Network Working Group
Internet-Draft
Expires: June 15, 2007

R. Stewart
Editor

December 12, 2006

Stream Control Transmission Protocol
draft-ietf-tsvwg-2960bis-03.txt

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on June 15, 2007.

Copyright Notice

Copyright (C) The IETF Trust (2006).

Abstract

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

SCTP is a reliable transport protocol operating on top of a connectionless packet network such as IP. It offers the following services to its users:

-- acknowledged error-free non-duplicated transfer of user data,
-- data fragmentation to conform to discovered path MTU size,
-- sequenced delivery of user messages within multiple streams, with an option for order-of-arrival delivery of individual user messages,
-- optional bundling of multiple user messages into a single SCTP packet, and
-- network-level fault tolerance through supporting of multi- homing at either or both ends of an association.

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

Table of Contents

1. Introduction 6
1.1. Motivation 6
1.2. Architectural View of SCTP 6
1.3. Functional View of SCTP 7
1.3.1. Association Startup and Takedown 8
1.3.2. Sequenced Delivery within Streams 9
1.3.3. User Data Fragmentation 9
1.3.4. Acknowledgement and Congestion Avoidance 9
1.3.5. Chunk Bundling 10
1.3.6. Packet Validation 10
1.3.7. Path Management 10
1.4. Key Terms 11
1.5. Abbreviations 15
1.6. Serial Number Arithmetic 15
1.7. Changes from RFC2960 16
2. Conventions 16
3. SCTP packet Format 16
3.1. SCTP Common Header Field Descriptions 17
3.2. Chunk Field Descriptions 18
3.2.1. Optional/Variable-length Parameter Format 20
3.2.2. Reporting of Unrecognized Parameters 22
3.3. SCTP Chunk Definitions 23
3.3.1. Payload Data (DATA) (0) 23
3.3.2. Initiation (INIT) (1) 25
3.3.3. Initiation Acknowledgement (INIT ACK) (2): 31
3.3.4. Selective Acknowledgement (SACK) (3): 35
3.3.5. Heartbeat Request (HEARTBEAT) (4): 39
3.3.6. Heartbeat Acknowledgement (HEARTBEAT ACK) (5): . . . 40
3.3.7. Abort Association (ABORT) (6): 41
3.3.8. Shutdown Association (SHUTDOWN) (7): 42
3.3.9. Shutdown Acknowledgement (SHUTDOWN ACK) (8): 42
3.3.10. Operation Error (ERROR) (9): 43

3.3.10.1. Invalid Stream Identifier (1) 45
3.3.10.2. Missing Mandatory Parameter (2) 45
3.3.10.3. Stale Cookie Error (3) 46
3.3.10.4. Out of Resource (4) 46
3.3.10.5. Unresolvable Address (5) 47
3.3.10.6. Unrecognized Chunk Type (6) 47
3.3.10.7. Invalid Mandatory Parameter (7) 48
3.3.10.8. Unrecognized Parameters (8) 48
3.3.10.9. No User Data (9) 48
3.3.10.10. Cookie Received While Shutting Down (10) 49
3.3.10.11. Restart of an Association with New Addresses (11) 49
3.3.10.12. User-Initiated Abort (12) 50
3.3.10.13. Protocol Violation (13) 50
3.3.11. Cookie Echo (COOKIE ECHO) (10): 51
3.3.12. Cookie Acknowledgement (COOKIE ACK) (11): 52
3.3.13. Shutdown Complete (SHUTDOWN COMPLETE) (14): 52
4. SCTP Association State Diagram 53
5. Association Initialization 56
5.1. Normal Establishment of an Association 57
5.1.1. Handle Stream Parameters 58
5.1.2. Handle Address Parameters 59
5.1.3. Generating State Cookie 61
5.1.4. State Cookie Processing 62
5.1.5. State Cookie Authentication 62
5.1.6. An Example of Normal Association Establishment . . . 63
5.2. Handle Duplicate or Unexpected INIT, INIT ACK, COOKIE ECHO, and COOKIE ACK 65
5.2.1. INIT received in COOKIE-WAIT or COOKIE-ECHOED State (Item B) 65
5.2.2. Unexpected INIT in States Other than CLOSED, COOKIE-ECHOED,COOKIE-WAIT and SHUTDOWN-ACK-SENT . . . 66
5.2.3. Unexpected INIT ACK 67
5.2.4. Handle a COOKIE ECHO when a TCB exists 67
5.2.5. Handle Duplicate COOKIE-ACK. 71
5.2.6. Handle Stale COOKIE Error 71
5.3. Other Initialization Issues 71
5.3.1. Selection of Tag Value 71
5.4. Path Verification 72
6. User Data Transfer 73
6.1. Transmission of DATA Chunks 75
6.2. Acknowledgement on Reception of DATA Chunks 77
6.2.1. Processing a Received SACK 80
6.3. Management of Retransmission Timer 82
6.3.1. RTO Calculation 82
6.3.2. Retransmission Timer Rules 83
6.3.3. Handle T3-rtx Expiration 85
6.4. Multi-homed SCTP Endpoints 86

6.4.1. Failover from Inactive Destination Address 87
6.5. Stream Identifier and Stream Sequence Number 87
6.6. Ordered and Unordered Delivery 87
6.7. Report Gaps in Received DATA TSNs 88
6.8. CRC32c Checksum Calculation 89
6.9. Fragmentation and Reassembly 90
6.10. Bundling 91
7. Congestion control 92
7.1. SCTP Differences from TCP Congestion control 93
7.2. SCTP Slow-Start and Congestion Avoidance 94
7.2.1. Slow-Start 95
7.2.2. Congestion Avoidance 96
7.2.3. Congestion Control 97
7.2.4. Fast Retransmit on Gap Reports 97
7.3. Path MTU Discovery 99
8. Fault Management 100
8.1. Endpoint Failure Detection 100
8.2. Path Failure Detection 100
8.3. Path Heartbeat 101
8.4. Handle "Out of the blue" Packets 103
8.5. Verification Tag 104
8.5.1. Exceptions in Verification Tag Rules 105
9. Termination of Association 106
9.1. Abort of an Association 106
9.2. Shutdown of an Association 107
10. Interface with Upper Layer 110
10.1. ULP-to-SCTP 110
10.2. SCTP-to-ULP 121
11. Security Considerations 124
11.1. Security Objectives 124
11.2. SCTP Responses To Potential Threats 124
11.2.1. Countering Insider Attacks 124
11.2.2. Protecting against Data Corruption in the Network . . 124
11.2.3. Protecting Confidentiality 125
11.2.4. Protecting against Blind Denial of Service Attacks . 125
11.2.4.1. Flooding 126
11.2.4.2. Blind Masquerade 127
11.2.4.3. Improper Monopolization of Services 128
11.3. Protection against Fraud and Repudiation 128
11.4. SCTP Interactions with Firewalls 129
11.5. Protection of Non-SCTP Capable Hosts. 129
12. Network Managment Considerations 130
13. Recommended Transmission Control Block (TCB) Parameters . . . 130
13.1. Parameters necessary for the SCTP instance 130
13.2. Parameters necessary per association (i.e. the TCB) . . . 130
13.3. Per Transport Address Data 132
13.4. General Parameters Needed 133
14. IANA Considerations 134

14.1. IETF-defined Chunk Extension 134
14.2. IETF-defined Chunk Parameter Extension 134
14.3. IETF-defined Additional Error Causes 135
14.4. Payload Protocol Identifiers 135
15. Suggested SCTP Protocol Parameter Values 136
16. Acknowledgements 136
Appendix A. Explicit Congestion Notification 137
Appendix B. CRC32c Checksum Calculation 139
Appendix C. ICMP Handling 141
17. References 147
17.1. Normative references 147
17.2. Informative References 148
Author's Address 149
Intellectual Property and Copyright Statements 150

1. Introduction

This section explains the reasoning behind the development of the Stream 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 [RFC0793] 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 [RFC0768]. The limitations which users have wished to bypass include the following:

-- TCP provides both reliable data transfer and strict order-of-transmission 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.

-- TCP is relatively vulnerable to denial of service attacks, such as SYN attacks.

