Network Working Group
INTERNET-DRAFT

R. R. Stewart
Q. Xie
Motorola
K. Morneault
C. Sharp
Cisco
H. J. Schwarzbauer
Siemens
T. Taylor
Nortel Networks
I. Rytina
Ericsson
M. Kalla
Telcordia
L. Zhang
UCLA
V. Paxson
ACIRI

expires in six months

October 15,1999

Simple Control Transmission Protocol
<draft-ietf-sigtran-sctp-01.txt>

Status of This Memo

Abstract

This document describes the Simple Control Transmission Protocol (SCTP). SCTP is designed to transport PSTN signalling messages over IP networks, but is capable of broader application.

SCTP is an application-level datagram transfer protocol operating on top of an unreliable datagram service such as UDP. It offers the following services to its users:

- -- acknowledged error-free non-duplicated transfer of user data
- -- application-level segmentation to conform to discovered MTU size
- -- sequenced delivery of user datagrams within multiple streams, with an option for order-of-arrival delivery of individual datagrams
- -- optional multiplexing of user datagrams into SCTP datagrams, subject to MTU size restrictions
- -- enhanced reliability through support of multi-homing at either or both ends of the association.

The design of SCTP includes appropriate congestion avoidance behaviour and resistance to flooding and masquerade attacks.

TABLE OF CONTENTS

1. Introduction

This section explains the reasoning behind the development of the Simple Control Transmission Protocol (SCTP), the services it offers, and the basic concepts needed to understand the detailed description of the protocol.

1.1 Motivation

TCP [RFC 793, "Transmission Control Protocol", Jon Postel ed., September 1981] has performed immense service as the primary means of reliable data transfer in IP networks. However, an increasing number of recent applications have found TCP too limiting, and have incorporated their own reliable data transfer protocol on top of UDP [RFC 768, "User Datagram Protocol", Jon Postel, August 1980]. The limitations which users have wished to bypass relate both to the intrinsic nature of TCP and to its typical implementation.

Intrinsic limitations:

-- TCP provides both reliable data transfer and strict order-of-arrival delivery of data. Some applications need reliable transfer without sequence maintenance, while others would be satisfied with partial ordering of the data. In both of these cases the head-of-line blocking offered by TCP causes unnecessary delay.

-- The stream-oriented nature of TCP is often an inconvenience. Applications must add their own record marking to delineate their messages, and must make explicit use of the push facility to ensure that a complete message is transferred in a

reasonable time.

-- The limited scope of TCP sockets complicates the task of providing highly-available data transfer capability using multi-homed hosts.

Limitations due to implementation:

-- TCP is generally implemented at the operating system level. Kernel limitations may constrain the maximum allowable number of simultaneous TCP connections to a number far below that required for certain applications.

-- TCP implementations do not generally allow the application to control the timers which determine how quickly a connection failure is discovered. Some applications are more critically dependent than others on timely initiation of recovery from such failures.

Transport of PSTN signalling across the IP network is an application for which all of these limitations of TCP are relevant. While this application directly motivated the development of SCTP, other applications may find SCTP a good match to their requirements.

1.2 Architectural View of SCTP

SCTP is viewed as a layer between the SCTP user application ("SCTP user" for short) and an unreliable end-to-end datagram service such as UDP. The basic service offered by SCTP is the reliable transfer of user datagrams between peer SCTP users. It performs this service within the context of an association between two SCTP Hosts. Chapter 9 of this document sketches the API which should exist at the boundary between the SCTP and the SCTP user layers.

SCTP is connection-oriented in nature, but the SCTP association is a broader concept than the TCP connection. SCTP provides the means for each SCTP endpoint to provide the other during association startup with a list of transport addresses (e.g. address/UDP port combinations) through which that endpoint can be reached and from which it will originate messages. The association spans transfers over all of the possible source/destination combinations which may be generated from the two endpoint lists.

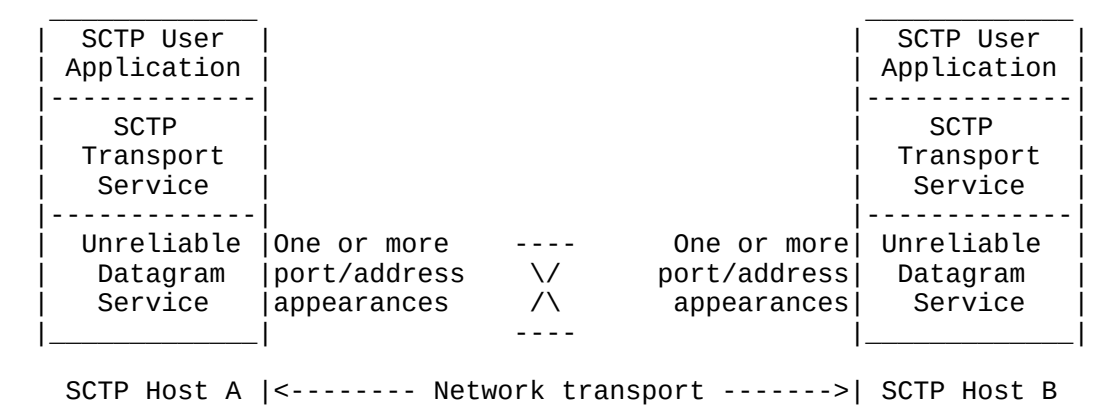


Figure 1: An SCTP Association

It is possible to have multiple associations active between the same pair of SCTP Hosts.

1.3 Functional View of SCTP

The SCTP transport service can be decomposed into a number of functions. These are depicted in Figure 2 and explained in the remainder of this section.

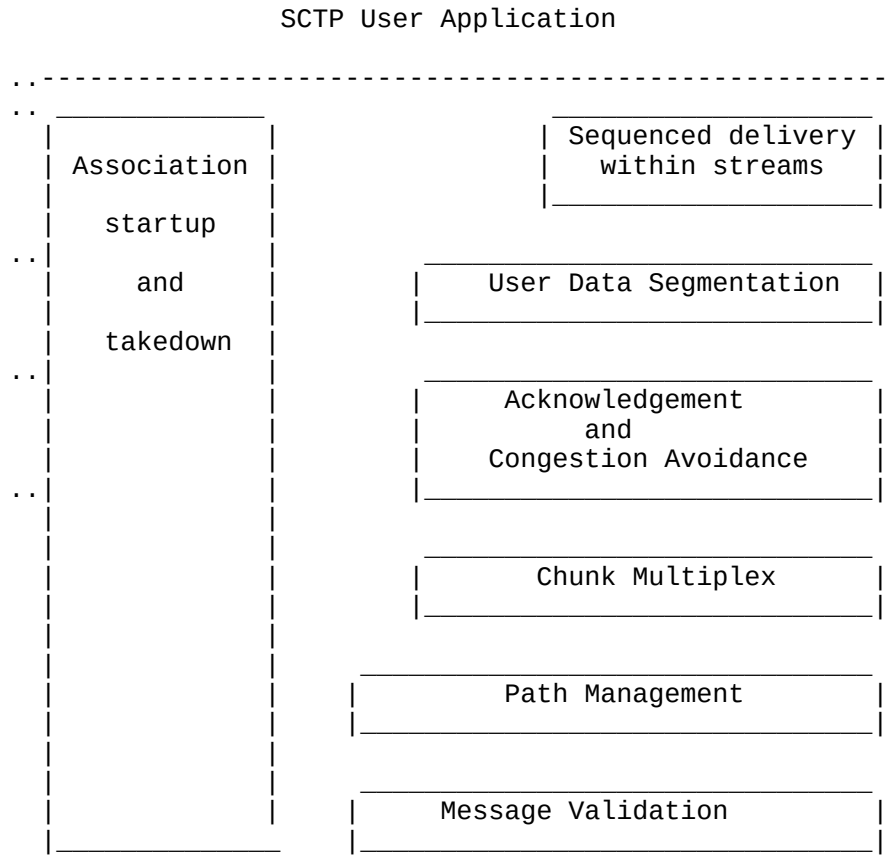


Figure 2: Functional View of the SCTP Transport Service

1.3.1 Association Startup and Takedown

An association is initiated by a request from the SCTP user (see the description of the ASSOCIATE primitive in Chapter 9). The startup sequence is described in chapter 4 of this document. It is designed to be resistant to flooding and masquerade attacks.

SCTP provides for graceful takedown of an active association on request from the SCTP user. See the description of the TERMINATE primitive in chapter 9. SCTP also allows ungraceful takedown, either on request from the user (ABORT primitive) or as a result of an error condition detected within the SCTP layer. Chapter 8 describes both the graceful and the ungraceful takedown procedures.

1.3.2 Sequenced Delivery within Streams

The term "stream" is used in SCTP to refer to a sequence of datagrams. This is in contrast to its usage in TCP, where it refers to a sequence of bytes.

The SCTP user can specify at association startup time the number of streams to be supported by the association. This number is negotiated with the remote end (see section 4.1.1). User datagrams are associated with stream numbers (SEND, RECEIVE primitives, Chapter 9). Internally, SCTP assigns a stream sequence number to each datagram passed to it by the SCTP user. On the receiving side, SCTP ensures that datagrams are delivered to the SCTP user in sequence within a given stream. However, while one stream may be blocked waiting for the next in-sequence user datagram, delivery from other streams may proceed.

SCTP provides a mechanism for bypassing the sequenced delivery service. Usr datagrams sent using this mechanism are delivered to the SCTP user as soon as they are received.

1.3.3 User Data Segmentation

SCTP can segment user datagrams to ensure that the SCTP datagram passed to the lower layer conforms to the path MTU. Segments are reassembled into complete datagrams before being passed to the SCTP user.

1.3.4 Acknowledgement and Congestion Avoidance

SCTP assigns a Transmission Sequence Number (TSN) to each user data segment or unsegmented datagram. The TSN is independent of any sequence number assigned at the stream level. The receiving end acknowledges all TSNs received, even if there are gaps in the sequence. In this way, reliable delivery is kept functionally separate from sequenced delivery.

The Acknowledgement and Congestion Avoidance function is responsible for message retransmission when timely acknowledgement has not been received. Message retransmission is conditioned by congestion avoidance procedures similar to those used for TCP.

See Chapters 5 and 6 for a detailed description of the protocol procedures associated with this function.

1.3.5 Chunk Multiplex

As described in Chapter 2, the SCTP datagram as delivered to the lower layer consists of a common header followed by one or more chunks. Each chunk may contain either user data or SCTP control information. The SCTP user has the option to request "bundling", or multiplexing of more than one user datagram into a single SCTP datagram. The chunk multiplex function of SCTP is responsible for assembly of the complete SCTP datagram and its disassembly at the receiving end.

1.3.6 Path Management

The sending SCTP user is able to manipulate the set of transport addresses used as destinations for SCTP datagrams, through the primitives described in Chapter 9. The SCTP path management function chooses the destination transport address for each outgoing SCTP datagram based on the SCTP user’s instructions and the currently perceived reachability status of the eligible destination set.

The path management function monitors reachability through heartbeat messages where other message traffic is inadequate to provide this information, and advises the SCTP user when reachability of any far-end transport address changes. The path management function is also responsible for reporting the eligible set of local transport addresses to the far end during association startup, and for reporting the transport addresses returned from the far end to the SCTP user.

At association start-up, a primary destination transport address is defined for each SCTP endpoint, and is used for normal sending of SCTP datagrams.

On the receiving end, the path management is responsible for verifying the existence of a valid SCTP association to which the inbound SCTP datagram belongs before passing it for further processing.

1.3.7 Message Validation

The common SCTP header includes a validation tag and an optional CRC field. A validation tag value is chosen by each end of the association during association startup. Messages received without the validation tag value expected by the receiver are discarded, as a protection against blind masquerade attacks and against stale datagrams from a previous association.

The CRC may optionally be set by the sender, to provide additional protection against data corruption in the network beyond that provided by lower layers (e.g. the UDP checksum).

1.4 Recapitulation of Key Terms

The language used to describe SCTP has been introduced in the previous sections. This section provides a consolidated list of the key terms and their definitions.

- SCTP user application (SCTP user): The logical higher-layer application entity which uses the services of SCTP, also called the Upper-layer Protocol (ULP).

- User datagram (user message): the unit of data delivery across the interface between SCTP and its user.

- User data: the content of user datagrams.

- SCTP datagram: the unit of data delivery across the interface between SCTP and the unreliable datagram service (e.g. UDP) which it is using. An SCTP datagram includes the common SCTP header, possible SCTP control chunks, and user data encapsulated within SCTP DATA chunks.

- SCTP host: a logical unit within which SCTP is running.

- Transport address: an address which serves as a source or destination for the unreliable datagram transport service used by SCTP. In IP networks, a transport address is defined by the combination of an IP address and a UDP port number.

- SCTP endpoint: a logical entity, comprising a set of eligible transport addresses at an SCTP host, which SCTP datagrams will be sent to and received from. Note, a transport address can only be included in one unique SCTP endpoint, i.e., it is NOT allowed to have the same SCTP transport address appear in more than one endpoints.

- SCTP association: a protocol relationship between SCTP endpoints, comprising the two SCTP endpoints and protocol state information including verification tags and the currently active set of Transmission Sequence Numbers (TSNs), etc.

- Chunk: a unit of information within an SCTP datagram, consisting of a chunk header and chunk-specific content.

- Transmission Sequence Number (TSN): a 32-bit sequence number used internally by SCTP. One TSN is attached to each chunk containing user data to permit the receiving SCTP endpoint to acknowledge its receipt and detect duplicate deliveries.

- Stream: a uni-directional logical channel established from one to another associated SCTP endpoints, within which all user datagrams are delivered in sequence except for those submitted to the unordered delivery service.

  Note: The relationship between stream numbers in opposite directions is strictly a matter of how the applications use them. It is the responsiblity of the SCTP user to create these correlations if they are so desired.

- Stream sequence number: a 16-bit sequence number used internally by SCTP to assure sequenced delivery of the user datagrams within a given stream. One stream sequence number is attached to each user datagram.

- Bundling: an optional multiplexing operation, whereby more than one user datagram may be carried in the same SCTP datagram. Each user datagram occupies its own DATA chunk.

- o Outstanding TSN (at an SCTP endpoint): a TSN (and the associated DATA chunk) which have been sent by the endpoint but for which it has not yet received an acknowledgement.

- o Unacknowledged TSN (at an SCTP endpoint): a TSN (and the associated DATA chunk) which have been received by the endpoint but for which an acknowledgement has not yet been sent.

- o Receiver Window (rwnd): The most recently advertised receiver window, in number of octets. This gives an indication of the space available in the receiver's inbound buffer.

- o Congestion Window (cwnd): An SCTP variable that limits the data, in number of octets, a sender can send into the network before receiving an acknowledgment on a particular destination Transport address.

- o Slow Start Threshold (ssthresh): An SCTP variable. This is the threshold which the endpoint will use to determine whether to perform slow start or congestion avoidance on a particular destination transport address. Ssthresh is in number of octets.

- o Transmission Control Block (TCB): an internal data structure created by an SCTP endpoint for each of its existing SCTP associations to other SCTP endpoints. TCB contains all the status and operational information for the endpoint to maintain and manage the corresponding association.

1.5. Abbreviations

MD5 - MD5 Message-Digest Algorithm [4]

NAT - Network Address Translation

RTO - Retransmission Time-out

RTT - Round-trip Time

RTTVAR - Round-trip Time Variation

SCTP - Simple Control Transmission Protocol

SRTT - Smoothed RTT

TCB - Transmission Control Block

TLV - Type-Length-Value Coding Format

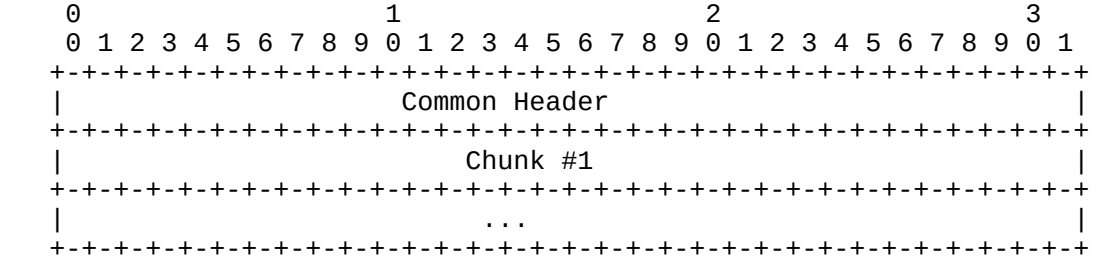
TSN - Transport Sequence Number

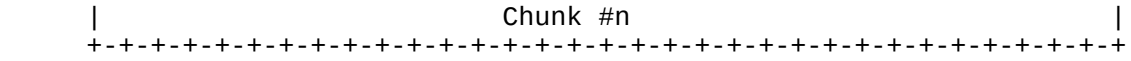
ULP - Upper-layer Protocol

2. SCTP Datagram Format

An SCTP datagram is composed of a common header and chunks. A chunk contains either control information or user data.

The SCTP datagram format is shown below:



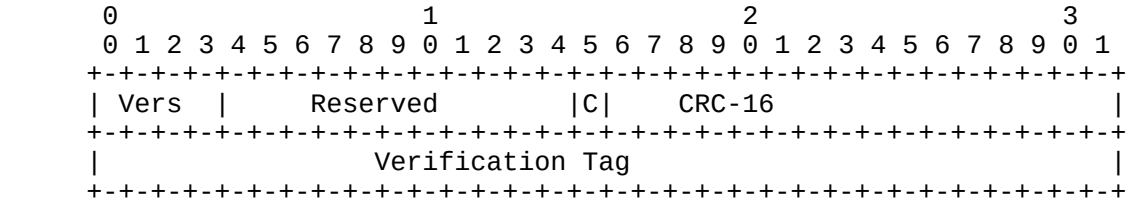


Multiple chunks can be multiplexed into one UDP SCTP datagram up to the MTU size except for the INIT, INIT ACK, and SHUTDOWN ACK chunks. These chunks MUST not be multiplexed with any other chunk in a datagram. See Section 5.10 for more details on chunk multiplexing.

If an user data message doesn't fit into one SCTP datagram it can be segmented into multiple chunks using the procedure defined in Section 5.9.

2.1 SCTP Common Header Field Descriptions

SCTP Common Header Format



Version: 4 bits, u_int

This field represents the version number of the SCTP protocol, and MUST be set to '0011'.

Verification Tag: 32 bit u_int

The receiver of this datagram uses the Verification Tag to identify the association. On transmit, the value of this Verification Tag MUST be set to the value of the Initiate Tag received from the peer endpoint during the association initialization.

For datagrams carrying the INIT chunk, the transmitter MUST set the Verification Tag to all 0's. If the receiver receives a datagram with an all-zeros Verification Tag field, it checks the Chunk ID immediately following the common header. If the Chunk Type is not INIT or SHUTDOWN ACK, the receiver MUST drop the datagram.

For datagrams carrying the SHUTDOWN-ACK chunk, the transmitter SHOULD set the Verification Tag to the Initiate Tag received from the peer endpoint during the association initialization, if known. Otherwise the Verification Tag MUST be set to all 0's.

Reserved:

Reserved bits MUST be set to 0 on transmit and should be ignored on reception.

C: 1 bit (Octet 2, Bit 8)

When the C-bit is set to 1, the CRC-16 field contains the CRC-16 (defined below).

When the C-bit is set to 0, the CRC-16 field is not used and MUST be set to 0.

CRC-16: (Octets 3 & 4)

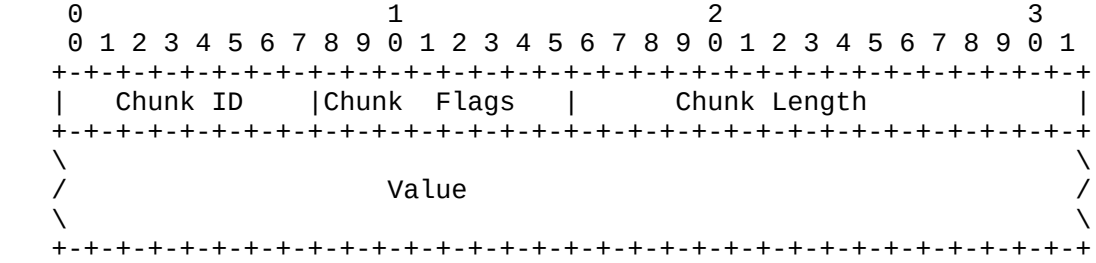
When the C Bit is set to 1, this field MUST contain a CRC-16. The CRC-16 used is defined in Section 4.2 of ITU Recommendation Q.703 [2].

