Host is Administratively Prohibited	[RFC792]
	12 Destination Host Unreachable for Type of Service	[RFC792]
	13 Communication Administratively Prohibited	[RFC1812]
	15 Precedence cutoff in effect	[RFC1812]
4	Source Quench	[RFC792]
5	Redirect	
	1 Redirect Datagram for the Host	[RFC792]
	3 Redirect Datagram for the Type of Service and Host	[RFC792]
6	Alternate Host Address	[JBP]
8	Echo	[RFC792]
11	Time Exceeded	[RFC792]
12	Parameter Problem	[RFC792,RFC1108]
13	Timestamp	[RFC792]
14	Timestamp Reply	[RFC792]
15	Information Request	[RFC792]
16	Information Reply	[RFC792]
17	Address Mask Request	[RFC950]
30	Traceroute	[RFC1393]
31	Datagram Conversion Error	[RFC1475]
32	Mobile Host Redirect	[Johnson]
39	SKIP	[Markson]
40	Photuris	[Simpson]

Type	Name/Codes	Reference
UNASSIGNED TYPE OR UNKNOWN GENERATOR:		
1	Unassigned	[JBP]
2	Unassigned	[JBP]
7	Unassigned	[JBP]
19	Reserved (for Security)	[Solo]
20-29	Reserved (for Robustness Experiment)	[ZSu]
33	IPv6 Where-Are-You	[Simpson]
34	IPv6 I-Am-Here	[Simpson]
35	Mobile Registration Request	[Simpson]
36	Mobile Registration Reply	[Simpson]
37	Domain Name Request	[Simpson]
38	Domain Name Reply	[Simpson]
41-255	Reserved	[JBP]

IPv6

Type	Name/Codes	Reference
HOST GENERATED:		
1	Destination Unreachable	[RFC 1885]
	4 Port Unreachable	

Type	Name/Codes	Reference
ROUTER GENERATED:		
1	Destination Unreachable	[RFC1885]
	0 No Route to Destination	
	1 Comm. w/Destination is Administratively Prohibited	
	2 Not a Neighbor	
	3 Address Unreachable	
2	Packet Too Big	[RFC1885]
	0	
3	Time Exceeded	[RFC1885]
	0 Hop Limit Exceeded in Transit	
	1 Fragment reassembly time exceeded	

Type	Name/Codes	Reference
BOTH ROUTER AND HOST GENERATED:		
4	Parameter Problem	[RFC1885]
	0 Erroneous Header Field Encountered	
	1 Unrecognized Next Header Type Encountered	
	2 Unrecognized IPv6 Option Encountered	

References

[BL73] Bell, D.E. & LaPadula, L.J., "Secure Computer Systems: Mathematical Foundations and Model", Technical Report M74-244, The MITRE Corporation, Bedford, MA, May 1973.

[Bra97] Bradner, S., "Key words for use in RFCs to Indicate Requirement Level", BCP 14, RFC 2119, March 1997.

[DoD85] US National Computer Security Center, "Department of Defense Trusted Computer System Evaluation Criteria", DoD 5200.28-STD, US Department of Defense, Ft. Meade, MD., December 1985.

[DoD87] US National Computer Security Center, "Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria", NCSC-TG-005, Version 1, US Department of Defense, Ft. Meade, MD., 31 July 1987.

[HA94] Haller, N., and R. Atkinson, "On Internet Authentication", RFC 1704, October 1994.

[HC98] Harkins, D., and D. Carrel, "The Internet Key Exchange (IKE)", RFC 2409, November 1998.

[HM97] Harney, H., and C. Muckenhirn, "Group Key Management Protocol (GKMP) Architecture", RFC 2094, July 1997.

[ISO] ISO/IEC JTC1/SC6, Network Layer Security Protocol, ISO-IEC DIS 11577, International Standards Organisation, Geneva, Switzerland, 29 November 1992.

[IB93] John Ioannidis and Matt Blaze, "Architecture and Implementation of Network-layer Security Under Unix", Proceedings of USENIX Security Symposium, Santa Clara, CA, October 1993.

[IBK93] John Ioannidis, Matt Blaze, & Phil Karn, "swIPe: Network-Layer Security for IP", presentation at the Spring 1993 IETF Meeting, Columbus, Ohio

[KA98a] Kent, S., and R. Atkinson, "IP Authentication Header", RFC 2402, November 1998.

[KA98b] Kent, S., and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.

[Ken91] Kent, S., "US DoD Security Options for the Internet Protocol", RFC 1108, November 1991.

[MSST97] Maughan, D., Schertler, M., Schneider, M., and J. Turner, "Internet Security Association and Key Management Protocol (ISAKMP)", RFC 2408, November 1998.

[Orm97] Orman, H., "The OAKLEY Key Determination Protocol", RFC 2412, November 1998.

[Pip98] Piper, D., "The Internet IP Security Domain of Interpretation for ISAKMP", RFC 2407, November 1998.

[Sch94] Bruce Schneier, Applied Cryptography, Section 8.6, John Wiley & Sons, New York, NY, 1994.

[SDNS] SDNS Secure Data Network System, Security Protocol 3, SP3, Document SDN.301, Revision 1.5, 15 May 1989, published in NIST Publication NIST-IR-90-4250, February 1990.

[SMPT98] Shacham, A., Monsour, R., Pereira, R., and M. Thomas, "IP Payload Compression Protocol (IPComp)", RFC 2393, August 1998.

[TDG97] Thayer, R., Doraswamy, N., and R. Glenn, "IP Security Document Roadmap", RFC 2411, November 1998.

[VK83] V.L. Voydock & S.T. Kent, "Security Mechanisms in High-level Networks", ACM Computing Surveys, Vol. 15, No. 2, June 1983.

Disclaimer

The views and specification expressed in this document are those of the authors and are not necessarily those of their employers. The authors and their employers specifically disclaim responsibility for any problems arising from correct or incorrect implementation or use of this design.

Author Information

Stephen Kent
BBN Corporation
70 Fawcett Street
Cambridge, MA 02140
USA

Phone: +1 (617) 873-3988
EMail: kent@bbn.com

Randall Atkinson
@Home Network
425 Broadway
Redwood City, CA 94063
USA

Phone: +1 (415) 569-5000
EMail: rja@corp.home.net

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.