Network Working Group
Internet-Draft
Expires: April 9, 2006

P. Lei
Cisco Systems, Inc.
P. Conrad
University of Delaware
October 6, 2005

TCP Mapping for Reliable Server Pooling Enhanced Mode
draft-ietf-rserpool-tcpmapping-03.txt

Status of this Memo

Copyright Notice

Abstract

This memo defines the shim protocol that maps the requirements of the ASAP protocol [5] to the capabilities of the TCP protocol [3]. In particular, this shim protocol adds the following capabilties that are required by ASAP, but not provided by TCP: (1) message orientation, (2) heartbeat messages, (3) multiple streams, and (4) undelivered message retrieval (if provided).

Table of Contents

1. Introduction

This memo defines the shim protocol that maps the requirements of the ASAP protocol [5] to the capabilities of the TCP protocol [3]. See [6] for details of these mapping requirements.

1.1. Brief overview of RSerPool

The RSerPool framework is designed to provide high availability for client/server applications. Servers (called "pool elements" or "PEs") are grouped into "pools". Each pool is named by a "pool handle", which is simply an octet string that identifies the pool. PEs join or leave a pool by registering and deregistering with an RSerPool nameserver (called an "ENRP server", after the ENRP protocol [8] that is used to share information among RSerPool nameservers).

Clients (called "pool users" or "PUs") that want to obtain service contact the ENRP server; the PU provides a pool handle, and the ENRP server returns a list of available servers. Associated with each server is a policy value; the PU then uses a "pool element selection policy" (such as round robin, or least used) to determine which of the PEs to contact.

The ASAP protocol [5] is used to communicate between the PU and the ENRP server, and between the PE and ENRP server, while the ENRP protocol [8] is used to communicate between and among RSerPool ENRP nameservers.

The RSerPool services framework provides two distinct channels, control and data, for delivery of RSerPool control messages and application layer data messages. Mappings provide the adaptions of the ASAP services to the specific transport protocols.

1.2. Role of the TCP Mapping Protocol

The actual application-specific communication between the PU and PE is defined by the application layer protocol, and does not use the ASAP protocol, per se. However, when failover services (such as forwarding of undelivered/unacknowledged messages) are desired by the application, all communication between a PU and a PE takes place through services provided by RSerPool, and more specifically, by the ASAP entity on the PU or PE. Furthermore, in order for the RSerPool framework to provide a consistent set of services for both application-specific data and RSerPool messages, this transport service must follow a particular model. This model includes features that are present in SCTP, but are lacking in TCP. Specifically, these are:

(1) Message Orientation. Messages must be framed within the TCP byte stream to allow for undelivered message retrieval (see below), and so that ASAP transport services can be consistent between SCTP and TCP.

(2) Heartbeat Messages. Heartbeats are needed so that a failed connection can be detected in a timely manner, even if that connection is idle. The "keepalive" mechanism provided in some TCP implementations (but not a standard feature of the protocol; see RFC1122) is not sufficient for this purpose.

(3) Retreival of Undelivered Messages. A PU can request that the ASAP layer detects when the transport layer association/connection between that PU and some PE has failed, and automatically failover to a new PE. In this case, it is necessary for the ASAP layer to determine which messages were not successfully delivered to the PE, retrieve them from the transport layer below, and resend them to the new PE. SCTP provides the RETRIEVE_UNSENT primitive (Section 10.1, item (E) of RFC2960) to enable this retrieval. TCP has no such facility.

To provide this capability over TCP, the mapping protocol provides another layer of acknowledgements on top of TCP; these acks are sent only when a message is actually delivered to the end system application (as opposed to when the message is handed up from the IP layer to the TCP layer). The sending side buffers messages until this acknowledgement is received, so that in the event of failover, these messages can be retrieved by the ASAP entity, and resent to the new PE.

This feature is optional and is NOT required. However, an appropriate error code must be returned to the upper layer if this feature is requested, but is unavailable (not implemented).

