Reliable Message Transport Protocol ERS

Protocol Version 0

Document Version 3.02

Allen Baker

Hewlett-Packard Company
INSO R&D Lab

August 24, 1998

Table of Contents

1. Revisions 4
1.1 Document Version 3.02 (6feb97) 4
1.2 Document Version 3.01 (5feb97) 4
1.3 Document Version 3.00 (24oct96) 4
1.4 Document Version 2.34 (6sep96) 4
1.5 Document Version 2.33 (4jun96) 4
1.6 Document Version 2.32 (3jun96) 4
1.7 Document Version 2.31 (30may96) 4
1.8 Document Version 2.30 (29may96) 5
1.9 Document Version 2.20 (28may96) 5
1.10 Document Version 2.10 (24may96) 5
1.11 Document Version 2.00 5
1.12 Document Version 1.00 5
2. Patents 6
3. Introduction 7
4. Interface 8
5. Protocol 9
5.1 Protocol Operation 9
5.1.1 RMTP Connection Management 9
5.1.1.1 Opening a Connection 9
5.1.1.2 Connection States 10
5.1.1.2.1 CLOSED 10
5.1.1.2.2 LISTEN 10
5.1.1.2.3 SYN-SENT 10
5.1.1.2.4 SYN-RCVD 11
5.1.1.2.5 OPEN 11
5.1.1.3 Closing a Connection 11
5.1.1.4 Detecting an Half-Open Connection 11
5.1.2 Version Negotiation 11
5.1.3 Data Communication 14
5.1.4 Reliable Communication 15
5.1.4.1 Segment Sequence Numbers 15
5.1.4.2 Checksums 15
5.1.4.3 Positive Acknowledgment of Segments 16
5.1.4.4 Retransmission Time-out 17
5.1.4.4.1 Algorithm for Computing the Retransmission Time-out 17
5.1.4.4.1.1 RTO Back-off During Retransmission 17
5.1.4.5 Connection Establishment Time-out 18
5.1.4.6 Connection Time-out 18
5.1.5 Flow Control and Window Management 19
5.1.5.1 Persist Timer 20
5.1.5.2 An Aside on Silly-Window Syndrome 20
5.2 RMTP Segments and Formats 21
5.2.1 RMTP Header Format 21
5.2.2 RMTP Header Fields 21
5.2.2.1 Control Bits 21
5.2.2.2 Version 22
5.2.2.3 Window Size 22
5.2.2.4 Checksum 22
5.2.2.5 Destination Connection 22
5.2.2.6 Acknowledged Sequence Number 22
5.2.2.7 Sequence Number 22
5.2.3 Connection Fields 23
5.2.3.1 Source Connection 23
5.2.3.2 Destination Connection 23
5.2.4 Segments 23
5.2.4.1 SYN Segment (0x01) 23
5.2.4.1.1 Segment Format 24

5.2.4.1.2 Fields 24
5.2.4.1.2.1 Source Connection..... 24
5.2.4.1.2.2 Max Segment Size..... 24
5.2.4.2 ASYN Segment (0x09)..... 25
5.2.4.2.1 Segment Format 25
5.2.4.2.2 Fields 25
5.2.4.2.2.1 Source Connection..... 25
5.2.4.2.2.2 Max Segment Size..... 26
5.2.4.3 ADTA Segment (0x0A)..... 26
5.2.4.3.1 Segment Format 26
5.2.4.3.2 Fields 27
5.2.4.3.2.1 Data Length..... 27
5.2.4.3.2.2 Message Data..... 27
5.2.4.4 RST Segment (0x04)..... 27
5.2.4.4.1 Segment Format 27
5.2.4.5 ACK Segment (0x08)..... 28
5.2.4.5.1 Segment Format 28
5.2.4.6 APRB Segment (0x18)..... 29
5.2.4.6.1 Segment Format 29

1. Revisions

1.1 Document Version 3.02 (6feb97)

- Updated Version Negotiation section.

1.2 Document Version 3.01 (5feb97)

- Added section describing version negotiation.

1.3 Document Version 3.00 (24oct96)

- Redesigned Segment formats to improve processing efficiency and to ensure that all fields are properly aligned.
- Changed sequence numbers from 16 to 32 bit.
- Changed connection timeout from 10 minutes to 3 minutes
- Changed persist timer from 120 seconds to 30 seconds.
- Renamed NUL bit to PRB (probe) bit.
- Renamed NUL segment to PRB segment.
- Added ACK to PRB segment and renamed it APRB segment.
- Removed requirement that initial sequence number is obtained from a clock.
- Rewrote section on computing usable window size.
- Made some minor wording improvements.

1.4 Document Version 2.34 (6sep96)

- Added information to Connection Establishment Time-out.
- Added brief justification for RMTP design to protocol introduction.
- Changed segment format pictures to better show bit and byte positions.

1.5 Document Version 2.33 (4jun96)

- Fixed bug in description of how to use CheckSum function.
- Improved description of connection fields.

1.6 Document Version 2.32 (3jun96)

- Changed protocol version number in version field from 1 to 0.
- Minor fixes to wording etc.

1.7 Document Version 2.31 (30may96)

- Fixed bug where ASYN segment was missing Segment Number and MaxSegment Size fields.
- Changed minimum value for Max Segment Size field in SYN and ASYN from 10 to 14.
- Added section on connection establishment time-out.
- Put ASYNs into connection state diagram.
- Added initial segment RTO value.
- Removed DTA segment; all data is sent in an ADTA.
- Added section on persist timer.

- Added back NUL bit and segment.
- Removed source connection field from segments in which it is not useful.
- Added more implementation hints.

1.8 Document Version 2.30 (29may96)

- Added zero padding of odd-last-byte to word size in checksum function.
- Removed NUL that was still in some segment headers.
- Added source and destination connection fields.
- Changed connection time out algorithm.

1.9 Document Version 2.20 (28may96)

- Reduced sequence numbers from 32 to 16 bits.

1.10 Document Version 2.10 (24may96)

- Simplified Retransmission time-out section.
- Simplified Half-open connection detection.
- Simplified and reduced Segment formats.
- Added algorithms for retransmission time-out, checksum, and window.
- Completed flow control section.
- Removed NUL control bit and NUL segment.
- Added DTA control bit.
- Rearranged control bits.
- Explicitly defined each legal control bit combination as a segment.
- Moved API into a separate document.

1.11 Document Version 2.00

- Removed CLOSE-WAIT state.
- Removed Extended Acknowledgment segment - EAK.
- Reduced size of Version field in header from 8 to 4 bits.
- Added 4-bit Window Size field to header.
- Removed Maximum Number of Outstanding Segments field from SYN segment.
- Elaborated Checksum and Retransmission sections.

1.12 Document Version 1.00

- Initial version; derived from Reliable Data Protocol (RDP) [RFC 1151].

2. Patents

The protocol described in this document is protected under the following international patents which are owned by the Hewlett-Packard Company:

1. DE69802792D1
2. DE69802792T2
3. DE69835314D1
4. DE69835314T2
5. EP0872991A2
6. EP0872991A3
7. EP0872991B1
8. EP0886411A2
9. EP0886411A3
10. EP0976058A1
11. EP0976058B1
12. US6202096 B1
13. US6502000
14. US6721286
15. WO1998047076A1

3. Introduction

An External Reference Specification (ERS) specifies its target object's look, feel, and functionality from an external perspective. The external perspective is a view of the target object from the outside and does not include requirements specifications, the target object's internal architectural and design details, or the target object's architectural relationships with other subsystems and modules (the target object's relationships to other subsystems and modules is described in the Architecture Specification). The external perspective includes all of the target object's hardware and software interfaces, protocols, performance characteristics, functionality, and operational procedures. Interfaces are inputs and outputs like those to and from users and other systems, subsystems and modules. An ERS describes its target object's interfaces and functionalities as a whole, without describing how those interfaces and functionalities are allocated to subsystems and modules within the target object.

The target object for this ERS is HP-JetSend's Reliable Message Transport Protocol layer (RMTP). This ERS specifies RMTP's interface and its protocol.

