

Expires in six months

25 February 1999

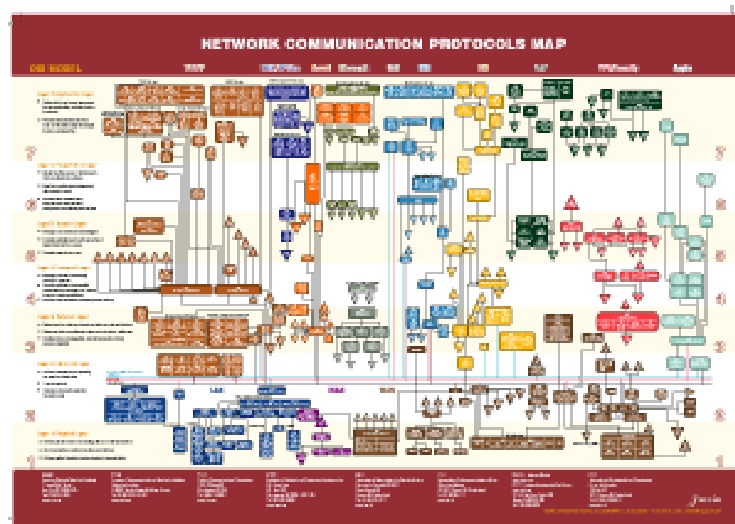
RELIABLE UDP PROTOCOL  
<draft-ietf-sigtran-reliable-udp-00.txt>

Status of This Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC 2026. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

To learn the current status of any Internet-Draft, please check the "lid-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), nic.nordu.net (Europe), munnari.oz.au (Pacific Rim), ftp.ietf.org (US East Coast), or ftp.isi.edu (US West Coast).



Network Communication Protocol Map. To order: <http://www.javvin.com/map.html>  
Easy to use network sniffing tool: <http://www.javvin.com/packet.html>

Abstract

This Internet Draft discusses Reliable UDP (RUDP). RUDP is a simple packet based transport protocol. RUDP is based on RFCs 1151 and 908 - Reliable Data Protocol. RUDP is layered on the UDP/IP Protocols and provides reliable in-order delivery (up to a maximum number of retransmissions) for virtual connections. RUDP has a very flexible design that would make it suitable for a variety of transport uses. One such use would be to transport telecommunication signaling protocols.

TABLE OF CONTENTS

- 1. Introduction.....3
  - 1.1 System Overview.....3
  - 1.2 Background.....5
  - 1.3 Data Structure Format.....5
  - 1.4 Feature Negotiation.....16
- 2. Future Potential Enhancements.....16
- 3. References.....17
- 4. Author's Addresses.....17

## 1. Introduction

This Internet Draft discusses a simple packet based transport protocol designed to support applications that require a reliable, sequenced packet based transport protocol. RUDP is based on RFCs 1151 and 908 - Reliable Data Protocol. RUDP is layered on the UDP/IP Protocols and provides reliable in-order delivery (up to a maximum number of retransmissions) for virtual connections.

### 1.1 Background

A reliable transport protocol is needed to transport of telephony signaling across IP networks. This reliable transport must be able to provide an architecture for a variety of applications (i.e. signaling protocols) requiring transport over IP.

Existing IP protocols have been scrutinized and it has been concluded that a new reliable transport mechanism for telecommunication signaling protocols is needed. This protocol should meet the following criteria:

- \* transport should provide reliable delivery up to a maximum number of retransmissions (i.e. avoid stale signaling messages).
- \* transport should provide in-order delivery.
- \* transport should be a message based.
- \* transport should provide flow control mechanism.
- \* transport should have low overhead, high performance.
- \* characteristics of each virtual connection should be configurable (i.e. timers).
- \* transport should provide a keep-alive mechanism.
- \* transport should provide error detection.
- \* transport should provide for secure transmission.

RUDP is designed to allow characteristics of each connection to be individually configured so that many protocols with different transport requirements can be implemented simultaneously on the same platform.