(4) Other features present in SCTP. Two other features present in SCTP are simulated by the TCP mapping layer, namely the multiple streams feature and the protocol payload identifier field (PPID). Strictly speaking these features are not necessary for RSerPool operation. However, it is trivial to provide these features in the TCP mapping layer, and providing them offers an important benefit, without significantly increasing the complexity of the protocol or the on-the-wire overhead. This is discussed further in Section 1.3

NOTE: PU-PE communication that is NOT mediated by the RSerPool framework is allowable when the application layer does not require data channel services. In this case, no mapping for application layer data is used regardless of the transport protocol used as they

will be tranported over the appropriate application-defined tranport protocol.

1.3. Consistency of Service

The main benefit of including provision for multiple streams and the PPID field in the RSerPool TCP mapping protocol is "consistency of service." By "consistency of service", we mean that the data transport service provided by RSerPool is consistent regardless of the underlying transport protocol used.

To see the benefit of consistency of service, consider an RSerPool enabled application that will be used by clients with a wide range of capabilities, e.g. wired desktop PCs at the one end, down to PDAs or cell phones at the other. On some of these platforms, SCTP may be available, while on others, only TCP will be available. If the multiple stream and PPID features are provided in the TCP mapping, then the application designer can design once to a single API, where regardless of the underlying transport used, multiple streams can be used to multiplex various kinds of messages. On PUs where SCTP is supported, an additional benefit is available, in that partial order delivery allows head of line blocking to be avoided. This is not the case for the systems where the TCP mapping is used, since all streams are mapped onto a single ordered TCP byte-stream. However, either way, the application designer doesn't have to be concerned; one can write to a single API, and the software will function correctly over either protocols, taking advantage of optimizations where available.

The alternative is to omit these features from the TCP mapping, and indicate that when the underlying protocol is TCP, that all data is implicitly sent over stream zero, and that the PPID field is ignored. However, this will encourage designers of application that run in a "mixed" environment to write to the "lowest common denominator" of functionality to avoid having to maintain special case code for each transport layer mapping. Thus, few applications will take advantage of the features offered by SCTP. The result may be that application with multiple streams will do their own multiplexing within the application layer protocol, send all data over stream zero, thus defeating the SCTP mechanism for avoiding head of line blocking. Similiarly, few applications will take advantage of the PPID field, meaning that it will be wasted space in the case where SCTP is available.

Of course, providing these extra features is not without some cost; in particular extra fields in the protocol header, and extra complexity in the implementation. Our compromise solution to the overhead of the extra fields is to include them in the data chunk definition, however to allow the service user to turn them off as an

optimization if they are not going to be used (see Section 3.3 and Section 3.2 for more details.)

As for the complexity, it turns out that because the TCP byte-stream preserves the order of each stream, providing for multiple streams is trivial; it involves only passing the stream number through just as the PPID is "passed through".

Thus, for minimal extra cost, "consistency of service" is preserved by including multiple streams and the PPID field in the TCP mapping. It is RECOMMENDED that any future mappings of RSerPool to other transport protocols SHOULD follow this model of providing "consistency of service" where possible.

2. Conventions Used In This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [2].

The terms "transmission sequence number" (TSN), "stream number", "stream identifier", "stream sequence number", and "payload protocol-id(PPID)" are used in this document with meanings analogous to their meanings in RFC2960 [4]; it is assumed that the reader is familiar with these terms as they appear in that document.

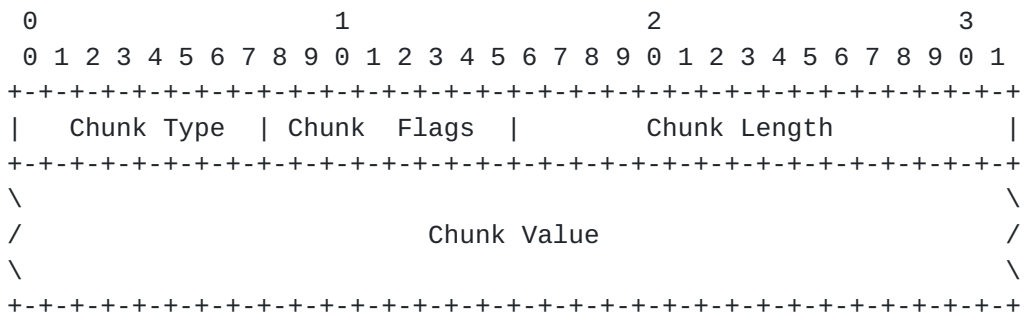
Comparisons and arithmetic on TSNs and stream sequence numbers are governed by the rules in Section 1.6 of RFC2960 [4].