4. Interface

The interface to RMTP is its API. The API provides the HP-JetSend Transport Independent API (TIAPI) with RMTP services.

The RMTP API is specified in a separate document - *Reliable Message Transport Protocol API*, Allen Baker, HP, 1996.

5. Protocol

RMTP is a simple to implement, lightweight protocol that adds reliable-ordered-delivery to a datagram transport. HP will use it for low traffic links between Internet devices where heavyweight stream connections like TCP's are too expensive to maintain.

The HP-JetSend architecture does not require that any stream transport be available to HP-JetSend. Therefore, it is possible that all HP-JetSend communications within a particular device may have to be conducted over a datagram transport. RMTP will at least insure that this communication is reliable and ordered, if not high performance.

Transport connections are expensive to implement in some HP peripheral architectures. In these architectures, the connection must be projected through several layers of firmware from the transport layer to the application layer that uses the connection. Each layer potentially has to store information about the connection.

RMTP is designed to allow multiple connections to run over a single UDP socket. This is intended to limit the multiple layer firmware resources required - in this case the resources required to set up multiple sockets.

5.1 Protocol Operation

5.1.1 RMTP Connection Management

RMTP 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. RMTP may be implemented using any transport protocol. It is intended for implementation using the services provided by a datagram transport such as UDP or IPX. It adds value to datagram transports by implementing reliability and ordered delivery.

RMTP uses the common Internet 3-way handshake to establish a connection. RMTP uses this connection handshake procedure to negotiate maximum segment sizes, synchronize sequence numbers and exchange connection identifiers.

5.1.1.1 Opening a Connection

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

The active Open request requires that a specific remote port and host address be specified with the request.

5.1.1.2 Connection States

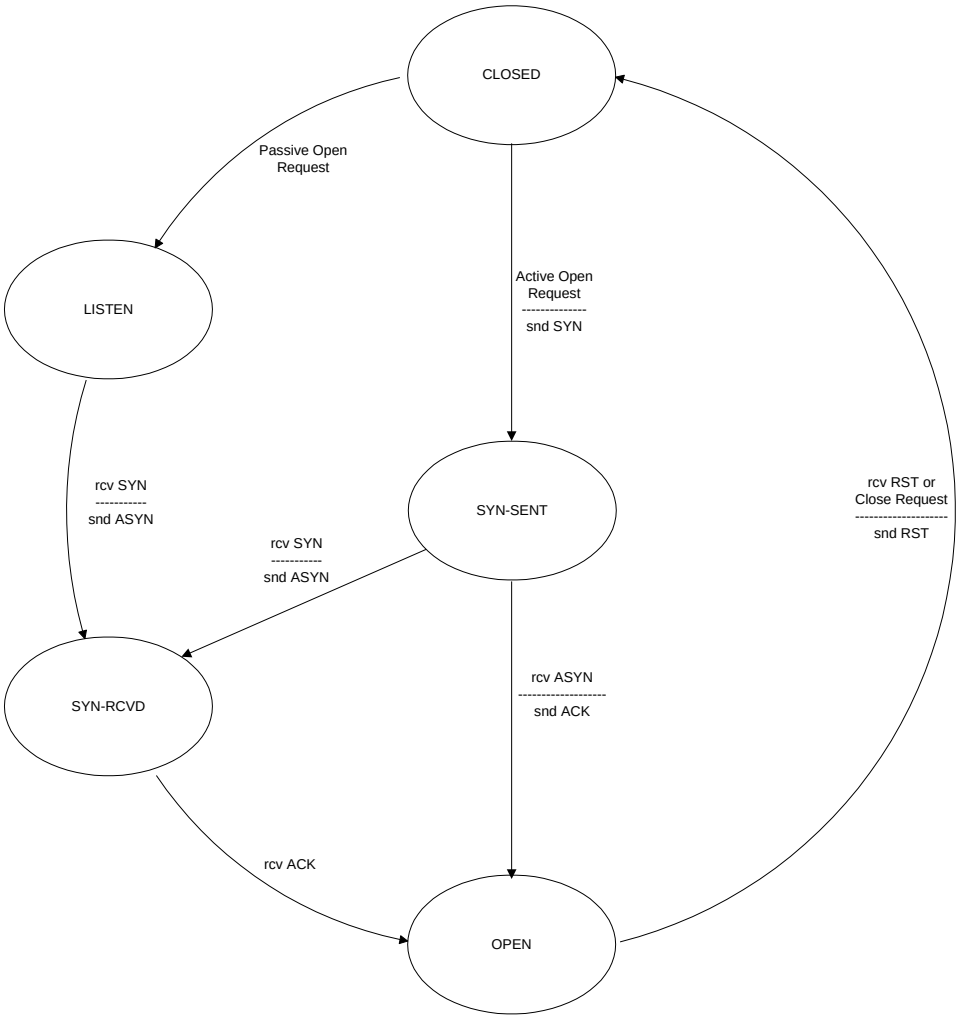


Figure 1

Each RMTP connection will progress through a series of states during its lifetime. The states are shown in **Figure 1** and are individually described below. In **Figure 1**, the ellipses represent the states of the RMTP finite state machine and the arcs represent changes in state. Each arc is annotated with the event causing the state change and the resulting output.

5.1.1.2.1 CLOSED

The CLOSED state exists when no connection exists.

5.1.1.2.2 LISTEN

The LISTEN state is entered after a passive Open request is processed. RMTP waits for an active request to establish a connection from a remote host.

5.1.1.2.3 SYN-SENT

The SYN-SENT state is entered after processing an active Open request. An initial sequence number is generated, and a SYN segment is sent to the remote host. RMTP then waits in the SYN-SENT state for acknowledgment of its Open request.

5.1.1.2.4 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 RMTP generates an initial sequence number for its side of the connection, and then sends the sequence number and an acknowledgment of the SYN segment to the remote host in an ASYN segment. It then waits for an acknowledgment.

The SYN-RCVD state is reached from the SYN-SENT state when a SYN segment is received from the remote host without an accompanying acknowledgment of the SYN segment sent to that remote host by the local RMTP. 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.

5.1.1.2.5 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 window size. In the Open state data may be sent between the two parties of the connection.

5.1.1.3 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, RMTP will send an RST segment to the other side of the connection and then enter the CLOSED state. RMTP discards segments received from the other side of the connection after the connection enters the CLOSED state. This simple connection closing facility requires that users determine that all data has been reliably delivered before requesting a close of the connection.

5.1.1.4 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.

If the RMTP connection is being used to send data, then RMTP will eventually detect and close a half-open connection when a send data request times out.

If the RMTP connection is being used to receive only, not send data, and if the application using RMTP has not received any data in a while, it should send something to test the liveliness of the connection. The TOPS OSP layer will handle this for HP-JetSend.

5.1.2 Version Negotiation

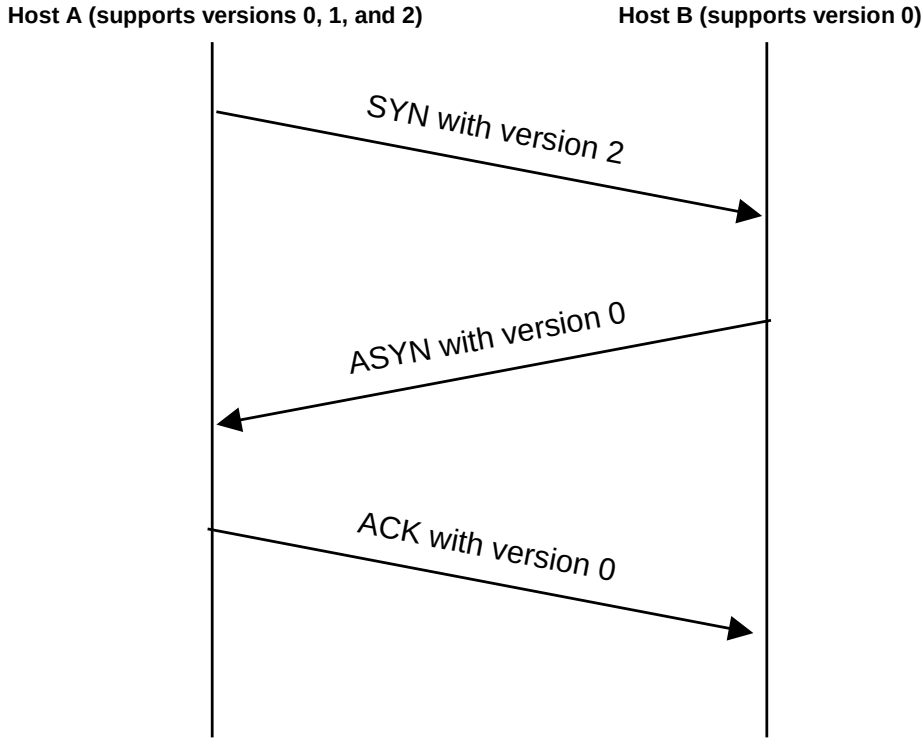
Each segment contains an RMTP version number. The version of the protocol that the two sides use for a connection is negotiated during the SYN - ASYN - ACK handshaking sequence that creates the connection. When two hosts contain different versions of RMTP, they use the lowest

