Reliable Data Protocol

RFC-908

David Velten

Robert Hinden

Jack Sax

BBN Communications Corporation

July 1984

Status of This Memo

This RFC specifies a proposed protocol for the ARPA Internet community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited.

Table of Contents

FIGURES

CHAPTER 1

Introduction

The Reliable Data Protocol (RDP) is designed to provide a reliable data transport service for packet-based applications such as remote loading and debugging. The protocol is intended to be simple to implement but still be efficient in environments where there may be long transmission delays and loss or non-sequential delivery of message segments.

Although this protocol was designed with applications such as remote loading and debugging in mind, it may be suitable for other applications that require reliable message services, such as computer mail, file transfer, transaction processing, etc.

Some of the concepts used come from a variety of sources. The authors wish credit to be given to Eric Rosen, Rob Gurwitz, Jack Haverty, and to acknowledge material adapted from "RFC-793, The Transmission Control Protocol", edited by Jon Postel. Thanks to John Linn for the checksum algorithm.

CHAPTER 2

General Description

2.1 Motivation

RDP is a transport protocol designed to efficiently support the bulk transfer of data for such host monitoring and control applications as loading/dumping and remote debugging. It attempts to provide only those services necessary, in order to be efficient in operation and small in size. Before designing the protocol, it was necessary to consider what minimum set of transport functions would satisfy the requirements of the intended applications.

The following is a list of requirements for such a transport protocol:

- Reliable delivery of packets is required. When loading or dumping a memory image, it is necessary that all memory segments be delivered. A 'hole' left in the memory image is not acceptable. However, the internet environment is a lossy one in which packets can get damaged or lost. So a positive acknowledgement and retransmission mechanism is a necessary component of the protocol.

- Since loading and dumping of memory images over the internet involves the bulk transfer of large amounts of data over a lossy network with potentially long delays, it is necessary that the protocol move data efficiently. In particular, unnecessary retransmission of segments should be avoided. If a single segment has been lost but succeeding segments correctly received, the protocol should not require the retransmission of all of the segments.

- Loading and dumping are applications that do not necessarily require sequenced delivery of segments, as long as all segments eventually are delivered. So the protocol need not force sequenced delivery. For these types of applications, segments may be delivered in the order in which they arrive.

- However, some applications may need to know that a particular piece of data has been delivered before sending the next. For example, a debugger will want to know that a command inserting a breakpoint into a host memory image has been delivered before sending a "proceed" command. If those segments arrived out of sequence, the intended results would not be achieved. The protocol should allow a user to optionally specify that a connection must deliver segments in sequence-number order.

- The loading/dumping and debugging applications are well-defined and lend themselves to easy packetization of the transferred data. They do not require a complex byte-stream transfer mechanism.

In order to combine the requirements for bulk transfers of data and reliable delivery, it is necessary to design a connection-oriented protocol using a three-way handshake to synchronize sequence numbers. The protocol seems to be approaching TCP in complexity, so why was TCP not, in fact, chosen? The answer is that TCP has some disadvantages for these applications. In particular:

- TCP is oriented toward a more general environment, supporting the transfer of a stream of bytes between two communicating parties. TCP is best suited to an environment where there is no obvious demarcation of data in a communications exchange. Much of the difficulty in developing a TCP implementation stems from the complexity of supporting this general byte-stream transfer, and thus a significant amount of complexity can be avoided by using another protocol. This is not just an implementation consideration, but also one of efficiency.

- Since TCP does not allow a byte to be acknowledged until all prior bytes have been acknowledged, it often forces unnecessary retransmission of data. Therefore, it does not meet another of the requirements stated above.

- TCP provides sequenced delivery of data to the application. If the application does not require such sequenced delivery, a large amount of resources are wasted in providing it. For example, buffers may be tied up buffering data until a segment with an earlier sequence number arrives. The protocol should not force its segment-sequencing desires on the application.

RDP supports a much simpler set of functions than TCP. The flow control, buffering, and connection management schemes of RDP are considerably simpler and less complex. The goal is a protocol that can be easily and efficiently implemented and that will serve a range of applications.

RDP functions can also be subset to further reduce the size of a particular implementation. For example, a target processor requiring down-loading from another host might implement an RDP module supporting only the passive Open function and a single connection. The module might also choose not to implement out-of-sequence acknowledgements.

2.2 Relation to Other Protocols

RDP is a transport protocol that fits into the layered internet protocol environment. Figure 1 illustrates the place of RDP in the protocol hierarchy:

```
 +-------+   +-----+     +-----+      +-------+
 |TELNET|   | FTP |     |Debug|  ... |Loader|   Application Layer
 +-------+   +-----+     +-----+      +-------+
    |           |           |             |
    +-----+-----+           +------+------+
          |                        |
       +-------+               +--------+
       |  TCP |               |  RDP  |        Transport Layer
       +-------+               +--------+
          |                      |
 +---------------------------------+
 | Internet Protocol & ICMP        |           Internetwork Layer
 +---------------------------------+
                |
             +-------------------------+
             | Network Access Protocol |       Network Layer
             +-------------------------+
```

Relation to Other Protocols
Figure 1

RDP provides the application layer with a reliable message transport service. The interface between users and RDP transfers data in units of messages. When implemented in the internet environment, RDP is layered on the Internet Protocol (IP), which provides an unreliable datagram service to RDP. Data is passed across the RDP/IP interface in the form of segments. RDP uses the standard IP interface primitives to send and receive RDP segments as IP datagrams. At the internet level, IP exchanges datagrams with the network layer. An internet packet may contain an entire datagram or a fragment of a datagram.

```
                                                      #       %
                                                        ?  *     !
                                                              @  )
  +-------+          +------+        +-----+         $  =   ^   +
  |       |Messages  |      |Segments|     | Datagrams   *
  | User  |<-------->| RDP  |<------>| IP  |<------->    Internet
  |       |          |      |        |     |          ,          ?
  +-------+          +------+        +-----+               !    )
                                                        *   %     $
                                                      @    ^   !
```

Form of Data Exchange Between Layers
Figure 2

If internetwork services are not required, it should be possible to use the RDP without the IP layer. As long as the encapsulating protocol provides the RDP with such necessary information as addressing and protocol demultiplexing, it should be possible to run RDP layered on a variety of different protocols.

CHAPTER 3

Protocol Operation

3.1 Protocol Service Objectives

The RDP protocol has the following goals:

- o RDP will provide a full-duplex communications channel between the two ports of each transport connection.
- o RDP will attempt to reliably deliver all user messages and will report a failure to the user if it cannot deliver a message. RDP extends the datagram service of IP to include reliable delivery.
- o RDP will attempt to detect and discard all damaged and duplicate segments. It will use a checksum and sequence number in each segment header to achieve this goal.
- o RDP will optionally provide sequenced delivery of segments. Sequenced delivery of segments must be specified when the connection is established.
- o RDP will acknowledge segments received out of sequence, as they arrive. This will free up resources on the sending side.