3. Packet Format

In the RSerPool TCP mapping protocol, each peer transmits a series of chunks over the TCP byte stream. The format of these chunks is similar to that of the chunk format of SCTP.

3.1. Basic Chunk Format

The figure below illustrates the field format for the chunks to be transmitted in the SCTP packet. Each chunk is formatted with a Chunk Type field, a chunk-specific Flag field, a Chunk Length field, and a Value field.


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Chunk Type
Chunk Flags
Chunk Length
Chunk Value

Chunk Type: 8 bits (unsigned integer)

This field identifies the type of information contained in the Chunk Value field. It takes a value from 0 to 254. The value of 255 is reserved for future use as an extension field.

The values of Chunk Types are defined as follows:

ID Value	Chunk Type
0	- Payload Data (DATA)
1	- Initiation (INIT)
2	- reserved by IETF
3	- Acknowledgement (ACK)
4	- Heartbeat Request (HEARTBEAT)
5	- Heartbeat Acknowledgement (HEARTBEAT ACK)
6 to 255	- reserved by IETF

Chunk Flags: 8 bits

The usage of these bits depends on the chunk type as given by the Chunk Type. Unless otherwise specified, they are set to zero on transmit and are ignored on receipt.

Chunk Length: 16 bits (unsigned integer)

This value represents the size of the chunk in bytes including the Chunk Type, Chunk Flags, Chunk Length, and Chunk Value fields. Therefore, if the Chunk Value field is zero-length, the Length field will be set to 4. The Chunk Length field does not count any padding.

Chunk Value: variable length

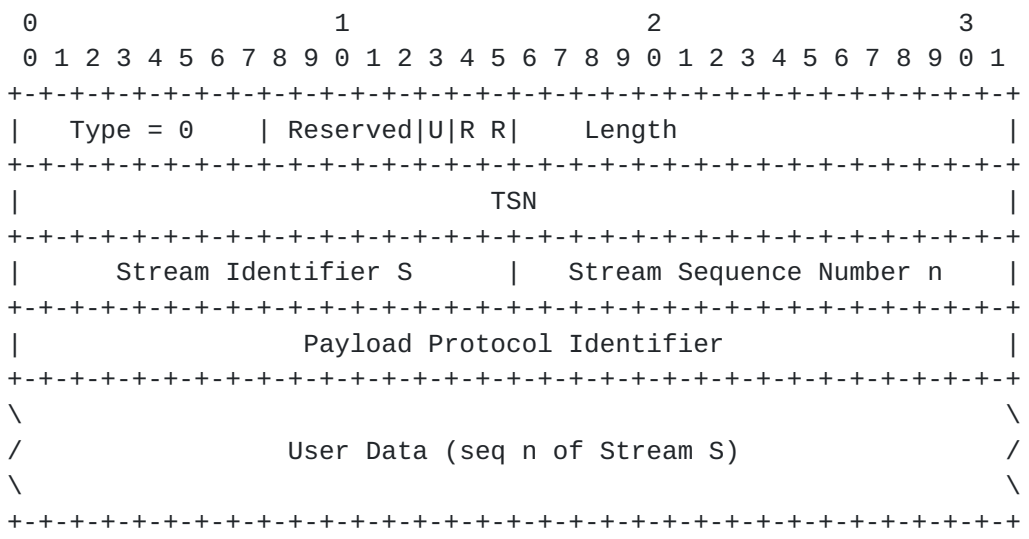
The Chunk Value field contains the actual information to be transferred in the chunk. The usage and format of this field is dependent on the Chunk Type.

The total length of a chunk (including Type, Length and Value fields) MUST be a multiple of 4 bytes. If the length of the chunk is not a multiple of 4 bytes, the sender MUST pad the chunk with all zero bytes and this padding is not included in the chunk length field. The sender should never pad with more than 3 bytes. The receiver MUST ignore the padding bytes.