of the two versions to connect to each other. The active side sends a SYN containing the version number of the highest version of the protocol it supports. The passive side responds with an ASYN containing the version number of the highest version of the protocol it supports or the version number that was contained in the SYN segment to which it is responding, whichever is lower. The active side responds to the ASYN with an ACK containing the same version number that was in the ASYN it is responding to. From this point forward, all segments sent through the connection contain that same version number that the active side sent in its ACK. Any segments containing any other version number are ignored. This is the agreed-upon version for the connection. During the lifetime of the connection each side will adhere to the rules of the agreed-upon version of RMTP.

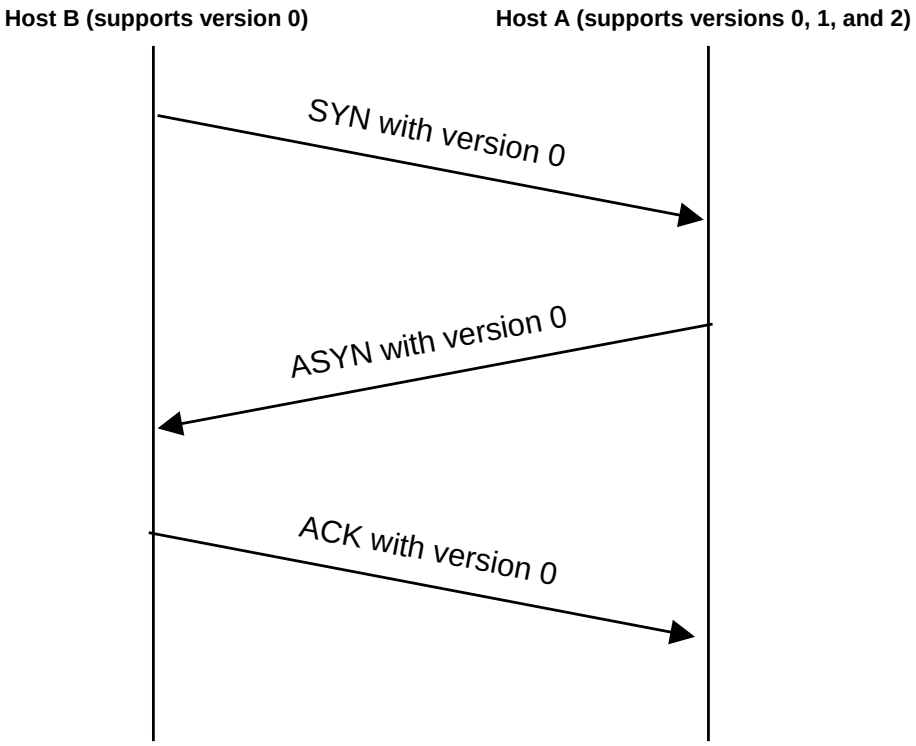
If one side cannot support the lowest of the two version numbers it will not complete the connection sequence and the two sides will not connect. All RMTP implementations must support, at a minimum, RMTP protocol version 0 to ensure backwards compatibility with all devices. Support for all other RMTP protocol versions is optional.

The following examples illustrate the negotiation. In the first two examples, host A supports protocol versions 0, 1 and 2. Host B supports version 0 only. In both examples, the handshaking will result in the two hosts agreeing upon version 0.

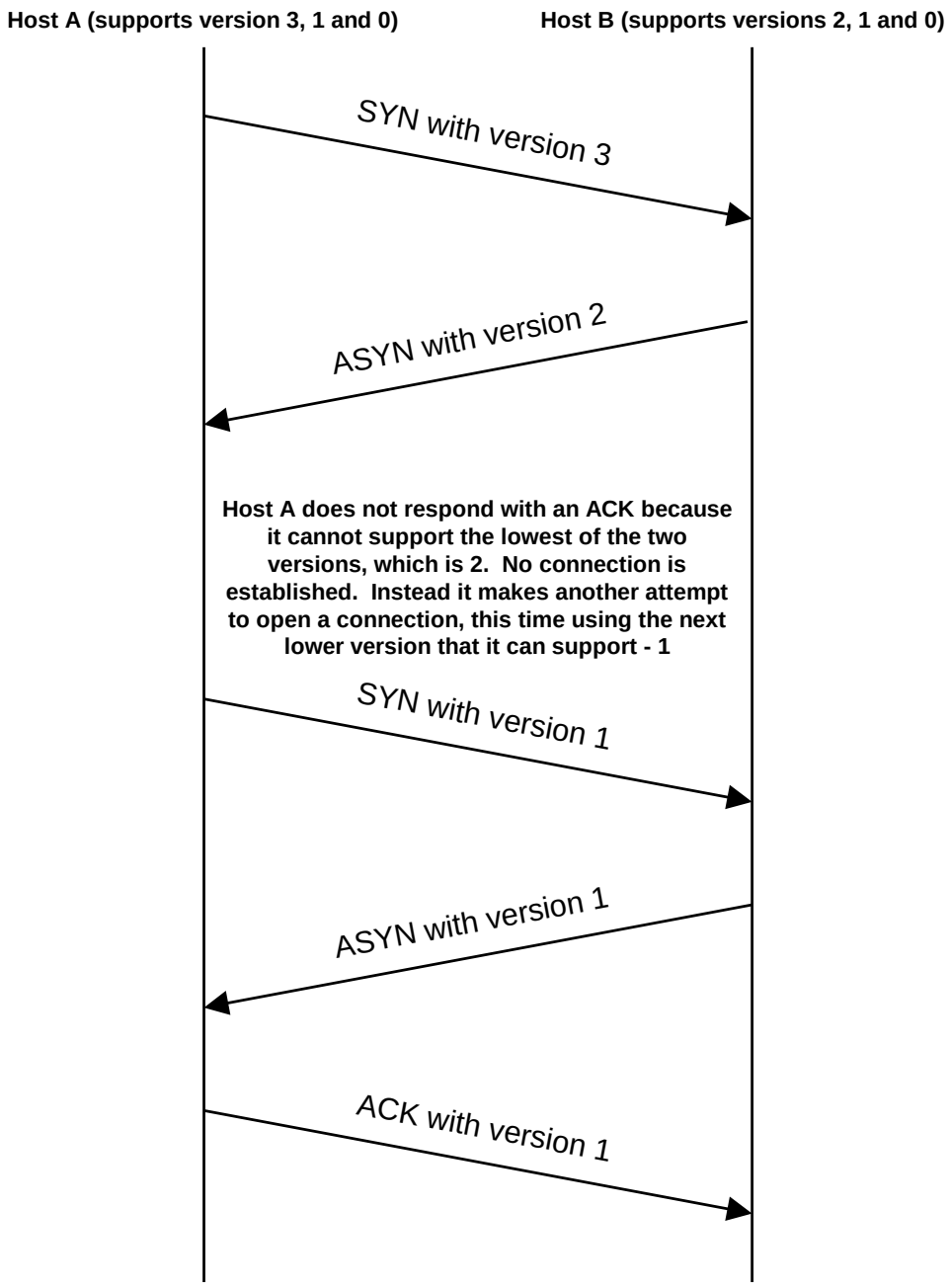
In the first example, host A is the active side and it tries to open a connection using version 2 of the protocol. Host B forces the version down to the highest level it can support which is 0.