### 1.3 Data Structure Format

#### 1. Six octet minimum RUDP header for data transmissions

Each UDP packet sent by RUDP must start with at least a six octet header. The first octet contains a series of single bit flags. The next three fields are each one octet in size. They are: Header length, Sequence

number, and Acknowledgment number. These three octets are followed by a two octet checksum.

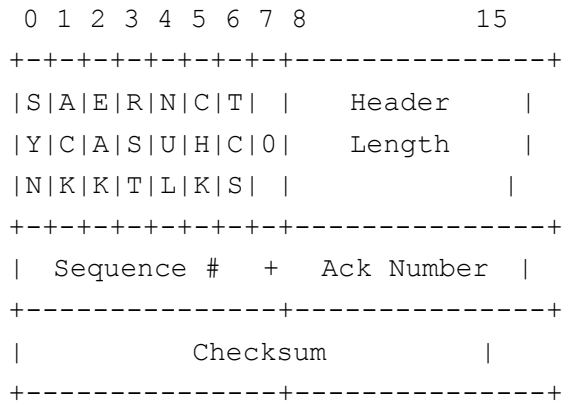


Figure 1, RUDP Header

Control bits

The control bits indicate what is present in the packet. The SYN bit indicate a synchronization segment is present. The ACK bit indicates the acknowledgment number in the header is valid. The EACK bit indicates an extended acknowledge segment is present. The RST bit indicates the packet is a reset segment. The NUL bit indicates the packet is a null segment. The TCS bit indicates the packet is a transfer connection state segment. The SYN, EACK, RST, and TCS bits are mutually exclusive. The ACK bit is always set when the NUL bit is set. The CHK bit indicates whether the Checksum field contains the checksum of just the header or the header and the body (data). If the CHK bit is zero, the Checksum field only contains the checksum of the header. If the CHK bit is one, the Checksum field contains the checksum of the header and data.

Header length.

The Header length field indicates where user data begins in the packet. If total length of the packet is greater than the value of the header length field, user data exists in the packet. User data cannot be present in packets with the EACK, NULL, or RST bits set. A packet with user data in it always has the ACK bit set and is called a data segment.

Sequence number.

Each packet contains a sequence number. When a connection is first opened, each peer randomly picks an initial sequence number. This sequence number is used in the SYN segments to open the connection. Each transmitter increments the sequence number before sending a data, null, or reset segment.

Acknowledgment Number.

The acknowledgment number field indicates to a transmitter the last in-sequence packet the receiver has received.

Checksum.

A checksum is always calculated on the RUDP header to ensure integrity. In addition, the checksum can be calculated on the header and body if the CHK bit is set to one. The checksum algorithm used in RUDP is the same algorithm used in UDP and TCP headers which is the 16-bit one's complement of the one's complement sum of bytes being checksummed.

## 2. SYN Segment

The SYN is used to establish a connection and synchronize sequence numbers between two hosts. The SYN segment also contains the negotiable parameters of the connection. All configurable parameters that the peer must know about are contained in this segment. This includes the maximum number of segments the local RUDP is willing to accept and option flags that indicate the features of the connection being established. A SYN segment must not be combined with user data. A SYN segment is also used to perform an auto reset on a connection. Auto reset is described later.

Figure 2 shows a SYN segment.



### Maximum Number of Outstanding Segments

The maximum number of segments that should be sent without getting an acknowledgment. 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. This is not a negotiable parameter. Each side must use the value provided by its peer when sending data.

### 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. A preliminary subset of flags are defined as follows:

Bit #	Bit Name	Description
0	not used	not used, must always be set to 1.
1	CHK	Data Checksum enable. If this bit is set then the checksum field will contain a checksum of the entire RUDP packet (header and data). This is a negotiable parameter.
2	REUSE	This bit must be set during an auto reset to indicate the previous negotiable parameters should be used. When this bit is set the following fields of the SYN should be set to zero by the sender and must be ignored by the receiver: Maximum Segment Size, Retransmission Timeout Value, Cumulative Ack Timeout Value, Max Retransmissions, Max Cumulative Ack, Max Out of Sequence, and Max Auto Reset.
3-7	Spare	

### Maximum Segment Size

The maximum number of octets that can be received by the peer sending the SYN segment. Each peer may specify a different value. Each peer must not send packets greater than the value of this field received from its peer during connection negotiation. This number includes the size of the RUDP header. This is not a negotiable parameter.