Transport of PSTN signaling 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 a connectionless packet network service such as IP. The remainder of this document assumes SCTP runs on top of IP. The basic service offered by SCTP is the reliable transfer of user

messages between peer SCTP users. It performs this service within the context of an association between two SCTP endpoints. Section 10 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 (Section 1.4) to provide the other endpoint (during association startup) with a list of transport addresses (i.e., multiple IP addresses in combination with an SCTP port) through which that endpoint can be reached and from which it will originate SCTP packets. The association spans transfers over all of the possible source/destination combinations which may be generated from each endpoint's lists.

```
    _____________                                      _____________
   |  SCTP User  |                                    |  SCTP User  |
   | Application |                                    | Application |
   |-------------|                                    |-------------|
   |    SCTP     |                                    |    SCTP     |
   |  Transport  |                                    |  Transport  |
   |   Service   |                                    |   Service   |
   |-------------|                                    |-------------|
   |             |One or more    ----      One or more|             |
   | IP Network  |IP address      \/        IP address| IP Network  |
   |   Service   |appearances     /\       appearances|   Service   |
   |_____________|               ----                 |_____________|

     SCTP Node A |<-------- Network transport ------->| SCTP Node B
```

Figure 1: An SCTP Association

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.

```
                          SCTP User Application

          ----------------------------------------------------
           _____________                  _____________________
          |             |                | Sequenced delivery |
          | Association |                |   within streams   |
          |             |                |____________________|
          |   startup   |
          |             |         ____________________________
          |     and     |        |    User Data Fragmentation |
          |             |        |____________________________|
          |   takedown  |
          |             |         ____________________________
          |             |        |     Acknowledgement        |
          |             |        |          and               |
          |             |        |    Congestion Avoidance    |
          |             |        |____________________________|
          |             |
          |             |         ____________________________
          |             |        |       Chunk Bundling       |
          |             |        |____________________________|
          |             |
          |             |     ____________________________
          |             |    |      Packet Validation         |
          |             |    |____________________________|
          |             |
          |             |     ____________________________
          |             |    |     Path Management            |
          |_____________|    |____________________________|
```

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 (or SEND) primitive in Section 10).

A cookie mechanism, similar to one described by Karn and Simpson in [RFC2522] , is employed during the initialization to provide protection against security attacks. The cookie mechanism uses a four-way handshake, the last two legs of which are allowed to carry user data for fast setup. The startup sequence is described in Section 5 of this document.

SCTP provides for graceful close (i.e., shutdown) of an active association on request from the SCTP user. See the description of the SHUTDOWN primitive in Section 10. SCTP also allows ungraceful close (i.e., abort), either on request from the user (ABORT

primitive) or as a result of an error condition detected within the SCTP layer. Section 9 describes both the graceful and the ungraceful close procedures.

SCTP does not support a half-open state (like TCP) wherein one side may continue sending data while the other end is closed. When either endpoint performs a shutdown, the association on each peer will stop accepting new data from its user and only deliver data in queue at the time of the graceful close (see Section 9).

1.3.2. Sequenced Delivery within Streams

The term "stream" is used in SCTP to refer to a sequence of user messages that are to be delivered to the upper-layer protocol in order with respect to other messages within the same stream. This is in contrast to its usage in TCP, where it refers to a sequence of bytes (in this document a byte is assumed to be eight bits).

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 5.1.1). User messages are associated with stream numbers (SEND, RECEIVE primitives, Section 10). Internally, SCTP assigns a stream sequence number to each message passed to it by the SCTP user. On the receiving side, SCTP ensures that messages 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 message, delivery from other streams may proceed.

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

1.3.3. User Data Fragmentation

When needed, SCTP fragments user messages to ensure that the SCTP packet passed to the lower layer conforms to the path MTU. On receipt, fragments are reassembled into complete messages 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 fragment or unfragmented message. The TSN is independent of any stream 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 stream delivery.

The acknowledgement and congestion avoidance function is responsible for packet retransmission when timely acknowledgement has not been received. Packet retransmission is conditioned by congestion avoidance procedures similar to those used for TCP. See Section 6 and Section 7 for a detailed description of the protocol procedures associated with this function.

1.3.5. Chunk Bundling

As described in Section 3, the SCTP packet 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 of more than one user messages into a single SCTP packet. The chunk bundling function of SCTP is responsible for assembly of the complete SCTP packet and its disassembly at the receiving end.

During times of congestion an SCTP implementation MAY still perform bundling even if the user has requested that SCTP not bundle. The user's disabling of bundling only affects SCTP implementations that may delay a small period of time before transmission (to attempt to encourage bundling). When the user layer disables bundling, this small delay is prohibited but not bundling that is performed during congestion or retransmission.

1.3.6. Packet Validation

A mandatory Verification Tag field and a 32 bit checksum field (see Appendix B for a description of the CRC32c checksum) are included in the SCTP common header. The Verification Tag value is chosen by each end of the association during association startup. Packets received without the expected Verification Tag value are discarded, as a protection against blind masquerade attacks and against stale SCTP packets from a previous association. The CRC32c checksum should be set by the sender of each SCTP packet to provide additional protection against data corruption in the network. The receiver of an SCTP packet with an invalid CRC32c checksum silently discards the packet.

1.3.7. Path Management

The sending SCTP user is able to manipulate the set of transport addresses used as destinations for SCTP packets through the primitives described in Section 10. The SCTP path management function chooses the destination transport address for each outgoing SCTP packet 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 heartbeats

when other packet 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 path is defined for each SCTP endpoint, and is used for normal sending of SCTP packets.

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

Note: Path Management and Packet Validation are done at the same time, so although described separately above, in reality they cannot be performed as separate items.

1.4. Key Terms

Some of 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.

- Active destination transport address: A transport address on a peer endpoint which a transmitting endpoint considers available for receiving user messages.

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

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

- Congestion Window (cwnd): An SCTP variable that limits the data, in number of bytes, a sender can send to a particular destination transport address before receiving an acknowledgement.

- Cumulative TSN Ack Point: The TSN of the last DATA chunk acknowledged via the Cumulative TSN Ack field of a SACK.

- Idle destination address: An address that has not had user messages sent to it within some length of time, normally the HEARTBEAT interval or greater.

- o Inactive destination transport address: An address which is considered inactive due to errors and unavailable to transport user messages.

- o Message = user message: Data submitted to SCTP by the Upper Layer Protocol (ULP).

- o Message Authentication Code (MAC): An integrity check mechanism based on cryptographic hash functions using a secret key. Typically, message authentication codes are used between two parties that share a secret key in order to validate information transmitted between these parties. In SCTP it is used by an endpoint to validate the State Cookie information that is returned from the peer in the COOKIE ECHO chunk. The term "MAC" has different meanings in different contexts. SCTP uses this term with the same meaning as in [RFC2104].

- o Network Byte Order: Most significant byte first, a.k.a., Big Endian.

- o Ordered Message: A user message that is delivered in order with respect to all previous user messages sent within the stream the message was sent on.

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

- o Path: The route taken by the SCTP packets sent by one SCTP endpoint to a specific destination transport address of its peer SCTP endpoint. Sending to different destination transport addresses does not necessarily guarantee getting separate paths.

- o Primary Path: The primary path is the destination and source address that will be put into a packet outbound to the peer endpoint by default. The definition includes the source address since an implementation MAY wish to specify both destination and source address to better control the return path taken by reply chunks and on which interface the packet is transmitted when the data sender is multi-homed.

- o Receiver Window (rwnd): An SCTP variable a data sender uses to store the most recently calculated receiver window of its peer, in number of bytes. This gives the sender an indication of the space available in the receiver's inbound buffer.

- o SCTP association: A protocol relationship between SCTP endpoints, composed of the two SCTP endpoints and protocol state information including Verification Tags and the currently active set of Transmission Sequence Numbers (TSNs), etc. An association can be uniquely identified by the transport addresses used by the endpoints in the association. Two SCTP endpoints MUST NOT have more than one SCTP association between them at any given time.

- o SCTP endpoint: The logical sender/receiver of SCTP packets. On a multi-homed host, an SCTP endpoint is represented to its peers as a combination of a set of eligible destination transport addresses to which SCTP packets can be sent and a set of eligible source transport addresses from which SCTP packets can be received. All transport addresses used by an SCTP endpoint must use the same port number, but can use multiple IP addresses. A transport address used by an SCTP endpoint must not be used by another SCTP endpoint. In other words, a transport address is unique to an SCTP endpoint.

- o SCTP packet (or packet): The unit of data delivery across the interface between SCTP and the connectionless packet network (e.g., IP). An SCTP packet includes the common SCTP header, possible SCTP control chunks, and user data encapsulated within SCTP DATA chunks.

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

- 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 bytes.

- o Stream: A uni-directional logical channel established from one to another associated SCTP endpoint, within which all user messages 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 responsibility of the SCTP user to create and manage these correlations if they are so desired.

- o Stream Sequence Number: A 16-bit sequence number used internally by SCTP to assure sequenced delivery of the user messages within a given stream. One stream sequence number is attached to each user message.

- o Tie-Tags: Two 32-bit random numbers that together make a 64- bit nonce. These Tags are used within a State Cookie and TCB so that a newly restarting association can be linked to the original association within the endpoint that did not restart and yet not reveal the true Verification Tags of an existing association.