In the second example, host B is the active side and it tries to open a connection using version 0 of the protocol. Although host A could support a version 2 connection, it agrees to settle for a version 0 connection since that is the best that host B can do.



If there is a discontinuity of versions that a host supports - say host A supports versions 0, 1 and 3 and host B supports versions 0, 1 and 2 - the connection attempt would fail. In this case the active side should lower its requested version to its next highest and try again to establish the connection. This next example illustrates this scenario.



5.1.3 Data Communication

Data flows through an RMTP connection in the form of segments. Each user message submitted with a Send request is packaged for transport as a single RMTP segment. Each RMTP segment is packaged as an RMTP header and one or more octets of data. RMTP 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 RMTP level, outgoing segments, as they are created, are queued as input to the datagram layer. Each segment is held by the sending RMTP until it is acknowledged by the foreign host. Incoming segments are queued as input to the user process. Segments are acknowledged when they have been queued as input to the user process or in the case of APRB segments, when they have been accepted by the receiving side.

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 RMTP determines that a buffer submitted with a Send request exceeds the maximum size segment permitted on the connection, RMTP will return an error to the user. In addition, RMTP 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.

5.1.4 Reliable Communication

RMTP implements a reliable message service through a number of mechanisms. These include the insertion of sequence numbers and checksums into segments, the positive acknowledgment of segment receipt, and time-out and retransmission of missing segments.

5.1.4.1 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. 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 (A)SYN segment and APRB segment, which cannot contain data, are exceptions. The (A)SYN and APRB segments are always sent with a unique sequence numbers. In the case of the (A)SYN segment, it is the initial sequence number.

5.1.4.2 Checksums

Checksums are included in packets in order that errors encountered during transmission may be detected. For Internet protocols such as TCP and UDP this is especially important because packets may have to cross wireless networks such as the Packet Radio Network and Atlantic Satellite Network where packets may be corrupted.

RMTP can not rely on its underlying datagram transport to do checksum computations. In UDP for example, it is a system option and is often not enabled. Each RMTP segment contains a checksum to allow the receiver to detect damaged segments. Like UDP and TCP, RMTP uses the 16-bit Internet checksum, which is specified in RFC-1071:

The following simple "C" function, derived from RFC-1071, computes the Internet checksum and returns its one’s complement for RMTP.

```
Tu16   InternetChecksum
       (
       void*  prmPacketPtr,
       Tu32   prmNumBytes
       )
       {
       Tu16*  lclWordPtr    =      {(Tu16*)prmPacketPtr};
       Tu32   lclNumWords   =      {prmNumBytes >> 1};
       Tu32   lclChkSum     =      {0};
```

```
Tu16   lclLastWord;
Tu32   lclCount;

/*
================================================
Sum 16 bits at a time into a 32 bit accumulator
------------------------------------------------ */
for (lclCount=0; lclCount<lclNumWords; lclCount++)
       {
       lclChkSum += *lclWordPtr++;
       }
/*
================================================
If the packet doesn't end on a word boundary, pad the
trailing byte with 0s to a word boundary and add it in too.
Make sure the byte order of the word we add in is the same
on the sending and receiving side.
------------------------------------------------ */
if (isodd(prmNumBytes))
       {
       lclLastWord  = *((Tu8*)prmPacketPtr+prmNumBytes-1);
       lclChkSum   += htons(lclLastWord);
       }
/*
================================================
Fold 32 bit checksum into 16 bits
------------------------------------------------ */
lclChkSum  = (lclChkSum >> 16) + (lclChkSum & 0xffff);
lclChkSum += (lclChkSum >> 16);
/*
================================================
Handle overflow and return complement
------------------------------------------------ */
return ((Tu16)( ~lclChkSum & 0xffff ));
}
```

To use the checksum, the sending side of RMTP sets the checksum field of a segment to all 0s. It then passes the complete segment including the zeroed out checksum field to InternetChecksum(). It places the result into the checksum field and sends the segment.

The receiving side receives the segment from the sender with the checksum field filled in. The receiving side passes the complete segment including the checksum field to InternetChecksum(). If the result is 0 then the segment passes the checksum test.

5.1.4.3 Positive Acknowledgment of Segments

RMTP assumes it has only an unreliable datagram service to deliver segments. To guarantee delivery of segments in this environment, RMTP uses positive acknowledgment and retransmission of segments. Each segment containing a unique sequence number (ADTA,SYN,ASYN,APRB) are acknowledged when they are correctly received and accepted by the destination host. Segments containing only an acknowledgment are not acknowledged. Damaged segments are discarded and are not acknowledged. Segments are retransmitted when there is no timely acknowledgment of the segment by the destination host.

RMTP uses a cumulative acknowledgment to acknowledge all segments up to a specified sequence number. The ACK control bit is set and the last acknowledged sequence number is placed in the Acknowledged Sequence Number field.

5.1.4.4 Retransmission Time-out

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 RMTP. The positive acknowledgment policy requires the receiver to acknowledge a segment only when the segment has been correctly received and accepted.

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

Because of the variability of the networks that compose an internetwork system, the retransmission time-out must be dynamically determined.

5.1.4.4.1 Algorithm for Computing the Retransmission Time-out

RMTP maintains a separate retransmission time-out value (RTO) for each connection. Each time a [non-retransmitted] segment is acknowledged, measure the elapsed time between sending the segment and receiving its acknowledgment. This measured elapsed time is the Round Trip Time (RTT). RMTP uses this simple and efficient algorithm from 4.3BSD UNIX to compute the connection's retransmission time-out (RTO):

```
Tu32  RetransmissionTimeOut (int RTT)
       {
       static  Tbool initialized  =  {FALSE};
       static  int   dev          =  {0};
       static  int   avg          =  {0};
       auto    int   RTO;
       auto    int   delta;

       if (! initialized)
              {
              avg = RTT << 3;
              initialized = TRUE;
              }

       delta  = RTT - (avg >> 3);
       avg   += delta;
       if (delta < 0) delta = (-delta);   /* abs(delta) */
       delta -= (dev >> 2);
       dev   += delta;
       RTO    = ((avg >> 2) + dev) >> 1;

       return(RTO);
       }
```

When the time since transmitting a segment on a connection exceeds the RTO for the connection and an ACK for the segment has not arrived, retransmit the segment.

For the first segment of the connection (the initial SYN or ASYN), initialize the RTO to 6 seconds.

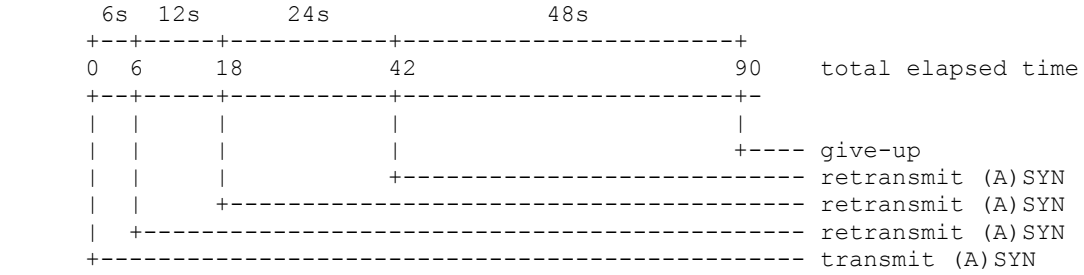
5.1.4.4.1.1 RTO Back-off During Retransmission

A problem occurs when a segment is retransmitted and an ACK is received. Is the ACK for the first transmission or the second?

To handle this, when RMTP receives an ACK for a segment that was retransmitted, it does not update the RTO with the normal algorithm described above. Instead, each time it has to retransmit a segment over a connection, RMTP doubles the RTO for that connection (RTO = 2*RTO). As soon as RMTP receives an ACK for a segment that has not been retransmitted, it goes back to using the normal RTO algorithm.