Section 5.8 defines the use of CRC-16 in SCTP.

IMPLEMENTATION NOTE: When the C bit is set to 0, an implementation MAY use the fixed value 0x30000000 as a sanity check on an inbound datagram. If the first long integer is not the fixed value the datagram MAY be discarded with no further processing.

2.2 Chunk Field Descriptions

The figure below illustrates the field format for the chunks to be transmitted in the SCTP datagram. Each chunk is formatted with a Chunk ID field, a Chunk-specific flag field, a Length field and a value field.



Chunk ID: 8 bits, u_int

This field identifies the type of information contained in the chunk value field. It takes a value from 0x00 to 0xFF. The value of 0xFE is reserved for vendor-specific extensions. The value of 0xFF is reserved for future use as an extension field. Procedures for extending this field by vendors are defined in Section 2.4.

The values of Chunk ID are defined as follows:

ID Value	Chunk Type
00000000	- Payload Data (DATA)
00000001	- Initiation (INIT)
00000010	- Initiation Acknowledgment (INIT ACK)
00000011	- Selective Acknowledgment (SACK)
00000100	- Heartbeat Request (HEARTBEAT)
00000101	- Heartbeat Acknowledgment (HEARTBEAT ACK)
00000110	- Abort (ABORT)
00000111	- Shutdown (SHUTDOWN)
00001000	- Shutdown Acknowledgment (SHUTDOWN ACK)
00001001	- Operation Error (ERROR)
00001010	- Responder Cookie (COOKIE)
00001011	- Cookie Acknowledgement (COOKIE ACK)
00001100 to 11111101	- reserved for future IETF usage
11111110	- Vendor-specific chunk extensions
11111111	- IETF-defined Chunk Extension

Chunk Flags: 8 bits

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

Chunk Length: 16 bits (u_int)

This value represents the size of the chunk in octets including the Chunk ID, Flags, Length and Value fields. Therefore, if the Value field is zero-length, the Length field will be set to 0x0004. The Length field does not include any padding.

Chunk Value: variable length

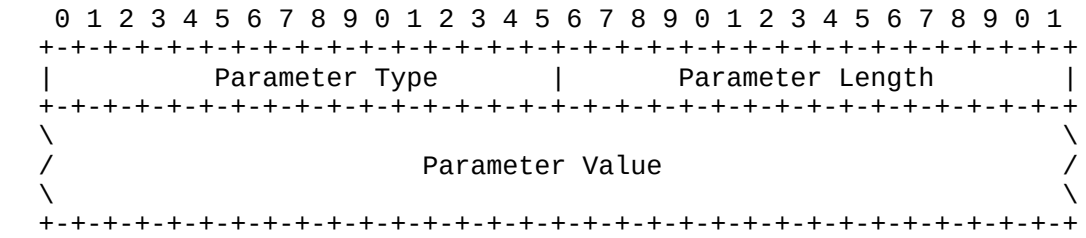
The Chunk Value field contains the actual information to be transferred in the chunk. The usage and format of this field is dependent on the Chunk ID. The Chunk Value field MUST be aligned on 32-bit boundaries. If the length of the chunk does not

align on 32-bit boundaries, it is padded at the end with all zero octets.

SCTP defined chunks are described in detail in Section 2.3. The guidelines for Vendor-Specific chunk extensions are discussed in Section 2.4. And the guidelines for IETF-defined chunk extensions can be found in Section 11.1 of this document.

2.2.1 Optional/Variable-length Parameter Format

The optional and variable-length parameters contained in a chunk are defined in a Type-Length-Value format as shown below.



Parameter Type: 16 bit u_int

The Type field is a 16 bit identifier of the type of parameter. It takes a value of 0x0000 to 0xFFFF.

The value of 0xFFFE is reserved for vendor-specific extensions if the specific chunk allows such extensions. The value of 0xFFFF is reserved for IETF-defined extensions. Values other than those defined in specific SCTP chunk description are reserved for use by IETF.

Parameter Length: 16 bit u_int

The Length field contains the size of the parameter in octets, including the Type, Length, and Value fields. Thus, a parameter with a zero-length Value field would have a Length field of 0x0004. The Length does not include any padding octets.

Parameter Value: variable-length.

The Value is dependent on the value of the Type field. The value field MUST be aligned on 32-bit boundaries. If the value field is not aligned on 32-bit boundaries it is padded at the end with all zero octets. The value field must be an integer number of octets.

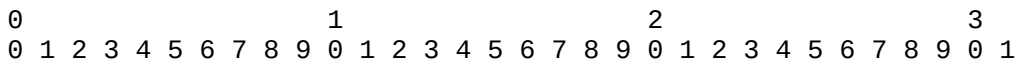
The actual SCTP parameters are defined in the specific SCTP chunk section. The guidelines for vendor-specific parameter extensions are discussed in Section 2.2.2. And the rules for IETF-defined parameter extensions are defined in Section 11.2.

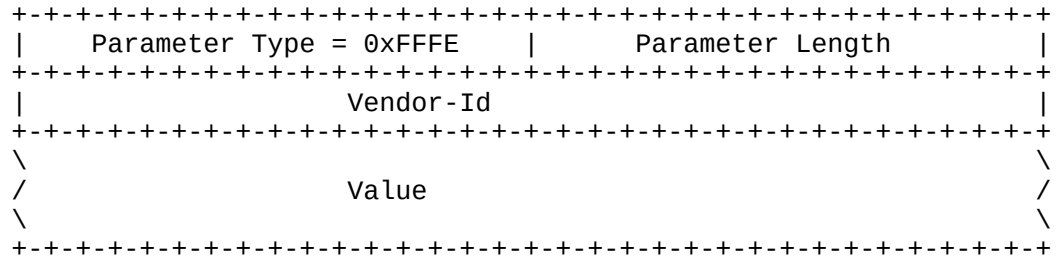
2.2.2 Vendor-Specific Extension Parameter Format

This is to allow vendors to support their own extended parameters not defined by the IETF. It MUST not affect the operation of SCTP.

Endpoints not equipped to interpret the vendor-specific information sent by a remote endpoint MUST ignore it (although it may be reported). Endpoints that do not receive desired vendor-specific information SHOULD make an attempt to operate without it, although they may do so (and report they are doing so) in a degraded mode.

A summary of the Vendor-Specific Extension format is shown below. The fields are transmitted from left to right.





Type: 16 bit u_int

0xFFFE for all Vendor-Specific parameters.

Length: 16 bit u_int

Indicate the size of the parameter in octets, including the Type, Length, Vendor-Id, and Value fields.

Vendor-Id: 32 bit u_int

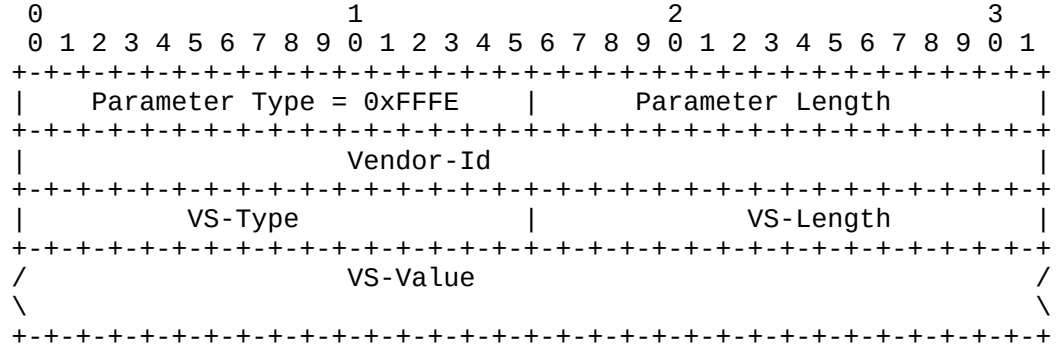
The high-order octet is 0 and the low-order 3 octets are the SMI Network Management Private Enterprise Code of the Vendor in network byte order, as defined in the Assigned Numbers (RFC 1700).

Value: variable length

The Value field is one or more octets. The actual format of the information is site or application specific, and a robust implementation SHOULD support the field as undistinguished octets.

The codification of the range of allowed usage of this field is outside the scope of this specification.

It SHOULD be encoded as a sequence of vendor type / vendor length / value fields, as follows. The parameter field is dependent on the vendor's definition of that attribute. An example encoding of the Vendor-Specific attribute using this method follows:



VS-Type: 16 bit u_int

This field identifies the parameter included in the VS-Value field. It is assigned by the vendor.

VS-Length: 16 bit u_int

This field is the length of the vendor-specific parameter and Includes the VS-Type, VS-Length and VS-Value (if included) fields.

VS-Value: Variable Length

This field contains the parameter identified by the VS-Type field. It's meaning is identified by the vendor.

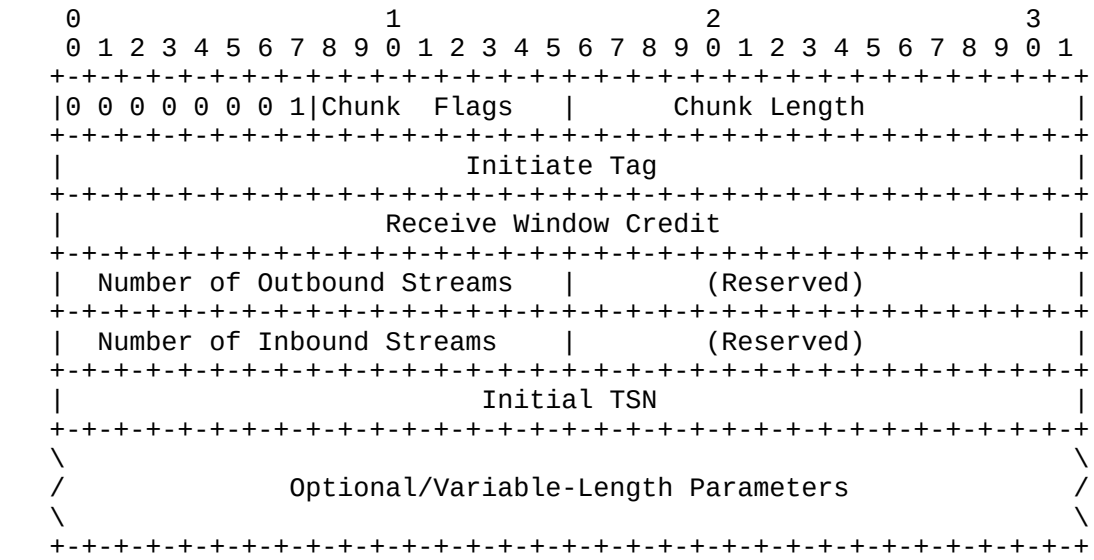
2.3 SCTP Chunk Definitions

This section defines the format of the different chunk types.

Note: integers in the chunk MUST be transmitted in network octet-order.

2.3.1 Initiation (INIT) (00000001)

This chunk is used to initiate a SCTP association between two endpoints. The format of the INIT message is shown below:



The INIT chunk contains the following parameters. Unless otherwise noted, each parameter MUST only be included once in the INIT chunk.

Fixed Parameters	Status
Initiate Tag	Mandatory
Receiver Window Credit	Mandatory
Number of Outbound Streams	Mandatory
Number of Inbound Streams	Mandatory
Initial TSN	Mandatory

Variable Parameters	Status	Type Value
IPv4 Address/Port (Note 1)	Optional	0x0005
IPv6 Address/Port (Note 1)	Optional	0x0006
Cookie Preservative	Optional	0x0009

Note 1: The INIT chunks may contain multiple addresses that may be IPv4 and/or IPv6 in any combination.

The sequence of parameters within an INIT may be processed in any order.

Vendor-specific parameters are allowed in INIT. However, they MUST be appended to the end of the above INIT chunk. The format of the vendor-specific parameters MUST follow the Type-Length-value format as defined in Section 2.2.2. In case an endpoint does not support the vendor-specific data received, it MUST ignore the additional fields.

Initiate Tag: 32 bit u_int

The receiver of the INIT (the responding end) records the value of the Initiate Tag parameter. This value MUST be placed into the Verification Tag field of every SCTP datagram that the responding end transmits within this association.

The valid range for Initiate Tag is from 0x1 to 0xffffffff. See Section 4.3.1 for more on selection of the tag value.

If the value of the Initiate Tag in a received INIT chunk is found to be 0x0, the receiver MUST treat it as an error and silently discard the datagram (or send Abort with tag=0??!).

Receive Window Credit (rwnd): 32 bit u_int

This field defines the maximum number of octets of outbound data the receiver of the INIT is allowed to have outstanding (i.e. sent and not acknowledged).

Number of Outbound Streams (OS): 16 bit u_int

Defines the number of outbound streams the sender of this INIT chunk wishes to create in this association. The value of 0 MUST NOT be used.

Number of Inbound Streams (MIS) : 16 bit u_int

Defines the maximum number of streams the sender of this INIT chunk allows the peer end to create in this association. The value 0 MUST NOT be used.

Initial TSN (I-TSN) : 32 bit u_int

Defines the initial TSN that the sender will use. This field MAY be set to the value of the Initiate Tag field.

The Reserved fields must be set to all 0 by the sender and ignored by the receiver.

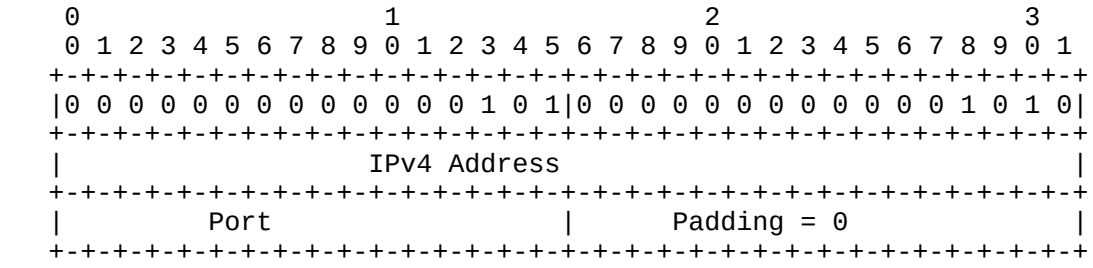
2.3.1.1 Optional or Variable Length Parameters

The following parameters follow the Type-Length-Value format as defined in Section 2.2.1. The IP address fields MUST come after the fixed-length fields.

Any extensions MUST come after the IP address fields.

IPv4 Address/Port

This parameter contains an IPv4 address/port for use as a destination transport address by the receiver.



IPv4 Address: 32bit u_int

Contains an IPv4 address of this endpoint. It is binary encoded.

Port Number: 16 bit u_int

Contains the UDP port number which the sender of this INIT wants to use for this address.

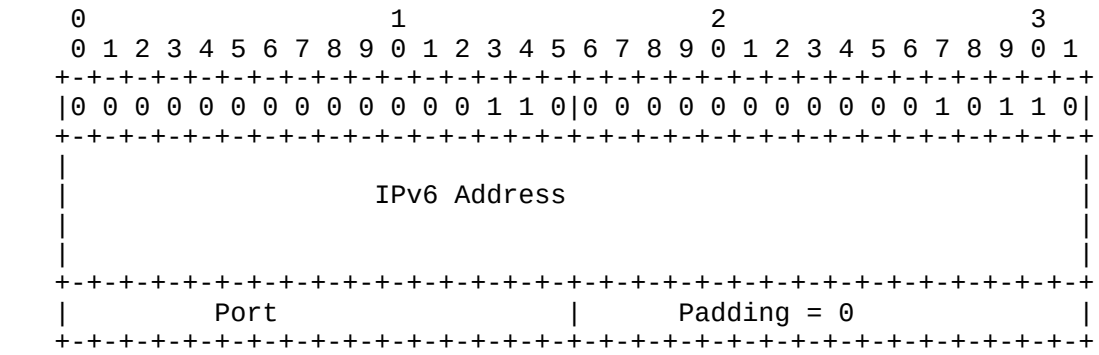
Padding: 16 bits

This field is set to 0x00 on transmit and ignored on receive.

IPv6 Address/Port:

This parameter contains an IPv6 address/port for use as a

destination transport address by the receiver.



IPv6 Address: 128 bit u_int

Contains an IPv6 address of the sender of this message. It is binary encoded.

Port Number: 16 bit u_int

Contains the UDP port number which the sender of this INIT wants to use for this address.

Padding: 16 bits

This field is set to 0x00 on transmit and ignored on receive.

The values passed in the IPv4 and IPv6 Address/Port parameters indicate to the other end of the association which transport addresses this end will support for the association being initiated. Within the association, any one of these addresses may appear in the source address field of a datagram sent from this (the initiating) end, and may be used as a destination of a datagram sent from the other (the responding) end.

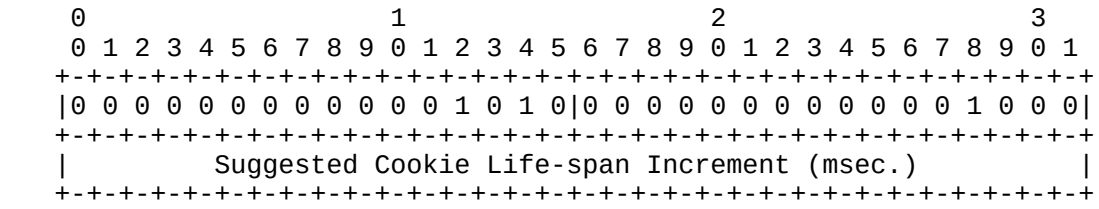
Note that an endpoint MAY be multi-homed. A multi-homed endpoint may have access to different types of network, thus more than one address type may be present in one INIT chunk, i.e., IPv4 and IPv6 addresses are allowed in the same INIT message.

More than one IP Address parameter can be included in an INIT chunk.

If the INIT contains a least one IP Address parameter, then only the transport addresses provided within the INIT may be used as destinations by the responding end. If the INIT does not contain any IP Address parameters, the responding end MUST use the source address associated with the received SCTP datagram as its sole destination address for the session.

Cookie Preservative

The sender of the INIT shall use this parameter to suggest to the receiver of the INIT for a longer life-span of the Responder Cookie.



Suggested Cookie Life-span Increment: 32bit u_int

This parameter indicates to the receiver the need for a cookie that expires in a longer period of time than that of the previous one. It is normally added to the INIT message during the second attempt of establishing an association with a peer after the first attempt

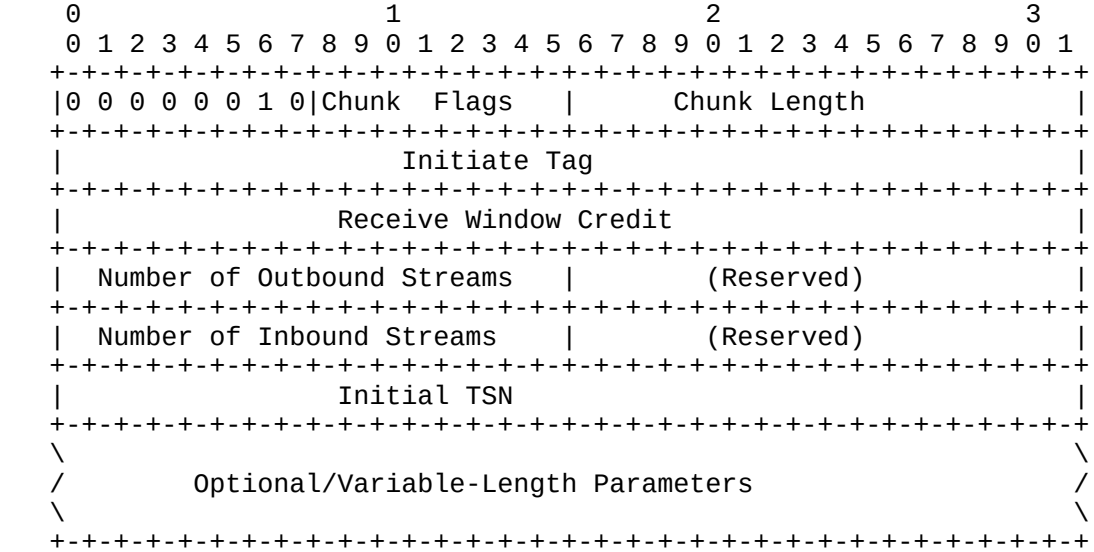
failed due to a Stale COOKIE report from the same peer. It is optional for the receiver to honor the suggested cookie life-span increment based upon its local security requirements.