3.2 RDP Connection Management

RDP is a connection-oriented protocol in which each connection acts as a full-duplex communication channel between two processes. Segments from a sender are directed to a port on the destination host. The two 8-bit source and destination port identifiers in the RDP header are used in conjunction with the network source and destination addresses to uniquely identify each connection.

3.2.1 Opening a Connection

Connections are opened by issuing the Open request, which can be either active or passive. A passive Open request puts RDP into the Listen state, during which it passively listens for a request to open a connection from a remote site. The active Open request attempts to establish a connection with a specified port at a remote site.

The active Open request requires that a specific remote port and host address be specified with the request. The passive Open may optionally specify a specific remote port and network address, or it may specify that an open be accepted from anyone. If a specific remote port and host address were specified, an arriving request to open a connection must exactly match the specified remote port and address.

3.2.2 Ports

Valid port numbers range from 1 to 255 (decimal). There are two types of ports: "well known" ports and "allocable" ports. Well-known ports have numbers in the range 1 to 63 (decimal) and allocable ports are given numbers in the range 64 to 255.

The user, when issuing an active Open request, must specify both the remote host and port and may optionally specify the local port. If the local port was not specified, RDP will select an unused port from the range of allocable ports. When issuing a passive Open request, the user must specify the local port number. Generally, in this case the local port will be a well-known port.

3.2.3 Connection States

An RDP connection will progress through a series of states during its lifetime. The states are shown in Figure 3 and are individually described below. In Figure 3, the boxes represent the states of the RDP FSM and the arcs represent changes in state. Each arc is annotated with the event causing the state change and the resulting output.

CLOSED

The CLOSED state exists when no connection exists and there is no connection record allocated.

LISTEN

The LISTEN state is entered after a passive Open request is processed. A connection record is allocated and RDP waits for an active request to establish a connection from a remote site.

SYN-SENT

The SYN-SENT state is entered after processing an active Open request. A connection record is allocated, an initial sequence number is generated, and a SYN segment is sent to the remote site. RDP then waits in the SYN-SENT state for acknowledgement of its Open request.

SYN-RCVD

The SYN-RCVD state may be reached from either the LISTEN state or from the SYN-SENT state. SYN-RCVD is reached from the LISTEN state when a SYN segment requesting a connection is received from a remote host. In reply, the local RDP generates an initial sequence number for its side of the connection, and then sends the sequence number and an acknowledgement of the SYN segment to the remote site. It then waits for an acknowledgement.

The SYN-RCVD state is reached from the SYN-SENT state when a SYN segment is received from the remote host without an accompanying acknowledgement of the SYN segment sent to that remote host by the local RDP. This situation is caused by simultaneous attempts to open a connection, with the SYN segments passing each other in transit. The action is to repeat the SYN segment with the same sequence number, but now including an ACK of the remote host's SYN segment to indicate acceptance of the Open request.


```
                        +------------+
         Passive Open   |            |<-------------------------+
       +----------------|   CLOSED   |                          |
       |   Request      |            |---------------+          |
       V                +------------+               |          |
+------------+                                       |          |
|            |                                       |          |
|   LISTEN   |                                       |          |
|            |                                       |          |
+------------+                                       |          |
       |                                   Active    |          |
       |  rcv SYN                       Open Request |          |
       | -----------                    ------------ |          |
       | snd SYN,ACK                      snd SYN    |          |
       V                   rcv SYN                   V          |
+------------+          -----------           +------------+    |
|            |          snd SYN,ACK           |            |    |
|  SYN-RCVD  |<-------------------------------|  SYN-SENT  |    |
|            |                                |            |    |
+------------+                                +------------+    |
       |  rcv ACK                       rcv SYN,ACK  |          |
       | ----------                    ------------- |          |
       |    xxx         +------------+    snd ACK    |          |
       |                |            |               |          |
       +--------------->|    OPEN    |<--------------+          |
                        |            |                          |
                        +------------+                          |
                    rcv RST   |   Close request                 |
                  ----------- |  ---------------                |
                      xxx     |     snd RST                     |
                              V                                 |
                        +------------+                          |
                        |            |                          |
                        | CLOSE-WAIT |--------------------------+
                        |            |  After a Delay
                        +------------+
```

RDP Connection State Diagram
Figure 3

OPEN

The OPEN state exists when a connection has been established by the successful exchange of state information between the two sides of the connection. Each side has exchanged and received such data as initial sequence number, maximum segment size, and maximum number of unacknowledged segments that may be outstanding. In the Open state data may be sent between the two parties of the connection.

CLOSE-WAIT

The CLOSE-WAIT state is entered from either a Close request or from the receipt of an RST segment from the remote site. RDP has sent an RST segment and is waiting a delay period for activity on the connection to complete.

3.2.4 Connection Record

The variables that define the state of a connection are stored in a connection record maintained for each connection. The following describes some of the variables that would be stored in a typical RDP connection record. It is not intended to be an implementation specification nor is it a complete description. The purpose of naming and describing some of the connection record fields is to simplify the description of RDP protocol operation, particularly event processing.

The connection record fields and their descriptions follow:

STATE

The current state of the connection. Legal values are OPEN, LISTEN, CLOSED, SYN-SENT, SYN-RCVD, and CLOSE-WAIT.

Send Sequence Number Variables:

SND.NXT

The sequence number of the next segment that is to be sent.

SND.UNA

The sequence number of the oldest unacknowledged segment.

SND.MAX

The maximum number of outstanding (unacknowledged) segments that can be sent. The sender should not send more than this number of segments without getting an acknowledgement.

SND.ISS

The initial send sequence number. This is the sequence number that was sent in the SYN segment.

Receive Sequence Number Variables:

RCV.CUR

The sequence number of the last segment received correctly and in sequence.

RCV.MAX

The maximum number of segments that can be buffered for this connection.

RCV.IRS

The initial receive sequence number. This is the sequence number of the SYN segment that established this connection.

RCVDSEQNO[n]

The array of sequence numbers of segments that have been received and acknowledged out of sequence.

Other Variables:

CLOSEWAIT

A timer used to time out the CLOSE-WAIT state.

SBUF.MAX

The largest possible segment (in octets) that can legally be sent. This variable is specified by the foreign host in the

SYN segment during connection establishment.

RBUF.MAX

The largest possible segment (in octets) that can be received. This variable is specified by the user when the connection is opened. The variable is sent to the foreign host in the SYN segment.