- 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.

- o 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.

- o Transport address: A Transport Address is traditionally defined by Network Layer address, Transport Layer protocol and Transport Layer port number. In the case of SCTP running over IP, a transport address is defined by the combination of an IP address and an SCTP port number (where SCTP is the Transport protocol).

- o Unacknowledged TSN (at an SCTP endpoint): A TSN (and the associated DATA chunk) which has been received by the endpoint but for which an acknowledgement has not yet been sent. Or in the opposite case, for a packet that has been sent but no acknowledgement has been received.

- o Unordered Message: Unordered messages are "unordered" with respect to any other message, this includes both other unordered messages as well as other ordered messages. An unordered message might be delivered prior to or later than ordered messages sent on the same stream.

- o User message: The unit of data delivery across the interface between SCTP and its user.

- o Verification Tag: A 32 bit unsigned integer that is randomly generated. The Verification Tag provides a key that allows a receiver to verify that the SCTP packet belongs to the current association and is not an old or stale packet from a previous association.

1.5. Abbreviations

MAC - Message Authentication Code [RFC2104]

RTO - Retransmission Time-out

RTT - Round-trip Time

RTTVAR - Round-trip Time Variation

SCTP - Stream Control Transmission Protocol

SRTT - Smoothed RTT

TCB - Transmission Control Block

TLV - Type-Length-Value Coding Format

TSN - Transmission Sequence Number

ULP - Upper-layer Protocol

1.6. Serial Number Arithmetic

It is essential to remember that the actual Transmission Sequence Number space is finite, though very large. This space ranges from 0 to 2**32 - 1. Since the space is finite, all arithmetic dealing with Transmission Sequence Numbers must be performed modulo 2**32. This unsigned arithmetic preserves the relationship of sequence numbers as they cycle from 2**32 - 1 to 0 again. There are some subtleties to computer modulo arithmetic, so great care should be taken in programming the comparison of such values. When referring to TSNs, the symbol "=<" means "less than or equal"(modulo 2**32).

Comparisons and arithmetic on TSNs in this document SHOULD use Serial Number Arithmetic as defined in [RFC1982] where SERIAL_BITS = 32.

An endpoint SHOULD NOT transmit a DATA chunk with a TSN that is more than 2**31 - 1 above the beginning TSN of its current send window. Doing so will cause problems in comparing TSNs.

Transmission Sequence Numbers wrap around when they reach 2**32 - 1. That is, the next TSN a DATA chunk MUST use after transmitting TSN = 2*32 - 1 is TSN = 0.

Any arithmetic done on Stream Sequence Numbers SHOULD use Serial Number Arithmetic as defined in [RFC1982] where SERIAL_BITS = 16. All other arithmetic and comparisons in this document uses normal

arithmetic.

1.7. Changes from RFC2960

SCTP was originally defined in [RFC2960] which this document obsoletes. Readers interested in the details of the various changes that this document incorporates are asked to consult [RFC4460].

2. Conventions

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, NOT RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in [RFC2119].

3. SCTP packet Format

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

The SCTP packet format is shown below:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Common Header                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Chunk #1                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           ...                                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Chunk #n                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Multiple chunks can be bundled into one SCTP packet up to the MTU size, except for the INIT, INIT ACK, and SHUTDOWN COMPLETE chunks. These chunks MUST NOT be bundled with any other chunk in a packet. See Section 6.10for more details on chunk bundling.

If a user data message doesn't fit into one SCTP packet it can be fragmented into multiple chunks using the procedure defined in Section 6.9.

All integer fields in an SCTP packet MUST be transmitted in network byte order, unless otherwise stated.

3.1. SCTP Common Header Field Descriptions

SCTP Common Header Format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Source Port Number        |     Destination Port Number   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Verification Tag                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Checksum                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Source Port Number: 16 bits (unsigned integer)

This is the SCTP sender's port number. It can be used by the receiver in combination with the source IP address, the SCTP destination port and possibly the destination IP address to identify the association to which this packet belongs. The port number 0 MUST NOT be used.

Destination Port Number: 16 bits (unsigned integer)

This is the SCTP port number to which this packet is destined. The receiving host will use this port number to de-multiplex the SCTP packet to the correct receiving endpoint/application. The port number 0 MUST NOT be used.

Verification Tag: 32 bits (unsigned integer)

The receiver of this packet uses the Verification Tag to validate the sender of this SCTP packet. 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, with the following exceptions:

- A packet containing an INIT chunk MUST have a zero Verification Tag.

- A packet containing a SHUTDOWN-COMPLETE chunk with the T-bit set MUST have the Verification Tag copied from the packet with the SHUTDOWN-ACK chunk.

- A packet containing an ABORT chunk may have the verification tag copied from the packet which caused the ABORT to be sent. For details see Section 8.4 and Section 8.5.

An INIT chunk MUST be the only chunk in the SCTP packet carrying it.

Checksum: 32 bits (unsigned integer)

This field contains the checksum of this SCTP packet. Its calculation is discussed in Section 6.8. SCTP uses the CRC32c algorithm as described in Appendix B for calculating the checksum

3.2. Chunk Field Descriptions

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Chunk Type  | Chunk  Flags  |        Chunk Length           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               \
/                          Chunk Value                          /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Type: 8 bits (unsigned integer)

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

```
ID Value    Chunk Type
-----       ----------
0          - Payload Data (DATA)
1          - Initiation (INIT)
2          - Initiation Acknowledgement (INIT ACK)
3          - Selective Acknowledgement (SACK)
4          - Heartbeat Request (HEARTBEAT)
5          - Heartbeat Acknowledgement (HEARTBEAT ACK)
6          - Abort (ABORT)
7          - Shutdown (SHUTDOWN)
8          - Shutdown Acknowledgement (SHUTDOWN ACK)
9          - Operation Error (ERROR)
10         - State Cookie (COOKIE ECHO)
11         - Cookie Acknowledgement (COOKIE ACK)
12         - Reserved for Explicit Congestion Notification Echo (ECNE)
13         - Reserved for Congestion Window Reduced (CWR)
14         - Shutdown Complete (SHUTDOWN COMPLETE)
15 to 62   - reserved by IETF
63         - IETF-defined Chunk Extensions
64 to 126  - reserved by IETF
127        - IETF-defined Chunk Extensions
128 to 190 - reserved by IETF
191        - IETF-defined Chunk Extensions
192 to 254 - reserved by IETF
255        - IETF-defined Chunk Extensions
```

Chunk Types are encoded such that the highest-order two bits specify the action that must be taken if the processing endpoint does not recognize the Chunk Type.

00 - Stop processing this SCTP packet and discard it, do not process any further chunks within it.

01 - Stop processing this SCTP packet and discard it, do not process any further chunks within it, and report the unrecognized chunk in an 'Unrecognized Chunk Type'.

10 - Skip this chunk and continue processing.

11 - Skip this chunk and continue processing, but report in an ERROR Chunk using the 'Unrecognized Chunk Type' cause of error.

Note: The ECNE and CWR chunk types are reserved for future use of Explicit Congestion Notification (ECN) - see Appendix A.

Chunk Flags: 8 bits

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

Chunk Length: 16 bits (unsigned integer)

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

Chunks (including Type, Length, and Value fields) are padded out by the sender with all zero bytes to be a multiple of 4 bytes long. This padding MUST NOT be more than 3 bytes in total. The Chunk Length value does not include terminating padding of the chunk. However, it does include padding of any variable-length parameter except the last parameter in the chunk. The receiver MUST ignore the padding.

Note: A robust implementation should accept the Chunk whether or not the final padding has been included in the Chunk Length.

Chunk Value: variable length

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

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

SCTP defined chunks are described in detail in Section 3.3. The guidelines for IETF-defined chunk extensions can be found in Section 14.1 of this document.

3.2.1. Optional/Variable-length Parameter Format

Chunk values of SCTP control chunks consist of a chunk-type-specific header of required fields, followed by zero or more parameters. The optional and variable-length parameters contained in a chunk are defined in a Type-Length-Value format as shown below.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Parameter Type       |       Parameter Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               \
/                       Parameter Value                         /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Parameter Type: 16 bits (unsigned integer)

The Type field is a 16 bit identifier of the type of parameter. It takes a value of 0 to 65534.

The value of 65535 is reserved for IETF-defined extensions. Values other than those defined in specific SCTP chunk description are reserved for use by IETF.

Chunk Parameter Length: 16 bits (unsigned integer)

The Parameter Length field contains the size of the parameter in bytes, including the Parameter Type, Parameter Length, and Parameter Value fields. Thus, a parameter with a zero-length Parameter Value field would have a Length field of 4. The Parameter Length does not include any padding bytes.