2.3.2 Initiation Acknowledgement (INIT ACK) (00000010):

The INIT ACK chunk is used to acknowledge the initiation of a SCTP association.

The parameter part of INIT ACK is formatted similarly to the INIT chunk. It uses two extra variable parameters: The Responder Cookie and the Unrecognized Parameter:

The format of the INIT ACK message is shown below:



The INIT ACK contains the following parameters. Unless otherwise noted, each parameter MUST only be included once in the INIT ACK chunk.

Fixed Parameters	Status
Initiate Tag	Mandatory
Receiver Window Credit	Mandatory
Number of Outbound Streams	Mandatory
Number of Inbound Streams	Mandatory
Initial TSN	Mandatory

Variable Parameters	Status	Type Value
Responder Cookie	Mandatory	0x0007
IPv4 Address (Note 1)	Optional	0x0005
IPv6 Address (Note 1)	Optional	0x0006
Unrecognized Parameters	Optional	0x0008

Note 1: The INIT ACK chunks may contain multiple addresses that may be IPv4 and/or IPv6 in any combination.

The Responder Cookie and Unrecognized Parameters use the Type-Length-Value format as defined in Section 2.2.1 and are described below. The other fields are defined the same as their counterparts in the INIT message.

2.3.2.1 Optional or Variable Length Parameters

Responder Cookie: variable size, depending on Size of Cookie

This field MUST contain all the necessary state and parameter information required for the sender of this INIT ACK to create the association, along with an MD5 digital signature (128-bit). See

Section 4.1.3 for details on Cookie definition. The Cookie MUST be padded with '0' to the next 32-bit word boundary; otherwise, the format of the Cookie is implementation-specific.

Unrecognized Parameters: Variable Size.

This parameter is returned to the originator of the INIT message if the receiver does not recognize one or more Optional/Variable-length parameters in the INIT chunk. This parameter field will contain the unrecognized parameters copied from the INIT message complete with TLV.

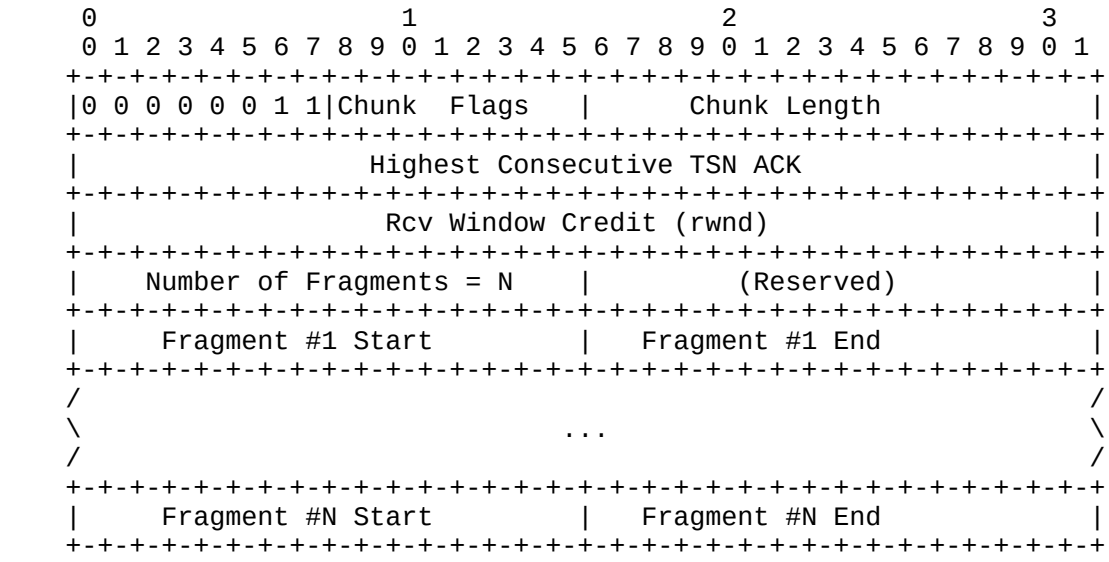
Vendor-Specific parameters are allowed in INIT ACK. However, they MUST be defined using the format described in Section 2.2.2, and be appended to the end of the INIT ACK chunk. In case the receiver of the INIT ACK does not support the vendor-specific parameters received, it MUST ignore those fields.

2.3.3 Selective Acknowledgement (SACK) (00000011):

This chunk is sent to the remote endpoint to acknowledge received DATA chunks and to inform the remote endpoint of gaps in the received subsequences of DATA chunks as represented by their TSNs.

The SACK MUST contain the Highest Consecutive TSN ACK and Rcv Window Credit (rwnd) parameters. By definition, the value of the Highest Consecutive TSN ACK parameter is the last TSN received at the time the Selective ACK is sent, before a break in the sequence of received TSNs occurs; the next TSN value following this one has not yet been received at the reporting end. This parameter therefore acknowledges receipt of all TSNs up to and including the value given.

The Selective ACK also contains zero or more fragment reports. Each fragment report acknowledges a subsequence of TSNs received following a break in the sequence of received TSNs. By definition, all TSNs acknowledged by fragment reports are higher than the value of the Highest Consecutive TSN ACK.



Chunk Flags:

Set to all zeros on transmit and ignored on receipt.

Highest Consecutive TSN ACK: 32 bit u_int

This parameter contains the TSN of the last DATA chunk received in sequence before a gap. Therefore, it must be lower than the Start TSN of the first fragment, if present.

Receive Window Credit (rwnd): 32 bit u_int

This field defines the new maximum number of octets of outbound data the receiver of this SACK is allowed to have outstanding (i.e. sent and not acknowledged).

Number of Fragments: 16 bit u_int

Indicates the number of TSN fragments included in this Selective ACK.

Reserved: 16 bit

Must be set to all 0 by the sender and ignored by the receiver.

Fragments:

These fields contain the ack fragments. They are repeated for each fragment up to the number of fragments defined in the Number of Fragments field. All DATA chunks with TSNs between the (Highest Consecutive TSN + Fragment Start) and (Highest Consecutive TSN + Fragment End) of each fragment are assumed to have been received correctly.

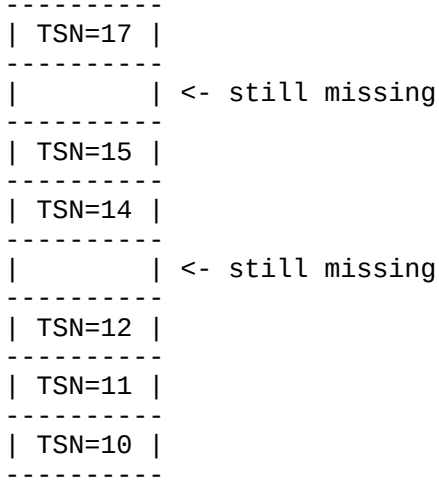
Fragment Start: 16 bit u_int

Indicates the Start offset TSN for this fragment. To calculate the actual TSN number the Highest Consecutive TSN is added to this offset number to yield the TSN. This calculated TSN identifies the first TSN in this fragment that has been received.

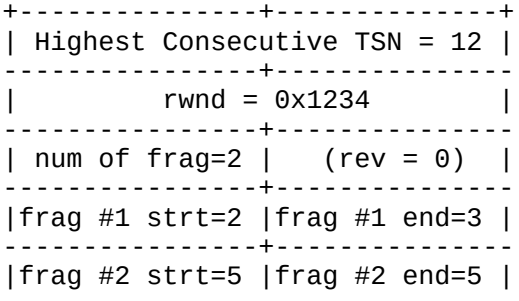
Fragment End: 16 bit u_int

Indicates the End offset TSN for this fragment. To calculate the actual TSN number the Highest Consecutive TSN is added to this offset number to yield the TSN. This calculated TSN identifies the TSN of the last DATA chunk received in this fragment.

For example, assume the receiver has the following datagrams newly arrived at the time when it decides to send a Selective ACK,



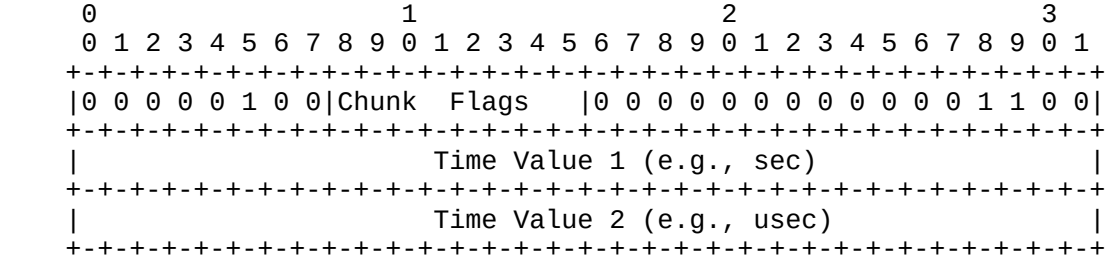
then, the parameter part of the Selective ACK MUST be constructed as follows (assuming the new rwnd is set to 0x1234 by the sender):



2.3.4 Heartbeat Request (HEARTBEAT) (00000100):

An endpoint should send this chunk to its peer endpoint of the current association to probe the reachability of a particular destination transport address defined in the present association.

The parameter fields MUST contain the time values.



Chunk Flags:

Set to zero on transmit and ignored on receipt.

Time Value 1: 32 bit u_int
Time Value 2: 32 bit u_int

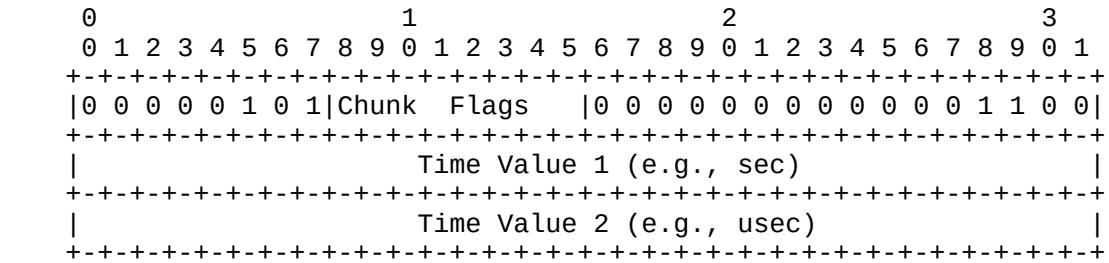
The Time Values contain a time value meaningful only to the sender. For example, Time Value 1 may be in units of seconds and Time Value 2 in units of microseconds. Their value should be set to the current time at which this Heartbeat Request is sent.

IMPLEMENTATION NOTE: For most systems the value is normally set by doing a system call to get the current time.

2.3.5 Heartbeat Acknowledgment (HEARTBEAT ACK) (00000101):

An endpoint should send this chunk to its peer endpoint as a response to a Heartbeat Request (see Section 7.3).

The parameter field MUST contain the time values.



Chunk Flags:

Set to zero on transmit and ignored on receipt.

Time Value 1: 32 bit u_int
Time Value 2: 32 bit u_int

The values of these two field SHALL be copied from the time values contained in the Heartbeat Request to which this Heartbeat acknowledgement is responding.

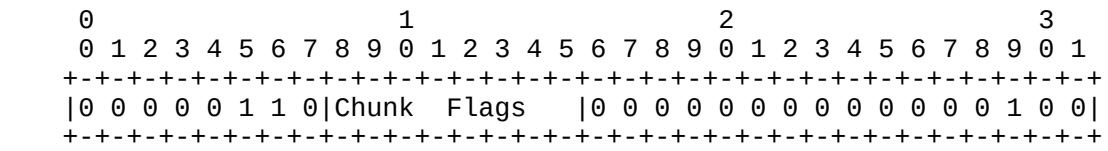
2.3.6 Abort Association (ABORT) (00000110):

The ABORT chunk is sent to the peer of an association to terminate the association. The Abort chunk has no parameters.

If an endpoint receives an INIT or INIT ACK missing a mandatory

parameter, it MUST send an ABORT message to its peer. It SHOULD include a Operational Error chunk with the Abort chunk to specify the reason.

If an endpoint receives an ABORT with a format error or for an association that doesn't exist, it drops the chunk and ignores it.

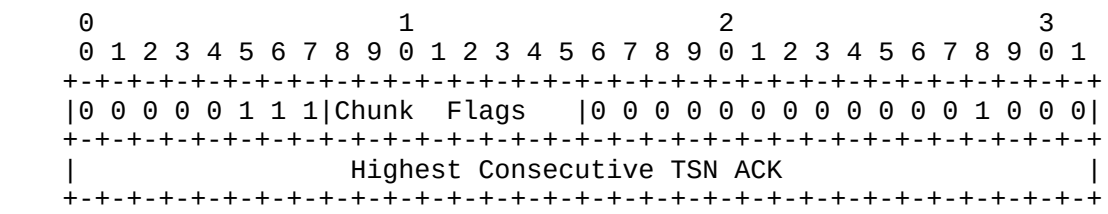


Chunk Flags:

Set to zero on transmit and ignored on receipt.

2.3.7 SHUTDOWN (00000111):

An endpoint in an association MUST use this chunk to initiate a graceful termination of the association with its peer. This chunk has the following format.



Chunk Flags:

Set to zero on transmit and ignored on receipt.

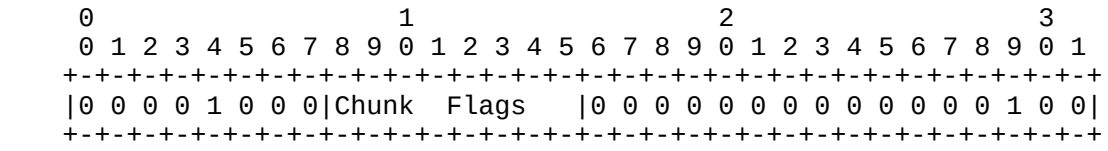
Highest Consecutive TSN ACK: 32 bit u_int

This parameter contains the TSN of the last chunk received in sequence before any gaps.

2.3.8 Shutdown Acknowledgment (SHUTDOWN ACK) (00001000):

This chunk MUST be used to acknowledge the receipt of the SHUTDOWN chunk at the completion of the shutdown process, see Section 8.2 for details.

The SHUTDOWN ACK chunk has no parameters.



Chunk Flags:

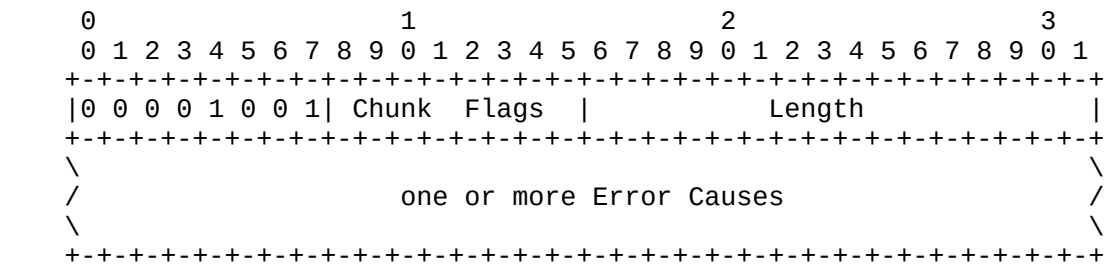
Set to zero on transmit and ignored on receipt.

Note: if the endpoint that receives the SHUTDOWN message does not have a TCB or tag for the sender of the SHUTDOWN, the receiver SHALL still respond. In such cases, the receiver SHALL send back a stand-alone SHUTDOWN ACK chunk in an SCTP datagram with the Verification Tag field of the common header filled with all '0's.

2.3.9 Operation Error (ERROR) (00001001):

This chunk is sent to the other endpoint in the association to notify certain error conditions. It contains one or more error causes. It has

the following parameters:



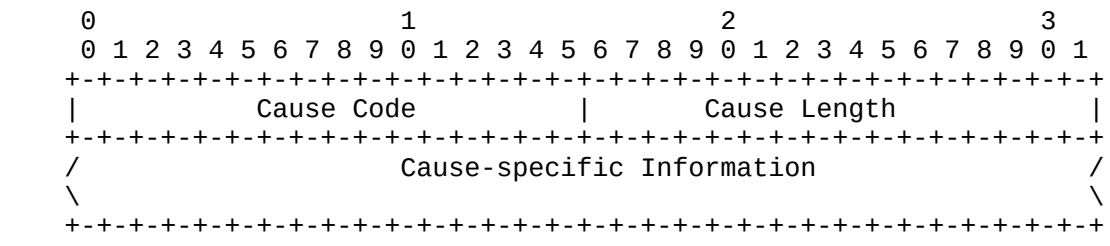
Chunk Flags:

  Set to zero on transmit and ignored on receipt.

Length:

  Set to the size of the chunk in octets, including the chunk header
  and all the Error Cause fields present.

Error causes are defined as variable-length parameters using the
format described in 2.2.1, i.e.:



Cause Code: 16 bit u_int

  Defines the type of error conditions being reported.

Cause Length: 16 bit u_int

  Set to the size of the parameter in octets, including the Cause Code,
  Cause Length, and Cause-Specific Information fields

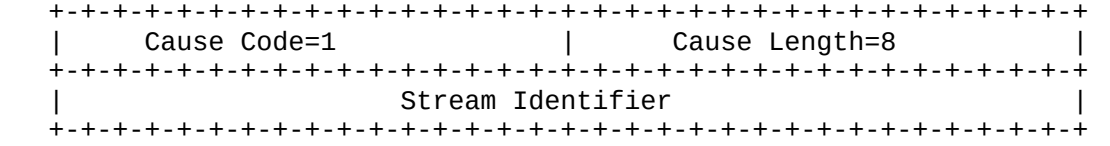
Cause-specific Information: variable length

  This field carries the details of the error condition.

Currently SCTP defines the following error causes:

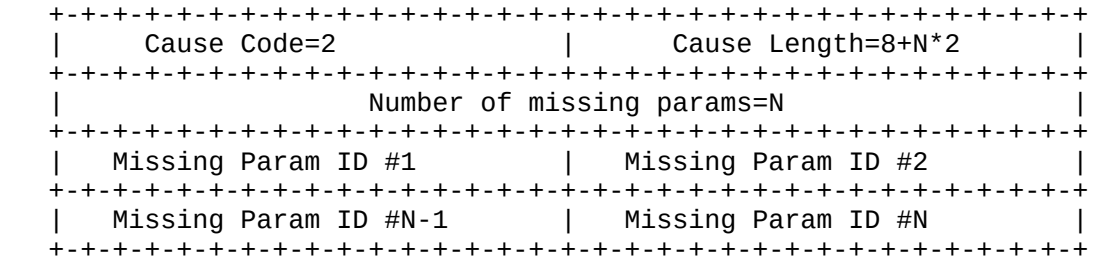
  Cause of error

  Invalid Stream Identifier



  Cause of error

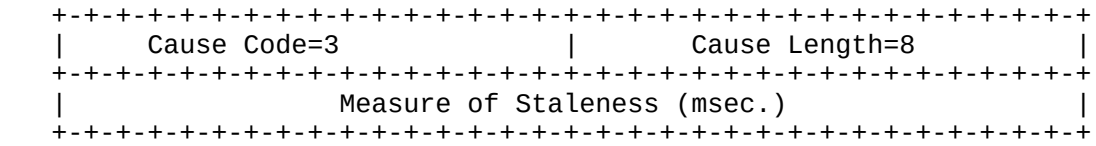
  Missing Mandatory Parameter



Each missing mandatory parameter ID should be specified in the message.

Cause of error

Stale Cookie Error: indicating the receiving of a valid cookie which is however expired.

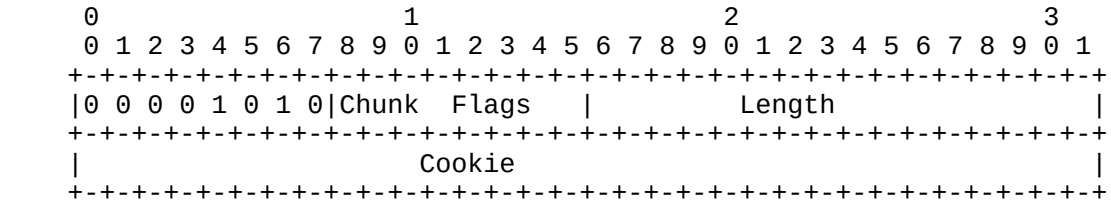


The sender of this error cause MAY choose to report how long past expiration the cookie is, by putting in the Measure of Staleness field the difference, in microseconds, between the current time and the time the cookie expired. If the sender does not wish to provide this information it should set Measure of staleness to 0.