### Retransmission Timeout Value

The timeout value for retransmission of unacknowledged packets. This value is specified in milliseconds. The valid range is 100 to 65536. This is a negotiable parameter, both peers must agree on the same value for this parameter.

#### Cumulative Ack Timeout Value

The timeout value for sending an acknowledgment segment if another segment is not sent. This value is specified in milliseconds. The valid range is 100 to 65536. This is a negotiable parameter, both peers must agree on the same value for this parameter. In addition, this parameter should be smaller than the Retransmission Timeout Value.

#### Null Segment Timeout Value

The timeout value for sending a null segment if a data segment has not been sent. Thus, the null segment acts as a keep-alive mechanism. This value is specified in milliseconds. The valid range is 0 to 65536. A value of 0 disables null segments. This is a negotiable parameter, both peers must agree on the same value for this parameter.

#### Transfer State Timeout Value

This timeout value indicate the amount of time the state information will be saved for a connection before an auto reset occurs. This value is specified in milliseconds. The valid range is 0 to 65536. This is a negotiable parameter, both peers must agree on the same value for this parameter. A value of 0 indicates the connection will be auto-reset immediately.

#### Max Retrans

The maximum number of times consecutive retransmission(s) will be attempted before the connection is considered broken. The valid range for this value is 0 to 255. A value of 0 indicates retransmission should be attempted forever. This is a negotiable parameter, both peers must agree on the same value for this parameter.

#### Max Cum Ack

The maximum number of acknowledgments that will be accumulated before sending an acknowledgment if another segment is not sent. The valid range for this value is 0 to 255. A value of 0 indicates an acknowledgment segment will be send immediately when a data, null, or reset segment is received. This is a negotiable parameter, both peers must agree on the same value for this parameter.

#### Max Out of Seq

The maximum number of out of sequence packets that will be accumulated before an EACK (Extended Acknowledgement) segment is sent. The valid range for this value is 0 to 255. A value of 0 indicates an EACK will be sent immediately if an out of order segment is received. This is a negotiable parameter, both peers must agree on the same value for this parameter.

Max Auto Reset

The maximum number of consecutive auto reset that will performed before a connection is reset. The valid range for this value is 0 to 255. A value of 0 indicates that an auto reset will not be attempted, the connection will be reset immediately if an auto reset condition occurs. This is a negotiable parameter, both peers must agree on the same value for this parameter. The consecutive auto reset counter is cleared once a connection is opened.

Connection Identifier

When opening a new connection each peer transmits a connection identifier that is unique among all RUDP current connections. Each side then saves the connection ID received. When an auto reset is performed, the peer shall send the saved connection ID originally received to indicate that an auto reset is being performed on the connection.

3. ACK Segment

The ACK Segment is used to acknowledge in-sequence segments. It contains both the next send sequence number and the acknowledgment sequence number in the RUDP header. The ACK segment may be sent as a separate segment, but it should be combined with data whenever possible. Data and Null segments must always include the ACK bit and Acknowledgment Number field. The size of a stand-alone ACK segment is six octets. Figure 3 shows a stand-alone ACK segment.

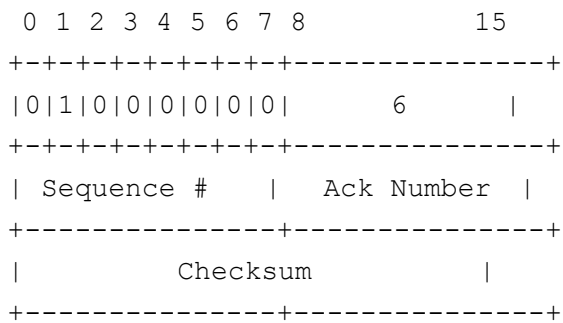


Figure 3, Stand-alone ACK segment

4. EACK segment

The EACK segment is used to acknowledge segments received out of sequence. It contains the sequence numbers of one or more segments received 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 header length is variable for the EACK segment. Its minimum value is seven and its maximum value depends on the maximum receive queue length. Figure 4 shows a stand-alone EACK segment.