Variables from Current Segment:

SEG.SEQ

The sequence number of the segment currently being processed.

SEG.ACK

The acknowledgement sequence number in the segment currently being processed.

SEG.MAX

The maximum number of outstanding segments the receiver is willing to hold, as specified in the SYN segment that established the connection.

SEG.BMAX

The maximum segment size (in octets) accepted by the foreign host on a connection, as specified in the SYN segment that established the connection.

3.2.5 Closing a Connection

The closing of a connection can be initiated by a Close request from the user process or by receipt of an RST segment from the other end of the connection. In the case of the Close request, RDP will send an RST segment to the other side of the connection and then enter the CLOSE-WAIT state for a period of time. While in the CLOSE-WAIT state, RDP will discard segments received from the other side of the connection. When the time-out period expires, the connection record is deallocated and the connection ceases to exist. This simple connection closing facility requires that users determine that all data has been

reliably delivered before requesting a close of the connection.

3.2.6 Detecting an Half-Open Connection

If one side of a connection crashes, the connection may be left with the other side still active. This situation is termed to be an half-open connection. For many cases, the active RDP will eventually detect the half-open connection and reset. Two examples of recovery from half-open connections are provided in sections 5.7 and 5.8. Recovery is usually achieved by user activity or by the crashed host's attempts to re-establish the connection.

However, there are cases where recovery is not possible without action by the RDP itself. For example, if all connection blocks are in use, attempts to re-establish a broken connection will be rejected. In this case, the RDP may attempt to free resources by verifying that connections are fully open. It does this by sending a NUL segment to each of the other RDPs. An acknowledgement indicates the connection is still open and valid.

To minimize network overhead, verification of connections should only be done when necessary to prevent a deadlock situation. Only inactive connections should be verified. An inactive connection is defined to be a connection that has no outstanding unacknowledged segments, has no segments in the user input or output queues, and that has not had any traffic for some period of time.

3.3 Data Communication

Data flows through an RDP connection in the form of segments. Each user message submitted with a Send request is packaged for transport as a single RDP segment. Each RDP segment is packaged as an RDP header and one or more octets of data. RDP will not attempt to fragment a large user message into smaller segments and re-assemble the message on the receiving end. This differs from a byte-stream protocol such as TCP which supports the transfer of an indeterminate length stream of data between ports, buffering data until it is requested by the receiver.

At the RDP level, outgoing segments, as they are created, are queued as input to the IP layer. Each segment is held by the sending RDP until it is acknowledged by the foreign host. Incoming segments are queued as input to the user process through the user interface. Segments are acknowledged when they have been accepted by the receiving RDP.

The receiving end of each connection specifies the maximum segment size it will accept. Any attempt by the sender to transmit a larger segment is an error. If RDP determines that a buffer submitted with a Send request exceeds the maximum size segment permitted on the connection, RDP will return an error to the user. In addition, RDP will abort a connection with an RST segment if an incoming segment contains more data than the maximum acceptable segment size. No attempt will be made to recover from or otherwise overcome this error condition.

If sequenced delivery of segments is necessary for a connection, the requirement must be stated when the connection is established. Sequenced delivery is specified when the Open request is made. Sequenced delivery of segments will then be the mode of delivery for the life of the connection.

3.4 Reliable Communication

RDP implements a reliable message service through a number of mechanisms. These include the insertion of sequence numbers and checksums into segments, the positive acknowledgement of segment receipt, and timeout and retransmission of missing segments.

3.4.1 Segment Sequence Numbers

Each segment transporting data has a sequence number that uniquely identifies it among all other segments in the same connection. The initial sequence number is chosen when the connection is opened and is selected by reading a value from a monotonically increasing clock. Each time a segment containing data is transmitted, the sequence number is incremented. Segments containing no data do not increment the sequence number. However, the SYN and NUL segments, which cannot contain data, are exceptions. The SYN segment is always sent with a unique sequence number, the initial sequence number. The NUL segment is

sent with the next valid sequence number.

3.4.2 Checksums

Each RDP segment contains a checksum to allow the receiver to detect damaged segments. RDP uses a non-linear checksum algorithm to compute a checksum that is 32-bits wide and operates on data in units of four octets (32 bits). The area that is covered by the checksum includes both the RDP header and the RDP data area.

If a segment contains a number of header and data octets that is not an integral multiple of 4 octets, the last octet is padded on the right with zeros to form a 32-bit quantity for computation purposes. The padding zeros are not transmitted as part of the segment. While computing the checksum, the checksum field itself is replaced with zeros. The actual algorithm is described in Section 4.2.1.

3.4.3 Positive Acknowledgement of Segments

RDP assumes it has only an unreliable datagram service to deliver segments. To guarantee delivery of segments in this environment, RDP uses positive acknowledgement and retransmission of segments. Each segment containing data and the SYN and NUL segments are acknowledged when they are correctly received and accepted by the destination host. Segments containing only an acknowledgement are not acknowledged. Damaged segments are discarded and are not acknowledged. Segments are retransmitted when there is no timely acknowledgement of the segment by the destination host.

RDP allows two types of acknowledgement. A cumulative acknowledgement is used to acknowledge all segments up to a specified sequence number. This type of acknowledgement can be sent using fixed length fields within the RDP header. Specifically, the ACK control flag is set and the last acknowledged sequence number is placed in the Acknowledgement Number field.

The extended or non-cumulative acknowledgement allows the receiver to acknowledge segments out of sequence. This type of acknowledgement is sent using the EACK control flag and the variable length fields in the RDP segment header. The variable length header fields are used to hold the sequence numbers of the acknowledged out-of-sequence segments.

The type of acknowledgement used is simply a function of the order in which segments arrive. Whenever possible, segments are acknowledged using the cumulative acknowledgement segment. Only out-of-sequence segments are acknowledged using the extended acknowledgement option.

The user process, when initiating the connection, cannot restrict the type of acknowledgement used on the connection. The receiver may choose not to implement out-of-sequence acknowledgements. On the other hand, the sender may choose to ignore out-of-sequence acknowledgements.

3.4.4 Retransmission Timeout

Segments may be lost in transmission for two reasons: they may be lost or damaged due to the effects of the lossy transmission media; or they may be discarded by the receiving RDP. The positive acknowledgement policy requires the receiver to acknowledge a segment only when the segment has been correctly received and accepted.

To detect missing segments, the sending RDP must use a retransmission timer for each segment transmitted. The timer is set to a value approximating the transmission time of the segment in the network. When an acknowledgement is received for a segment, the timer is cancelled for that segment. If the timer expires before an acknowledgement is received for a segment, that segment is retransmitted and the timer is restarted.