Guidelines for IETF-defined Error Cause extensions are discussed in Section 11.3 of this document.

2.3.10 Encryption Cookie (COOKIE) (00001010):

This chunk is used only during the initialization of an association. It is sent by the initiator of an association to its peer to complete the initialization process. This chunk MUST precede any DATA chunk sent within the association, but MAY be bundled with one or more DATA chunks in the same datagram.



Chunk Flags: 8 bit

Set to zero on transmit and ignored on receipt.

Length: 16 bit u_int

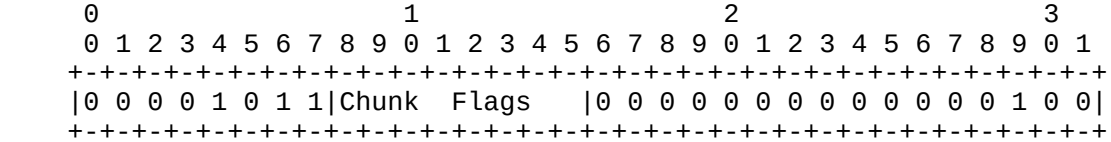
Set to the size of the chunk in octets, including the 4 octets of the chunk header and the size of the Cookie.

Cookie: variable size

This field must contain the exact cookie received in a previous INIT ACK.

2.3.11 Cookie Acknowledgment (COOKIE ACK) (00001011):

This chunk is used only during the initialization of an association. It is used to acknowledge the receipt of a COOKIE chunk. This chunk MUST precede any DATA chunk sent within the association, but MAY be bundled with one or more DATA chunks in the same SCTP datagram.

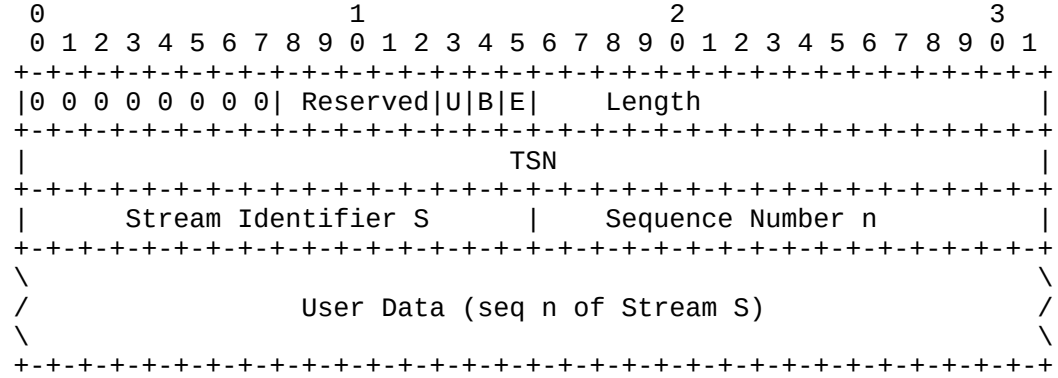


Chunk Flags:

Set to zero on transmit and ignored on receipt.

2.3.12 Payload Data (DATA) (00000000):

The following format MUST be used for the DATA chunk:



Note: the TSN, stream identifier, and sequence number MUST be transmitted in network byte order.

Reserved: 5 bits
  should be set to all ’0’s and ignored by the receiver.

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

  After reassembly (if necessary), unordered data chunks MUST be dispatched to the upper layer by the receiver without any attempt of re-ordering.

  Note, if an unordered user message is segmented, each segment MUST have its U bit set to 1.

B bit: 1 bit

  The (B)eginning segment bit, if set, indicates the first segment of an SCTP user message.

E bit: 1 bit
  The (E)nding segment bit, if set, indicates the last segment of an SCTP user message.

A non-segmented user message shall have both the B and E bits set to 1. Setting both B and E bits to 0 indicates a middle segment of a multi-segment SCTP user message, as summarized in the following table:

B E	Description
1 0	First piece of a segmented SCTP user message.
0 0	Middle piece of a segmented user message
0 1	Last piece of a segmented SCTP user message.
1 1	Un-segmented Message

Length: 16 bits (16 bit u_int)

  This field indicates the length of the DATA chunk in octets. It includes the Type field, the Reserved field, the U and B/E bits, the Length field, TSN, the Stream Identifier, the Stream Sequence Number, and the User Data fields. It does not include any padding.

TSN : 32 bits (32 bit u_int)

  This value represents the TSN for this DATA chunk.

Stream Identifier S: 16 bit u_int

  Identifies the stream to which the following user data belongs.

Sequence Number n: 16 bit u_int

  This value presents the sequence number of the following user
  data within the stream S. Valid range is 0x0 to 0xFFFF.

  Note, when a user message is segmented by SCTP for transport, the
  same sequence number MUST be carried in each of the segments of
  the message.

User Data: variable length

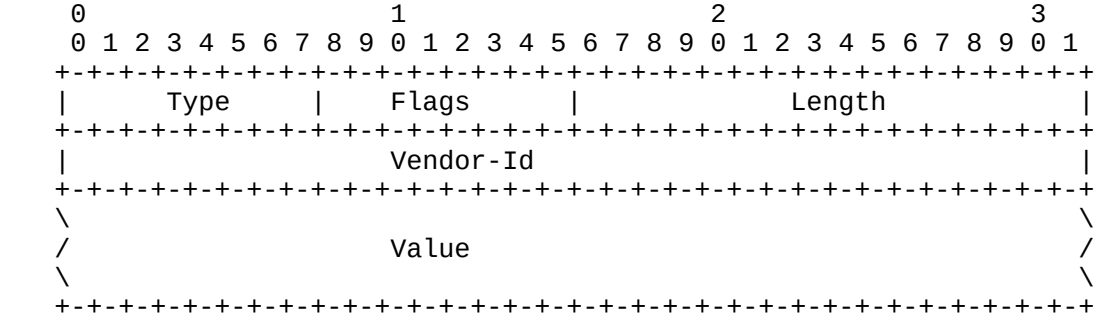
  This is the payload user data. The implementation MUST pad the end
  of the data to a 32 bit boundary with 0 octets. Any padding should
  NOT be included in the length field.

2.4 Vendor-Specific Chunk Extensions

This Chunk type is available to allow vendors to support their own extended data formats not defined by the IETF. It MUST not affect the operation of SCTP.

Endpoints not equipped to interpret the vendor-specific chunk sent by a remote endpoint MUST ignore it. Endpoints that do not receive desired vendor specific information SHOULD make an attempt to operate without it, although they may do so (and report they are doing so) in a degraded mode.

A summary of the Vendor-Specific Chunk format is shown below. The fields are transmitted from left to right.



Type: 8 bit u_int

   0xFE for all Vendor-Specific chunks.

Flags: 8 bit u_int

   Vendor specific flags.

Length: 16 bit u_int

   Size of this Vendor-Specific chunks in octets, including the Type, Flags, Length, Vendor-Id, and Value fields.

Vendor-Id: 32 bit u_int

   The high-order octet is 0 and the low-order 3 octets are the SMI Network Management Private Enterprise Code of the Vendor in network byte order, as defined in the Assigned Numbers (RFC 1700).

Value: Variable length

The Value field is one or more octets. The actual format of the information is site or application specific, and a robust implementation SHOULD support the field as undistinguished octets.

The codification of the range of allowed usage of this field is outside the scope of this specification.

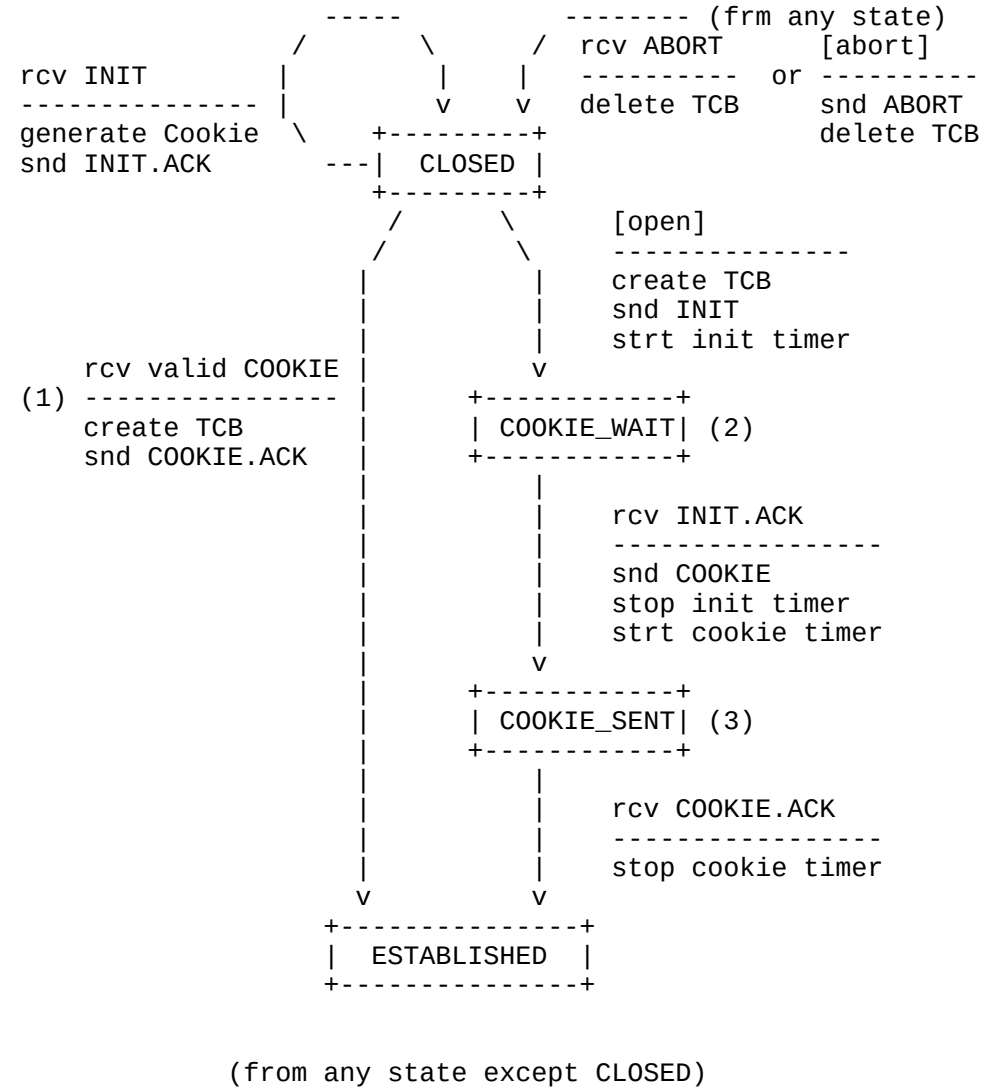
3. SCTP Association State Diagram

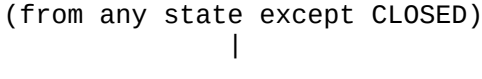
During the lifetime of an SCTP association, the SCTP endpoints progress from one state to another in response to various events. The events that may potentially advance an endpoint's state include:

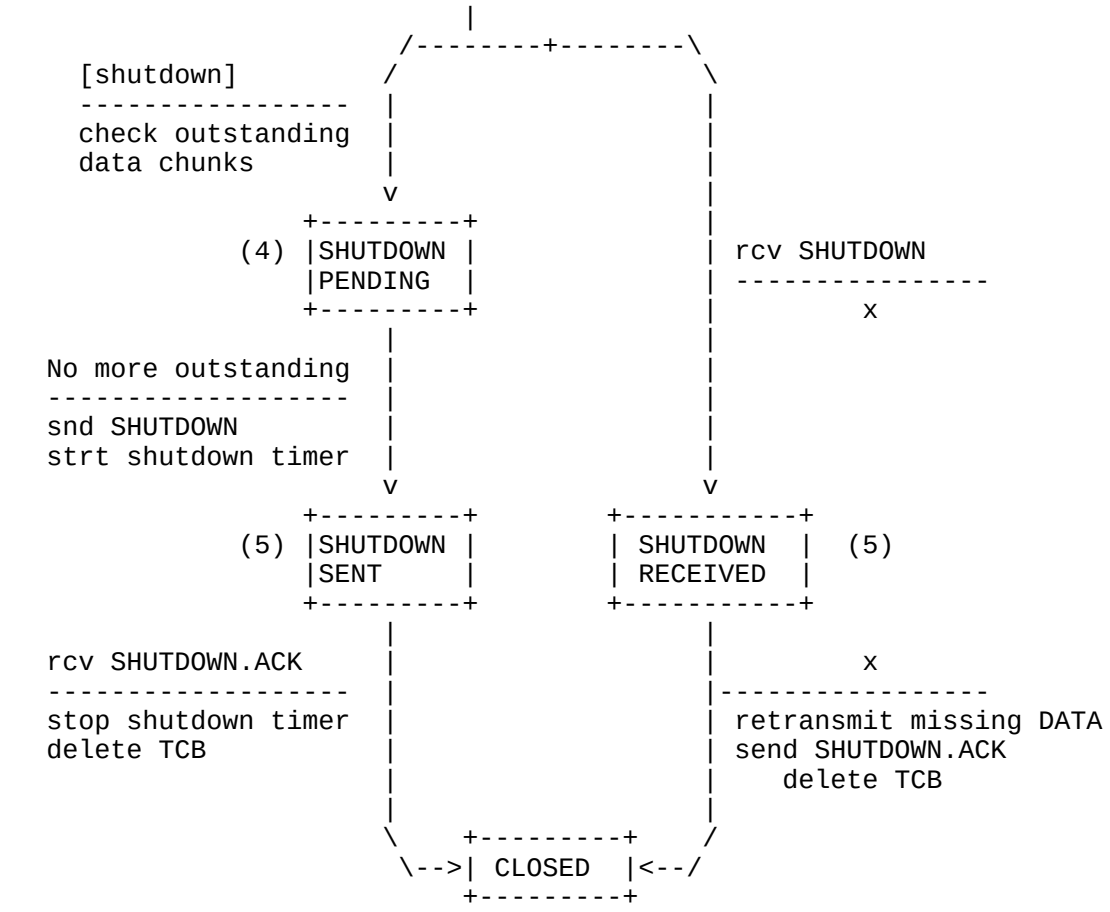
- o SCTP user primitive calls, e.g., [open], [shutdown], [abort],
- o reception of INIT, COOKIE, ABORT, SHUTDOWN, etc. control chunks, or
- o some timeout events.

The state diagram in the figures below illustrates state changes, together with the causing events and resulting actions. Note that some of the error conditions are not shown in the state diagram. Full description of all special cases should be found in the text.

Note, chunk names are given in all capital letters, while parameter names have the first letter capitalized, e.g., COOKIE chunk type vs. Cookie parameter.







Note:

(1) If the received COOKIE is invalid (i.e., failed to pass the authentication check), the receiver MUST silently discard the datagram. Or, if the received COOKIE is expired (see Section 4.1.5), the receiver SHALL send an ERROR chunk back. In either case, the receiver SHALL stay in the closed state.

(2) If the init timer expires, the endpoint SHALL retransmit INIT and re-start the init timer without changing state. This SHALL be repeated up to 'Max.Init.Retransmit' times. After that, the endpoint SHALL abort the initialization process and report the error to SCTP user.

(3) If the cookie timer expires, the endpoint SHALL retransmit COOKIE and re-start the cookie timer without changing state. This SHALL be repeated up to 'Max.Init.Retransmit' times. After that, the endpoint SHALL abort the initialization process and report the error to SCTP user.

(4) In SHUTDOWN-SENT state the endpoint SHALL acknowledge any received DATA chunks without delay

(5) In SHUTDOWN-RECEIVED state, the endpoint MUST NOT accept any new send request from its SCTP user.

4. Association Initialization

Before the first data transmission can take place from one SCTP endpoint ("A") to another SCTP endpoint ("Z"), the two endpoints must complete an initialization process in order to set up an SCTP association between them.

The SCTP user at an endpoint SHOULD use the ASSOCIATE primitive to initialize an SCTP association to another SCTP endpoint.

IMPLEMENTATION NOTE: From an SCTP-user's point of view, an association may be implicitly opened, without an Associate primitive

(see 9.1 B) being invoked, by the initiating endpoint's sending of the first user data to the destination endpoint. The initiating SCTP will assume default values for all mandatory and optional parameters for the INIT/INIT ACK.

Once the association is established, unidirectional streams will be open for data transfer on both ends (see Section 4.1.1).

A cookie mechanism is employed during the initialization to provide protection against security attacks. The cookie mechanism uses a four-way handshaking, but the last two legs of which are allowed to carry user data for fast setup.

4.1 Normal Establishment of an Association

The initialization process consists of the following steps (assuming that SCTP endpoint "A" tries to set up an association with SCTP endpoint "Z" and "Z" accepts the new association):

A) "A" shall first send an INIT message to "Z". In the INIT, "A" must provide its security tag "Tag_A" in the Initiate Tag field. Tag_A shall be a random number in the range of 0x1 to 0xffffffff (see 4.3.1 for Tag value selection). After sending the INIT, "A" enters the COOKIE-WAIT state.

B) "Z" shall respond immediately with an INIT ACK message. In the message, besides filling in other parameters, "Z" must set the Verification Tag field to Tag_A, and also provide its own security tag "Tag_Z" in the Initiate Tag field.

   Moreover, "Z" shall generate and send along with the INIT ACK a responder cookie. See Section 4.1.3 for responder cookie generation.

   Note: after sending out INIT ACK with the cookie, "Z" should not allocate any resources, nor keep any states for the new association. Otherwise, "Z" will be vulnerable to resource attacks.

C) Upon reception of the INIT ACK from "Z", "A" shall leave COOKIE-WAIT state and enter the COOKIE-SENT state. "A" now sends the cookie received in the INIT ACK message in a cookie chunk. The cookie chunk can be bundled with any pending DATA chunks, but it MUST be the first chunk in the datagram.

D) Upon reception of the COOKIE chunk, Endpoint "Z" will reply with a COOKIE-ACK chunk after building a TCB and marking itself to the established state. A COOKIE-ACK chunk may also be combined with any pending DATA chunks (and/or SACK chunks), but the COOKIE-ACK chunk must be the first chunk in the datagram.

  IMPLEMENTATION NOTE: a implementation may choose to send the communication up primitive to the SCTP user upon reception of a valid cookie.

E) Upon reception of the COOKIE ACK, endpoint "A" will move from the COOKIE-SENT state to the ESTABLISHED state, stopping its INIT timer.

  IMPLEMENTATION NOTE: a implementation may choose to send the communication up primitive to the SCTP user upon reception of a the COOKIE ACK.

Note: no DATA chunk shall be carried in the INIT or INIT ACK message.

Note: if an endpoint receives an INIT, INIT ACK, or COOKIE chunk but decides not to establish the new association due to lack of resources, etc., it shall respond to the chunk with an ABORT chunk. The Verification Tag field of the common header must be set to equal the Initiate Tag value of the peer.

Note: After the reception of the first data chunk in an association the receiver MUST immediately respond with a SACK to acknowledge the data chunk, subsequent acknowledgements should be done as described in section 5.2.

Note: When a SCTP endpoint sends a INIT or INIT ACK it MUST include all of its transport addresses in the parameter section. This is because it may NOT be possible to control the "sending" address that a receiver of a SCTP datagram sees. A receiver thus MUST know every address that may be a source address for a peer SCTP endpoint, this assures that the inbound SCTP datagram can be matched to the proper association.

4.1.1 Handle Stream Parameters

In the INIT and INIT ACK messages, the sender of the message shall indicate the number of outbound streams (OS) it wishes to have in the association, as well as the maximal inbound streams (MIS) it will accept from the other endpoint.

After receiving these stream configuration information from the other side, each endpoint shall perform the following check: if the peer’s MIS is less than the endpoint’s OS, meaning that the peer is incapable of supporting all the outbound streams the endpoint wants to configure, the endpoint MUST either settle with MIS outbound streams, or abort the association and report to its upper layer the resources shortage at its peer.

After the association is initialized, the valid outbound stream identifier range for either endpoint shall be 0 to min(local OS, remote MIS)-1.