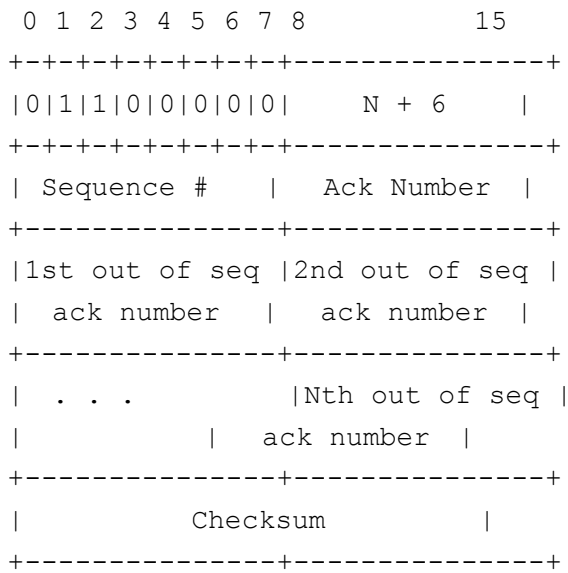


Figure 4, EACK segment

5. RST segment

The RST segment is used to close or reset a connection. Upon receipt of an RST segment, the sender must stop sending new packets, but must continue to attempt delivery of packets already accepted from the API. The RST is sent as a separate segment and does not include any data. Figure 5 shows a RST segment.

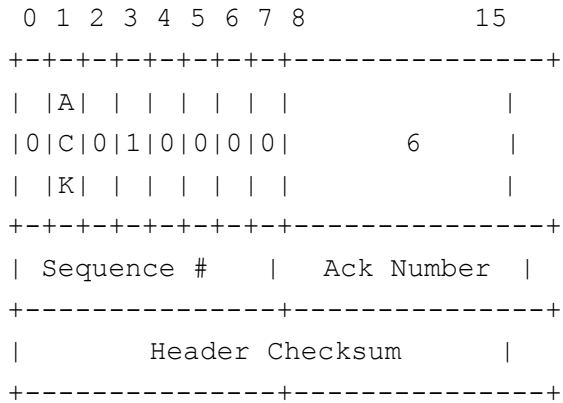


Figure 5, RST segment

6. NUL segment

The NUL segment is used to determine if the other side of a connection is still active. Thus, the NUL segment performs a keep-alive function. When a NUL segment is received, an RUDP implementation must immediately acknowledge the segment if a valid connection exists and the segment sequence number is the next one in sequence. The segment is then discarded. The NUL must be combined with an ACK in a segment but never combined with user data. Figure 6 shows a NUL segment.

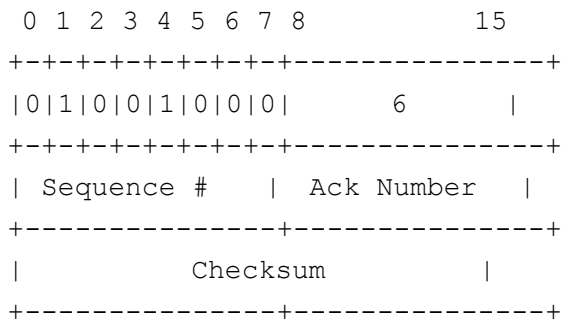


Figure 6, NUL segment

7. TCS Segment

The TCS is used to transfer the state of connection. Figure 7 shows a TCS segment.