3.5 Flow Control and Window Management

RDP employs a simple flow control mechanism that is based on the number of unacknowledged segments sent and the maximum allowed number of outstanding (unacknowledged) segments. Each RDP connection has an associated set of flow control parameters that include the maximum number of outstanding segments for each side of a connection. These parameters are specified when the connection is opened with the Open request, with each side of the connection specifying its own parameters. The parameters are

passed from one host to another in the initial connection segments.

The values specified for these parameters should be based on the amount and size of buffers that the RDP is willing to allocate to a connection. The particular RDP implementation can set the parameters to values that are optimal for its buffering scheme. Once these parameters are set they remain unchanged throughout the life of the connection.

RDP employs the concept of a sequence number window for acceptable segment sequence numbers. The left edge of the window is the number of the last in-sequence acknowledged sequence number plus one. The right edge of the window is equal to the left edge plus twice the allowed maximum number of outstanding segments. The allowed maximum number of outstanding segments is the number of segments the transmitting RDP software is allowed to send without receiving any acknowledgement.

The flow control and window management parameters are used as follows. The RDP module in the transmitting host sends segments until it reaches the connection's segment limit specified by the receiving process. Once this limit is reached, the transmitting RDP module may only send a new segment for each acknowledged segment.

When a received segment has a sequence number that falls within the acceptance window, it is acknowledged. If the sequence number is equal to the left-hand edge (i.e., it is the next sequence number expected), the segment is acknowledged with a cumulative acknowledgement (ACK). The acceptance window is adjusted by adding one to the value of the edges. If the sequence number is within the acceptance window but is out of sequence, it is acknowledged with a non-cumulative acknowledgement (EACK). The window is not adjusted, but the receipt of the out-of-sequence segment is recorded.

When segments are acknowledged out of order, the transmitting RDP module must not transmit beyond the acceptance window. This could occur if one segment is not acknowledged but all subsequent segments are received and acknowledged. This condition will fix the left edge of the window at the sequence number of the unacknowledged segment. As additional segments are transmitted, the next segment to be sent will approach and eventually overtake the right window edge. At this point all transmission of new segments will cease until the unacknowledged segment is acknowledged.

3.6 User Interface

The user interface to RDP is implementation dependent and may use system calls, function calls or some other mechanism. The list of requests that follows is not intended to suggest a particular implementation.

OPEN Request

Opens a connection. Parameters include type (active or passive), local port, remote port, remote host address, maximum segment size, maximum number of unacknowledged segments, delivery mode (sequenced or non-sequenced). The connection id, including any allocated port number, is returned to the user.

SEND Request

Sends a user message. Parameters include connection identifier, buffer address and data count.

RECEIVE Request

Receives a user message. Parameters include connection identifier, buffer address and data count.

CLOSE Request

Closes a specified connection. The single parameter is the connection identifier.

STATUS Request

Returns the status of a connection. The parameters include the connection identifier and the address of the status buffer.

3.7 Event Processing

This section describes one possible sequence for processing events. It is not intended to suggest a particular implementation, but any actual implementation should vary from this description only in detail and not significantly in substance. The following are the kinds of events that may occur:

USER REQUESTS

- Open
- Send
- Receive
- Close
- Status

ARRIVING SEGMENT

- Segment Arrives

TIMEOUTS

- Retransmission Timeout
- Close-Wait Timeout

User request processing always terminates with a return to the caller, with a possible error indication. Error responses are given as a character string. A delayed response is also possible in some situations and is returned to the user by whatever event or pseudo interrupt mechanism is available. The term "signal" is used to refer to delayed responses.

Processing of arriving segments usually follows this general sequence: the sequence number is checked for validity and, if valid, the segment is queued and processed in sequence-number order. For all events, unless a state change is specified, RDP remains in the same state.

3.7.1 User Request Events

The following scenarios demonstrate the processing of events caused by the issuance of user requests:

```
Open Request


  CLOSED STATE

    Create a connection record
    If none available
      Return "Error - insufficient resources"
    Endif

    If passive Open
      If local port not specified
        Return "Error - local port not specified"
      Endif
      Generate SND.ISS
      Set SND.NXT = SND.ISS + 1
          SND.UNA = SND.ISS
      Fill in SND.MAX, RMAX.BUF from Open parameters
      Set State = LISTEN
      Return
    Endif


    If active Open
      If remote port not specified
        Return "Error - remote port not specified"
      Endif
      Generate SND.ISS
      Set SND.NXT = SND.ISS + 1
          SND.UNA = SND.ISS
      Fill in SND.MAX, RMAX.BUF from Open parameters
      If local port not specified
        Allocate a local port
      Endif
      Send <SEQ=SND.ISS><MAX=SND.MAX><MAXBUF=RMAX.BUF><SYN>
      Set State = SYN-SENT
      Return (local port, connection identifier)
    Endif
```



```
  LISTEN STATE
  SYN-SENT STATE
  SYN-RCVD STATE
  OPEN STATE
  CLOSE-WAIT STATE

    Return "Error - connection already open"


Close Request

  OPEN STATE

    Send <SEQ=SND.NXT><RST>
    Set State = CLOSE-WAIT
    Start TIMWAIT Timer
    Return

  LISTEN STATE

    Set State = CLOSED
    Deallocate connection record
    Return

  SYN-RCVD STATE
  SYN-SENT STATE

    Send <SEQ=SND.NXT><RST>
    Set State = CLOSED
    Return


  CLOSE-WAIT STATE

    Return "Error - connection closing"

  CLOSE STATE

    Return "Error - connection not open"
```

```
Receive Request

  OPEN STATE

    If Data is pending
      Return with data
     else
      Return with "no data" indication
    Endif

  LISTEN STATE
  SYN-RCVD STATE
  SYN-SENT STATE

    Return with "no data" indication

  CLOSE STATE
  CLOSE-WAIT STATE

    Return "Error - connection not open"


Send Request

  OPEN STATE

    If SND.NXT < SND.UNA + SND.MAX
      Send <SEQ=SND.NXT><ACK=RCV.CUR><ACK><Data>
      Set SND.NXT = SND.NXT + 1
      Return
     else
      Return "Error - insufficient resources to send data"
    Endif


  LISTEN STATE
  SYN-RCVD STATE
  SYN-SENT STATE
  CLOSE STATE
  CLOSE-WAIT STATE

    Return "Error - connection not open"


Status Request

  Return with:
```



```
    State of connection (OPEN, LISTEN, etc.)
    Number of segments unacknowledged
    Number of segments received not given to user
    Maximum segment size for the send side of the connection
    Maximum segment size for the receive side of the connection
```