4.1.2 Handle Address Parameters

During the association initialization, an endpoint shall use the following rules to discover and collect the destination transport address(es) to its peer.

On reception of an INIT or INIT ACK message, the receiver shall record any transport addresses specified as parameters in the INIT or INIT ACK message, and use only these addresses as destination transport addresses when sending subsequent datagrams to its peer. If NO destination transport addresses are specified in the INIT or INIT ACK message, then the source address from which the message arrived should be used as the destination transport address for all datagrams.

4.1.3 Generating Responder Cookie

When sending an INIT ACK as a response to an INIT message, the sender of INIT ACK should create a responder cookie and send it as part of the INIT ACK. Inside this responder cookie, the sender should include a security signature, a time stamp on when the cookie is created, and the lifespan of the cookie, along with all the information necessary for it to establish the association.

The following steps SHOULD be taken to generate the cookie:

1) create an association TCB using information from both the received INIT and the outgoing INIT ACK messages,

2) in the TCB, set the creation time to the current time of day, and the lifespan to the protocol parameter ’Valid.Cookie.Life’,

3) attach a private security key to the TCB and generate a 128-bit MD5 signature from the key/TCB combination (see [4] for details on MD5), and

4) generate the responder cookie by combining the TCB and the

resultant MD5 signature.

After sending the INIT ACK with the cookie, the sender SHOULD delete the TCB and any other local resource related to the new association, so as to prevent resource attacks.

The private key should be a cryptographic quality random number with a sufficient length. Discussion in RFC-1750 [1] can be helpful in selection of the key.

4.1.4 Cookie Processing

When a cookie is received from its peer in an INIT ACK message, the receiver of the INIT ACK MUST immediately send a COOKIE chunk to its peer and MAY piggy-back any pending DATA chunks on the outbound cookie chunk. The sender should also start a timer, and retransmit the cookie chunk until a COOKIE ACK is received or the endpoint is marked unreachable.

4.1.5 Cookie Authentication

When an endpoint receives a COOKIE chunk from another endpoint with which it has no association, it shall take the following actions:

1) compute an MD5 signature using the TCB data carried in the cookie along with the receiver's private security key,

2) authenticate the cookie by comparing the computed MD5 signature against the one carried in the cookie. If this comparison fails, the datagram, including the COOKIE and the attached user data, should be silently discarded,

3) compare the creation time stamp in the cookie to the current local time, if the elapsed time is longer than the lifespan carried in the cookie, then the datagram, including the COOKIE and the attached user data, SHOULD be discarded and the endpoint MUST transmit a stale cookie operational error to the sending endpoint,

4) if the cookie is valid, create an association to the sender of the COOKIE message with the information in the TCB data carried in the COOKIE, and enter the ESTABLISHED state,

5) acknowledge any DATA chunk in the datagram following the rules defined in Section 5.2, and,

6) send a COOKIE ACK chunk to the sender acknowledging reception of the cookie. The COOKIE ACK MAY be piggybacked with any DATA chunk or SACK chunk (if a DATA chunk is present in the received datagram a SACK MUST be sent in the acknowledgement).

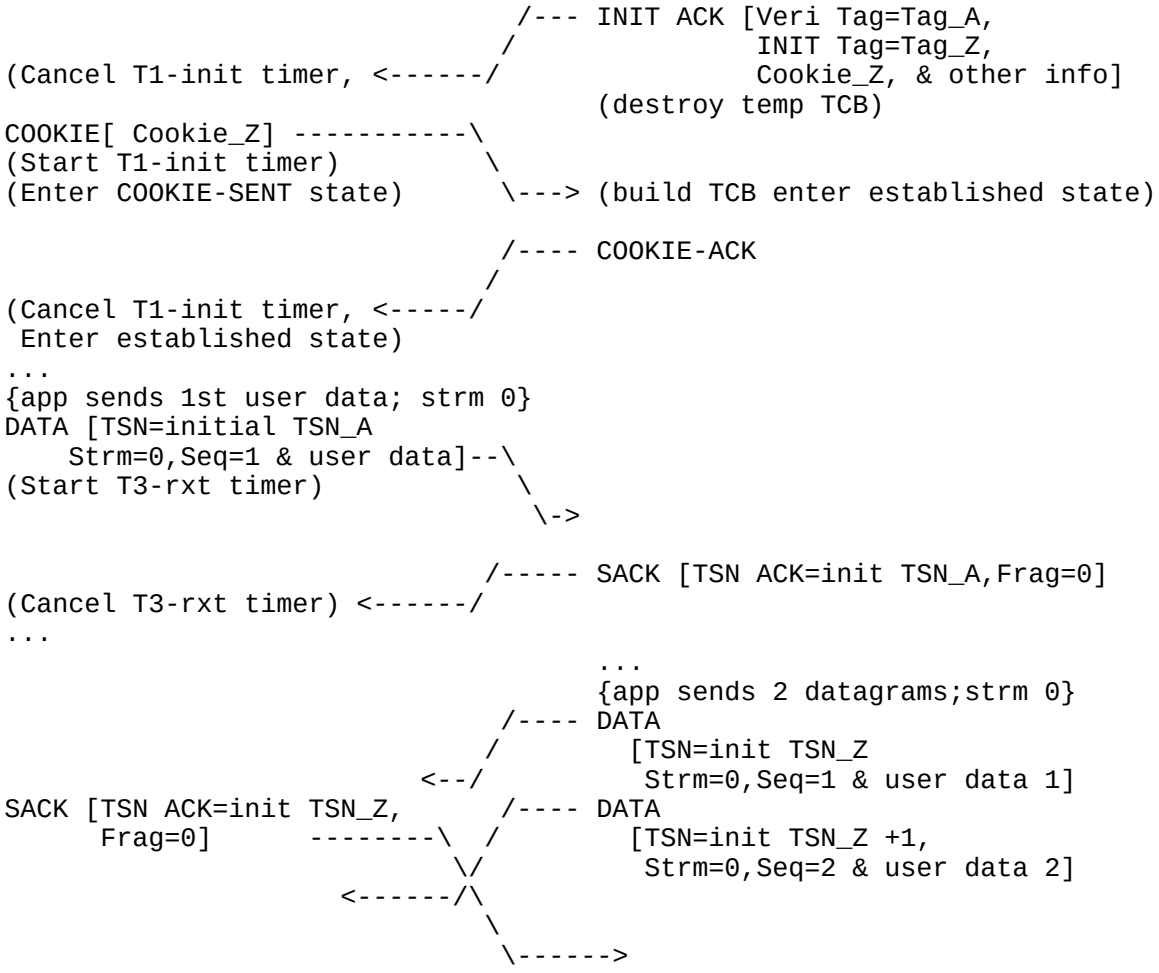
Note that if a COOKIE is received from an endpoint with which the receiver of the COOKIE has an existing association, the proceedures in section 4.2 should be followed.

4.1.6 An Example of Normal Association Establishment

In the following example, "A" initiates the association and then sends a user datagram to "Z", then "Z" sends two user datagrams to "A" later:

```
Endpoint A                                          Endpoint Z

{app sets association with Z}
INIT [INIT Tag=Tag_A
      & other info]  --------\
(Start T1-init timer)         \
(Enter COOKIE-WAIT state)      \---> (compose temp TCB and Cookie_Z)
```



Note that If T1-init timer expires at "A" after the INIT or COOKIE chunks are sent, the same INIT or cookie chunk with the same Initiate Tag (i.e., Tag_A) or cookie shall be retransmitted and the timer restarted. This shall be repeated Max.Init.Retransmit times before "A" considers "Z" unreachable and reports the failure to its upper layer. When retransmitting the INIT, the endpoint SHALL following the rules defined in 5.3 to determine the proper timer value.

4.2 Handle Duplicate INIT, INIT ACK, COOKIE, and COOKIE ACK

At any time during the life of an association (in one of the possible states) between an endpoint and its peer, one of the setup chunks may be received from the peer, the receiver shall process such a duplicate has described in this section.

The following scenarios can cause duplicated chunks:

A) The peer has crashed without being detected, and re-started itself and sent out a new Chunk trying to restore the association,

B) Both sides are trying to initialize the association at about the same time,

C) The chunk is a staled datagram that was used to establish the present association or a past association which is no longer in existence, or

D) The chunk is a false message generated by an attacker.

In case A), the endpoint shall reset the present association and set a new association with its peer. Case B) is unique and is discussed in Section 4.2.1. However, in cases C) and D), the endpoint must retain the present association.

The rules in the following sections shall be applied in order to identify and correctly handle these cases.

4.2.1 Handle Duplicate INIT in COOKIE-WAIT or COOKIE-SENT State

This usually indicates an initialization collision.

In such a case, each of the two side shall respond to the other side with an INIT ACK, with the Verification Tag field of the common header set to the tag value received from the INIT message, and the Initiate Tag field set to its own tag value (the same tag used in the INIT message sent out by itself). Each responder shall also generate a cookie with the INIT ACK.

After that, no other actions shall be taken by either side, i.e., the endpoint shall not change its state, and the T1-init timer shall be let running. The normal procedures for handling cookies will resolve the duplicate INITs to a single association.

4.2.2 Handle Duplicate INIT in Other States

Upon reception of the duplicated INIT, the receiver shall follow the normal procedures for handling a INIT message, i.e. generate a INIT ACK with a cookie.

In the outbound INIT ACK, the Verification Tag field of the common header shall be set to the peer tag value (from the INIT message), and the Initiate Tag field set to its own tag value (unchanged from the existing association). A cookie should also be included generated with the current time and a updated TCB based upon the INIT message. And no further actions shall be taken.

4.2.3 Handle Duplicate INIT ACK

If an INIT ACK is received by an endpoint in any state other than the COOKIE-WAIT state, the endpoint should discard the INIT ACK message. A duplicate INIT ACK usually indicates the processing of a old INIT or duplicated INIT message.

4.2.4 Handle Duplicate Cookie

When a duplicated COOKIE chunk is received in any state for an existing association the following rules shall be applied:

1) compute an MD5 signature using the TCB data carried in the cookie along with the receiver’s private security key,

2) authenticate the cookie by comparing the computed MD5 signature against the one carried in the cookie. If this comparison fails, the datagram, including the COOKIE and the attached user data, should be silently discarded (this is case C or D above).

3) compare the timestamp in the cookie to the current time, if the cookie is older than the lifespan carried in the cookie, the datagram, including the COOKIE and the attached user data, should be discarded and the endpoint MUST transmit a stale cookie error to the sending endpoint only if the Verification tags of the cookie’s TCB does NOT match the current tag values in the association (this is case C or D above).

4) If the cookie proves to be valid, unpack the TCB into a temporary TCB.

5) If the Verification Tags in the Temporary TCB matches the Verification Tags in the existing TCB, the cookie is a duplicate cookie. A cookie ack should be sent to the peer endpoint but NO update should be made to the existing TCB.

6) If the the local Verification Tag in the temporary TCB
   does not match the local Verification Tag in the existing
   TCB, then the cookie is a old stale cookie and does
   not correspond to the existing association (case C above).
   The datagram should be silently discarded.

7) If the Peers Verification Tag in the temporary TCB does not
   match the Peers Verification Tag in the existing TCB
   then a restart of the peer has occurred (case A above) and the
   endpoint should report the restart and respond with a COOKIE-ACK
   message; Updating the Verification Tag, starting sequence
   number, and network information of its peer from the temporary
   TCB to the existing TCB. After which the temporary TCB may be
   discarded.

  IMPLEMENTATION NOTE: It is an implementation decision on how
  to handle any pending datagrams. The implementation may elect
  to either A) send all messages up to its upper layer with the
  restart report, or B) automatically requeue any datagrams
  pending, marking all to the unsent state and assigning
  new TSN's at the time of initial transmit based upon the
  updated starting sequence number (as defined in section 5.5).

4.2.5 Handle Duplicate COOKIE-ACK.

At any state other than COOKIE-Sent a endpoint may receive a
duplicated COOKIE-ACK chunk. If so, the chunk should be silently
discarded.

4.2.6 Handle Stale COOKIE Error

A stale cookie error indicates one of a number of possible events:

A) that the association failed to completely setup before the
   cookie issued by the sender was processed.

B) an old cookie was processed after setup completed.

C) an old cookie is received from someone that the receiver is
   not interested in having a association with and the ABORT
   message was lost.

When processing a stale cookie a endpoint should first examine
if an association is in the process of being setup, i.e. the
association is in the COOKIE-SENT state. In all cases if
the association is NOT in the COOKIE-SENT state, the stale
cookie message should be silently discarded.

If the association is in the COOKIE-SENT state, the endpoint
may elect one of three alternatives.

1) Send a new INIT message to the endpoint, to generate
   a new cookie and re-attempt the setup procedure.

2) Discard the TCB and report to the upper layer the
   inability to setup the association.

3) Send a new INIT message to the endpoint, adding a cookie
   preservative requesting a time extentsion on the life of the
   cookie. When calculating the time extension, an implementation
   SHOULD use Round Trip Time (RTT) information generated from
   the original COOKIE <-> Stale COOKIE timing and should add no more
   than 1,000,000 microseconds beyond the RTT. Long cookie lives will
   make a endpoint more subject to a replay attack.

4.3 Other Initialization Issues

4.3.1 Selection of Tag Value

Initiate Tag values should be selected from the range of 0x1 to 0xffffffff. It is very important that the Tag value be randomized to help protect against "man in the middle" and "sequence number" attacks. It is suggested that RFC 1750 [1] be used for the Tag randomization.

Moreover, the tag value used by either endpoint in a given association MUST never be changed during the lifetime of the association. However, a new tag value MUST be used each time the endpoint tears-down and then re-establishes the association to the same peer.

4.3.2 Initiation from behind a NAT

When a NAT is present between two endpoints, the endpoint that is behind the NAT, i.e., one that does not have a publicly available network address, shall take one of the following options:

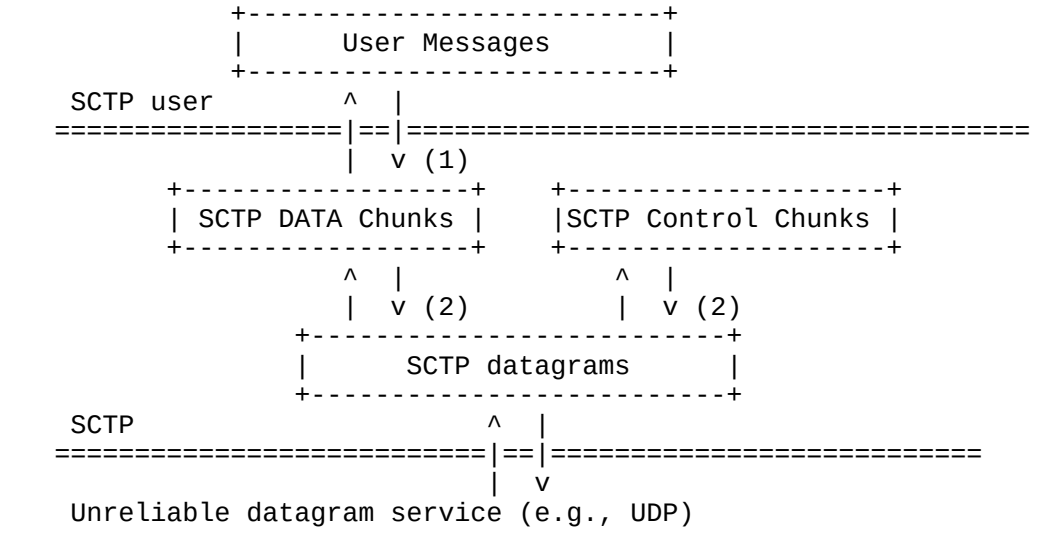
A) Indicate that only one address can be used by including no transport addresses in the INIT message (Section 2.3.1.1). This will make the endpoint that receives this Initiation message to consider the sender as only having that one address. This method can be used for a dynamic NAT, but any multi-homing configuration at the endpoint that is behind the NAT will not be visible to its peer, and thus not be taken advantage of.

B) Indicate all of its networks in the Initiation by specifying all the actual IP addresses and ports that the NAT will substitute for the endpoint. This method requires that the endpoint behind the NAT must have pre-knowledge of all the IP addresses and ports that the NAT will assign.

5. User Data Transfer

For transmission efficiency, SCTP defines mechanisms for bundling of small user messages and segmentation of large user messages.

The following diagram depicts the flow of user messages through SCTP.



Note:
(1) When converting user messages into Data chunks, SCTP sender will segment user messages larger than the current path MTU into multiple data chunks. The segmented message will be reassembled from data chunks before delivery by the SCTP receiver.

(2) Multiple data and control chunks may be multiplexed by the sender into a single SCTP datagram for transmission, as long as the final size of the datagram does not exceed the current path MTU. The receiver will de-multiplex the datagram back into

chunks.

The bundling and segmentation mechanisms, as detailed in Sections 5.9 and 5.10, are optional to implement by the data sender, but they MUST be implemented by the data receiver, i.e., a SCTP receiver MUST be prepared to receive and process bundled or segmented data.

5.1 Transmission of DATA Chunks

The following general rules SHALL be applied by the sender for transmission and/or retransmission of outbound DATA chunks:

A) At any given time, the sender MUST NOT transmit new data onto any destination transport address if it has rwnd or more octets of data outstanding. The outstanding data size is defined as the total size of ALL data chunks outstanding.

B) At any given time, the sender MUST NOT transmit new data onto a given transport address if it has cwnd or more octets of data outstanding on that transport address.

C) When the time comes for the sender to transmit, before sending new DATA chunks, the sender MUST first transmit any outstanding DATA chunks which are marked for retransmission.

D) Then, the sender can send out as many new DATA chunks as Rule A and Rule B above allow.

Note: multiple DATA chunks committed for transmission MAY be bundled in a single packet, unless bundling is explicitly disallowed by ULP of the data sender. Also, it is implementation specific whether to allow the bundling of retransmission DATA chunks with new DATA chunks.

Note: if there are any unacknowledged DATA chunks (e.g., due to delayed ack), the sender should create a SACK and bundle it with the outbound DATA chunk, as long as the size of the final SCTP datagram does not exceed the current MTU. See Section 5.2.

> IMPLEMENTATION Note: if the window is full (i.e., transmission is disallowed by Rule A and/or Rule B), the sender MAY still accept send requests from its upper layer, but SHALL transmit no more DATA chunks until some or all of the outstanding DATA chunks are acknowledged and transmission is allowed by Rule A and Rule B again.

In all cases, if a transmission or retransmission of a DATA chunk is made, the sender shall start the retransmission timer (T3-rxt) if it is not currently running. The T3-rxt timer value must be adjusted according to the timer rules defined in Sections 5.3.2, and 5.3.3.

Note: If rwnd is set to 0, and all outstanding DATA chunks are acknowledged, a sender is allowed to send ONE new DATA chunk up to the size of current path MTU, to probe the receiver for changes to rwnd.

5.2 Acknowledgment on Reception of DATA Chunks

The SCTP receiver MUST always acknowledge the SCTP sender about the reception of each DATA chunk.

The guidelines on delayed acknowledgment algorithm specified in Section 4.2 of RFC 2581 [3] SHOULD be followed. In particular, a SCTP receiver MUST NOT excessively delay acknowledgments. Specifically, an acknowledgement SHOULD be generated for at least every second datagram received, and SHOULD be generated within 200 ms of the arrival of any unacknowledged datagram.