Chunk Parameter Value: variable-length.

The Parameter Value field contains the actual information to be transferred in the parameter.

The total length of a parameter (including Type, Parameter Length and Value fields) MUST be a multiple of 4 bytes. If the length of the parameter is not a multiple of 4 bytes, the sender pads the Parameter at the end (i.e., after the Parameter Value field) with all zero bytes. The length of the padding is not included in the parameter length field. A sender SHOULD NOT pad with more than 3 bytes. The receiver MUST ignore the padding bytes.

The Parameter Types are encoded such that the highest-order two bits specify the action that must be taken if the processing endpoint does not recognize the Parameter Type.

00 - Stop processing this parameter; do not process any further parameters within this chunk.

01 - Stop processing this parameter, do not process any further parameters within this chunk, and report the unrecognized parameter in an 'Unrecognized Parameter', as described in Section 3.2.2.

10 - Skip this parameter and continue processing.

11 - Skip this parameter and continue processing but report the unrecognized parameter in an 'Unrecognized Parameter', as described in Section 3.2.2.

Please note that in all four cases an INIT-ACK or COOKIE-ECHO chunk is sent. In the 00 or 01 case the processing of the parameters after the unknown parameter is canceled, but no processing already done is rolled back.

The actual SCTP parameters are defined in the specific SCTP chunk sections. The rules for IETF-defined parameter extensions are defined in Section 14.2. Note that a parameter type MUST be unique across all chunks. For example, the parameter type '5' is used to represent an IPv4 address (see Section 3.2.2). The value '5' then is reserved across all chunks to represent an IPv4 address and MUST NOT be reused with a different meaning in any other chunk.

3.2.2. Reporting of Unrecognized Parameters

If the receiver of an INIT chunk detects unrecognized parameters and has to report them according to Section 3.2.1, it MUST put the 'Unrecognized Parameter' parameter(s) in the INIT-ACK chunk sent in response to the INIT-chunk. Note that if the receiver of the INIT chunk is NOT going to establish an association (e.g., due to lack of resources), an 'Unrecognized Parameter' would NOT be included with any ABORT being sent to the sender of the INIT.

If the receiver of an INIT-ACK chunk detects unrecognized parameters and has to report them according to Section 3.2.1, it SHOULD bundle the ERROR chunk containing the 'Unrecognized Parameters' error cause with the COOKIE-ECHO chunk sent in response to the INIT-ACK chunk. If the receiver of the INIT-ACK cannot bundle the COOKIE-ECHO chunk with the ERROR chunk, the ERROR chunk MAY be sent separately but not before the COOKIE-ACK has been received.

Note: Any time a COOKIE-ECHO is sent in a packet, it MUST be the first chunk.

3.3. SCTP Chunk Definitions

This section defines the format of the different SCTP chunk types.

3.3.1. Payload Data (DATA) (0)

The following format MUST be used for the DATA chunk:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 0    | Reserved|U|B|E|    Length                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             TSN                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Stream Identifier S      |   Stream Sequence Number n    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Payload Protocol Identifier                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               \
/                 User Data (seq n of Stream S)                 /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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 to '1', indicates that this is an unordered DATA chunk, and there is no Stream Sequence Number assigned to this DATA chunk. Therefore, the receiver MUST ignore the Stream Sequence Number field.

After re-assembly (if necessary), unordered DATA chunks MUST be dispatched to the upper layer by the receiver without any attempt to re-order.

If an unordered user message is fragmented, each fragment of the message MUST have its U bit set to '1'.

B bit: 1 bit

The (B)eginning fragment bit, if set, indicates the first fragment of a user message.

E bit: 1 bit

The (E)nding fragment bit, if set, indicates the last fragment of a user message.

An unfragmented user message shall have both the B and E bits set to '1'. Setting both B and E bits to '0' indicates a middle fragment of a multi-fragment user message, as summarized in the following table:

B	E	Description
1	0	First piece of a fragmented user message
0	0	Middle piece of a fragmented user message
0	1	Last piece of a fragmented user message
1	1	Unfragmented Message

Table 1: Fragment Description Flags

When a user message is fragmented into multiple chunks, the TSNs are used by the receiver to reassemble the message. This means that the TSNs for each fragment of a fragmented user message MUST be strictly sequential.

Length: 16 bits (unsigned integer)

This field indicates the length of the DATA chunk in bytes from the beginning of the type field to the end of the user data field excluding any padding. A DATA chunk with one byte of user data will have Length set to 17 (indicating 17 bytes).

A DATA chunk with a user data field of length L will have the length field set to (16 + L) (indicating 16+L bytes) where L MUST be greater than 0.

TSN : 32 bits (unsigned integer)

This value represents the TSN for this DATA chunk. The valid range of TSN is from 0 to 4294967295 (2**32 - 1). TSN wraps back to 0 after reaching 4294967295.

Stream Identifier S: 16 bits (unsigned integer)

Identifies the stream to which the following user data belongs.

Stream Sequence Number n: 16 bits (unsigned integer)

This value represents the stream sequence number of the following user data within the stream S. Valid range is 0 to 65535.

When a user message is fragmented by SCTP for transport, the same stream sequence number MUST be carried in each of the fragments of the message.

Payload Protocol Identifier: 32 bits (unsigned integer)

This value represents an application (or upper layer) specified protocol identifier. This value is passed to SCTP by its upper layer and sent to its peer. This identifier is not used by SCTP but can be used by certain network entities, as well as by the peer application, to identify the type of information being carried in this DATA chunk. This field must be sent even in fragmented DATA chunks (to make sure it is available for agents in the middle of the network). Note that this field is NOT touched by an SCTP implementation, therefore its byte order is NOT necessarily Big Endian. The upper layer is responsible for any byte order conversions to this field.

The value 0 indicates no application identifier is specified by the upper layer for this payload data.

User Data: variable length

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

3.3.2. Initiation (INIT) (1)

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 1    |  Chunk Flags  |      Chunk Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Initiate Tag                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Advertised Receiver Window Credit (a_rwnd)          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Number of Outbound Streams   |  Number of Inbound Streams    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Initial TSN                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               \
/              Optional/Variable-Length Parameters              /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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
Advertised Receiver Window Credit	Mandatory
Number of Outbound Streams	Mandatory
Number of Inbound Streams	Mandatory
Initial TSN	Mandatory

Variable Parameters	Status	Type Value
IPv4 Address (Note 1)	Optional	5
IPv6 Address (Note 1)	Optional	6
Cookie Preservative	Optional	9
Reserved for ECN Capable (Note 2)	Optional	32768 (0x8000)
Host Name Address (Note 3)	Optional	11
Supported Address Types (Note 4)	Optional	12

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

Note 2: The ECN capable field is reserved for future use of Explicit Congestion Notification.

Note 3: An INIT chunk MUST NOT contain more than one Host Name address parameter. Moreover, the sender of the INIT MUST NOT combine any other address types with the Host Name address in the INIT. The receiver of INIT MUST ignore any other address types if the Host Name address parameter is present in the received INIT chunk.

Note 4: This parameter, when present, specifies all the address types the sending endpoint can support. The absence of this parameter indicates that the sending endpoint can support any address type.

IMPLEMENTATION NOTE: If an INIT chunk is received with known parameters that are not optional parameters of the INIT chunk then the receiver SHOULD process the INIT chunk and send back an INIT-ACK. The receiver of the INIT chunk MAY bundle an ERROR chunk with the COOKIE-ACK chunk later. However, restrictive implementations MAY send back an ABORT chunk in response to the INIT chunk.

The Chunk Flags field in INIT is reserved and all bits in it should be set to 0 by the sender and ignored by the receiver. The sequence of parameters within an INIT can be processed in any order.

Initiate Tag: 32 bits (unsigned integer)

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 packet that the receiver of the INIT transmits within this association.

The Initiate Tag is allowed to have any value except 0. See Section 5.3.1 for more on the selection of the tag value.

If the value of the Initiate Tag in a received INIT chunk is found to be 0, the receiver MUST treat it as an error and close the association by transmitting an ABORT.

Advertised Receiver Window Credit (a_rwnd): 32 bits (unsigned integer)

This value represents the dedicated buffer space, in number of bytes, the sender of the INIT has reserved in association with this window. During the life of the association this buffer space SHOULD not be lessened (i.e. dedicated buffers taken away from this association); however, an endpoint MAY change the value of a_rwnd it sends in SACK chunks.

Number of Outbound Streams (OS): 16 bits (unsigned integer)

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.

Note: A receiver of an INIT with the OS value set to 0 SHOULD abort the association.