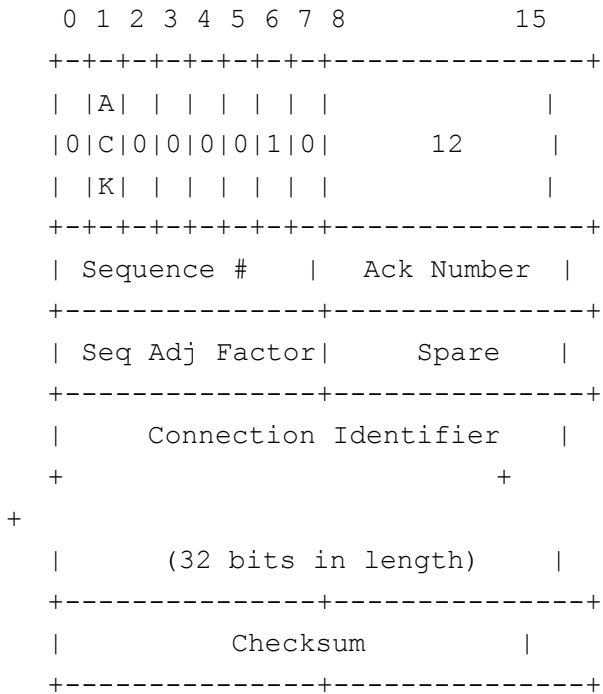


Figure 7, TCS segment

Sequence Number

The sequence number field contains the initial sequence number selected for this connection.

Acknowledgment Number

The acknowledgment number field indicates to a transmitted that last in-sequence packet the receiver has received.

Seq Adj Factor

This field is used during transfer of state to adjust sequence numbers between the old and current connections.

## Connection Identifier

When opening a new connection each peer transmits a connection identifier that is unique among all current RUDP connections. Each side then saves the connection ID received. This field is used to inform the peer connection of the connection record that is being transferred to this connection.

### 1.3.1 Detailed Design

A separate internet draft is being prepared which details in connections state transitions in Specification and Description Language (SDL) format. It also contains more details on the internal design of RUDP.

### 1.3.2 Feature Description

The following features are supported by RUDP. In the following description, transmitter and receiver refer to either clients or servers that are sending or receiving a data segment respectively on a connection. Client refers to the peer that initiates the connection and Server refers to the peer that listened for a connection. A connection is defined as an interface that serves a unique peer IP address/UDP port pair. A server or a client can have multiple connections on a particular IP address/UDP port, each connection will be with a unique peer IP address/UDP port pair.

#### 1. Retransmission timer.

The transmitter has a retransmission timer with a configurable time-out value. This timer is initialized every time a data, null, or reset segment is sent and there is not a segment currently being timed. If an acknowledgment for this data segment is not received by the time the timer expires, all segments that have been sent but not acknowledged are retransmitted. The Retransmission timer is re-started when the timed segment is received, if there is still one or more packets that have been sent but not acknowledged. The recommended value of the retransmission timer is 600 milliseconds.

#### 2. Retransmission Counter.

The transmitter maintains a counter of the number of times a segment has been retransmitted. The maximum value of this counter is configurable with a recommended value is 2. If this counter exceeds its maximum, the connection will be considered broken. Refer to paragraph item 14 below, Broken Connection Handling, for a description of how RUDP handles a broken connection.

#### 3. Stand-alone acknowledgments.

A stand-alone acknowledgment segment is a segment that contains only acknowledgment information. Its sequence number field contains the sequence number of the next data, null, or reset segment to be sent.

#### 4. Piggyback acknowledgments.

Whenever a receiver sends a data, null, or reset segment to the transmitter, the receiver includes the sequence number of the last in-sequence data, null, or reset segment received from the transmitter in the acknowledgment number field of the header of the segment being sent by the receiver.

#### 5. Cumulative acknowledge counter.

The receiver maintains a counter of unacknowledged segments received without an acknowledgment being sent to the transmitter. The maximum value of this counter is configurable. If this counter's maximum is exceeded, the receiver sends either a stand-alone acknowledgment, or an extended acknowledgment if there are currently any out-of-sequence segments. The recommended value for the cumulative acknowledge counter is 3.

#### 6. Out-of-sequence acknowledgments counter.

The receiver maintains a counter of the number of segments that have arrived out-of-sequence. Each time this counter exceeds its configurable maximum, an extended acknowledgment segment containing the sequence numbers of all current out-of-sequence segments that have been received is sent to the transmitter. The counter is then reset to zero. The recommended value for the out-of-sequence acknowledgements counter is 3.

When a transmitter receives an Extended Acknowledgment, it retransmits the missing data segments to the receiver.