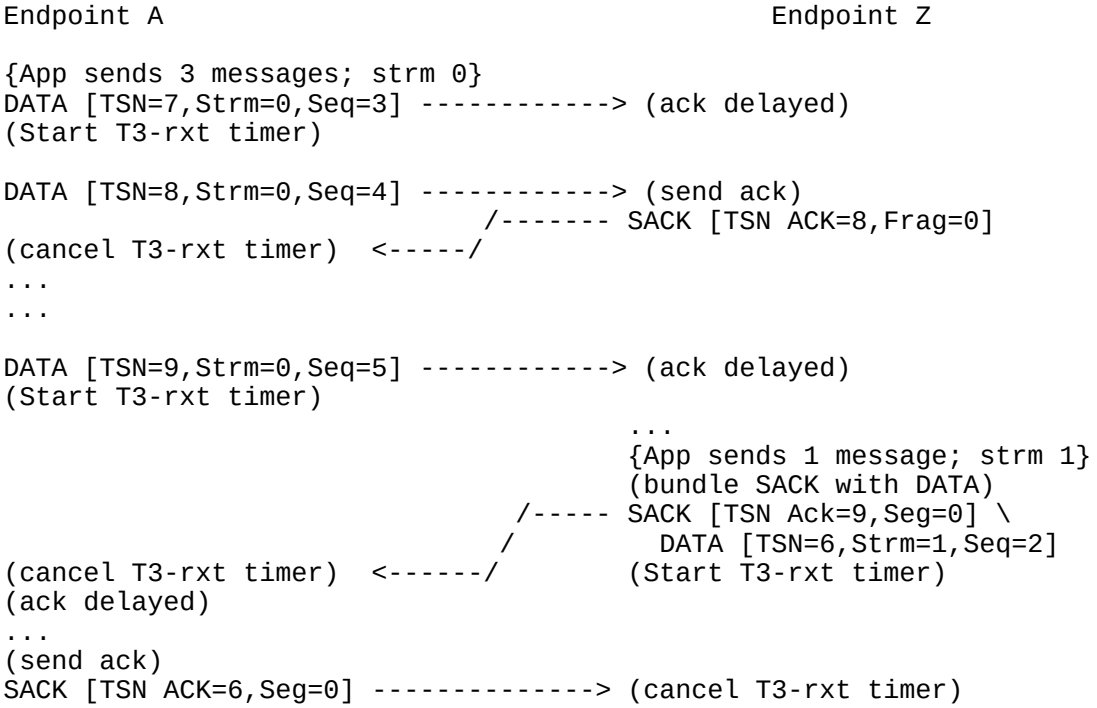
> IMPLEMENTATION NOTE: the maximal delay for generating an acknowledgement may need to be configurable by the SCTP user,

either statically or dynamically, in order to meet the specific timing requirement of the signaling protocol being carried.

Acknowledgments MUST be sent in SACK control chunks. A SACK chunk can acknowledge the reception of multiple DATA chunks. See Section 2.3.3 for SACK chunk format. In particular, the SCTP receiver MUST fill the Highest Consecutive TSN ACK field to indicate the highest consecutive TSN number it has received, and any received segments beyond the highest consecutive TSN SHALL also be reported.

Upon reception of the SACK, the data sender MUST adjust its total outstanding data count and the outstanding data count on those destination addresses for which one or more data chunks is acknowledged by the SACK.

The following example illustrates the use of delayed acknowledgments:



5.3 Management of Retransmission Timer

SCTP uses a retransmission timer T3-rxt to ensure data delivery in the absence of any feedback from the remote data receiver. The duration of this timer is referred to as RTO (retransmission timeout).

The computation and management of RTO in SCTP follows closely with how TCP manages its retransmission timer. To compute the current RTO, an SCTP sender maintains two state variables per destination transport address: SRTT (smoothed round-trip time) and RTTVAR (round-trip time variation).

5.3.1 RTO Calculation

The rules governing the computation of SRTT, RTTVAR, and RTO are as follows:

C1) Until an RTT measurement has been made for a packet sent to the given destination transport address, set RTO to the protocol parameter 'RTO.Initial'.

C2) When the first RTT measurement R is made, set SRTT <- R, RTTVAR <- R/2, and RTO <- SRTT + K * RTTVAR, where K = 4.

C3) When a new RTT measurement R' is made, set

RTTVAR <- beta * RTTVAR + (1 - beta) * |SRTT - R'|

SRTT <- alpha * SRTT + (1 - alpha) * R'

(The value of SRTT used in the update to RTTVAR is its value *before* updating SRTT itself using the second assignment.)

The above are computed using alpha=1/8 and beta=1/4.

After the computation, update RTO <- SRTT + 4 * RTTVAR.

C4) It is RECOMMENDED that new RTT measurements should be made no more than once per round-trip for a given destination transport address. There are two reasons for this recommendation: first, it appears that measuring more often does not in practice yield any significant benefit [5]; second, if measurements are made more often, then the values of alpha and beta in rule C3 above must be adjusted so that SRTT and RTTVAR still adjust to changes at roughly the same rate (in terms of how many round trips it takes them to reflect new value) as they would if making only one measurement per round-trip and using alpha and beta as given in rule C3.

C5) Karn's algorithm: RTT measurements MUST NOT be made using packets that were retransmitted (and thus for which it is ambiguous whether the reply was for the first instance of the packet or a later instance).

C6) Whenever RTO is computed, if it is less than 1 second then it is rounded up to 1 second. The reason for this rule is that RTOs that do not have a high minimum value are susceptible to unnecessary timeouts [5].

C7) A maximum value may be placed on RTO provided it is at least 60 seconds.

There is no requirement for the clock granularity G used for computing RTT measurements and the different state variables, other than

G1) Whenever RTTVAR is computed, if RTTVAR = 0, then adjust RTTVAR <- G.

Experience has shown that finer clock granularities (<= 100 msec) perform somewhat better than more coarse granularities.

5.3.2 Retransmission Timer Rules

The rules for managing the retransmission timer are as follows:

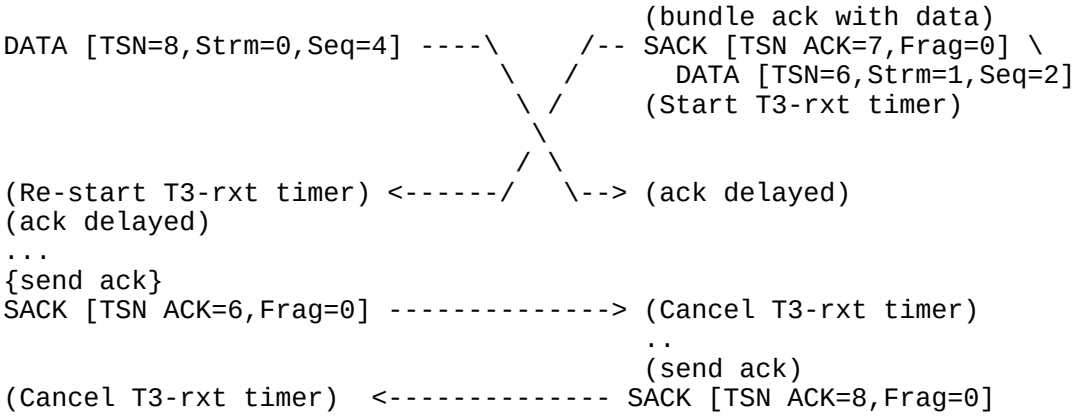
R1) Every time a packet containing data is sent (including a retransmission), if the T3-rxt timer is not running, start it running so that it will expire after RTO seconds. The RTO used here is that obtained after any doubling due to retransmission as discussed in rule E3 below.

R2) Whenever all outstanding data has been acknowledged, turn off the retransmission timer.

R3) Whenever a SACK is received that acknowledges new data chunks including the one with the earliest outstanding TSN (i.e., moving the cumulative ACK point forward), restart T3-rxt retransmission timer so that it will expire after RTO seconds (for the current value of RTO).

The following example shows the use of various timer rules (assuming the receiver uses delayed acks).

```
Endpoint A                                          Endpoint Z
{App sends 2 messages; strm 0}
Data [TSN=7,Strm=0,Seq=3] ------------> (ack delayed)
(Start T3-rxt timer)
                                      {App sends 1 message; strm 1}
```



5.3.3 Handle T3-rxt Expiration

Whenever the retransmission timer T3-rxt expires, do the following:

E1) Adjust ssthresh with rules defined in Section 6.2.3.

E2) Determine how many of the earliest (i.e., lowest TSN) outstanding Data chunks will fit into a single packet, subject to the MTU constraints for the path corresponding to the destination transport address. Call this value K. Retransmit those K data chunks in a single packet, and set cwnd <- MTU.

E3) Set RTO <- RTO * 2 ("back off the timer"). The maximum value discussed in rule C7 above may be used to provide an upper bound to this doubling operation.

E4) Start the retransmission timer, per rule R1 above.

Note, the data sender MAY use a value smaller than the RTO when start the retransmission timer IF the sender has fewer than cwnd octets of outstanding data on the transport address to which the retransmission is being sent.

[EditorÆs Note: if the receiver is multi-homed and the retrans is to be sent to an alternate address, how this rapid retran rule should apply??]

Note that after retransmitting, once a new RTT measurement is obtained (which can only happen when new data has been sent and acknowledged, per rule C5, or for a measurement made from a Heartbeat [see Section 7.3]), the computation in rule C3 is performed, including the computation of RTO, which may result in "collapsing" RTO back down after it has been subject to doubling (rule E3).

The final rule for managing the retransmission timer concerns failover:

F1) Whenever SCTP switches from the current destination transport address to a different one (per section ???), the current retransmission timer is left running. As soon as SCTP transmits a packet containing data to the new transport address, restart the timer, using the RTO value for the path to the new address.

5.4 Multi-homed SCTP Endpoints

An SCTP endpoint is considered multi-homed if there are more than one transport addresses that can be used as a destination address to reach that endpoint.

Moreover, at the sender side, one of the multiple destination addresses of the multi-homed receiver endpoint shall be selected as the primary destination transport address by the UPL (see Section 9 for details).

At association initiation, the initial primary destination transport addresses are:

- for the sender of the INIT message, the transport address that the INIT is sent on. This may be changed upon reception of the destination transport address list in the INIT ACK message from the peer.

- for the sender of the INTI ACK message, any valid transport address obtained from the INIT message.

When the SCTP sender is transmitting to the multi-homed receiver, by default the transmission SHOULD always take place on the primary transport address, unless the SCTP user explicitly specifies the destination transport address to use.

If possible, acknowledgements SHOULD be transmitted to the same destination transport address from which the acknowledged DATA or control chunks were received (Note: when acknowledging multiple DATA chunks in a single SACK, this may not be possible).

Some of the destination transport addresses may become inactive due to either the occurrance of certain error conditions or adjustments from SCTP user.

In the case where the primary destination transport address becomes inactive, or the SCTP user tries to explicitly send to an inactive destination transport address, the SCTP sender should either send the chunk to an alternate active destination transport address, or to report an error. This is implementation specific.

Also, when the SCTP receiver is multi-homed, an SCTP sender SHOULD always try to retransmit a chunk to an active destination transport address that is different from the original destination address used to transmit that chunk. Retransmissions do not affect the total outstanding data count. However, if the data chunk is retransmitted onto a different destination address, the outstanding data counts on the new destination address and the old destination address where the data chunk was originally sent to shall be adjusted accordingly.

5.5 Stream Identifier and Sequence Number

Every DATA chunk MUST carry a valid stream identifier. If a DATA chunk with an invalid stream identifier is received, the receiver shall respond immediately with an ERROR message with cause set to Invalid Stream Identifier (see Section 2.3.9) and discard the DATA chunk.

The stream sequence number in all the streams shall start from 0x0 when the association is established. Also, when the stream sequence number reaches the value 0xffff the next sequence number shall be set to 0x0.

5.6 Ordered and Un-ordered Delivery

Normally, the SCTP receiver shall ensure the DATA chunks within any given stream be delivered to the upper layer according to the order of their stream sequence number. If there are DATA chunks arriving out of order of their stream sequence number, the receiver MUST hold the received DATA chunks from delivery until they are re-ordered.

However, an SCTP sender can indicate that no ordered delivery is required on a particular DATA chunk within the stream by setting the U flag of the DATA chunk to 1.

In this case, the receiver must ignore the sequence number field of the data chunk, bypass the ordering mechanism and immediately delivery the data to the upper layer (after re-assembly if the user data is segmented by the sender).

This provides an effective way of transmitting "out-of-band" data in a given stream. Also, a stream can be used as an "unordered" stream by simply setting the U flag to 1 in each outbound DATA chunk from that stream.

IMPLEMENTATION NOTE: An implementation, when sending an unordered DATA chunk, may choose to place the DATA chunk in an outbound datagram at the head of the outbound transmission queue if possible.

5.7 Report Gaps in Received DATA TSNs

Upon the reception of a new DATA chunk, an SCTP receiver shall examine the continuity of the TSNs received. If the receiver detects that gaps exist in the received DATA chunk sequence, an SACK with fragment reports shall be sent back immediately.

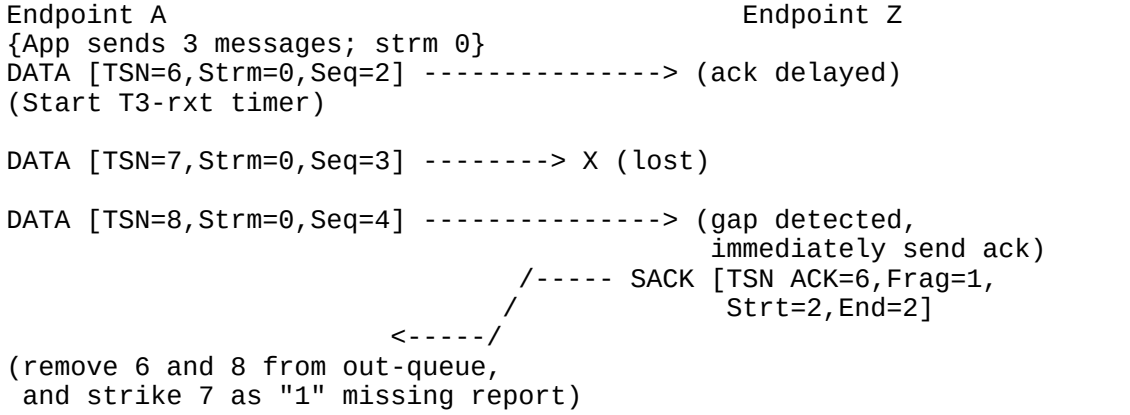
Based on the segment reports from the SACK, the data sender can calculate the missing DATA chunks and make decisions on whether to retransmit them (see Section 5.3 for details).

Multiple gaps can be reported in one single SACK (see Section 2.3.3).

Note that when the data sender is multi-homed, the SCTP receiver SHOULD always try to send the SACK to the same network from where the last DATA chunk was received.

Upon the reception of the SACK, the data sender SHALL remove all DATA chunks which have been acknowledged by the SACK. The data sender MUST also treat all the DATA chunks which fall into the gaps between the fragments reported by the SACK as "missing". The number of "missing" reports for each outstanding DATA chunk MUST be recorded by the data sender in order to make retransmission decision, see Section 6.2.4 for details.

The following example shows the use of SACK to report a gap.



Note: in order to keep the size of the outbound SCTP datagram not to exceed the current path MTU, the maximal number of fragments that can be reported within a single SACK chunk is limited. When a single SACK can not cover all the fragments needed to be reported due to the MTU limitation, the endpoint SHALL send only one SACK, reporting the fragments from the lowest to highest TSNs, within the size limit set by the MTU, and leave the remaining highested TSN fragment numbers unacknowledged.

5.8 CRC-16 Utilization

When sending a datagram, the sender can choose to strengthen the data integrity of the transmission by including the CRC-16 value calculated on the datagram, as described below.

After the datagram is constructed (containing the SCTP common header and one or more control or DATA chunks), the sender shall:

1) fill in the proper Version number and Verification Tag in the common header,

2) set the C Bit to '1' and fill the 16 bit CRC-16 field with '0',

3) calculate the CRC-16 value of the whole datagram, including the SCTP common header and all the chunks,

   It shall be the ones complement of the sum (modulo 2) of:
   a) the remainder of x k (x 15 + x 14 + x 13 + x 12 + x 11 + x 10 + x 9 + x 8 + x 7 + x 6 + x 5 + x 4 + x 3 + x 2 + x + 1) divided (modulo 2) by the generator polynomial x 16 + x 12 + x 5 + 1, where k is the number of bits in the SCTP frame not including the CRC-16 bits, and
   b) the remainder of the division (modulo 2) by the generator polynomial x 16 + x 12 + x 5 + 1, of the product of x 16 by the content of the frame not including the CRC-16 bits.

4) put the resultant value into the CRC-16 field, and leave the rest of the bits unchanged.

When a datagram is received, the receiver MUST first check the C Bit. If the C Bit is set, the receiver SHALL:

1) store the received CRC-16 value aside,

2) replace the 16 bits of the CRC-16 with '0' and calculate a CRC-16 value of the whole received datagram,

3) verify that the calculated CRC-16 value is the same as the received CRC-16 value, If not, the receiver MUST treat the datagram as an invalid SCTP datagram.

If the C Bit is not set, the receiver MUST NOT perform the above CRC-16 check.

The default procedure of handling invalid SCTP datagrams is to silently discard them.

5.9 Segmentation

Segmentation SHALL be performed by the data sender if the user message to be sent has a large size that causes the outbound SCTP datagram size exceeding the current MTU.

When determining when to segment, the SCTP implementation MUST take into account the SCTP datagram header as well as the DATA chunk header. The implementation MAY also take account of the space required for a SACK chunk.

  IMPLEMENTATION NOTE: if the data receiver is multi-homed, the latest MTU of the current primary destination address shall be used.

  IMPLEMENTATION NOTE: if segmentation is not support by the sender, an error should be reported to the sender's SCTP user if the data to be sent has a size exceeding the current MTU. In such cases the Send primitive discussed in Section 9.1 would need to return an error to the upper layer.

Segmentation takes the following steps:

1) the data sender SHALL break the large user message into a series of DATA chunks, each with a total size smaller than the current MTU minus the SCTP common header size (i.e., 8 octets),

2) the data sender MUST then assign, in sequence, a separate TSN to each of the DATA chunks in the series,

3) the data sender MUST also set the B/E bits of the first DATA chunk in the series to '10', the B/E bits of the last DATA chunk in the series to '01', and the B/E bits of all other DATA chunks in the series to '00'.

The data receiver MUST recognize the segmented DATA chunks, by examining the B/E bits in each of the received DATA chunks, and queue the segmented DATA chunks for re-assembly. Then, it shall pass the re-assembled user message to the specific stream for re-ordering and final dispatching.

5.10 Bundling and Multiplexing

An SCTP sender achieves data bundling by simply including multiple DATA chunks in one outbound SCTP datagram. Note that the total size of the resultant SCTP datagram, including the SCTP common header, MUST be less or equal to the current MTU.

> IMPLEMENTATION NOTE: if the data receiver is multi-homed, the latest MTU of the current primary destination address shall be used.

When multiplexing control chunks with DATA chunks, control chunks have the priority and MUST be placed first in the outbound SCTP datagram and be transmitted first. The transmitter MUST transmit DATA chunks within a SCTP datagram in increasing order of TSN.

Partial chunks MUST NOT be placed in a SCTP datagram.

The receiver MUST process the chunks in order in the datagram. The receiver uses the chunk length field to determine the end of a chunk and beginning of the next chunk taking account of the fact that all chunks end on a thirty-two-bit word boundary. If the receiver detects a partial chunk, it MUST drop the chunk.

6. Congestion control

[Editors Notes.. this section is still being reworked and may have some changes]

Congestion control is one of the basic functions in the SCTP protocol. It is likely that adequate resources will be allocated to SCTP traffic to assure prompt delivery of time-critical SCTP data, thus it would be unlikely, during normal operations, that SCTP transmissions encounter severe congestion condition. However SCTP must prepare itself for adverse operational conditions, which can develop upon partial network failures or unexpected traffic surge. In such situations SCTP must follow correct congestion control steps to recover from congestion quickly in order to get data delivered as soon as possible. In the absence of network congestion, these preventive congestion control algorithms will show no impact on the protocol performance.

The congestion control algorithms used by SCTP are based on RFC 2581 [3], "TCP Congestion Control". This section describes how the algorithms defined in RFC 2581 are adopted for use in SCTP. We first list differences in protocol designs between TCP and SCTP, and then describe SCTP's congestion control scheme. The description will use the same terminology as in TCP congestion control whenever appropriate.

6.1 SCTP Differences from TCP Congestion control

One difference from TCP is that Selective Acknowledgment function (SACK) is designed into SCTP, rather than an enhancement that is added to the protocol later as the case for TCP. SCTP SACK carries different semantic meanings from that of TCP SACK. TCP considers the information carried in the SACK as advisory information only. In SCTP, any DATA chunk that has been acknowledged by SACK, including

DATA that arrived at the receiving end out of order, are considered having been delivered to the destination application, and the sender is free to discard the local copy. Thus the value of cwnd controls the number of outstanding data; it is not the upper bound between the highest acknowledged sequence number and the latest DATA chunk that can be sent within the congestion window, as the case in TCP. SCTP SACK leads to different implementations of fast-retransmit and fast-recovery from that of TCP.