5.1.4.5 Connection Establishment Time-out

Since RMTP does not have an estimate of the round trip time to a foreign host when it sends a SYN (or ASYN) segment to it, RMTP uses a similar approach as that used by the 4.3BSD UNIX TCP implementation for giving up on a connection establishment attempt. If RMTP does not receive a response to a (A)SYN within 90 seconds it gives up.



If during (A)SYN retransmission back-off, the RTO for the connection is computed to be 90 seconds or more in size, RMTP will give up the attempt to establish a connection. Given the initial value of 6 for RTO, this allows for an initial transmission followed by no more than three re-transmissions.

```
If ((RTO = 2*RTO) >= 90SECONDS) GiveUp();
```

5.1.4.6 Connection Time-out

If during retransmission back-off, the RTO for a connection is computed to be 3 minutes or more in size, RMTP will treat this event as if it had received a RST segment from the remote host. RMTP closes the connection.

```
If ((RTO = 2*RTO) >= 3MINUTES) CloseConnection();
```

5.1.5 Flow Control and Window Management

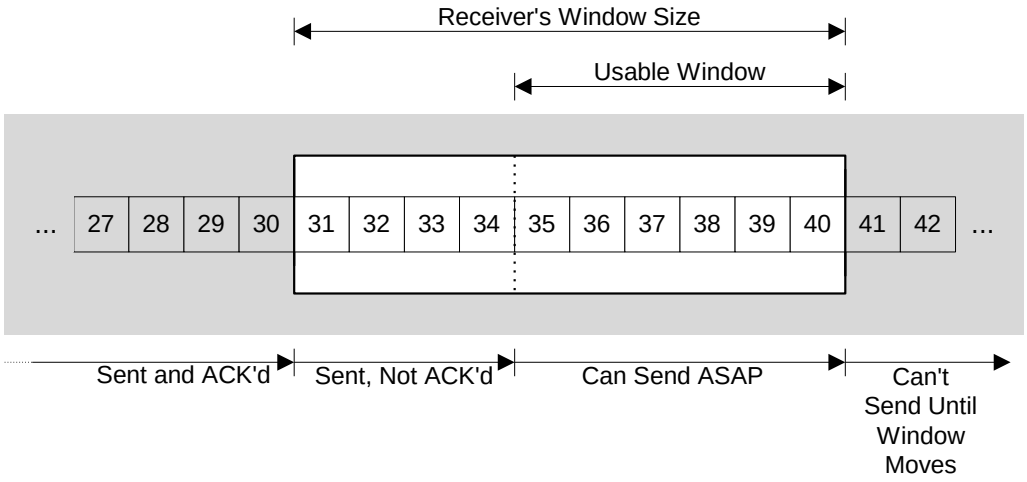


Figure 2

RMTP employs a sliding window without selective or negative acknowledgments for flow control. The maximum window size is 255 (8-bit Window Size field in header). **Figure 2** illustrates the sliding window mechanism. In the figure, segments with sequence numbers 27 through 42 are shown. The window advertised by the receiver is called the Receiver's Window Size. The Receivers Window Size covers segments 31 through 40, meaning that the receiver has acknowledged all segments up to and including number 30, and has advertised a window size of 10. The receiver advertises its window size by placing the value in the Window Size field of each of the segments it sends. It's size is 10 because that was the value in the Window Size field of the last segment that the sender received. The receiver can stop the flow of data to itself by allowing its advertised window size to go to zero.

The window size is relative to the acknowledged sequence number. The sender computes its Usable Window, which is how many segments it can send immediately.

Over time the sliding window moves to the right, as the receiver ACKs segments. The relative motion of the two ends of the window increases or decreases the size of the window. Three terms are used to describe the movement of the right and left edges of the window:

1. The window *closes* as the left edge advances to the right. This happens when segments are sent and ACK'd. If a window closes completely - goes to 0 - the advertiser must eventually reopen it using an ACK.
2. The window *opens* when the right edge moves to the right, allowing more segments to be sent. This happens when new buffers are made available for receiving segments on the receiver's side.
3. The window *shrinks* when the right edge moves to the left. This potentially wastes network bandwidth because segments that may have already been sent cannot be accepted by the receiver. **RMTP does not do window shrinking**.

This seemingly complicated flow control mechanism is very easy to implement. To implement the sliding window, the receiving side simply fills in the window field in an ASYN, ADTA, APRB, or ACK segment with the number of buffers it has available for receiving segments.

The sending side computes the usable window size using the *acknowledged sequence number* (AckSeqNum) and *advertised window size* (AdvrtzdWinSize) from any ASYN,ADTA,ACK,or APRB that it receives together with the *last sent and ACK'd sequence number* (LastAckdSeqNum) and a count of the *number of sent but not ACK'd segments* (NumSentNotAckd):

```
if (AckdSeqNum >= LastAckdSeqNum)
        {
        UsableWindow =
            (
            AckdSeqNum     -
            LastAckdSeqNum +
            AdvrtzdWinSize -
            NumSentNotAckd
            );
        }
```

The sending side then immediately sends all the unsent segments that it has queued, up to and including UsableWindow segments (the ones labeled "Can Send ASAP" in **Figure 2**).

5.1.5.1 Persist Timer

A persist timer is set when the receiving end of a connection can accept no more data segments, stopping RMTP from sending segments. (In a sliding window protocol it advertises a window of 0.) Since ACKs are not acknowledged, there is a chance that a future window update, allowing RMTP to send some segments, can get lost. If the sending side has segments waiting to be sent, it needs to probe the receiver periodically to determine if the window was re-opened and the notifying ACK was lost.

RMTP sets a persist timer for 30 seconds when the receiver's advertised window size goes to 0. After the 30 second interval, RMTP checks to see if it has a segment queued up to send. If not it simply resets the 30 second timer and waits to repeat the process. If so, it sends an APRB segment to probe the receiver. When it receives the ACK to the APRB, if the window is still 0, it resets the timer and waits to repeat the process. As soon as the ACK to the probe indicates that the window has re-opened, the sending RMTP stops the persist timer process and sends the next segment.

This gives rise to an RMTP requirement: a receiving RMTP with a window of 0 must always be able to accept an APRB segment and ACK it. Also, the receiving implementation must guard against allowing the receipt of an APRB segment itself to close its window - which could perpetuate the closed window indefinitely.

5.1.5.2 An Aside on Silly-Window Syndrome

If a sending side can transmit segments faster than a receiving side can consume them, this flow control mechanism will begin to exhibit something like the silly-window syndrome [RFC 813]. When this happens, flow control will resemble a request-reply protocol. However, unlike a byte-stream protocol like TCP, RMTP does not become inefficient in its usage of network bandwidth because the ratio of user data transmitted to segment header size does not go down. Bottom line: an RMTP implementation does not have to concern itself with managing silly-window syndrome.