3.2. DATA Chunk

The figure below illustrates the field format for the DATA chunk. Note that it is nearly identical to the format of the DATA chunk in SCTP. The following differences should be noted: (1) No B and E bits for fragmentation (2) the second, third, and fourth 32-bit words are all optional, and can be supressed (independently) through flags in the INIT message (as explained further below.)

All ASAP protocol messages and application-layer data (when sent through the RSerPool framework data channel) MUST be carried in DATA chunks.



Reserved, and bits marked "R": 5 bits and 2 bits, respectively

The sender SHOULD set all '0's and the receiver SHOULD ignore.

U bit: 1 bit

The (U)nordered bit, if set to '1', indicates that this is an unordered DATA chunk, and there is no Stream Sequence Number assigned to this DATA chunk. Therefore, the receiver MUST ignore the Stream Sequence Number field.

Length: 16 bits (unsigned integer)

This field indicates the length of the DATA chunk in bytes from the beginning of the type field to the end of the user data field excluding any padding.

TSN : 32 bits (unsigned integer)

This value represents the TSN for this DATA chunk. The valid range of TSN is from 0 to 4294967295 (2**32 - 1). TSN wraps back to 0 after reaching 4294967295. In the TCP mapping protocol (unlike in SCTP), this field MUST start at 0 with each new connection. Also note that this field is redundant; TSN of each data chunk is implicit by its position in the TCP byte stream. It does server the purpose of providing additional robustness against errors in a sender or receiver protocol implementation, however if desired, it MAY be supressed for the lifetime of the TCP connection via a flag in the INIT chunk.

Stream Identifier S: 16 bits (unsigned integer)

Identifies the stream to which the following user data belongs. if desired, the entire 32-bit word containing Stream Identifier and Stream Sequence Number MAY be supressed to save bandwidth; in this case, all data chunks MUST be interpreted by the receiver to be part of stream 0, and should be delivered to the end user as such.

Stream Sequence Number n: 16 bits (unsigned integer)

This value represents the stream sequence number of the following user data within the stream S. Valid range is 0 to 65535. Note that this field, like TSN, is redundant, but it provides extra robustness.

Note that unlike SCTP, there is no concept of fragmentation in the TCP mapping protocol.

Payload Protocol Identifier: 32 bits (unsigned integer)

This value represents an application (or upper layer) specified protocol identifier. This value is passed to the TCP mapping layer from the upper layer and sent to its peer. This identifier is not used by the TCP mapping layer but can be used by certain network entities as well as the peer application to identify the type of information being carried in this DATA chunk.

The IANA assigned SCTP protocol payload identifier value for ASAP (11) MUST be used for the transport of ASAP messages (eg. RSerPool control channel messages).

The value 0 indicates no application identifier is specified by the upper layer for this payload data. If the application has no use for this field, it can be supressed for all data chunks

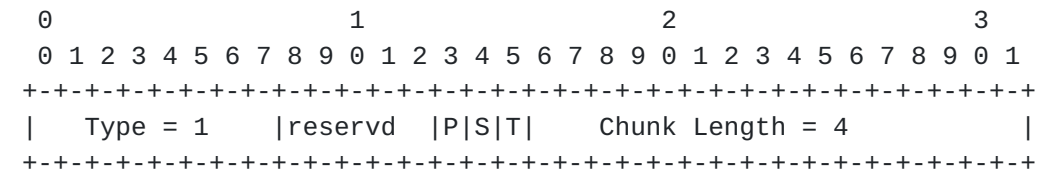
over the lifetime of the TCP connection via a flag in the INIT message.

User Data: variable length

This is the payload user data. The implementation MUST pad the end of the data to a 4 byte boundary with all-zero bytes. Any padding MUST NOT be included in the length field. A sender MUST never add more than 3 bytes of padding.

3.3. INIT Chunk

The figure below illustrates the field format for the INIT chunk. The INIT chunk is transmitted only once at the beginning of the TCP connection. The purpose of the INIT chunk is to transmit flags indicating which fields will be enabled/disabled in all subsequent data chunks over the lifetime of the TCP connection.