Number of Inbound Streams (MIS) : 16 bits (unsigned integer)

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.

Note: There is no negotiation of the actual number of streams but instead the two endpoints will use the min(requested, offered). See Section 5.1.1 for details.

Note: A receiver of an INIT with the MIS value of 0 SHOULD abort the association.

Initial TSN (I-TSN) : 32 bits (unsigned integer)

Defines the initial TSN that the sender will use. The valid range is from 0 to 4294967295. This field MAY be set to the value of the Initiate Tag field.

3.3.2.1. Optional/Variable Length Parameters in INIT

The following parameters follow the Type-Length-Value format as defined in Section 3.2.1. Any Type-Length-Value fields MUST come after the fixed-length fields defined in the previous section.

IPv4 Address Parameter (5)

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Type = 5               |      Length = 8               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        IPv4 Address                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IPv4 Address: 32 bits (unsigned integer)

Contains an IPv4 address of the sending endpoint. It is binary encoded.

IPv6 Address Parameter (6)

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Type = 6           |          Length = 20          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                         IPv6 Address                          |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IPv6 Address: 128 bit (unsigned integer)

Contains an IPv6 address of the sending endpoint. It is binary encoded.

Note: A sender MUST NOT use an IPv4-mapped IPv6 address [RFC2373]. but should instead use an IPv4 Address Parameter for an IPv4 address.

Combined with the Source Port Number in the SCTP common header, the value passed in an IPv4 or IPv6 Address parameter indicates a transport address the sender of the INIT will support for the association being initiated. That is, during the lifetime of this association, this IP address can appear in the source address field of an IP datagram sent from the sender of the INIT, and can be used as a destination address of an IP datagram sent from the receiver of the INIT.

More than one IP Address parameter can be included in an INIT chunk when the INIT sender is multi-homed. Moreover, a multi-homed endpoint may have access to different types of network, thus more than one address type can be present in one INIT chunk, i.e., IPv4 and IPv6 addresses are allowed in the same INIT chunk.

If the INIT contains at least one IP Address parameter, then the source address of the IP datagram containing the INIT chunk and any additional address(es) provided within the INIT can be used as destinations by the endpoint receiving the INIT. If the INIT does not contain any IP Address parameters, the endpoint receiving the INIT MUST use the source address associated with the received IP datagram as its sole destination address for the association.

Note that not using any IP address parameters in the INIT and INIT-ACK is an alternative to make an association more likely to work across a NAT box.

Cookie Preservative (9)

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Type = 9             |          Length = 8           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Suggested Cookie Life-span Increment (msec.)          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Suggested Cookie Life-span Increment: 32 bits (unsigned integer)

This parameter indicates to the receiver how much increment in milliseconds the sender wishes the receiver to add to its default cookie life-span.

This optional parameter should be added to the INIT chunk by the sender when it re-attempts establishing an association with a peer to which its previous attempt of establishing the association failed due to a stale cookie operation error. The receiver MAY choose to ignore the suggested cookie life-span increase for its own security reasons.

Host Name Address (11)

The sender of INIT uses this parameter to pass its Host Name (in place of its IP addresses) to its peer. The peer is responsible for resolving the name. Using this parameter might make it more likely for the association to work across a NAT box.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Type = 11            |          Length               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                           Host Name                           /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Host Name: variable length

This field contains a host name in "host name syntax" per RFC1123 Section 2.1 [RFC1123]. The method for resolving the host name is out of scope of SCTP.

Note: At least one null terminator is included in the Host Name string and must be included in the length.

Supported Address Types (12)

The sender of INIT uses this parameter to list all the address types it can support.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Type = 12            |          Length               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Address Type #1        |        Address Type #2        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             ......                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Address Type: 16 bits (unsigned integer)

This is filled with the type value of the corresponding address TLV (e.g., IPv4 = 5, IPv6 = 6, Hostname = 11).

3.3.3. Initiation Acknowledgement (INIT ACK) (2):

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

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

The format of the INIT ACK chunk is shown below:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 2    |  Chunk Flags  |      Chunk Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Initiate Tag                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Advertised Receiver Window Credit                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Number of Outbound Streams   |  Number of Inbound Streams    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Initial TSN                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               \
/              Optional/Variable-Length Parameters              /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Initiate Tag: 32 bits (unsigned integer)

The receiver of the INIT ACK records the value of the Initiate Tag parameter. This value MUST be placed into the Verification Tag field of every SCTP packet that the INIT ACK receiver transmits within this association.

The Initiate Tag MUST NOT take the value 0. See Section 5.3.1 for more on the selection of the Initiate Tag value.

If the value of the Initiate Tag in a received INIT ACK chunk is found to be 0, the receiver MUST destroy the association discarding its TCB. The receiver MAY send an ABORT for debugging purpose.

Advertised Receiver Window Credit (a_rwnd): 32 bits (unsigned integer)

This value represents the dedicated buffer space, in number of bytes, the sender of the INIT ACK has reserved in association with this window. During the life of the association this buffer space SHOULD not be lessened (i.e. dedicated buffers taken away from this association).

Number of Outbound Streams (OS): 16 bits (unsigned integer)

Defines the number of outbound streams the sender of this INIT ACK chunk wishes to create in this association. The value of 0 MUST NOT be used, and the value MUST NOT be greater than the MIS value sent in the INIT chunk.

Note: A receiver of an INIT ACK with the OS value set to 0 SHOULD destroy the association discarding its TCB.

Number of Inbound Streams (MIS) : 16 bits (unsigned integer)

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

Note: There is no negotiation of the actual number of streams but instead the two endpoints will use the min(requested, offered). See Section 5.1.1 for details.

Note: A receiver of an INIT ACK with the MIS value set to 0 SHOULD destroy the association discarding its TCB.

Initial TSN (I-TSN) : 32 bits (unsigned integer)

Defines the initial TSN that the INIT-ACK sender will use. The valid range is from 0 to 4294967295. This field MAY be set to the value of the Initiate Tag field.

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

Variable Parameters	Status	Type Value
State Cookie	Mandatory	7
IPv4 Address (Note 1)	Optional	5
IPv6 Address (Note 1)	Optional	6
Unrecognized Parameter	Optional	8
Reserved for ECN Capable (Note 2)	Optional	32768 (0x8000)
Host Name Address (Note 3)	Optional	11

Note 1: The INIT ACK chunks can contain any number of IP address parameters that can be IPv4 and/or IPv6 in any combination.

Note 2: The ECN capable field is reserved for future use of Explicit Congestion Notification.

Note 3: The INIT ACK chunks MUST NOT contain more than one Host Name address parameter. Moreover, the sender of the INIT ACK MUST NOT combine any other address types with the Host Name address in the INIT ACK. The receiver of the INIT ACK MUST ignore any other address types if the Host Name address parameter is present.

IMPLEMENTATION NOTE: An implementation MUST be prepared to receive a INIT ACK that is quite large (more than 1500 bytes) due to the variable size of the state cookie AND the variable address list. For example if a responder to the INIT has 1000 IPv4 addresses it wishes to send, it would need at least 8,000 bytes to encode this in the INIT ACK.

IMPLEMENTATION NOTE: If an INIT-ACK chunk is received with known parameters that are not optional parameters of the INIT-ACK chunk, then the receiver SHOULD process the INIT-ACK chunk and send back a COOKIE-ECHO. The receiver of the INIT-ACK chunk MAY bundle an ERROR chunk with the COOKIE-ECHO chunk. However, restrictive implementations MAY send back an ABORT chunk in response to the INIT-ACK chunk.

In combination with the Source Port carried in the SCTP common header, each IP Address parameter in the INIT ACK indicates to the receiver of the INIT ACK a valid transport address supported by the sender of the INIT ACK for the lifetime of the association being initiated.

If the INIT ACK contains at least one IP Address parameter, then the source address of the IP datagram containing the INIT ACK and any additional address(es) provided within the INIT ACK may be used as destinations by the receiver of the INIT-ACK. If the INIT ACK does not contain any IP Address parameters, the receiver of the INIT-ACK MUST use the source address associated with the received IP datagram as its sole destination address for the association.

The State Cookie and Unrecognized Parameters use the Type-Length-Value format as defined in Section 3.2.1 and are described below. The other fields are defined the same as their counterparts in the INIT chunk.

3.3.3.1. Optional or Variable Length Parameters

State Cookie

Parameter Type Value: 7

Parameter Length: variable size, depending on Size of Cookie

Parameter Value:

This parameter value MUST contain all the necessary state and parameter information required for the sender of this INIT ACK to create the association, along with a Message Authentication Code (MAC). See Section 5.1.3 for details on State Cookie definition.

Unrecognized Parameter:

Parameter Type Value: 8

Parameter Length: Variable Size.