3.7.2 Segment Arrival Events

The following scenarios describe the processing of the event caused by the arrival of a RDP segment from a remote host. The assumption is made that the segment was addressed to the local port associated with the connection record.

```
If State = CLOSED

  If RST set
    Discard segment
    Return
  Endif

  If ACK or NUL set
     Send <SEQ=SEG.ACK + 1><RST>
     Discard segment
     Return
   else
     Send <SEQ=0><RST><ACK=RCV.CUR><ACK>
     Discard segment
     Return
  Endif

Endif


If State = CLOSE-WAIT
  If RST set
     Set State = CLOSED
     Discard segment
     Cancel TIMWAIT timer
     Deallocate connection record
   else
     Discard segment
  Endif
  Return
Endif
```

```
If State = LISTEN

  If RST set
    Discard the segment
    Return
  Endif

  If ACK or NUL set
    Send <SEQ=SEG.ACK + 1><RST>
    Return
  Endif

  If SYN set
    Set RCV.CUR = SEG.SEQ
        RCV.IRS = SEG.SEQ
        SND.MAX = SEG.MAX
        SBUF.MAX = SEG.BMAX
    Send <SEQ=SND.ISS><ACK=RCV.CUR><MAX=RCV.MAX><BUFMAX=RBUF.MAX>
         <ACK><SYN>
    Set State = SYN-RCVD
    Return
  Endif

  If anything else (should never get here)
    Discard segment
    Return
  Endif
Endif

If State = SYN-SENT

  If ACK set
    If RST clear and SEG.ACK != SND.ISS
      Send <SEQ=SEG.ACK + 1><RST>
    Endif
    Discard segment; Return
  Endif

  If RST set
    If ACK set
      Signal "Connection Refused"
      Set State =  CLOSED
      Deallocate connection record
    Endif
    Discard segment
    Return
  Endif
```



```
  If SYN set
    Set RCV.CUR = SEG.SEQ
        RCV.IRS = SEG.SEQ
        SND.MAX = SEG.MAX
        RBUF.MAX = SEG.BMAX
    If ACK set
      Set SND.UNA = SEG.ACK
      State = OPEN
      Send <SEQ=SND.NXT><ACK=RCV.CUR><ACK>
     else
      Set State = SYN-RCVD
      Send <SEQ=SND.ISS><ACK=RCV.CUR><MAX=RCV.MAX><BUFMAX=RBUF.MAX>
             <SYN><ACK>
    Endif
    Return
  Endif

  If anything else
    Discard segment
    Return
  Endif
Endif

If State = SYN-RCVD

  If RCV.IRS < SEG.SEQ =< RCV.CUR + (RCV.MAX * 2)
    Segment sequence number acceptable
   else
    Send <SEQ=SND.NXT><ACK=RCV.CUR><ACK>
    Discard segment
    Return
  Endif


  If RST set
    If passive Open
       Set State = LISTEN
    else
       Set State = CLOSED
       Signal "Connection Refused"
       Discard segment
       Deallocate connection record
    Endif
    Return
  Endif
```

```
  If SYN set
    Send <SEQ=SEG.ACK + 1><RST>
    Set State = CLOSED
    Signal "Connection Reset"
    Discard segment
    Deallocate connection record
    Return
  Endif

  If EACK set
     Send <SEQ=SEG.ACK + 1><RST>
     Discard segment
     Return
  Endif

  If ACK set
    If SEG.ACK = SND.ISS
       Set State = OPEN
     else
       Send <SEQ=SEG.ACK + 1><RST>
       Discard segment
       Return
    Endif
   else
    Discard segment
    Return
  Endif

  If Data in segment or NUL set
    If the received segment is in sequence
       Copy the data (if any) to user buffers
       Set RCV.CUR=SEG.SEQ
       Send <SEQ=SND.NXT><ACK=RCV.CUR><ACK>
     else
       If out-of-sequence delivery permitted
          Copy the data (if any) to user buffers
       Endif
       Send <SEQ=SND.NXT><ACK=RCV.CUR><ACK><EACK><RCVDSEQNO1>
                 ...<RCVDSEQNOn>
    Endif
  Endif

Endif
```



```
If State = OPEN

  If RCV.CUR < SEG.SEQ =< RCV.CUR + (RCV.MAX * 2)
    Segment sequence number acceptable
   else
    Send <SEQ=SND.NXT><ACK=RCV.CUR><ACK>
    Discard segment and return
  Endif

  If RST set
    Set State = CLOSE-WAIT
    Signal "Connection Reset"
    Return
  Endif

  If NUL set
    Set RCV.CUR=SEG.SEQ
    Send <SEQ=SND.NXT><ACK=RCV.CUR><ACK>
    Discard segment
    Return
  Endif

  If SYN set
    Send <SEQ=SEG.ACK + 1><RST>
    Set State = CLOSED
    Signal "Connection Reset"
    Discard segment
    Deallocate connection record
    Return
  Endif

  If ACK set
    If SND.UNA =< SEG.ACK < SND.NXT
      Set SND.UNA = SEG.ACK
      Flush acknowledged segments
    Endif
  Endif

  If EACK set
    Flush acknowledged segments
  Endif
```

```
  If Data in segment
   If the received segment is in sequence
     Copy the data to user buffers
     Set RCV.CUR=SEG.SEQ
     Send <SEQ=SND.NXT><ACK=RCV.CUR><ACK>
    else
     If out-of-sequence delivery permitted
        Copy the data to user buffers
     Endif
     Send <SEQ=SND.NXT><ACK=RCV.CUR><ACK><EACK><RCVDSEQNO1>
                 ...<RCVDSEQNOn>
   Endif
  Endif
Endif
```

3.7.3 Timeout Events

Timeout events occur when a timer expires and signals the RDP. Two types of timeout events can occur, as described below:

RETRANSMISSION TIMEOUTS

```
  If timeout on segment at head of retransmission queue
     Resend the segment at head of queue
     Restart the retransmission timer for the segment
     Requeue the segment on retransmission queue
     Return
  Endif
```

CLOSE-WAIT TIMEOUTS

```
  Set State = CLOSED
  Deallocate connection record
  Return
```

CHAPTER 4

RDP Segments and Formats

The segments sent by the application layer are encapsulated in headers by the transport, internet and network layers, as follows:

```
+----------------+
| Network Access |
|     Header     |
+----------------+
|   IP Header    |
+----------------+
|   RDP Header   |
+----------------+
|     D          |
|      A         |
|       T        |
|        A       |
+----------------+
```

Segment Format
Figure 4

4.1 IP Header Format

When used in the internet environment, RDP segments are sent using the version 4 IP header as described in RFC791, "Internet Protocol." The RDP protocol number is ??? (decimal). The time-to-live field should be set to a reasonable value for the network.