Chunk Type: 8 bits (unsigned integer)

0x01, indicating INIT chunk

Chunk Flags: 8 bits

The five high order bits are reserved; the sender SHOULD set them to zero, and the receiver SHOULD ignore them.

The three low order bits are to be intepreted as follows;

0x01: The TSN value is omitted in subsequent DATA chunks. Additionally, this indicates that the receiver of the INIT chunk SHOULD omit the TSN on ACK chunks.
0x02: The Stream Number/Stream Seq Number is omitted in subsequent DATA chunks
0x04: The PPID value is omitted in subsequent DATA chunks

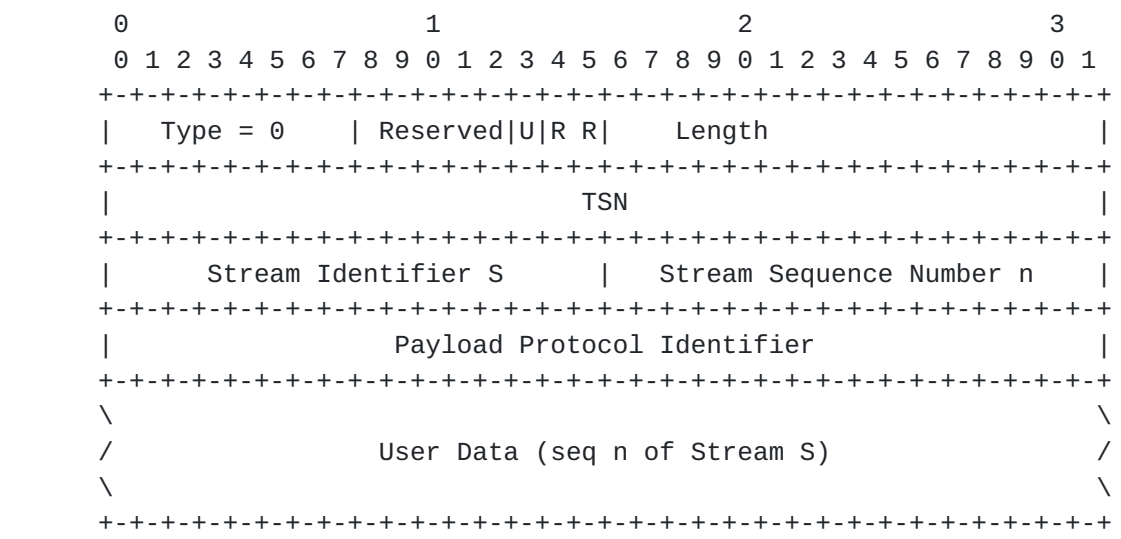
Chunk Length: 16 bits (unsigned integer)

MUST be set to 0x0004.

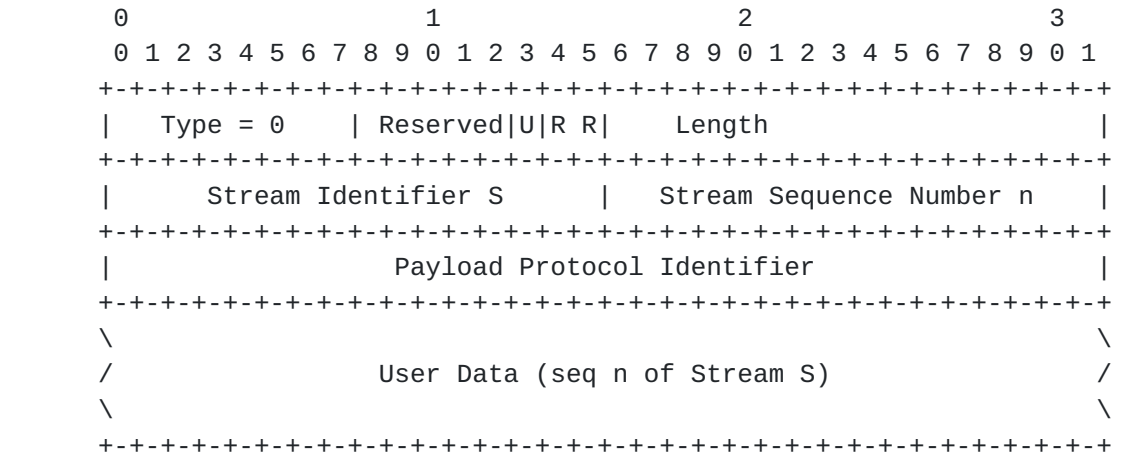
NOTE: Unlike SCTP, there is no INIT-ACK defined in the RSerPool TCP mapping protocol.