5.2 RMTP Segments and Formats

5.2.1 RMTP Header Format

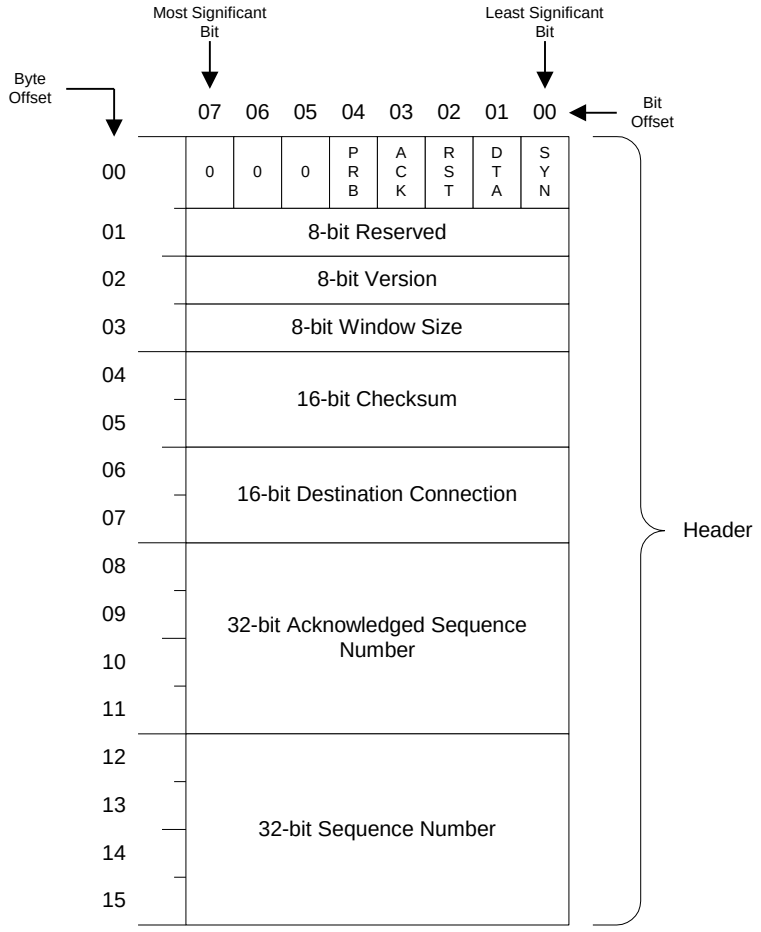


Figure 3

Every RMTP segment is prefaced with an RMTP header. The format of the header is shown in **Figure 3**. Byte offsets from the beginning of the segment are numbered from top to bottom. Bit positions are numbered from right to left beginning with the least significant bit and ending with the most significant bit. The numerical value of a set bit is two raised to the power of the bit position.

All multi-byte words in a segment are transmitted across the network in network byte order – that is in big-endian form.

5.2.2 RMTP Header Fields

These fields are common to all RMTP segments and so are described only once, in this section.

5.2.2.1 Control Bits

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

0) SYN Establish connection and synchronize sequence numbers.
1) DTA The segment contains data.

2) RST Close the connection.
3) ACK The segment contains an acknowledgment.
4) PRB The segment is a sequenced and ACK-able probe.
5) Unused - set to 0.
6) Unused - set to 0.
7) Unused - set to 0.

The control bits are used in combinations to create these segment types:

SYN	DTA	RST	ACK	PRB	Name	Meaning
1	0	0	0	0	SYN	Request to establish connection.
1	0	0	**1**	0	ASYN	ACK request for connection.
0	**1**	0	**1**	0	ADTA	Data segment, piggybacked ACK.
0	0	**1**	0	0	RST	Close connection.
0	0	0	**1**	0	ACK	ACK with no data.
0	0	0	**1**	**1**	APRB	ACK-able probe.

5.2.2.2 Version

The version number of the protocol described by this document. Current value is zero (0).

5.2.2.3 Window Size

The Advertised Window Size of the side of the connection that sent this segment. Its contents are defined in section 5.1.5.

- In a RST segment this field is ignored.

5.2.2.4 Checksum

The segment checksum. Its contents are defined in section 5.1.4.2.

5.2.2.5 Destination Connection

The connection to which the receiving side of RMTP should route this segment.

- In a SYN segment this field must be set to 0.
- In an ASYN segment this field is set to the value that was contained in the Source Connection field of the SYN that the ASYN is acknowledging.
- In ADTA, RST, ACK, and APRB segments this field is set to the value that was contained in the Source Connection field of the (A)SYN that this side of the connection received during the connection establishment process.

5.2.2.6 Acknowledged Sequence Number

The sequence number of the last segment that the sender of this segment has received and for which the sender of this segment has received all earlier sequence numbers.

- In a SYN segment this field is ignored.
- In a RST segment this field is ignored.

5.2.2.7 Sequence Number

The sequence number of this segment.

- In an ACK segment this field is ignored.
- In a RST segment this field is ignored.

- In the SYN segment this field contains the initial sequence number of the connection for the sending side.
- In the ASYN segment this field contains the initial sequence number of the connection for the sending side.

5.2.3 Connection Fields

5.2.3.1 Source Connection

The connection number of the sending connection. In a SYN or ASYN segment this field is set to any 16 bit value except 0. Thereafter, any segments sent to the connection that sent the SYN or ASYN use this value in the destination connection field.

This field is in the header as a convenience. Both UDP end-points for all HP-JetSend RMTP connections are:

hostIPAddress.HP-JetSendWellKnownUDPPort

All RMTP connections are multiplexed through the well-known HP-JetSend UDP port. Therefor, Sockets or TLI can support only one connection between any single pair of hosts. This means that RMTP can derive the identity of the receiving connection from the source IPAddress information it obtains when it receives a segment. However, it is easier for an RMTP implementation to de-multiplex on an implementation defined connection identifier than on an IPAddress.

The connection field also allows RMTP to open multiple connections through the same socket to a single remote host. Currently HP-JetSend is defined in such a way that it will not make use of multiple RMTP connections to a single host.

(If an RMTP implementation can open a different socket for each connection and de-multiplex using the socket then the connection numbers will not add any value in that implementation (although they must still be used because the implementation on the other side of the connection may be relying upon them). An implementation can use a unique socket for each connection by opening a UDP socket for each new connection, binding the local endpoint to the local well-known HP-JetSend port, and calling connect() to associate the foreign host's well-known HP-JetSend port with the socket's foreign endpoint. RMTP is designed so that HP can implement it in constrained peripheral firmware environments. In these environments it may not be possible to open multiple sockets with the same UDP port as the local endpoint.)

5.2.3.2 Destination Connection

The connection number of the connection to which the receiving RMTP should route this segment.

5.2.4 Segments

All segments except the ADTA segment are fixed size. ADTA, since it is used to carry user data, is variable sized.

5.2.4.1 SYN Segment (0x01)

RMTP uses SYN to make a request to its remote peer to open a connection and synchronize sequence numbers. The SYN segment also

contains information to inform the remote peer of the maximum segment size it can accept and the sending side's connection number. (Notice that the side that sends the SYN must create a connection number for the connection it is trying to establish before it sends the SYN).

5.2.4.1.1 Segment Format

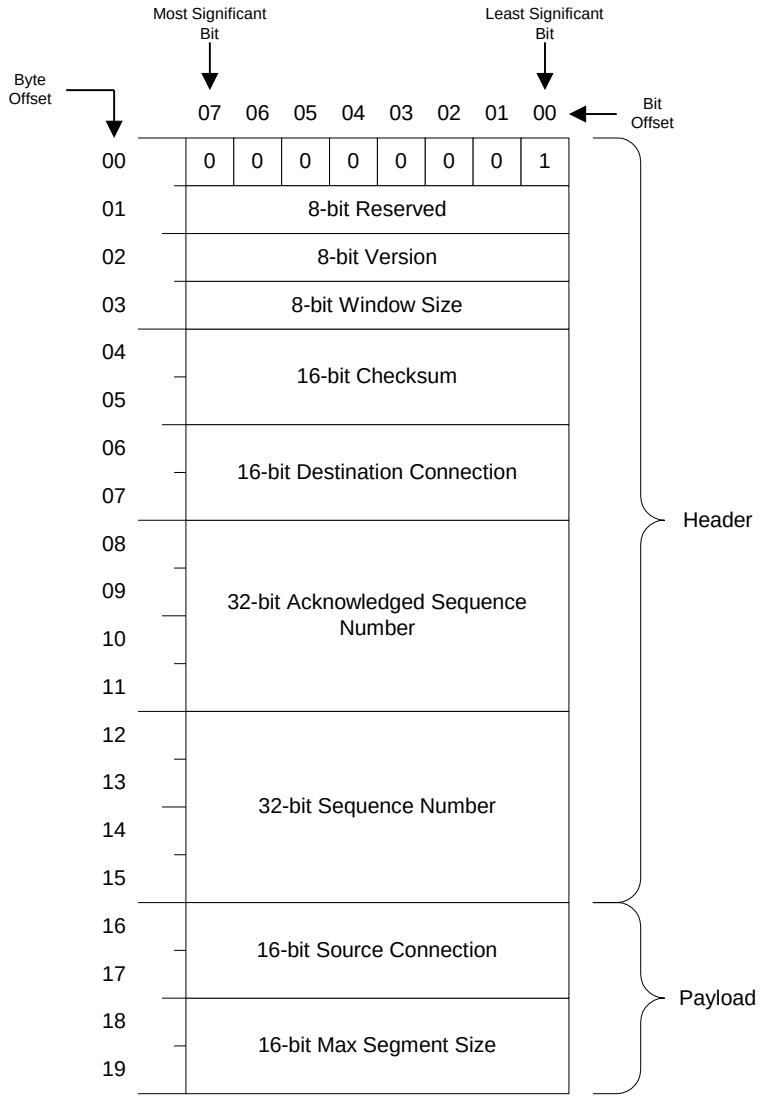


Figure 4

5.2.4.1.2 Fields

5.2.4.1.2.1 Source Connection

The source connection field value in a SYN segment is implementation defined but cannot be 0. It will be used by the side that sent this segment to identify this connection for the duration of this connection's existence.