Parameter Value:
This parameter is returned to the originator of the INIT chunk when the INIT contains an unrecognized parameter which has a value that indicates that it should be reported to the sender. This parameter value field will contain unrecognized parameters copied from the INIT chunk complete with Parameter Type, Length and Value fields.

3.3.4. Selective Acknowledgement (SACK) (3):

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

The SACK MUST contain the Cumulative TSN Ack, Advertised Receiver Window Credit (a_rwnd), Number of Gap Ack Blocks, and Number of Duplicate TSNs fields.

By definition, the value of the Cumulative TSN Ack parameter is the last TSN received before a break in the sequence of received TSNs occurs; the next TSN value following this one has not yet been received at the endpoint sending the SACK. This parameter therefore acknowledges receipt of all TSNs less than or equal to its value.

The handling of a_rwnd by the receiver of the SACK is discussed in detail in Section 6.2.1.

The SACK also contains zero or more Gap Ack Blocks. Each Gap Ack Block acknowledges a subsequence of TSNs received following a break in the sequence of received TSNs. By definition, all TSNs acknowledged by Gap Ack Blocks are greater than the value of the Cumulative TSN Ack.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 3    |Chunk  Flags   |      Chunk Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Cumulative TSN Ack                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Advertised Receiver Window Credit (a_rwnd)           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Number of Gap Ack Blocks = N  |  Number of Duplicate TSNs = X |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Gap Ack Block #1 Start       |   Gap Ack Block #1 End        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                                                               /
\                              ...                              \
/                                                               /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Gap Ack Block #N Start      |  Gap Ack Block #N End         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Duplicate TSN 1                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                                                               /
\                              ...                              \
/                                                               /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Duplicate TSN X                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Flags: 8 bits

Set to all zeros on transmit and ignored on receipt.

Cumulative TSN Ack: 32 bits (unsigned integer)

This parameter contains the TSN of the last DATA chunk received in sequence before a gap. In the case where no DATA chunk has been received, this value is set to the peer's Initial TSN minus one.

Advertised Receiver Window Credit (a_rwnd): 32 bits (unsigned integer)

This field indicates the updated receive buffer space in bytes of the sender of this SACK, see Section 6.2.1 for details.

Number of Gap Ack Blocks: 16 bits (unsigned integer)

Indicates the number of Gap Ack Blocks included in this SACK.

Number of Duplicate TSNs: 16 bit

This field contains the number of duplicate TSNs the endpoint has received. Each duplicate TSN is listed following the Gap Ack Block list.

Gap Ack Blocks:

These fields contain the Gap Ack Blocks. They are repeated for each Gap Ack Block up to the number of Gap Ack Blocks defined in the Number of Gap Ack Blocks field. All DATA chunks with TSNs greater than or equal to (Cumulative TSN Ack + Gap Ack Block Start) and less than or equal to (Cumulative TSN Ack + Gap Ack Block End) of each Gap Ack Block are assumed to have been received correctly.

Gap Ack Block Start: 16 bits (unsigned integer)

Indicates the Start offset TSN for this Gap Ack Block. To calculate the actual TSN number the Cumulative TSN Ack is added to this offset number. This calculated TSN identifies the first TSN in this Gap Ack Block that has been received.

Gap Ack Block End: 16 bits (unsigned integer)

Indicates the End offset TSN for this Gap Ack Block. To calculate the actual TSN number the Cumulative TSN Ack is added to this offset number. This calculated TSN identifies the TSN of the last DATA chunk received in this Gap Ack Block.

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

```
                          ----------
                          | TSN=17 |
                          ----------
                          |        | <- still missing
                          ----------
                          | TSN=15 |
                          ----------
                          | TSN=14 |
                          ----------
                          |        | <- still missing
                          ----------
                          | TSN=12 |
                          ----------
                          | TSN=11 |
                          ----------
                          | TSN=10 |
                          ----------
```

then, the parameter part of the SACK MUST be constructed as follows (assuming the new a_rwnd is set to 4660 by the sender):

```
                    +--------------------------------+
                    |   Cumulative TSN Ack = 12      |
                    +--------------------------------+
                    |        a_rwnd = 4660           |
                    +----------------+---------------+
                    | num of block=2 | num of dup=0  |
                    +----------------+---------------+
                    |block #1 strt=2 |block #1 end=3 |
                    +----------------+---------------+
                    |block #2 strt=5 |block #2 end=5 |
                    +----------------+---------------+
```

Duplicate TSN: 32 bits (unsigned integer)

Indicates the number of times a TSN was received in duplicate since the last SACK was sent. Every time a receiver gets a duplicate TSN (before sending the SACK) it adds it to the list of duplicates. The duplicate count is re-initialized to zero after sending each SACK.

For example, if a receiver were to get the TSN 19 three times it would list 19 twice in the outbound SACK. After sending the SACK if it received yet one more TSN 19 it would list 19 as a duplicate once in the next outgoing SACK.

3.3.5. Heartbeat Request (HEARTBEAT) (4):

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

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 4    | Chunk  Flags  |      Heartbeat Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               \
/            Heartbeat Information TLV (Variable-Length)        /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Flags: 8 bits

Set to zero on transmit and ignored on receipt.

Heartbeat Length: 16 bits (unsigned integer)

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

Heartbeat Information: variable length

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

Variable Parameters	Status	Type Value
Heartbeat Info	Mandatory	1

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Heartbeat Info Type=1      |         HB Info Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                  Sender-specific Heartbeat Info               /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Sender-specific Heartbeat Info field should normally include information about the sender's current time when this HEARTBEAT chunk is sent and the destination transport address to which this HEARTBEAT is sent (see Section 8.3).

3.3.6. Heartbeat Acknowledgement (HEARTBEAT ACK) (5):

An endpoint should send this chunk to its peer endpoint as a response to a HEARTBEAT chunk (see Section 8.3). A HEARTBEAT ACK is always sent to the source IP address of the IP datagram containing the HEARTBEAT chunk to which this ack is responding.

The parameter field contains a variable length opaque data structure.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 5    | Chunk  Flags  |    Heartbeat Ack Length       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               \
/            Heartbeat Information TLV (Variable-Length)        /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Flags: 8 bits

Set to zero on transmit and ignored on receipt.

Heartbeat Ack Length: 16 bits (unsigned integer)

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

Heartbeat Information: variable length

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

Variable Parameters	Status	Type Value
Heartbeat Info	Mandatory	1

3.3.7. Abort Association (ABORT) (6):

The ABORT chunk is sent to the peer of an association to close the association. The ABORT chunk may contain Cause Parameters to inform the receiver about the reason of the abort. DATA chunks MUST NOT be bundled with ABORT. Control chunks (except for INIT, INIT ACK and SHUTDOWN COMPLETE) MAY be bundled with an ABORT but they MUST be placed before the ABORT in the SCTP packet, or they will be ignored by the receiver.

If an endpoint receives an ABORT with a format error or no TCB is found, it MUST silently discard it. Moreover, under any circumstances, an endpoint that receives an ABORT MUST NOT respond to that ABORT by sending an ABORT of its own.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 6    |Reserved     |T|           Length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               \
/                   zero or more Error Causes                   /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Flags: 8 bits

Reserved: 7 bits

Set to 0 on transmit and ignored on receipt.

T bit: 1 bit

The T bit is set to 0 if the sender filled in the Verification Tag expected by the peer. If the Verification Tag is reflected, the T bit MUST be set to 1. Reflecting means that the sent Verification Tag is the same as the received one.

Note: Special rules apply to this chunk for verification, please see Section 8.5.1 for details.

Length: 16 bits (unsigned integer)

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

See Section 3.3.10 for Error Cause definitions.

3.3.8. Shutdown Association (SHUTDOWN) (7):

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 7    | Chunk  Flags  |      Length = 8               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Cumulative TSN Ack                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Flags: 8 bits

Set to zero on transmit and ignored on receipt.

Length: 16 bits (unsigned integer)

Indicates the length of the parameter. Set to 8.

Cumulative TSN Ack: 32 bits (unsigned integer)

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

Note: Since the SHUTDOWN message does not contain Gap Ack Blocks, it cannot be used to acknowledge TSNs received out of order. In a SACK, lack of Gap Ack Blocks that were previously included indicates that the data receiver reneged on the associated DATA chunks. Since SHUTDOWN does not contain Gap Ack Blocks, the receiver of the SHUTDOWN shouldn't interpret the lack of a Gap Ack Block as a renege. (see Section 6.2 for information on reneging)

3.3.9. Shutdown Acknowledgement (SHUTDOWN ACK) (8):

This chunk MUST be used to acknowledge the receipt of the SHUTDOWN chunk at the completion of the shutdown process, see Section 9.2 for details.