The following examples illustrate the use of the flags in the INIT message.

Example 1: Flags in the INIT are 0x00. Data chunk format is:

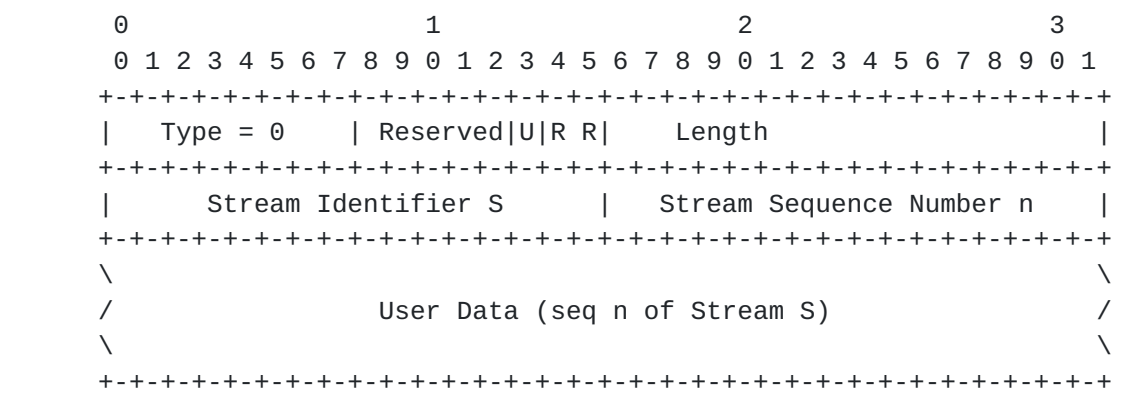


Example 2: Flags in the INIT are 0x01. Data chunk format is:



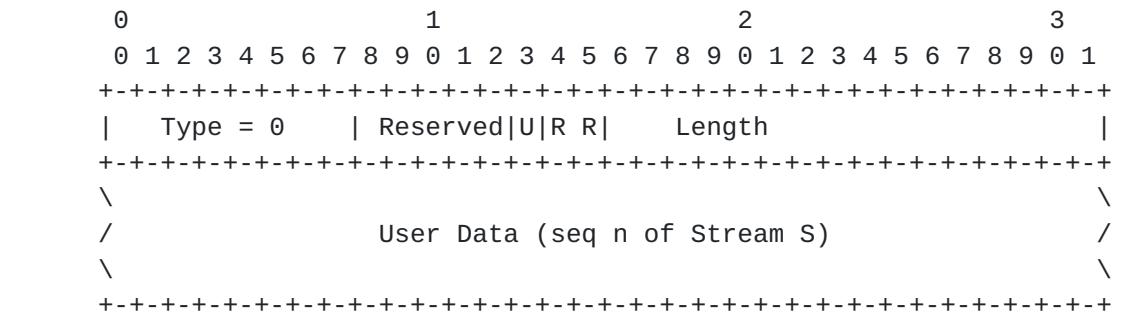
In this example, TSN is implicit from the order in which the chunk appears in the TCP byte stream.

Example 3: Flags in the INIT are 0x05. Data chunk format is:



In this example, TSN is implicit from the order in which the chunk appears in the TCP byte stream, and PPID is treated as zero.