5.2.4.1.2.2 Max Segment Size

The maximum segment size that this side of the RMTP connection will accept. This size is set once for each side and used

throughout the lifetime of the connection. The maximum segment size must be at least 21.

5.2.4.2 ASYN Segment (0x09)

RMTP uses ASYN to acknowledge a request from its remote peer to open a connection and synchronize sequence numbers. The ASYN segment also contains information to inform the remote peer of the maximum segment size it can accept, its own initial sequence number, and its own connection number. (Notice that the side that sends the ASYN must create a connection number for the connection it is agreeing to establish before it sends the ASYN).

5.2.4.2.1 Segment Format

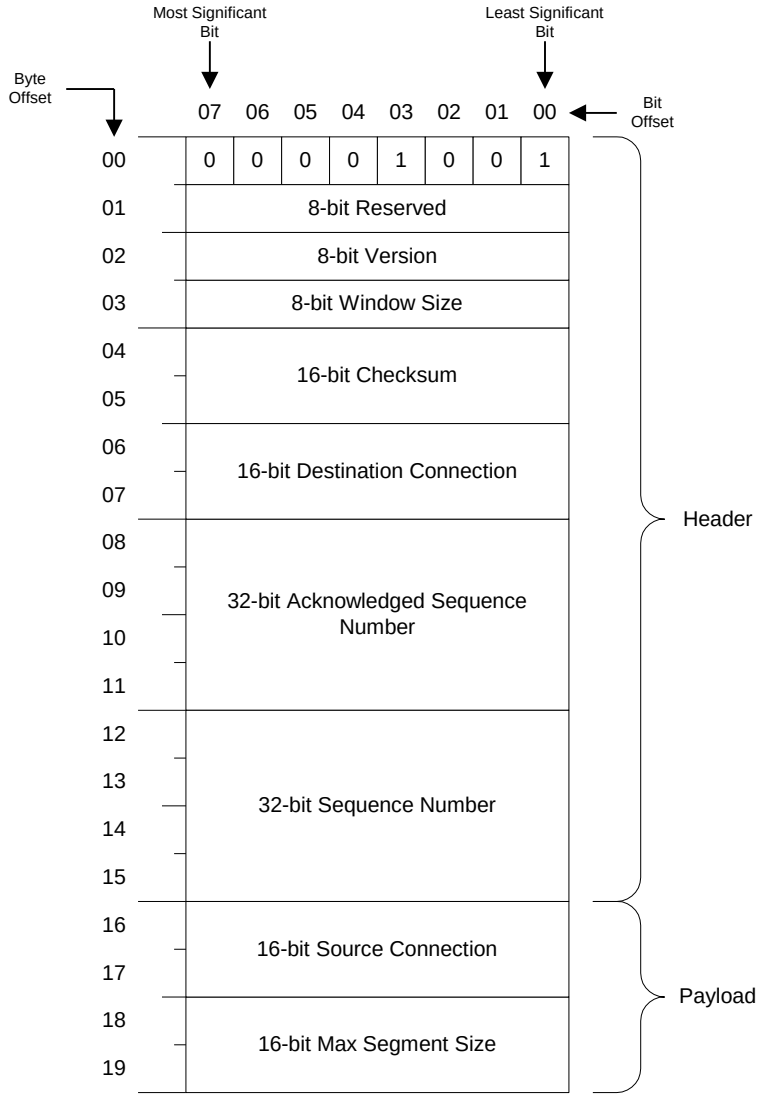


Figure 5

5.2.4.2.2 Fields

5.2.4.2.2.1 Source Connection

The source connection field value in a ASYN segment is implementation defined but cannot be 0. It will be used by the side that sends this segment to identify this connection for the duration of this connection's existence.

5.2.4.2.2.2 Max Segment Size

The maximum segment size that this side of the RMTP connection will accept. This size is set once for each side and used throughout the lifetime of the connection. The maximum segment size must be at least 21.

5.2.4.3 ADTA Segment (0x0A)

RMTP uses ADTA to send user data and a cumulative acknowledgment to its remote peer. RMTP uses this segment to piggy-back an acknowledgment on a data segment.

5.2.4.3.1 Segment Format

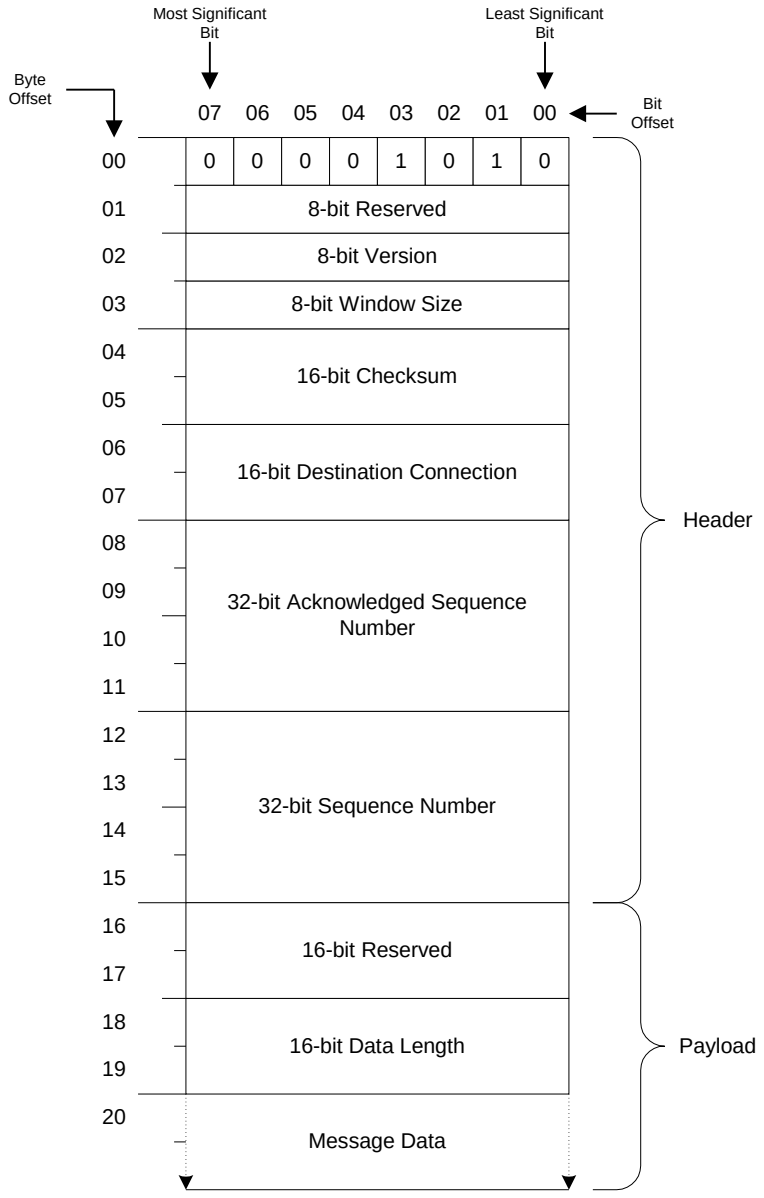


Figure 6

5.2.4.3.2 Fields

5.2.4.3.2.1 Data Length

The length in octets (bytes) of the Message Data field.