The SHUTDOWN ACK chunk has no parameters.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 8    |Chunk  Flags   |      Length = 4               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Flags: 8 bits

Set to zero on transmit and ignored on receipt.

3.3.10. Operation Error (ERROR) (9):

An endpoint sends this chunk to its peer endpoint to notify it of certain error conditions. It contains one or more error causes. An Operation Error is not considered fatal in and of itself, but may be used with an ABORT chunk to report a fatal condition. It has the following parameters:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 9    | Chunk  Flags  |           Length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               \
/                    one or more Error Causes                   /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Flags: 8 bits

Set to zero on transmit and ignored on receipt.

Length: 16 bits (unsigned integer)

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

Error causes are defined as variable-length parameters using the format described in 3.2.1, i.e.:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Cause Code          |       Cause Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                    Cause-specific Information                 /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Cause Code: 16 bits (unsigned integer)

Defines the type of error conditions being reported.

Cause Code Value	Cause Code
1	Invalid Stream Identifier
2	Missing Mandatory Parameter
3	Stale Cookie Error
4	Out of Resource
5	Unresolvable Address
6	Unrecognized Chunk Type
7	Invalid Mandatory Parameter
8	Unrecognized Parameters
9	No User Data
10	Cookie Received While Shutting Down
11	Restart of an Association with New Addresses
12	User Initiated Abort
13	Protocol Violation

Cause Length: 16 bits (unsigned integer)

Set to the size of the parameter in bytes, 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.

Section 3.3.10.1 - Section 3.3.10.13 define error causes for SCTP. Guidelines for the IETF to define new error cause values are discussed in Section 14.3.

3.3.10.1. Invalid Stream Identifier (1)

Cause of error

Invalid Stream Identifier: Indicates endpoint received a DATA chunk sent to a nonexistent stream.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Cause Code=1              |      Cause Length=8           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Stream Identifier      |         (Reserved)            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Stream Identifier: 16 bits (unsigned integer)

Contains the Stream Identifier of the DATA chunk received in error.

Reserved: 16 bits

This field is reserved. It is set to all 0's on transmit and Ignored on receipt.

3.3.10.2. Missing Mandatory Parameter (2)

Cause of error

Missing Mandatory Parameter: Indicates that one or more mandatory TLV parameters are missing in a received INIT or INIT ACK.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Cause Code=2              |      Cause Length=8+N*2       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Number of missing params=N                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Missing Param Type #1       |   Missing Param Type #2       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Missing Param Type #N-1     |   Missing Param Type #N       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Number of Missing params: 32 bits (unsigned integer)

This field contains the number of parameters contained in the Cause-specific Information field.

Missing Param Type: 16 bits (unsigned integer)

Each field will contain the missing mandatory parameter number.

3.3.10.3. Stale Cookie Error (3)

Cause of error

Stale Cookie Error: Indicates the receipt of a valid State Cookie that has expired.

```
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |     Cause Code=3              |       Cause Length=8          |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                 Measure of Staleness (usec.)                  |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Measure of Staleness: 32 bits (unsigned integer)

This field contains the difference, in microseconds, between the current time and the time the State Cookie expired.

The sender of this error cause MAY choose to report how long past expiration the State Cookie is by including a non-zero value in the Measure of Staleness field. If the sender does not wish to provide this information it should set the Measure of Staleness field to the value of zero.

3.3.10.4. Out of Resource (4)

Cause of error

Out of Resource: Indicates that the sender is out of resource. This is usually sent in combination with or within an ABORT.

```
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |     Cause Code=4              |      Cause Length=4           |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

3.3.10.5. Unresolvable Address (5)

Cause of error

Unresolvable Address: Indicates that the sender is not able to resolve the specified address parameter (e.g., type of address is not supported by the sender). This is usually sent in combination with or within an ABORT.

```
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |     Cause Code=5              |      Cause Length             |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 /                  Unresolvable Address                         /
 \                                                               \
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Unresolvable Address: variable length

The unresolvable address field contains the complete Type, Length and Value of the address parameter (or Host Name parameter) that contains the unresolvable address or host name.

3.3.10.6. Unrecognized Chunk Type (6)

Cause of error

Unrecognized Chunk Type: This error cause is returned to the originator of the chunk if the receiver does not understand the chunk and the upper bits of the 'Chunk Type' are set to 01 or 11.

```
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |     Cause Code=6              |      Cause Length             |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 /                  Unrecognized Chunk                           /
 \                                                               \
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Unrecognized Chunk: variable length

The Unrecognized Chunk field contains the unrecognized Chunk from the SCTP packet complete with Chunk Type, Chunk Flags and Chunk Length.

3.3.10.7. Invalid Mandatory Parameter (7)

Cause of error

Invalid Mandatory Parameter: This error cause is returned to the originator of an INIT or INIT ACK chunk when one of the mandatory parameters is set to a invalid value.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Cause Code=7              |      Cause Length=4           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

3.3.10.8. Unrecognized Parameters (8)

Cause of error

Unrecognized Parameters: This error cause is returned to the originator of the INIT ACK chunk if the receiver does not recognize one or more Optional TLV parameters in the INIT ACK chunk.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Cause Code=8              |      Cause Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                  Unrecognized Parameters                      /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Unrecognized Parameters: variable length

The Unrecognized Parameters field contains the unrecognized parameters copied from the INIT ACK chunk complete with TLV. This error cause is normally contained in an ERROR chunk bundled with the COOKIE ECHO chunk when responding to the INIT ACK, when the sender of the COOKIE ECHO chunk wishes to report unrecognized parameters.

3.3.10.9. No User Data (9)

Cause of error

No User Data: This error cause is returned to the originator of a

DATA chunk if a received DATA chunk has no user data.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Cause Code=9              |      Cause Length=8           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                  TSN value                                    /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

TSN value: 32 bits (+unsigned integer)

The TSN value field contains the TSN of the DATA chunk received with no user data field.

This cause code is normally returned in an ABORT chunk (see Section 6.2)

3.3.10.10. Cookie Received While Shutting Down (10)

Cause of error

Cookie Received While Shutting Down: A COOKIE ECHO was received While the endpoint was in SHUTDOWN-ACK-SENT state. This error is usually returned in an ERROR chunk bundled with the retransmitted SHUTDOWN ACK.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Cause Code=10             |      Cause Length=4           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

3.3.10.11. Restart of an Association with New Addresses (11)

Cause of error

Restart of an association with new addresses: An INIT was received on an existing association. But the INIT added addresses to the association that were previously NOT part of the association. The new addresses are listed in the error code. This ERROR is normally sent as part of an ABORT refusing the INIT (see Section 5.2).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Cause Code=11         |      Cause Length=Variable    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                       New Address TLVs                        /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Note: Each New Address TLV is an exact copy of the TLV that was found in the INIT chunk that was new, including the Parameter Type and the Parameter length.

3.3.10.12. User-Initiated Abort (12)

Cause of error

This error cause MAY be included in ABORT chunks that are sent because of an upper layer request. The upper layer can specify an Upper Layer Abort Reason that is transported by SCTP transparently and MAY be delivered to the upper layer protocol at the peer.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Cause Code=12         |      Cause Length=Variable    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                    Upper Layer Abort Reason                   /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

3.3.10.13. Protocol Violation (13)

Cause of error

This error cause MAY be included in ABORT chunks that are sent because an SCTP endpoint detects a protocol violation of the peer that is not covered by the error causes described in Section 3.3.10.1 to Section 3.3.10.12. An implementation MAY provide additional information specifying what kind of protocol violation has been detected.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Cause Code=13         |      Cause Length=Variable    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                    Additional Information                     /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

3.3.11. Cookie Echo (COOKIE ECHO) (10):

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 packet.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 10   |Chunk  Flags   |         Length                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                     Cookie                                    /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Flags: 8 bit

Set to zero on transmit and ignored on receipt.

Length: 16 bits (unsigned integer)

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

Cookie: variable size

This field must contain the exact cookie received in the State Cookie parameter from the previous INIT ACK.

An implementation SHOULD make the cookie as small as possible to insure interoperability.

Note: A Cookie Echo does NOT contain a State Cookie Parameter; instead, the data within the State Cookie's Parameter Value becomes the data within the Cookie Echo's Chunk Value. This allows an implementation to change only the first two bytes of the State Cookie parameter to become a Cookie Echo Chunk.

3.3.12. Cookie Acknowledgement (COOKIE ACK) (11):

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 11   |Chunk  Flags   |     Length = 4                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Flags: 8 bits

Set to zero on transmit and ignored on receipt.