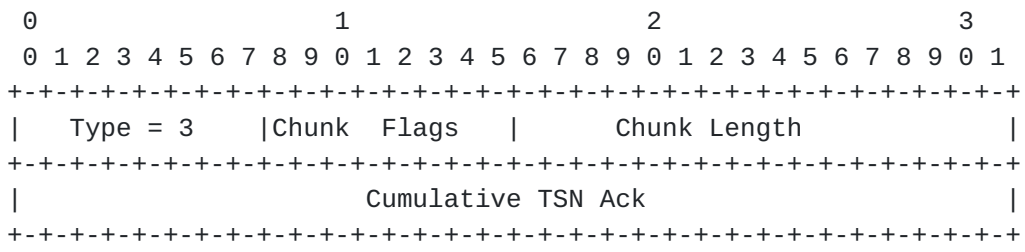
Example 4: Flags in the INIT are 0x07. Data chunk format is:



In this example, TSN and Stream Sequence Number are implicit from the order in which the chunk appears in the TCP byte stream, and Stream Identifier and PPID are always considered zero.

3.4. ACK Chunk

The figure below illustrates the field format for the ACK chunk. An RSerPool TCP mapping layer entity MUST transmit exactly one corresponding ACK chunk over the TCP connection immediately after delivering each DATA chunk to the upper layer. Each such ACK chunk SHOULD contain the TSN corresponding to the DATA chunk that was delivered, unless inclusion of the TSN has been supressed by receipt of the 0x01 flag value in a previous INIT chunk from the data sender to the data receiver.



Chunk Flags: 8 bits

Set to all zeros on transmit and ignored on receipt.

Cumulative TSN Ack: 32 bits (unsigned integer)

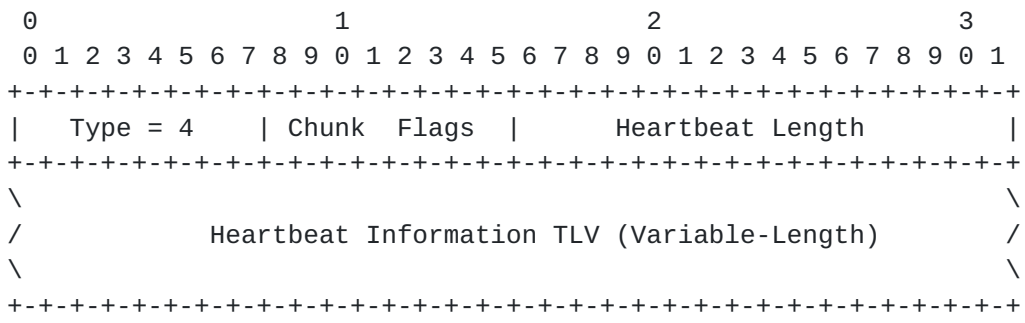
This field is OPTIONAL, and SHOULD NOT be included if the side sending the ack previously received a value of 0x01 in the INIT from the peer. The receiver of an ACK chunk can determine whether the TSN field is present by checking whether the length of the ACK chunk is 4 bytes or 8 bytes.

This field contains the TSN of the last DATA chunk delivered to the upper layer protocol. Note that this value is implicit from the position of the ACK chunk in the TCP byte stream.

3.5. HEARTBEAT Chunk

The figure below illustrates the field format for the HEARTBEAT chunk.

The parameter field contains the Heartbeat Information which is a variable length opaque data structure understood only by the sender.



Chunk Flags: 8 bits

Set to zero on transmit and ignored on receipt.

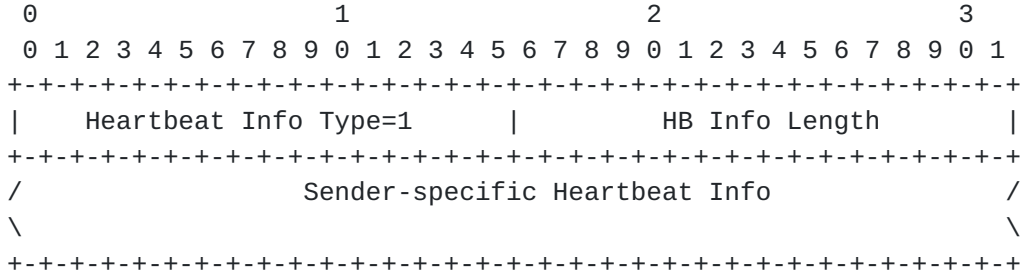
Heartbeat Length: 16 bits (unsigned integer)

Set to the size of the chunk in bytes, including the chunk header and the Heartbeat Information field.