All other fields should be set as specified in RFC-791.

4.2 RDP Header Format

Every RDP segment is prefaced with an RDP header. The format of the header is shown in Figure 5 below. The RDP header is variable in length and its size is indicated by a field in a fixed location within the header.

```
     0             0 0   1         1
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
    +-+-+-+-+-+-+---+---------------+
    |S|A|E|R|N| |Ver|    Header     |
  0 |Y|C|A|S|U|0|No.|    Length     |
    |N|K|K|T|L| |   |               |
    +-+-+-+-+-+-+---+---------------+
  1 | Source Port   |   Dest. Port  |
    +---------------+---------------+
  2 |          Data  Length         |
    +---------------+---------------+
  3 |                               |
    +---    Sequence Number      ---+
  4 |                               |
    +---------------+---------------+
  5 |                               |
    +--- Acknowledgement Number  ---+
  6 |                               |
    +---------------+---------------+
  7 |                               |
    +---        Checksum         ---+
  8 |                               |
    +---------------+---------------+
  9 |     Variable Header Area      |
    .                               .
    .                               .
    |                               |
    +---------------+---------------+
```

RDP Header Format
Figure 5

4.2.1 RDP Header Fields

Control Flags

This 8-bit field occupies the first octet of word one in the header. It is bit encoded with the following bits currently defined:

Bit #	Bit Name	Description
0	SYN	Establish connection and synchronize sequence numbers.
1	ACK	Acknowledge field significant.
2	EACK	Non-cumulative (Extended) acknowledgement.
3	RST	Reset the connection.
4	NUL	This is a null (zero data length) segment.
5		Unused.

Note that the SYN and RST are sent as separate segments and may not contain any data. The ACK may accompany any message. The NUL segment must have a zero data length, but may be accompanied by ACK and EACK information. The other control bit is currently unused and is defined to be zero.

Version Number

This field occupies bits 6-7 of the first octet. It contains the version number of the protocol described by this document. Current value is one (1).

Header Length

The length of the RDP header in units of two (2) octets, including this field. This field allows RDP to find the start of the Data field, given a pointer to the head of the segment. This field is 8 bits in length. For a segment with no variable header section, the header length field will have the value 9.

Source and Destination Ports

The Source and Destination Ports are used to identify the processes in the two hosts that are communicating with each other. The combination of the port identifiers with the source and destination addresses in the network access

protocol header serves to fully qualify the connection and constitutes the connection identifier. This permits RDP to distinguish multiple connections between two hosts. Each field is 8 bits in length, allowing port numbers from 0 to 255 (decimal).

Data Length

The length in octets of the data in this segment. The data length does not include the RDP header. This field is 16 bits in length.

Sequence Number

The sequence number of this segment. This field is 32 bits in length.

Acknowledgement Number

If the ACK bit is set in the header, this is the sequence number of the segment that the sender of this segment last received correctly and in sequence. Once a connection is established this should always be sent. This field is 32 bits in length.

Checksum

This field is a 32-bit checksum of the segment header and data. The algorithm description below includes two variables, the checksum accumulator and the checksum pointer. The checksum accumulator is an actual 32-bit register in which the checksum is formed. The checksum pointer is included for purposes of description, to represent the operation of advancing through the data four octets (32-bits) at a time. It need not be maintained in a register by an implementation.

1) The checksum pointer is set to zero, to correspond to the beginning of the area to be checksummed. The checksum accumulator is also initialized to zero before beginning the computation of the checksum.

2) The 32-bit memory word located at the address referenced by the checksum pointer is added arithmetically to the checksum accumulator. Any carry propagated out of the checksum accumulator is ignored. The checksum field itself is replaced with zeros when being added to the checksum

accumulator.

3) The checksum accumulator is rotated left one bit position. The checksum pointer is advanced to correspond to the address of the next 32-bit word in the segment.

4) Steps 2 and 3 are repeated until the entire segment has been summed. If a segment contains a number of header and data octets that is not an integral multiple of 4 octets, the last octet is padded on the right with zeros to form a 32-bit quantity for computation purposes.

Variable Header Area

This area is used to transmit parameters for the SYN and EACK segments.

4.3 SYN Segment

The SYN is used to establish a connection and synchronize sequence numbers between two hosts. The SYN segment also contains information to inform the remote host of the maximum number of segments the local RDP is willing to accept and the maximum segment size it can accept. The SYN may be combined with an ACK in a segment but is never combined with user data.

4.3.1 SYN Segment Format

```
      0             0 0   1         1
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
     +-+-+-+-+-+-+---+---------------+
   0 |1|0|0|0|0|0|0 1| Header Length |
     +-+-+-+-+-+-+---+---------------+
   1 | Source Port   |   Dest. Port  |
     +---------------+---------------+
   2 |       Data  Length = 0        |
     +---------------+---------------+
   3 |                               |
     +---    Sequence Number      ---+
   4 |                               |
     +---------------+---------------+
   5 |                               |
     +--- Acknowledgement Number  ---+
   6 |                               |
     +---------------+---------------+
   7 |                               |
     +---        Checksum         ---+
   8 |                               |
     +---------------+---------------+
   9 | Max. # of Outstanding Segments|
     +---------------+---------------+
  10 |       Max. Segment Size       |
     +-------------------------------+
  11 |      Options Flag Field       |
     +---------------+---------------+
```

SYN Segment Format
Figure 6

4.3.2 SYN Segment Fields

Sequence Number

Contains the initial sequence number selected for this connection.

Acknowledgement Number

This field is valid only if the ACK flag is set. In that case, this field will contain the sequence number of the SYN segment received from the other RDP.

Maximum Number of Outstanding Segments

The maximum number of segments that should be sent without getting an acknowledgement. This is used by the receiver as a means of flow control. The number is selected during connection initiation and may not be changed later in the life of the connection.

Maximum Segment Size

The maximum size segment in octets that the sender should send. It informs the sender how big the receiver's buffers are. The specified size includes the length of the IP header, RDP header, and data. It does not include the network access layer's header length.

Options Flag Field

This field of two octets contains a set of options flags that specify the set of optional functions that are desired for this connection. The flags are defined as follows:

Bit #	Bit Name	Description
0	SDM	Sequenced delivery mode.

The sequenced delivery mode flag specifies whether delivery of segments to the user is sequenced (delivered in sequence-number order) or non-sequenced (delivered in arrival order, regardless of sequence number). A value of 0 specifies non-sequenced delivery of segments, and a value of 1 specifies sequenced delivery.

4.4 ACK Segment

The ACK segment is used to acknowledge in-sequence segments. It contains both the next send sequence number and the acknowledgement sequence number in the RDP header. The ACK segment may be sent as a separate segment, but it should be combined with data whenever possible. Data segments must always include the ACK bit and Acknowledgement Number field.