The biggest difference between SCTP and TCP, however, is multi-homing. SCTP is designed to establish robust communication associations between two end points each of which may be reachable by more than one transport address. Potentially different addresses may lead to distinguished data paths between the two points, thus ideally one may need a separate set of congestion control parameters for each of the paths. Given SCTP is the first transport protocol whose design specifically takes multihoming issue into consideration, however, there is no experience at this point regarding the use of multiple addresses, or the likelyhood of each address pair representing a separate path. To proceed with caution, we make the following assumptions:

- o the sender does not change the use of source address often, if at all.
- o the sender always uses the same destination address until being instructed by the upper layer otherwise.
- o the sender keeps a separate congestion control parameter set for each of the destination addresses. The parameters should decay if the address is not used for a long enough time period.
- o For each of the destination addresses, do slow-start upon the first transmission to that address.

6.2 SCTP Slow-Start and Congestion Avoidance

The slow start and congestion avoidance algorithms MUST be used by a SCTP sender to control the amount of outstanding data being injected into the network. The congestion control in SCTP is employed in regard to the association, not to an individual stream. In some situations it may be beneficial for a SCTP sender to be more conservative than the algorithms allow, however a SCTP sender MUST NOT be more aggressive than the following algorithms allow.

Like TCP, an SCTP sender uses the following three control variables to regulate its transmission rate.

- o Receiver advertised window size (rwnd), which is set by the receiver based on its available buffer space for incoming packets.
- o Congestion control window (cwnd), which is adjusted by the sender based on observed network conditions.
- o Slow-start threshold (ssthresh), which is also used by the sender to distinguish congestion control and congestion avoidance phases.

SCTP also requires one additional control variable, cwnd2, which is used during congestion avoidance phase to facilitate cwnd adjustment.

6.2.1 Slow-Start

Beginning data transmission into a network with unknown conditions requires SCTP to probe the network to determine the available capacity. The slow start algorithm is used for this purpose at the beginning of a transfer, or after repairing loss detected by the retransmission timer.

- o The initial value of cwnd MUST be less than or equal to 2*MTU octets.
- o The initial value of ssthresh MAY be arbitrarily high (for example, some implementations use the size of the receiver advertised window).
- o Whenever cwnd is greater than zero, the sender is allowed to have cwnd octets of data outstanding on that transport address.
- o When cwnd is less than or equal to ssthresh AND the sender has cwnd or more

data outstanding on the transport address, cwnd is incremented by the total number of octets of all new data chunks acknowledged in each SACK received (piggy-backed or stand-alone SACKs).
- When the sender does not transmit data on a given transport address, the cwnd of the transport address should be adjusted to max(cwnd / 2, 2*MTU) per RTT.

Note: a piggy-backed SACK is one that is bundled with a DATA chunk.

6.2.2 Congestion Avoidance

Whenever cwnd is increased to be equal or greater than ssthresh, cwnd should be incremented by MTU per RTT if at time the sender has cwnd or more octets of data outstanding on that transport address.

In practice an implementation can achieve this goal in the following way:

- cwnd2 is initialized to 0.
- Whenever cwnd is equal or greater than ssthresh, upon each SACK arrival, increase cwnd2 by the total number of octets of all new chunks acknowledged in that SACK.
- When cwnd2 is equal or greater than cwnd and before the arrival of the SACK the sender has cwnd or more octets of data outstanding, increase cwnd by MTU, and reset cwnd2 to (cwnd2 - cwnd).

6.2.3 Congestion Control

Upon detection of packet losses from SACK, the sender should do the following:

```
ssthresh = max(cwnd/2, 2*MTU)
cwnd = ssthresh/2
```

Basically, a packet loss causes cwnd to be cut in half(or 1/4??).

When the T3-rxt timer expires, SCTP should perform slow start by setting cwnd = MTU, and assure that no more than one DATA chunk will be in flight until the sender receives acknowledgment for successful delivery.

6.2.4 Fast Retransmit on Gap Reports

In the absence of data losses, a SCTP receiver performs delayed acknowledgment. However whenever a receiver notices a hole in the arriving TSN sequence, it should start sending a SACK for every packet arrival.

At the sender end, whenever the sender notices a hole in a SACK, it should wait for 3 further SACKs before taking action. If the 3 subsequent SACKs report the same TSN(s) missing, the sender shall:

1) mark the DATA chunk(s) for retransmission,
2) adjust its ssthresh and cwnd according to the formula described in Section 6.2.3.
3) Restart T3-rxt timer if it is running, and
4) start retransmission procedure, as described in Section 5.3.3.

A straightforward implementation of the above requires that the sender keeps a counter for each TSN hole first reported by a SACK; the counter keeps track of whether 3 subsequent SACKs have reported the same hole.

Because cwnd in SCTP bounds the number of outstanding TSN’s, the effect of TCP fast-recovery is achieved automatically with no adjustment to the control window size. [Is this still true??]

6.3 Path MTU Discovery

[Editor's Note: text to be provided by Vern]

6.4 Discussion

[The following discussion needs to be updated for byte-based algorithm]

There is one important difference between the SCTP congestion control, as described above, and TCP congestion control. In the latter the control behavior is measured in unit of packets. That is, upon slow start, a TCP connection doubles cwnd value per RTT as measured by the number of packets. In SCTP, cwnd value is doubled per RTT as measured by the number of DATA chunks. Similarly, during congestion avoidance, in the absence of packet losses a TCP connection increases cwnd by one data packet per RTT, while a SCTP association increases cwnd by one DATA chunk per RTT.

Ideally, congestion control should be performed by controlling the number of outstanding packets. However because the DATA chunk size is a variable, there does not seem a simple and reliable way to translate "one datagram" to a suitable number of chunks. Although the SCTP congestion control design, as described in this section, represents a simple starting point, it may have potential negative impact on the application performance depending on the relative sizes of DATA chunks and packet MTU. For example, if the MTU is 5 times of the average DATA chunk size, it would take 5 times longer for a SCTP association to open up its cwnd to the same size of that of a TCP connection. During a slow start congestion recovery, limiting the transmission to cwnd DATA chunks may lead to sending half-full packets, even when more application data is waiting to be transmitted.

We suggest that the current design be discussed, and revised if deemed necessary.

7. Fault Management

7.1 Endpoint Failure Detection

The data sender shall keep a counter on the total number of consecutive retransmissions to its peer (including retransmissions to ALL the destination transport addresses of the peer if it is multi-homed).

If the value of this counter exceeds the limit defined in the protocol parameter 'Max.Retransmits', the data sender shall consider the peer endpoint unreachable and shall stop transmitting any more data to it. In addition, the data sender shall report the failure to the upper layer, and optionally report back all outstanding datagrams remaining in its outbound queue.

The counter shall be reset each time a datagram is received from the peer endpoint.

7.2 Path Failure Detection

When the remote endpoint is multi-homed, the data sender should keep a 'retrans.count' counter for each of the destination transport addresses of the remote endpoint.

This count should be incremented each time the data sender retransmits an outstanding datagram which was originally sent to the destination transport address.

When the value in 'retrans.count' exceeds half of the value of the protocol parameter 'Max.Retransmits', the data sender should mark the corresponding destination transport address as inactive, and a notification may optionally be sent to the upper layer.

When an outstanding datagram is acknowledged, the data sender should clear the 'retrans.count' counter of the destination transport address to which the datagram was sent. In the case of a retransmitted datagram (due to time-out or SACK) the destination transport address last sent to should be used to determine which 'retrans.count' to clear.

7.3 Path Heartbeat

By default, an SCTP endpoint shall monitor the reachability of the idle destination transport address(es) of its peer by sending HEARTBEAT messages periodically to the destination transport address(es).

A destination transport address should be considered idle if no datagram has been sent to it for a certain period of time, no matter if it is marked active and inactive.

> IMPLEMENTATION NOTE: When multiple idle destination transport addresses exist, it is recommended that the endpoint sends heartbeat messages on a Round-Robin basis, with priority given to active idle destination transport addresses.

The upper layer can optionally initiate the following functions:

A) disable heartbeat on a given association,
B) re-enable heart beat on a given association, and,
C) request an on-demand heartbeat on a given association.

The endpoint should keep a 'heartbeat.sent.count' counter for each destination transport address to record the number of HEARTBEAT messages sent to that destination transport address yet not acknowledged upon.

When the value of this counter reaches the protocol parameter 'Max.HeartBeat.Misses', the endpoint should also mark that destination address as inactive if it is not so marked. The endpoint may also optionally report to the upper layer the un-reachablility of the transport address.

The sender of the HEARTBEAT message should include in the message the current time when the message is sent out.

The receiver of the HEARTBEAT should immediately respond with a HEARTBEAT ACK that contains the time value copied out from the received HEARTBEAT message.

Upon the receipt of the HEARTBEAT ACK, the sender of the HEARTBEAT should clear the 'heartbeat.sent.count' of the destination transport address to which the HEARTBEAT was sent, and mark the destination transport address as active if it is not so marked. The endpoint may also optionally report to the upper layer the reachablility of the transport address. It also should perform an RTT measurement for that destination transport address using the time value carried in the HEARTBEAT ACK message.

The suggested interval for heart beat interval is 4000 ms, and may be dynamically adjusted by adding the current RTT measurement if it is available.

7.4 Verification Tag

Except for INIT, the sender of any SCTP datagram MUST include the destination endpoint's Tag in the Verification Tag field of the message. In the case of INIT, the sender should set the Verification Tag to 0.

When sending a SHUTDOWN ACK message, the sender is allowed to either to use the destination endpoint's Tag or fill the Verification Tag field with 0.

When receiving an SCTP datagram (except for INIT and SHUTDOWN ACK), the receiver MUST ensure that the value in the Verification Tag field of the received message matches its own Tag. If the values do not match, the receiver shall silently discard the datagram and shall NOT process it.

The receiver of a SHUTDOWN ACK message shall accept the message regardless the Verification Tag field is filled with the correct Tag or 0x0.

8. Termination of Association

All existing associations should be terminated when an endpoint exits from service. An association can be terminated by either close or shutdown.

8.1 Close of an Association

When an endpoint decides to close down an association, it shall send an ABORT message to its peer endpoint.

No acknowledgment is required for ABORT message. When the peer endpoint receives the Abort, after checking the Verification Tag, the peer shall remove the association from its record, and shall report the termination to its upper layer.

8.2 Shutdown of an Association

An endpoint in an association may decide to gracefully shutdown the association. This will guarantee that all outstanding datagrams from the peer of the shutdown initiator be delivered before the association terminates.

The initiator shall send a SHUTDOWN message to the peer of the association, and shall include the last highest consecutive TSN it has received from the peer in the 'Highest Consecutive TSN ACK' field. It shall then start the T2-shutdown timer and enter the Shutdown-SENT state. If the timer expires, the initiator must re-send the SHUTDOWN with the updated last TSN received from its peer. When retransmitting the SHUTDOWN, the rules in 5.3 SHALL be followed to determine the proper timer value. The sender of the SHUTDOWN message may also optionally include a SACK to indicate any gaps by bundling both the SACK and SHUTDOWN message together.

Note the sender of a shutdown should limit the number of retransmissions of the shutdown message to the protocol parameter 'Max.Retransmits'. If Max.Retransmits is exceeded the endpoint should destroy the TCB and may report the endpoint has unreachable to the upper layer.

Upon the reception of the SHUTDOWN, the peer shall enter the Shutdown-received state, and shall verify, by checking the TSN ACK field of the message, that all its outstanding datagrams have been received by the initiator.

If there are still outstanding datagrams left, the peer shall mark them for retransmission and start the retransmit procedure as defined in Section 5.3.

While in Shutdown-SENT state, the initiator shall immediately respond to each inbound user datagram from the peer with a SACK and restart the T2-shutdown timer.

If there is no more outstanding datagrams, the peer shall send a SHUTDOWN ACK and then remove all record of the association.

Upon the receipt of the SHUTDOWN ACK, the initiator shall stop the

T2-shutdown timer and remove all record of the association.

Note: that it should be the responsibility of the initiator to assure that all the outstanding datagrams on its side have been resolved before it initiates the shutdown procedure.

Note: an endpoint shall reject any new data request from its upper layer if it is Shutdown-SENT or Shutdown-RECEIVED state until completion of the sequence.

Note: if an endpoint is in a Shutdown-SENT state and receives an INIT message from its peer, it should discard the INIT message and retransmit the shutdown message. The sender of the INIT should respond with a stand-alone SHUTDOWN ACK in an SCTP datagram with the Verification Tag field of its common header set to 0, and let the normal T1-init timer cause the INIT message to be retransmitted and thus restart the association.

9. Interface with Upper Layer

The Upper Layer Protocols (ULP) shall request for services by passing primitives to SCTP and shall receive notifications from SCTP for various events.

The primitives and notifications described in this section should be used as a guideline for implementing SCTP. The following functional description of ULP interface primitives is, at best, fictional. We must warn readers that different SCTP implementations may have different ULP interfaces. However, all SCTPs must provide a certain minimum set of services to guarantee that all SCTP implementations can support the same protocol hierarchy. This section specifies the functional interfaces required of all SCTP implementations.

Sections 9.1 and 9.2 model interface between SCTP and the Upper Layer Protocols. Section 9.3 models interfaces to a Layer Management entity. The Layer Management functions could be implemented as part of the upper layer itself or as a separate entity.

9.1 ULP-to-SCTP

The following sections functionally characterize a ULP/SCTP interface. The notation used is similar to most procedure or function calls in high level languages.

The ULP primitives described below specify the basic functions the SCTP must perform to support inter-process communication. Individual implementations must define their own exact format, and may provide combinations or subsets of the basic functions in single calls.

A) Initialize

Format: INITIALIZE ([local port], [local eligible transport address list])
-> local SCTP instance name

This primitive allows SCTP to initialize its internal data structures and allocate necessary resources for setting up its operation environment. Note that once SCTP is initialized, ULP can communicate directly with other endpoints without re-invoking this primitive.

A local SCTP instance name will be returned to the ULP by the SCTP.

Mandatory attributes:

None.

Optional attributes:

The following types of attributes may be passed along with

the primitive:

- o local port - UDP port number, if ULP wants it to be specified;

- o local eliglible transport address list - A list of eliglible transport addresses that the local SCTP endpoint should bind. By default all transport interface cards should be used by the local SCTP host if no list is given.

B) Associate

Format: ASSOCIATE(local SCTP instance name, destination addr info, stream count [,eligible transport address list] [,timer info])
-> association id [,destination net list] [,outbound stream count]

This primitive allows the upper layer to initiate an association to a specific peer endpoint. The peer endpoint shall be specified by one of the transport addresses which define the endpoint (see section 1.1). If the local SCTP instance has not been initialized, the ASSOCIATE is considered an error. The set of transport addresses specified in the "eligible transport address list" shall be used has valid destinations when sending to the peer endpoint. If this parameter is not specified, the associate command will consider all of the transport addresses returned by the INIT ACK message has valid.

An association id, which is a local handle to the SCTP association, will be returned on successful establishment of the association. If SCTP is not able to open an SCTP association with the peer endpoint, an error is returned.

Implementor’s Note: If ASSOCIATE primitive is implemented as a blocking function call, the ASSOCIATE primitive can return association parameters in addition to the association id upon successful establishment. If ASSOCIATE primitive is implemented as a non-blocking call, only the association id shall be returned and association parameters shall be passed using the COMMUNICATION UP notification.

The association parameters shall include the destination addresses of the peer as well as the outbound stream count. One of the transport address from the set of destination addresses will be used as default primary destination address for sending datagrams to this peer. The returned "destination net list" can be used by the ULP to override the default primary destination transport address or to force sending a datagram on a specific network.

Mandatory attributes:

- o local SCTP instance name - obtained from the initialize operation.

- o destination addr info - specified as one of the transport addresses of the peer endpoint with which the association is to be established.

- o stream count - the number of streams the ULP would like to open at the beginning of the association.

Optional attributes:

- o eligible transport address list - a list of transport addresses that the endpoint is allowed to use for sending datagrams to the peer. By default, all transport addresses of the peer are available.

- o timer info - Timer selection and its operation syntax -- to indicate to SCTP an alternative timer the SCTP should use for its operation.

C) Terminate

Format: TERMINATE(association id)
-> result

Gracefully terminates an association. Any locally queued datagrams will be delivered to the peer. The association will be terminated only after the peer acknowledges all the messages sent. A success code will be returned on successful termination of the association. If attempting to terminate the association results in a failure, an error code shall be returned.

Mandatory attributes:

- association id - local handle to the SCTP association

Optional attributes:

None.

D) Abort

Format: ABORT(association id)
-> result

Ungracefully terminates an association. Any locally queued datagrams will be discarded and an ABORT message is sent to the peer. A success code will be returned on successful abortion of the association. If attempting to abort the association results in a failure, an error code shall be returned.

Mandatory attributes:

- association id - local handle to the SCTP association

Optional attributes:

None.

E) Send

Format: SEND(association id, buffer address, byte count
[,context] [,stream id] [,life time] [,destination transport address]
[,un-order flag] [,no-bundle flag])

This is the main method to send datagrams via SCTP.

Mandatory attributes:

- association id - local handle to the SCTP association
- buffer address - the location where the payload to be transmitted is stored;
- byte count - The size of the payload in number of octets;

Optional attributes:

- context - optional information that will be carried in the sending failure notification to the ULP if the transportation of this datagram fails.
- stream id - to indicate which stream to send the data on. If not specified, stream 0 will be used.
- life time - specifies the life time of the message. The message will not be sent by SCTP after the life time expires. This parameter can be used to avoid efforts to transmit stale datagrams. SCTP notifies the ULP, if the datagram cannot be initiated to transport (i.e. sent to the destination via SCTP’s send primitive) within the life time variable. How ever, the message will be transmitted if a TSN has been assigned to the message before the life time expired.
- destination transport address - specified as one of the destination transport addresses of the peer endpoint to which this message

should be sent. Whenever possible, SCTP should use this destination transport address for sending the datagram, instead of the current primary destination transport address.

- o un-order flag - this flag, if present, indicates that the user would like the data delivered in an un-ordered fashion to the remote peer.

- o no-bundle flag - Instructs SCTP not to bundle the user data with other outbound DATA chunks. Note: SCTP may still bundle even when this flag is present, when faced with network congestion.

F) Set Primary

Format: SETPRIMARY(association id, destination transport address) -> result

Instructs the local SCTP to use the specified destination transport address as primary destination address for sending datagrams.

The result of attempting this operation shall be returned. If the specified destination transport address is not present in the "destination transport address list" returned earlier in an associate command or communication up notification, an error shall be returned.

Mandatory attributes:

- o association id - local handle to the SCTP association

- o destination transport address - specified as one of the transport addresses of the peer endpoint, which should be used as primary address for sending datagrams. This overrides the current primary address information maintained by the local SCTP endpoint.

G) Receive

Format: RECEIVE(association id, buffer address, buffer size [,stream id]) -> byte count [,transport address] [,stream id] [,sequence number]

This primitive shall read the first datagram in the SCTP in-queue to ULP, if there is one available, into the specified buffer. The size of the datagram read, in octets, will be returned. It may, depending on the specific implementation, also return other information such as the sender’s address, the stream id on which it is received, whether there are more datagrams available for retrieval, etc. For ordered messages, their sequence number may also be returned.

Depending upon the implementation, if this primitive is invoked when no datagram is available the implementation should return an indication of this condition or should block the invoking process until data does become available.

Mandatory attributes:

- o association id - local handle to the SCTP association

- o buffer address - the memory location indicated by the ULP to store the received datagram.

- o buffer size - the maximum size of data to be received, in octets.

Optional attributes:

- o stream id - to indicate which stream to receive the data on.

H) Status

Format: STATUS(association id) -> status data

This primitive shall return a data block containing the following information:

receive window size,
send window size,
connection state,
number of buffers awaiting acknowledgement,
number of buffers pending receipt,
primary destination address,
round trip time on primary destination address,
retransmission time out value on primary destination address,
other destination addresses,
round trip times on other destination addresses.

Mandatory attributes:

o association id - local handle to the SCTP association

Optional attributes:

None.

I) Change Heartbeat

Format: CHANGEHEARTBEAT(association id, new state)
-> result

Instructs the local SCTP to enable or disable heart beat on the specified association.

The result of attempting this operation shall be returned.

Mandatory attributes:

o association id - local handle to the SCTP association

o new state - the new state of heart beat for this association (either enabled or disabled).