#### 7. Cumulative acknowledge timer.

When a receiver has any segments that it has not acknowledged or if it has segments on its out-of-sequence queue, it waits a maximum amount of time before sending a stand-alone acknowledgment or an extended acknowledgment, respectively. When this timer expires, if there are segments on the out-of-sequence queue, an extended acknowledgment is sent. Otherwise, if there are any segments currently unacknowledged, a stand-alone acknowledgment is sent. The recommended value of the cumulative acknowledgement timer is 300 milliseconds.

The cumulative acknowledge timer is restarted whenever an acknowledgment is sent in a data, null, or reset segment, provided that there are no segments currently on the out-of-sequence queue. If there are segments on the out-of-sequence queue, the timer is not restarted, so that another extended acknowledgment will be sent when it expires again.

#### 8. Null segment timer

The client maintains a timer which is started when the connection is opened and is reset every time a data segment is sent. If the client's null segment timer expires, the client sends a null segment to the server.

The null segment is acknowledged by the server if its sequence number is valid. The server maintains a null segment timer with a time-out value of twice the client's time-out value. The server's timer is reset whenever a data or null segment is received from the client. If the server's null segment timer expires, the connection is considered broken. Refer to paragraph item 14 below, Broken Connection Handling, for a description of how RUDP handles a broken connection. The recommended value of the Null segment timer is 2 seconds.

#### 9. Auto Reset

Either side of a connection can initiate an auto reset. An auto reset can be caused by the retransmission count exceeding its maximum, the expiration of the server's Null segment timer, or the expiration of the transfer state timer. An auto reset will cause both peers to reset their current states including flushing retransmission and out-of-sequence queues and then reset their initial sequence number and re-negotiate the connection. Each peer will notify its Upper Layer Protocol (ULP) of the auto reset. There is a consecutive reset counter that sets the maximum number of auto-reset that can occur without the connection opening. If this counter exceeds its maximum the connection is reset. The recommended value for the consecutive reset counter is 3.

#### 10. Receiver Input Queue Size

The size of the receiver's input queue is a configurable parameter. The recommended value of the receiver input queue size is 32 packets. The input queue size acts as a flow control mechanism.

#### 11. Congestion control and slow start

RUDP does not provide any congestion control or slow start algorithms.

#### 12. UDP port numbers

RUDP doesn't place any restrictions on which UDP port numbers are used. Valid port numbers are ports not defined in RFC 1700.

#### 13. Support for redundant connections

If an RUDP connection fails, the Upper Layer Protocol will be signaled and the transfer state timer will be started. The ULP can initiate the transfer of this state to another RUDP connection via an API call and RUDP will transfer the state information to the new connection ensuring that packets are not duplicated or lost. If the ULP does not transfer the state to another connection before the Transfer State Timer expires, the connection state is lost and buffers of the queues of the connection are freed. The time-out value for the Transfer State timer is configurable. The recommended value for the Transfer State timer is 1 second.

#### 14. Broken connection handling

An RUDP connection is considered broken if either of the following situations occur:

- The Retransmission Timer expires and the Retransmission Count has been exceeded its maximum.
- A server's Null Segment Timer expires.

If either of the above two situations occur and the Transfer State timeout value is non-zero, the ULP will be notified of the connection failure via the Connection failure signal of the API and the Transfer State Timer will be started. If the Transfer State Timer expires, an Auto Reset is performed and the ULP is notified via the Connection auto reset signal of the API.

If the transfer state timeout value is zero, then an auto reset will be performed immediately when either of the above two situations occur. The ULP will be notified of a connection failure via the Connection auto reset signal of the API.

#### 15. Retransmission Algorithm

Retransmission occurs as a result of receiving an EACK segment or the time-out of the Retransmission timer.

When an EACK segment is received. The segments specified in the message are removed from the unacknowledged sent queue. The segments to be retransmitted are determined by examining the Ack Number and the last out of seq ack number in the EACK segment. All segments between but not including these two sequence numbers that are on the unacknowledged sent queue are retransmitted.

When a retransmission time-out occurs, all messages on the unacknowledged sent queue are retransmitted.