4.4.1 ACK Segment Format

```
     0             0 0   1         1
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
    +-+-+-+-+-+-+---+---------------+
  0 |0|1|0|0|0|0|0 1| Header Length |
    +-+-+-+-+-+-+---+---------------+
  1 | Source Port   |   Dest. Port  |
    +---------------+---------------+
  2 |          Data  Length         |
    +---------------+---------------+
  3 |                               |
    +---    Sequence Number      ---+
  4 |                               |
    +---------------+---------------+
  5 |                               |
    +--- Acknowledgement Number  ---+
  6 |                               |
    +---------------+---------------+
  7 |                               |
    +---        Checksum         ---+
  8 |                               |
    +---------------+---------------+
    |                               |
    |             Data              |
    .                               .
    .                               .
    +---------------+---------------+
```

ACK Segment Format
Figure 7

4.4.2 ACK Segment Fields

Data Length

A non-zero Data Length field indicates that there is data present in the segment.

Sequence Number

The value of the Sequence Number field is advanced to the next sequence number only if there is data present in the segment. An ACK segment without data does not use sequence number space.

Acknowledgement Number

The Acknowledgement Number field contains the sequence number of the last segment received in sequential order.

4.5 Extended ACK Segment

The EACK segment is used to acknowledge segments received out of sequence. It contains the sequence numbers of one or more segments received with a correct checksum, but out of sequence. The EACK is always combined with an ACK in the segment, giving the sequence number of the last segment received in sequence. The EACK segment may also include user data.

4.5.1 EACK Segment Format

The EACK segment has the format shown in Figure 8.

4.5.2 EACK Segment Fields

Data Length

A non-zero Data Length field indicates that there is data present in the segment.

Sequence Number

The value of the Sequence Number field is advanced to the next sequence number only if there is data present in the segment. An EACK segment without data does not use sequence number space.

Acknowledgement Number

The Acknowledgement Number field contains the sequence number of the last segment received in sequential order.

Sequence # Received OK

Each entry is the sequence number of a segment that was received with a correct checksum, but out of sequence.

```
      0             0 0   1         1
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
     +-+-+-+-+-+-+---+---------------+
   0 |0|1|1|0|0|0|0 1| Header Length |
     +-+-+-+-+-+-+---+---------------+
   1 | Source Port   |   Dest. Port  |
     +---------------+---------------+
   2 |          Data  Length         |
     +---------------+---------------+
   3 |                               |
     +---    Sequence Number      ---|
   4 |                               |
     +---------------+---------------+
   5 |                               |
     +--- Acknowledgement Number  ---+
   6 |                               |
     +---------------+---------------+
   7 |                               |
     +---        Checksum         ---+
   8 |                               |
     +---------------+---------------+
   9 |                               |
     +--- Sequence # Received OK  ---+
  10 |                               |
     +---------------+---------------+
  11 |                               |
     +--- Sequence # Received OK  ---+
  12 |                               |
     +---------------+---------------+
     :               .               :
     :               .               :
     :               .               :
     +---------------+---------------+
     |                               |
     |             Data              |
     |                               |
     +---------------+---------------+
```

EACK Segment Format
Figure 8

4.6 RST Segment

The RST segment is used to close or reset a connection. Upon receipt of an RST segment, the sender must stop sending and must abort any unserviced requests. The RST is sent as a separate segment and does not include any data.

4.6.1 RST Segment Format

```
      0             0 0   1         1
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
     +-+-+-+-+-+-+---+---------------+
   0 |0|0|0|1|0|0|0 1| Header Length |
     +-+-+-+-+-+-+---+---------------+
   1 | Source Port   |   Dest. Port  |
     +---------------+---------------+
   2 |       Data  Length = 0        |
     +---------------+---------------+
   3 |                               |
     +---    Sequence Number      ---+
   4 |                               |
     +---------------+---------------+
   5 |                               |
     +--- Acknowledgement Number  ---+
   6 |                               |
     +---------------+---------------+
   7 |                               |
     +---        Checksum         ---+
   8 |                               |
     +-------------------------------+
```

RST Segment Format
Figure 9

4.7 NUL Segment

The NUL segment is used to determine if the other side of a connection is still active. When a NUL segment is received, an RDP implementation must acknowledge the segment if a valid connection exists and the segment sequence number falls within the acceptance window. The segment is then discarded. The NUL may be combined with an ACK in a segment but is never combined with user data.

4.7.1 NUL segment format

```
       0             0 0   1         1
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
      +-+-+-+-+-+-+---+---------------+
    0 |0|0|0|0|1|0|0 1| Header Length |
      +-+-+-+-+-+-+---+---------------+
    1 | Source Port   |   Dest. Port  |
      +---------------+---------------+
    2 |       Data  Length = 0        |
      +---------------+---------------+
    3 |                               |
      +---    Sequence Number      ---+
    4 |                               |
      +---------------+---------------+
    5 |                               |
      +--- Acknowledgement Number  ---+
    6 |                               |
      +---------------+---------------+
    7 |                               |
      +---        Checksum         ---+
    8 |                               |
      +-------------------------------+
```

NUL Segment Format
Figure 10

CHAPTER 5

Examples of Operation

5.1 Connection Establishment

This is an example of a connection being established between Host A and Host B. Host B has done a passive Open and is in LISTEN state. Host A does an active Open to establish the connection.

```
             Host A                         Host B

Time State                                              State

1.    CLOSED                                             LISTEN

2.    SYN-SENT    <SEQ=100><SYN> --->

3.                                <--- <SEQ=200><ACK=100><SYN,ACK>
                                                        SYN-RCVD

4.    OPEN    <SEQ=101><ACK=200> --->                    OPEN

5.      <SEQ=101><ACK=200><Data> --->

6.                                <--- <SEQ=201><ACK=101>
```


5.2 Simultaneous Connection Establishment

This is an example of two hosts trying to establishing connections to each other at the same time. Host A sends a SYN request to Host B at the same time Host B sends a SYN request to Host A.

```
     Host A                         Host B

Time State                                              State

1.   CLOSED                                             CLOSED

2.   SYN-SENT <SEQ=100><SYN>  --->
                              <--- <SEQ=200><SYN>      SYN-SENT

3.   SYN-RCVD                                           SYN-RCVD
   <SEQ=100><ACK=200><SYN,ACK> --->
                              <--- <SEQ=200><ACK=100><SYN,ACK>

4.   OPEN                                               OPEN
```