Heartbeat Information: variable length

Defined as a variable-length parameter using the format described in Section 3.2.1, i.e.:

Variable Parameters	Status	Type Value
Heartbeat Info	Mandatory	1

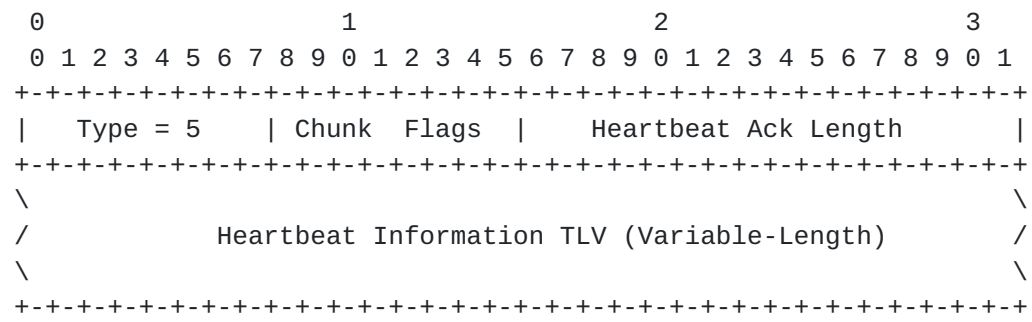


3.6. HEARTBEAT ACK Chunk

The figure below illustrates the field format for the HEARTBEAT ACK chunk. The RSerPool TCP mapping layer MUST transmit exactly one HEARTBEAT ACK chunk in response to each HEARTBEAT chunk received. The Heartbeat Information parameter received in the HEARTBEAT chunk

MUST be included in the HEARTBEAT ACK chunk.

The parameter field contains a variable length opaque data structure which was received in the HEARTBEAT.



Chunk Flags: 8 bits

Set to zero on transmit and ignored on receipt.

Heartbeat Ack Length: 16 bits (unsigned integer)

Set to the size of the chunk in bytes, including the chunk header and the Heartbeat Information field.

Heartbeat Information: variable length

This field MUST contain the Heartbeat Information parameter of the Heartbeat Request to which this Heartbeat Acknowledgement is responding.

Variable Parameters	Status	Type Value
Heartbeat Info	Mandatory	1

4. Protocol Operations

[TBD: In this section describe the basic operation of the protocol. Most of this is already pretty much spelled out in the descriptions of the packets, but a few details need to be ironed out, mainly how and under what conditions the TCP mapping layer decides that a failure occured. Perhaps the upper layer needs to be able to specify a timeout value for data, and a heartbeat interval? Are there any other details that need to specified in this section?]

5. Security Considerations

There are no known additional security considerations over what is already present for TCP.

6. IANA Considerations

[Open Issue TBD: Will there be an enumeration of the various transport layer mappings that must be registered with IANA?]

7. Acknowledgements

8. References

[1] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

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

[3] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[4] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, October 2000.

[5] Stewart, R., "Aggregate Server Access Protocol (ASAP)", draft-ietf-rserpool-asap-12 (work in progress), July 2005.

[6] Lei, P. and P. Conrad, "Services Provided By Reliable Server Pooling", draft-ietf-rserpool-service-02 (work in progress), October 2005.

[7] Tuexen, M., "Architecture for Reliable Server Pooling", draft-ietf-rserpool-arch-10 (work in progress), July 2005.

[8] Stewart, R., "Endpoint Handlespace Redundancy Protocol (ENRP)", draft-ietf-rserpool-enrp-12 (work in progress), July 2005.

Authors' Addresses

Peter Lei
Cisco Systems, Inc.
8735 W Higgins Rd, Suite 300
Chicago, IL 60631
US

Phone: +1 773 695 8201
Email: peterlei@cisco.com

Phillip T. Conrad
University of Delaware
Dept. of Computer and Information Sciences
103 Smith Hall
Newark, DE 19716
US

Phone: +1 302 831 8622
Email: conrad@acm.org
URI: http://udel.edu/~pconrad

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.