#### 16. Signals to Upper Layer Protocol (ULP)

The following signals are sent to the ULP via the API. These are used to signal asynchronous events to the ULP:

Connection open - This signal is generated when the state of the connection transitions to Open.

Connection refused - This signal is generated when the state of a connection transitions to the Closed state from other than the Close Wait state.

Connection closed - This signal is generated when the state of a connection transitions from the Close Wait state to the Closed state.

Connection failure - This signal is generated when a connection is declared broken, as described in section 1.3.2, item 15 (Retransmission Algorithm) above.

Connection auto reset - This signal is generated when a connection auto resets. This is an indication to the ULP that data may have been lost and RUDP is attempting to return the connection to the Open state.

## 17. Checksum Algorithm

The checksum algorithm used in RUDP is the same algorithm used in UDP and TCP headers which is the 16-bit one's complement of the one's complement sum of data being checksummed. The checksum is calculated over the entire RUDP packet if negotiated at the time the connection was opened. The negotiation is based on setting the CHK bit of the options field in the SYN segment used to open the connection. Otherwise, the checksum is calculated over the RUDP header only. Refer to RFC 1071 for implementation information for this algorithm.

## 18. FEC

RUDP does not define a procedure for generate Forward Error Correction (FEC). It is compatible with FEC algorithms that generate duplicate packets because it will throw away any duplicate packets it receives.

## 19. Security

RUDP will be compatible with the IPsec standard for secure IP communications.

## 1.4 Feature Negotiation

When client initiates a connection its sends a SYN segment which contains the negotiable parameters defined by the ULP via the API. The server can accept these parameters by echoing them back in its SYN with ACK response or propose different parameters in its SYN with ACK response. The client can then choose to accept the parameters sent by the server by sending an ACK to establish the connection or it can refuse the connection by sending send a RST. Features cannot be re-negotiated during an auto reset.

## 2.0 Future Potential Enhancements

RUDP should support a symmetrical mode of operation in addition to the current client/server mode of operation. This would allow both sides to actively start the connection.

RUDP should support the ability to expand the sequence and acknowledge fields from their current one octet length to two octets. This will allow transmission windows of greater than 255 buffer to be used.

Investigate use of the Nagle algorithm to improve network efficiency.

### 3.0 References

- [1] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification", RFC 791, USC/Information Sciences Institute, September 1981.
- [2] Postel, J., "User Datagram Protocol", RFC 768, USC/Information Sciences Institute, August 1980.
- [3] Postel, J. (ed.), "Transmission Control Protocol", RFC 793, USC/Information Sciences Institute, September 1981.
- [4] Velten, D., Hinden, R. and Sax, J., "Reliable Data Protocol", RFC 908, BBN Communications Corporation, July 1984.
- [5] Partridge, C. and Hinden, R., "Version 2 of the Reliable Data Protocol", RFC 1151, BBN Corporation, April 1990.
- [6] Braden, R., "Computing the Internet Checksum", RFC 1071, ISI, September 1988
- [7] V. Jacobson, "Congestion Avoidance and Control," Computer Communication Review, Val. 18, no. 4, pp. 314-329, Aug. 1988.
- [8] W. Stevens, RFC 2001 "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", January 1997
- [9] V. Jacobson, "Modified TCP Congestion Avoidance Algorithm", April 30, 1990.
- [10] Z. Wang, J. Crowcroft, A Dual-Window Model for Flow and Congestion Control, The Distributed Computing Engineering Journal, Institute of Physics/British Computer Society/IEEE, Vol 1, No 3, page 162-172, May 1994.
- [11] Romanow, Allyn, "TCP over ATM: Some Performance Results", ATM Forum/93-784

### 4.0 Author's Addresses

Tom Bova  
Cisco Systems  
13615 Dulles Technology Drive  
Herndon, VA 20171  
USA

Tel: +1-703-484-3331  
Email: tbova@cisco.com

Ted Krivoruchka  
Cisco Systems  
13615 Dulles Technology Drive  
Herndon, VA 20171  
USA

Tel: +1-703-484-3325  
Email: tedk@cisco.com