5.3 Lost Segments

This is an example of what happens when a segment is lost. It shows how segments can be acknowledged out of sequence and that only the missing segment need be retransmitted. Note that in this and the following examples "EA" stands for "Out of Sequence Acknowledgement."

```
Time   Host A                            Host B

1.     <SEQ=100><ACK=200><Data>  --->

2.                               <--- <SEQ=201><ACK=100>

3.     <SEQ=101><ACK=200><Data> (segment lost)

4.

5.     <SEQ=102><ACK=200><Data>  --->

6.                               <--  <SEQ=201><ACK=100><EA=102>

7.     <SEQ=103><ACK=200><Data>  --->

8.                               <--- <SEQ=201><ACK=100>
                                        <EA=102,103>

9.     <SEQ=101><ACK=200><Data>  --->

10.                              <--- <SEQ=201><ACK=103>

11.    <SEQ=104><ACK=200><Data>  --->

12.                              <--- <SEQ=201><ACK=104>
```


5.4 Segments Received Out of Order

This an example of segments received out of order. It further illustrates the use of acknowledging segments out of order to prevent needless retransmissions.

```
Time     Host A                            Host B

1.   <SEQ=100><ACK=200><Data>  --->

2.                             <--- <SEQ=201><ACK=100>

3.   <SEQ=101><ACK=200><Data> (delayed)

4.

5.   <SEQ=102><ACK=200><Data>  --->

6.                             <--- <SEQ=201><ACK=100><EA=102>

7.   <SEQ=103><ACK=200><Data>  --->
                              ---> (delayed segment 101 arrives)

8.                             <--- <SEQ=201><ACK=103>

9.   <SEQ=104><ACK=200><Data>  --->

10.                            <--- <SEQ=201><ACK=104>
```

5.5 Communication Over Long Delay Path

This is an example of a data transfer over a long delay path. In this example, Host A is permitted to have as many as five unacknowledged segments. The example shows that it is not necessary to wait for an acknowledgement in order to send additional data.

```
Time        Host A                      Host B

1.    <SEQ=100><ACK=200><Data> -1->
2.    <SEQ=101><ACK=200><Data> -2->
3.    <SEQ=102><ACK=200><Data> -3->
                               -1-> (received)
4.                            <-4-  <SEQ=201><ACK=100>
5.    <SEQ=103><ACK=200><Data> -5->
                               -2-> (received)
6.                            <-6-  <SEQ=201><ACK=101>
7.    <SEQ=104><ACK=200><Data> -7->
                               -3-> (received)
8.                            <-8-  <SEQ=201><ACK=102>
                  (received)  <-4-
9.    <SEQ=105><ACK=200><Data> -9->
                               -5-> (received)
10.                           <-10- <SEQ=201><ACK=103>
                  (received)  <-6-
11.   <SEQ=106><ACK=200><Data> -11->
                               -7-> (received)
12.                           <-12- <SEQ=201><ACK=104>
                  (received)  <-8-
13.                            -9-> (received)
14.                           <-13- <SEQ=201><ACK=105>
                  (received)  <-10-
15.                            -11-> (received)
16.                           <-14- <SEQ=201><ACK=106>
                  (received)  <-12-
17.               (received)  <-13-
18.               (received)  <-14-
```


5.6 Communication Over Long Delay Path With Lost Segments

This is an example of communication over a long delay path with a lost segment. It shows that by acknowledging segments out of sequence, only the lost segment need be retransmitted.

```
Time       Host A                     Host B

1. <SEQ=100><ACK=200><Data>  -1->
2. <SEQ=101><ACK=200><Data>  -2->
3. <SEQ=102><ACK=200><Data>  -3->
                             -1-> (received)
4.                          <-4-  <SEQ=201><ACK=100>
5. <SEQ=103><ACK=200><Data> (segment lost)
                             -2-> (received)
6.                          <-5-  <SEQ=201><ACK=101>
7. <SEQ=104><ACK=200><Data>  -6->
                             -3-> (received)
8.                          <-7-  <SEQ=201><ACK=102>
                 (received) <-4-
9. <SEQ=105><ACK=200><Data>  -8->
10.
                 (received) <-5-
11. <SEQ=106><ACK=200><Data> -10->
                             -6-> (received)
12.                         <-11- <SEQ=201><ACK=102><EA=104>
                 (received) <-7-
                             -8-> (received)
13.                         <-12- <SEQ=201><ACK=102><EA=104,105>
                             -10-> (received)
14.                         <-13- <SEQ=201><ACK=102><EA=104-106>
                 (received) <-11-
15. <SEQ=103><ACK=200><Data> -14->
                 (received) <-12-
16.              (received) <-13-
                             -14-> (received)
17.                         <-15- <SEQ=201><ACK=106>
18.
19.              (received) <-15-
```

5.7 Detecting a Half-Open Connection on Crash Recovery

This is an example of a host detecting a half-open connection due to the crash and subsequent restart of the host. In this example, Host A crashes during a communication session, then recovers and tries to reopen the connection. During the reopen attempt, it discovers that a half-open connection still exists and it then resets the other side. Both sides were in the OPEN state prior to the crash.

```
   Host A                                   Host B

Time

1.  OPEN                                     OPEN
   (crash!)               <--- <SEQ=200><ACK=100><ACK>

2.  CLOSED                                   OPEN
   (recover)

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

4.  SYN-SENT                                 OPEN
     (!)                  <--- <SEQ=200><ACK=100><ACK>

5.  SYN-SENT                                 OPEN
            <SEQ=101><RST> --->             (abort)

6.  SYN-SENT                                 CLOSED

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


5.8 Detecting a Half-Open Connection from the Active Side

This is another example of detecting a half-open connection due to the crash and restart of a host involved in a connection. In this example, host A again crashes and restarts. Host B is still active and tries to send data to host A. Since host A has no knowledge of the connection, it rejects the data with an RST segment, causing host B to reset the connection.

```
         Host A                               Host B

Time

1.   (crash!)                                          OPEN

2.   CLOSED                 <--- <SEQ=200><ACK=100><Data> OPEN

3.   CLOSED  <SEQ=101><RST> --->                       (abort)

4.   CLOSED                                            CLOSED
```

APPENDIX A

Implementing a Minimal RDP

It is not necessary to implement the entire RDP specification to be able to use RDP. For simple applications such as a loader, where size of the protocol module may be important, a subset of RDP may be used. For example, an implementation of RDP for loading may employ the following restrictions:

- Only one connection and connection record is supported. This is the connection used to load the device.
- A single, well-known port is used as the loader port. Allocable ports are not implemented.
- Only the passive Open request is implemented. Active Opens are not supported.
- The sequenced delivery option is not supported. Messages arriving out of order are delivered in the order they arrive.
- If efficiency is less important than protocol size, the extended acknowledgement feature need not be supported.

INDEX