J) Request HeartBeat

Format: REQUESTHEARTBEAT(association id, transport address)

Instructs the local SCTP to perform a HeartBeat on the specified transport address of the given association. The results of the HeartBeat should update the RTT information.

Mandatory attributes:

o association id - local handle to the SCTP association

o transport address - the transport address of the association on which a heartbeat should be issued.

K) Get RTT Report

Format: GETRTTREPORT(association id, transport address)
-> rtt result

Instructs the local SCTP to report the current RTT measurement on the specified transport address of the given association. The returned result can be an intager containing the most recent RTT in milliseconds.

Mandatory attributes:

o association id - local handle to the SCTP association

o transport address - the transport address of the association on which the RTT measurement is to be reported.

9.2 SCTP-to-ULP

It is assumed that the operating system or application environment provides a means for the SCTP to asynchronously signal the ULP process. When SCTP does signal an ULP process, certain information is passed to the ULP.

A) DATA ARRIVE notification

SCTP shall invoke this notification on the ULP when a datagram is successfully received and ready for retrieval.

The following may be optionally be passed with the notification:

- o association id - local handle to the SCTP association
- o stream id - to indicate which stream the data is received on.

B) SEND FAILURE notification

If a datagram can not be delivered SCTP shall invoke this notification on the ULP.

The following may be optionally be passed with the notification:

- o data - the location ULP can find the un-delivered datagram.
- o context - optional information associated with this datagram (see D in section 9.1).
- o association id - local handle to the SCTP association

C) NETWORK STATUS CHANGE notification

When a destination transport address is marked down (e.g., when SCTP detects a failure), or marked up (e.g., when SCTP detects a recovery), SCTP shall invoke this notification on the ULP.

The following shall be passed with the notification:

- o destination transport address - This indicates the destination transport address of the peer endpoint affected by the change;
- o new-status - This indicates the new status.

D) COMMUNICATION UP notification

This notification is used when SCTP becomes ready to send or receive datagrams, or when a lost communication to an endpoint is restored.

> IMPLEMENTATION NOTE: If ASSOCIATE primitive is implemented as a blocking function call, the association parameters are returned as a result of the ASSOCIATE primitive itself. In that case, COMMUNICATION UP notification is optional at the association initiator’s side.

The following shall be passed with the notification:

- o status - This indicates what type of event that has occurred;
- o association id - local handle to the SCTP association
- o destination transport address list - the set of transport addresses of the peer
- o outbound stream count - the maximum number of streams allowed to be used in this association by the ULP

E) COMMUNICATION LOST notification

When SCTP loses communication to an endpoint completely or detects that the endpoint has performed an abort or graceful shutdown operation, it shall invoke this notification on the ULP.

The following shall be passed with the notification:

- o status - This indicates what type of event that has occurred;
- o association id - local handle to the SCTP association

The following may be optionally passed with the notification:

- o unsent-datagrams - The number and location of un-sent datagrams still in hold by SCTP;
- o unacknowledged-datagrams - The number and location of datagrams that were attempted to be transported to the destination, but were not acknowledged when the loss of communication was detected.
- o last-acked - the sequence number last acked by that peer endpoint;
- o last-sent - the sequence number last sent to that peer endpoint;
- o received-but-not-delivered - datagrams that were received by SCTP but not yet delivered to the ULP.

Note: the un-send data report may not be accurate for those user messages which are segmented by SCTP during transmission.

9.3 Interfaces to Layer Management

This section models interfaces to a Layer Management (LM) entity, which manages resources that have transport layer-wide impact. Layer Management consists of primitives related to the management of SCTP performed as a subset of systems management.

9.3.1 LM-to-SCTP

The following ULP-to-SCTP primitives from section 9.1 could be implemented as part of the LM entity.

INITIALIZE
SETPRIMARY
ABORT
STATUS

9.3.2 SCTP-to-LM

The following SCTP-to-ULP primitives from section 9.2 could be implemented as part of the LM entity.

SEND FAILURE notification
NETWORK STATUS CHANGE notification
COMMUNICATION UP notification
COMMUNICATION LOST notification

10. Security Considerations

10.1 Security Objectives

As a common transport protocol designed to reliably carry time-sensitive user messages, such as billing or signalling messages for telephony services, between two networked endpoints, SCTP has the following security objectives.

- availability of reliable and timely data transport services
- integrity of the user-to-user information carried by SCTP

10.2 SCTP Responses To Potential Threats

It is clear that SCTP may potentially be used in a wide variety of risk situations. It is important for operator(s) of the SCTP Hosts concerned to analyze their particular situations and decide on the appropriate counter-measures.

Where the SCTP Host serves a group of users, it is probably operating as part of a professionally managed corporate or service provider network. It is reasonable to expect that this management includes an appropriate security policy framework. [RFC 2196, "Site Security Handbook", B. Fraser Ed., September 1997] should be consulted for guidance.

The case is more difficult where the SCTP Host is operated by a private user. The service provider with whom that user has a contractual arrangement SHOULD provide help to ensure that the user's site is secure, ranging from advice on configuration through downloaded scripts and security software.

10.2.1 Countering Insider Attacks

The principles of the Site Security Handbook [ ] should be applied to minimize the risk of theft of information or sabotage by insiders. These include publication of security policies, control of access at the physical, software, and network levels, and separation of services.

10.2.2 Protecting against Data Corruption in the Network

Where the risk of undetected errors in datagrams delivered by the lower layer transport services is considered to be too great, additional checksum protection may be required. The question is whether this is appropriately provided as an SCTP service because it is needed by most potential users of SCTP, or whether instead it should be provided by the SCTP user application. (The SCTP protocol overhead, as opposed to the signalling payload, is protected adequately by the UDP checksum and measures taken in SCTP to prevent replay attacks and masquerade.) In any event, the checksum must be specifically designed to ensure that it detects the errors left behind by the UDP checksum.

10.2.3 Protecting Confidentiality

In most cases, the risk of breach of confidentiality applies to the signalling data payload, not to the SCTP or lower-layer protocol overheads. If that is true, encryption of the SCTP user data only may be considered. As with the supplementary checksum service, user data encryption may be performed either by the SCTP user application or as a service of SCTP itself. If it is performed by SCTP, the user data must be encrypted before any checksum is applied.

Particularly for mobile users, the requirement for confidentiality may include the masking of IP addresses and ports. In this case IPSEC ESP should be used instead of application-level encryption. Similarly, where other reasons prompt the use of the IPSEC ESP service, application-level encryption is unnecessary. It will be up to the SCTP Host operators to configure the application appropriately.

Regardless of which level performs the encryption, the IPSEC ISAKMP service should be used for key management.

Operators should consult [RFC 2401, "Security Architecture for the Internet Protocol", S. Kent, R. Atkinson, November 1998] for information on the configuration of IPSEC services between hosts with and without intervening firewalls.

10.2.4 Protecting against Blind Denial of Service Attacks

A blind attack is one where the attacker is unable to intercept or otherwise see the content of data flows passing to and from the target SCTP Host where it is not a party to the association. Blind denial of service attacks may take the form of flooding, masquerade, or improper monopolization of services.

10.2.4.1 Flooding

The objective of flooding is to cause loss of service and incorrect behaviour at target systems through resource exhaustion, interference with legitimate transactions, and exploitation of buffer-related software bugs. Flooding may be directed either at the SCTP Host or at resources in the intervening IP Access Links or the Internetwork. Where the latter entities are the target, flooding will manifest itself as loss of network services, including potentially the breach of any firewalls in place.

In general, protection against flooding begins at the equipment design level, where it includes measures such as:

- avoiding commitment of limited resources before determining that the request for service is legitimate
- giving priority to completion of processing in progress over the acceptance of new work
- identification and removal of duplicate or stale queued requests for service.

Network equipment should be capable of generating an alarm and log if a suspicious increase in traffic occurs. The log should provide information such as the identity of the incoming link and source address(es) used which will help the network or SCTP Host operator to take protective measures. Procedures should be in place for the operator to act on such alarms if a clear pattern of abuse emerges.

The design of SCTP is resistant to flooding attacks, particularly in its use of a four-way start-up handshake, its use of a cookie to defer commitment of resources at the responding SCTP Host until the handshake is completed, and its use of a verification tag to prevent insertion of extraneous messages into the flow of an established association.

10.2.4.2 Masquerade

Masquerade can be used to deny service in several ways:

- by tying up resources at the target SCTP Host to which the impersonated host has limited access. For example, the target host may by policy permit a maximum of one SCTP association with the impersonated SCTP Host. The masquerading attacker may attempt to establish an association purporting to come from the impersonated host so that the latter cannot do so when it requires it.
- by deliberately allowing the impersonation to be detected, thereby provoking counter-measures which cause the impersonated host to be locked out of the target SCTP Host
- by interfering with an established association by inserting extraneous content such as a SHUTDOWN request.

SCTP prevents masquerade through IP spoofing by use of the four-way startup handshake. Because the initial exchange is memoryless, no lockout mechanism is triggered by masquerade attacks. SCTP protects against insertion of extraneous messages into the flow of an established association by use of the verification tag.

Logging of received INIT requests and abnormalities such as unexpected INIT ACKs might be considered as a way to detect patterns of hostile activity. However, the potential usefulness of such logging must be weighed against the increased SCTP startup processing it implies, rendering the SCTP Host more vulnerable to flooding attacks. Logging is pointless without the establishment of operating procedures to review and analyze the logs on a routine

basis.

10.2.4.3 Improper Monopolization of Services

Attacks under this heading are performed openly and legitimately by the attacker. They are directed against fellow users of the target SCTP Host or of the shared resources between the attacker and the target host. Possible attacks include the opening of a large number of associations between the attacker's host and the target, or transfer of large volumes of information within a legitimately-established association.

Such attacks take advantage of policy deficiencies at the target SCTP Host. Defense begins with a contractual prohibition of behaviour directed to denial of service to others. Policy limits should be placed on the number of associations per adjoining SCTP Host. SCTP user applications should be capable of detecting large volumes of illegitimate or "no-op" messages within a given association and either logging or terminating the association as a result, based on local policy.

10.3 Protection against Fraud and Repudiation

The objective of fraud is to obtain services without authorization and specifically without paying for them. In order to achieve this objective, the attacker must induce the SCTP user application at the target SCTP Host to provide the desired service while accepting invalid billing data or failing to collect it. Repudiation is a related problem, since it may occur as a deliberate act of fraud or simply because the repudiating party kept inadequate records of service received.

Potential fraudulent attacks include interception and misuse of authorizing information such as credit card numbers, blind masquerade and replay, and man-in-the middle attacks which modify the messages passing through a target SCTP association in real time.

The interception attack is countered by the confidentiality measures discussed in section 10.2.3 above.

Section 10.2.4.2 describes how SCTP is resistant to blind masquerade attacks, as a result of the four-way startup handshake and the validation tag. The validation tag and TSN together are protections against blind replay attacks, where the replay is into an existing association.

However, SCTP does not protect against man-in-the-middle attacks where the attacker is able to intercept and alter the messages sent and received in an association. Where a significant possibility of such attacks is seen to exist, or where possible repudiation is an issue, the use of the IPSEC AH service is recommended to ensure both the integrity and the authenticity of the messages passed.

SCTP also provides no protection against attacks originating at or beyond the SCTP Host and taking place within the context of an existing association. Prevention of such attacks should be covered by appropriate security policies at the host site, as discussed in section 10.2.1.

11. IANA Consideration

This protocol may be extended through IANA in three ways:
 -- through definition of additional chunk types,
 -- through definition of additional parameter types, or
 -- through definition of additional cause codes within Operation Error chunks

11.1 IETF-defined Chunk Extension

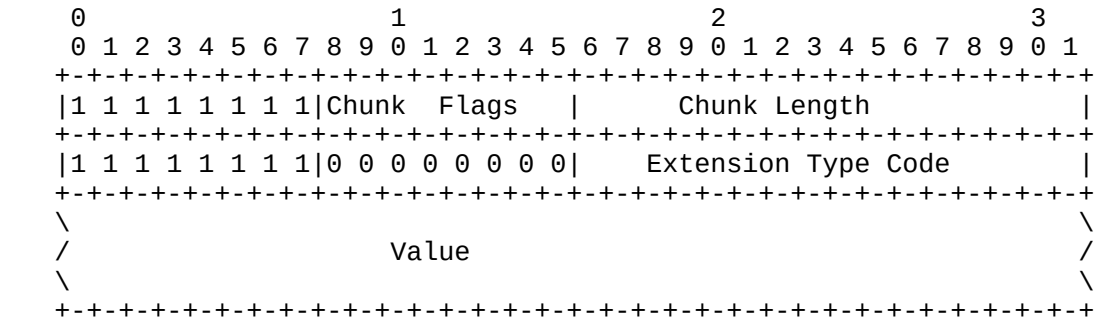
The appropriate use of specific chunk types is an integral part of the SCTP protocol. In consequence, the intention is that new IETF-defined chunk types MUST be supported by standards-track RFC documentation. As a transitional step, a new chunk type MAY be introduced in an Experimental RFC. Chunk type codes MUST remain permanently associated with the original documentation on the basis of which they were allocated. Thus if the RFC supporting a given chunk type is deprecated in favour of a new document, the corresponding chunk type code value is also deprecated and a new code value is allocated in association with the replacement document.

The documentation for a new chunk code type must include the following information:
(a) a long and short name for the new chunk type;
(b) a detailed description of the structure of the chunk, which MUST conform to the basic structure defined in section 2.2;
(c) a detailed definition and description of intended use of each field within the chunk, including the chunk flags if any;
(d) a detailed procedural description of the use of the new chunk type within the operation of the protocol.

If the primary numbering space reserved for IETF use (0x00 to 0xFD) is exhausted, new codes shall subsequently be allocated in the extension range 0x0000 through 0xFFFF. Chunks allocated in this range MUST conform to the following structure:

First word (32 bits):
as shown in section 2.2, with chunk type code equal to 0xFF.

Second word:
first octet MUST be all 1's (0xFF). Next octet MUST be all 0's (0x00). Final two octets contain the allocated extension code value.



11.2 IETF-defined Chunk Parameter Extension

The allocation of a new chunk parameter type code from the IETF numbering space MUST be supported by RFC documentation. As with chunk type codes, parameter type codes are uniquely associated with their supporting document and MUST be replaced if new documentation is provided. This documentation may be Informational, Experimental, or standards-track at the discretion of the IESG. It MUST contain the following information:
(a) Name of the parameter type.
(b) Detailed description of the structure of the parameter field. This structure MUST conform to the general type-length-value format described in section 2.2.1.
(c) Detailed definition of each component of the parameter value.
(d) Detailed description of the intended use of this parameter type, and an indication of whether and under what circumstances multiple instances of this parameter type may be found within the same chunk.

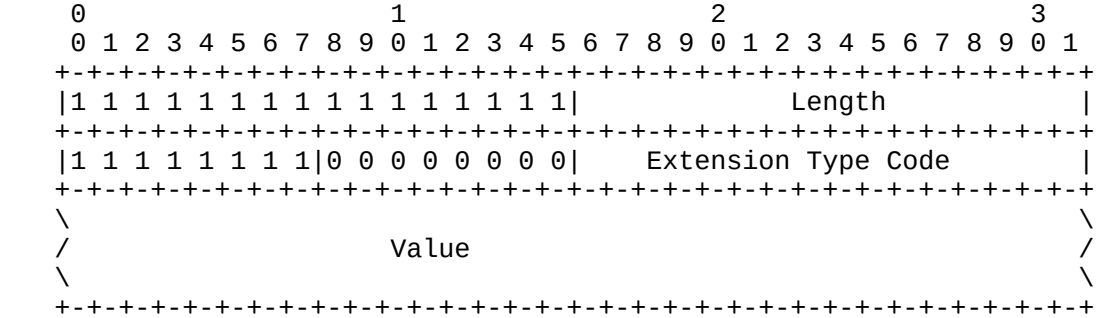
Additional parameter type codes may be allocated initially from the range 0x0000 through 0xFFFD. If this space is exhausted, extension codes shall be allocated in the range 0x0000 through 0xFFFF. Where an extension code has been allocated, the format of the parameter must

conform to the following structure:

First word (32 bits):
  contains the parameter type code 0xFFFF and parameter length as
  described in section 2.2.1.

Second word:
  first octet MUST be all 1's (0xFF). Next octet MUST be all 0's
  (0x00). Final two octets contain the allocated extension code
  value.

The Value portion of the parameter, if any, follows the second word.



11.3 IETF-defined Additional Error Causes

Additional cause codes may be allocated in the range 0x0004 to 0xFFFF upon receipt of any permanently-available public documentation containing the following information:
(a) Name of the error condition.
(b) Detailed description of the conditions under which an SCTP endpoint should issue an Operation Error with this cause code.
(c) Expected action by the SCTP endpoint which receives an Operation Error chunk containing this cause code.
(d) Detailed description of the structure and content of data fields which accompany this cause code.

The initial word (32 bits) of a cause code parameter MUST conform to the format shown in section 2.3.9, i.e.:
 -- first two octets contain the cause code value
 -- last two octets contain length of the cause parameter.

12. Suggested SCTP Protocol Parameter Values

The following protocol parameters are recommended:

```
RTO.Initial              -  3 seconds
Valid.Cookie.Life        -  5 seconds
Max.Retransmits          - 10 attempts
Max.Init.Retransmit      -  8 attempts
Max.HeartBeat.Misses     -  3 attempts
```

Miscellaneous protocol variables/counters:

```
'retrans.count'        - per association counter
'heartbeat.sent.count' - per destination transport address counter
```

13. Acknowledgments

The authors wish to thank Mark Allman, Richard Band, Scott Bradner, Ram Dantu, R. Ezhirpavai, Sally Floyd, Matt Holdrege, Henry Houh, Gary Lehecka, Lyndon Ong, Kelvin Porter, Heinz Prantner, Jarno Rajahalme, A. Sankar, Greg Sidebottom, Brian Wyld, and many others for their invaluable comments.

14. Authors' Addresses

Randall R. Stewart
Motorola, Inc.
1501 W. Shure Drive, #2315
Arlington Heights, IL 60004
USA
Tel: +1-847-632-7438
EMail: rstewar1@email.mot.com

Qiaobing Xie
Motorola, Inc.
1501 W. Shure Drive, #2309
Arlington Heights, IL 60004
USA
Tel: +1-847-632-3028
EMail: qxie1@email.mot.com

Ken Morneault
Cisco Systems Inc.
13615 Dulles Technology Drive
Herndon, VA. 20171
USA
Tel: +1-703-484-3323
EMail:kmorneau@cisco.com

Chip Sharp
Cisco Systems Inc.
7025 Kit Creek Road
Research Triangle Park, NC 27709
USA
Tel: +1-919-472-3121
EMail:chsharp@cisco.com

Hanns Juergen Schwarzbauer
SIEMENS AG
Hofmannstr. 51
81359 Munich
Germany
EMail: HannsJuergen.Schwarzbauer@icn.siemens.de
Tel: +49-89-722-24236

Tom Taylor
Nortel Networks
1852 Lorraine Ave.
Ottawa, Ontario
Canada K1H 6Z8
EMail:taylor@nortelnetworks.com
Tel: +1-613-736-0961

Ian Rytina
Ericsson Australia
37/360 Elizabeth Street
Melbourne, Victoria 3000
Australia
Tel:
EMail:ian.rytina@ericsson.com

Malleswar Kalla
Telcordia Technologies
MCC 1J211R
445 South Street
Morristown, NJ 07960
USA
EMail: kalla@research.telcordia.com
Tel: +1-973-829-5212

Lixia Zhang
UCLA Computer Science Department
4531G Boelter Hall
Los Angeles, CA 90095-1596
USA
Tel: +1-310-825-2695
EMail: lixia@cs.ucla.edu

Vern Paxson
ACIRI
1947 Center St., Suite 600,
Berkeley, CA 94704-1198
USA
Tel: +1-510-642-4274 x 302
EMail: vern@aciri.org

15. References

[1] Eastlake , D. (ed.), "Randomness Recommendations for Security", RFC 1750, December 1994.

[2] ITU-T Recommendation Q.703 "Q.703 - Signaling link", July 1996.

[3] Allman, M., Paxson, V., and Stevens, W., "TCP Congestion Control", RFC 2581, April 1999.

[4] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, August 1999.

[5] M. Allman and V. Paxson, "On Estimating End-to-End Network Path Properties", Proc. SIGCOMM ’99, 1999.

This Internet Draft expires in 6 months from October 1999.