5.2.4.3.2.2 Message Data

The user data that RMTP has been told to send to a remote host.

5.2.4.4 RST Segment (0x04)

RMTP uses RST to tell its remote peer that the connection is closed.

5.2.4.4.1 Segment Format

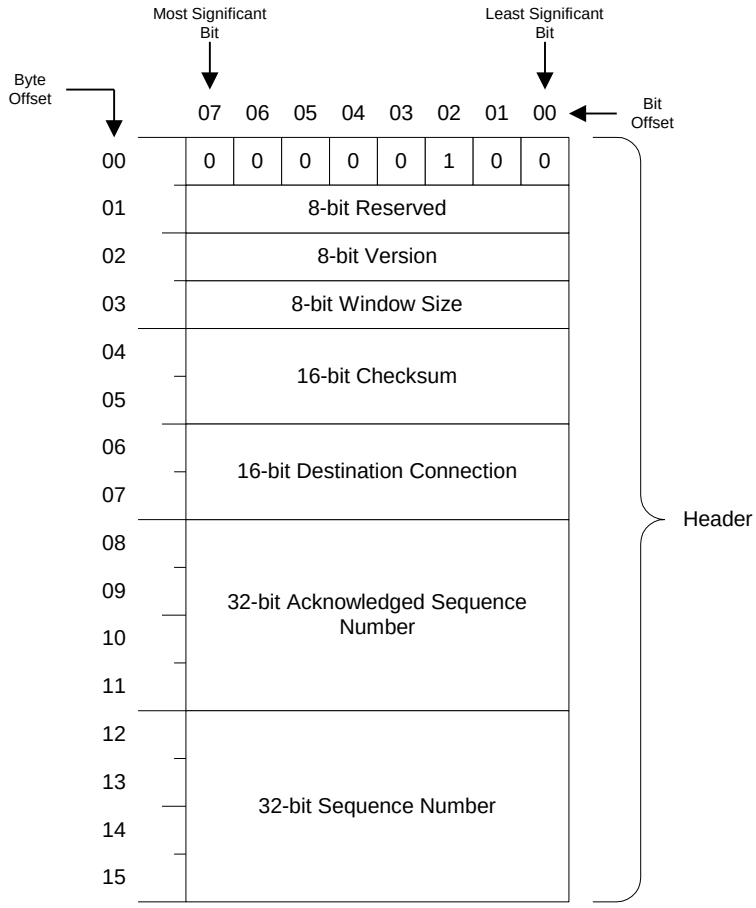


Figure 7

5.2.4.5 ACK Segment (0x08)

RMTP uses ADTA to send a cumulative acknowledgment to its remote peer. RMTP may send an ACK at any time. It may use ACKs to acknowledge the same sequence number as often as it likes. This is how it can re-open a closed receive window.

5.2.4.5.1 Segment Format

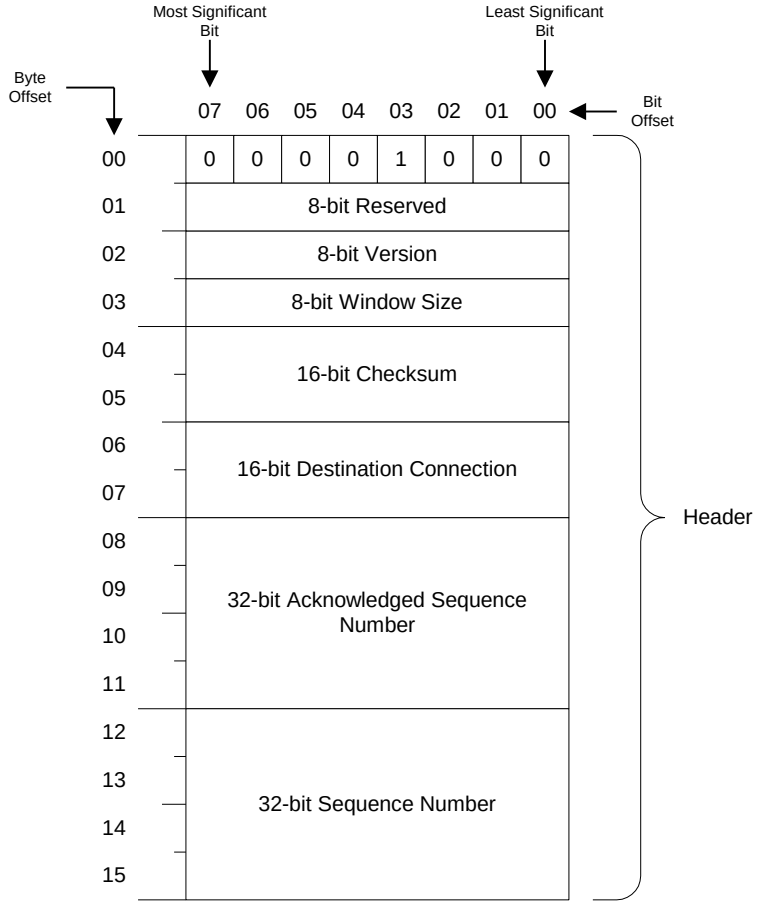


Figure 8

5.2.4.6 APRB Segment (0x18)

RMTP uses an APRB (**A**ck'd **PR**o**B**e) segment to probe the other side for lost window size updates when the window size is 0. It is a sequenced segment so it must be ACK'd. Its use is described in 5.1.5.1.

5.2.4.6.1 Segment Format

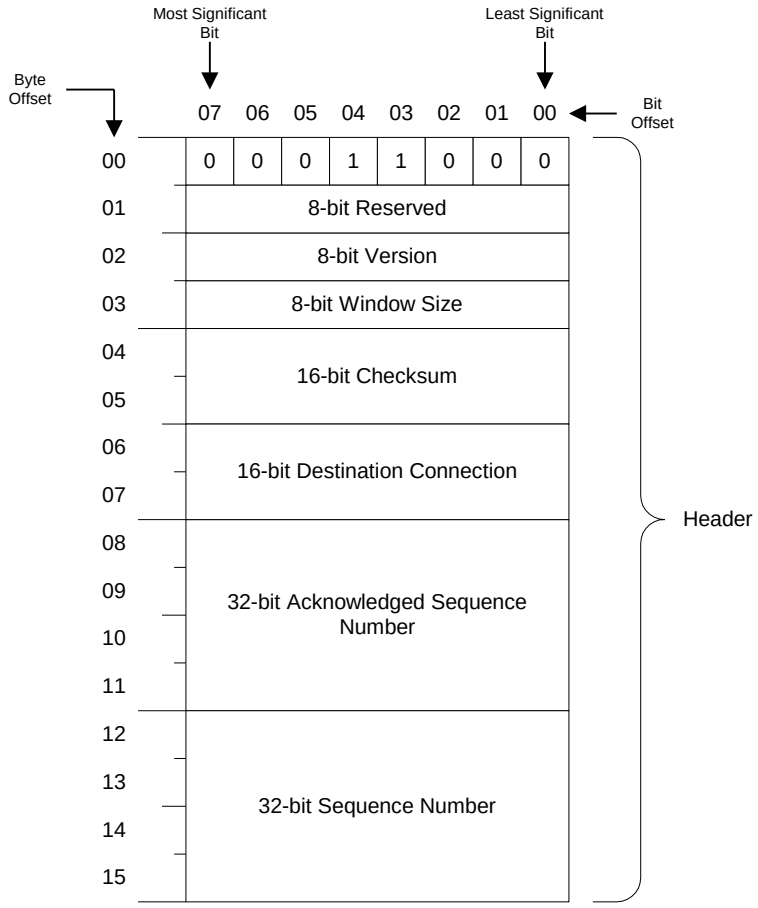

Most Significant Bit
Least Significant Bit
Byte Offset
Bit Offset
07 06 05 04 03 02 01 00
00
0 0 0 1 1 0 0 0
01
8-bit Reserved
02
8-bit Version
03
8-bit Window Size
04
05
16-bit Checksum
06
07
16-bit Destination Connection
08
09
10
11
32-bit Acknowledged Sequence Number
12
13
14
15
32-bit Sequence Number
Header

Figure 9