3.3.13. Shutdown Complete (SHUTDOWN COMPLETE) (14):

This chunk MUST be used to acknowledge the receipt of the SHUTDOWN ACK chunk at the completion of the shutdown process, see Section 9.2 for details.

The SHUTDOWN COMPLETE chunk has no parameters.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 14   |Reserved     |T|      Length = 4               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Flags: 8 bits

Reserved: 7 bits

Set to 0 on transmit and ignored on receipt.

T bit: 1 bit

The T bit is set to 0 if the sender filled in the Verification Tag expected by the peer. If the Verification Tag is reflected, the T bit MUST be set to 1. Reflecting means that the sent Verification Tag is the same as the received one.

Note: Special rules apply to this chunk for verification, please see Section 8.5.1 for details.

4. SCTP Association State Diagram

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

o SCTP user primitive calls, e.g., [ASSOCIATE], [SHUTDOWN], [ABORT],

o Reception of INIT, COOKIE ECHO, 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 ECHO chunk type vs. State Cookie parameter. If more than one event/message can occur which causes a state transition it is labeled (A), (B) etc.

```
                    -----          -------- (frm any state)
                  /       \      /  rcv ABORT      [ABORT]
 rcv INIT        |         |    |   ----------  or ----------
 --------------- |         v    v   delete TCB     snd ABORT
 generate Cookie  \    +---------+                delete TCB
 snd INIT ACK       ---|  CLOSED |
                       +---------+
                        /      \      [ASSOCIATE]
                       /        \     ---------------
                      |          |    create TCB
                      |          |    snd INIT
                      |          |    strt init timer
       rcv valid      |          |
```

```
        COOKIE  ECHO     |          v
    (1) ---------------- |      +------------+
        create TCB       |      | COOKIE-WAIT| (2)
        snd COOKIE ACK   |      +------------+
                         |          |
                         |          |    rcv INIT ACK
                         |          |    -----------------
                         |          |    snd COOKIE ECHO
                         |          |    stop init timer
                         |          |    strt cookie timer
                         |          v
                         |      +--------------+
                         |      | COOKIE-ECHOED| (3)
                         |      +--------------+
                         |          |
                         |          |    rcv COOKIE ACK
                         |          |    -----------------
                         |          |    stop cookie timer
                         v          v
                       +---------------+
                       |  ESTABLISHED  |
                       +---------------+


                     (from the ESTABLISHED state only)
                                    |
                                    |
                          /--------+--------\
       [SHUTDOWN]         /                   \
       -------------------|                    |
       check outstanding  |                    |
       DATA chunks        |                    |
                          v                    |
                    +---------+                |
                    |SHUTDOWN-|                | rcv SHUTDOWN
                    |PENDING  |                |------------------
                    +---------+                | check outstanding
                          |                    | DATA chunks
     No more outstanding  |                    |
     ---------------------|                    |
     snd SHUTDOWN         |                    |
     strt shutdown timer  |                    |
                          v                    v
                    +---------+        +-----------+
                (4) |SHUTDOWN-|        | SHUTDOWN- |  (5,6)
                    |SENT     |        | RECEIVED  |
                    +---------+        +-----------+
                          |  \                 |
```

```
    (A) rcv SHUTDOWN ACK  |   \               |
    ----------------------|    \              |
    stop shutdown timer   |     \rcv:SHUTDOWN |
    send SHUTDOWN COMPLETE|      \  (B)       |
    delete TCB            |       \           |
                          |        \          | No more outstanding
                          |         \         |-----------------
                          |          \        | send SHUTDOWN ACK
    (B)rcv SHUTDOWN       |           \       | strt shutdown timer
    ----------------------|            \      |
    send SHUTDOWN ACK     |             \     |
    start shutdown timer  |              \    |
    move to SHUTDOWN-     |               \   |
    ACK-SENT              |                |  |
                          |                v  |
                          |            +-----------+
                          |            | SHUTDOWN- | (7)
                          |            | ACK-SENT  |
                          |            +----------+-
                          |                  | (C)rcv SHUTDOWN COMPLETE
                          |                  |-----------------
                          |                  | stop shutdown timer
                          |                  | delete TCB
                          |                  |
                          |                  | (D)rcv SHUTDOWN ACK
                          |                  |--------------
                          |                  | stop shutdown timer
                          |                  | send SHUTDOWN COMPLETE
                          |                  | delete TCB
                          |                  |
                          \    +---------+    /
                           \-->| CLOSED  |<--/
                               +---------+
```

Figure 3: State Transition Diagram of SCTP

Notes:

1) If the State Cookie in the received COOKIE ECHO is invalid (i.e., failed to pass the integrity check), the receiver MUST silently discard the packet. Or, if the received State Cookie is expired (see Section 5.1.5), the receiver MUST send back an ERROR chunk. In either case, the receiver stays in the CLOSED state.

2) If the T1-init timer expires, the endpoint MUST retransmit INIT and re-start the T1-init timer without changing state. This MUST be repeated up to 'Max.Init.Retransmits' times. After that, the endpoint MUST abort the initialization process and report the error to SCTP user.

3) If the T1-cookie timer expires, the endpoint MUST retransmit COOKIE ECHO and re-start the T1-cookie timer without changing state. This MUST be repeated up to 'Max.Init.Retransmits' times. After that, the endpoint MUST abort the initialization process and report the error to SCTP user.

4) In SHUTDOWN-SENT state the endpoint MUST 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.

6) In SHUTDOWN-RECEIVED state, the endpoint MUST transmit or retransmit data and leave this state when all data in queue is transmitted.

7) In SHUTDOWN-ACK-SENT state, the endpoint MUST NOT accept any new send request from its SCTP user.

The CLOSED state is used to indicate that an association is not created (i.e., doesn't exist).

5. 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 Section 10.1 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 are open for data transfer on both ends (see Section 5.1.1).

5.1. 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" first sends an INIT chunk to "Z". In the INIT, "A" must provide its Verification Tag (Tag_A) in the Initiate Tag field. Tag_A SHOULD be a random number in the range of 1 to 4294967295 (see Section 5.3.1 for Tag value selection). After sending the INIT, "A" starts the T1-init timer and enters the COOKIE-WAIT state.

B) "Z" shall respond immediately with an INIT ACK chunk. The destination IP address of the INIT ACK MUST be set to the source IP address of the INIT to which this INIT ACK is responding. In the response, besides filling in other parameters, "Z" must set the Verification Tag field to Tag_A, and also provide its own Verification Tag (Tag_Z) in the Initiate Tag field.

   Moreover, "Z" MUST generate and send along with the INIT ACK a State Cookie. See Section 5.1.3 for State Cookie generation.

   Note: After sending out INIT ACK with the State Cookie parameter, "Z" MUST 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 stop the T1-init timer and leave COOKIE-WAIT state. "A" shall then send the State Cookie received in the INIT ACK chunk in a COOKIE ECHO chunk, start the T1-cookie timer, and enter the COOKIE-ECHOED state.

   Note: The COOKIE ECHO chunk can be bundled with any pending outbound DATA chunks, but it MUST be the first chunk in the packet and until the COOKIE ACK is returned the sender MUST NOT send any other packets to the peer.

D) Upon reception of the COOKIE ECHO chunk, Endpoint "Z" will reply with a COOKIE ACK chunk after building a TCB and moving to the ESTABLISHED state. A COOKIE ACK chunk may be bundled with any pending DATA chunks (and/or SACK chunks), but the COOKIE ACK chunk MUST be the first chunk in the packet.

   IMPLEMENTATION NOTE: An implementation may choose to send the Communication Up notification to the SCTP user upon reception of a valid COOKIE ECHO chunk.

E) Upon reception of the COOKIE ACK, endpoint "A" will move from the COOKIE-ECHOED state to the ESTABLISHED state, stopping the T1-cookie timer. It may also notify its ULP about the successful establishment of the association with a Communication Up notification (see Section 10).

An INIT or INIT ACK chunk MUST NOT be bundled with any other chunk. They MUST be the only chunks present in the SCTP packets that carry them.

An endpoint MUST send the INIT ACK to the IP address from which it received the INIT.

Note: T1-init timer and T1-cookie timer shall follow the same rules given in Section 6.3.

If an endpoint receives an INIT, INIT ACK, or COOKIE ECHO chunk but decides not to establish the new association due to missing mandatory parameters in the received INIT or INIT ACK, invalid parameter values, or lack of local resources, it SHOULD respond with an ABORT chunk. It SHOULD also specify the cause of abort, such as the type of the missing mandatory parameters, etc., by including the error cause parameters with the ABORT chunk. The Verification Tag field in the common header of the outbound SCTP packet containing the ABORT chunk MUST be set to the Initiate Tag value of the peer.

Note that a COOKIE ECHO chunk that does NOT pass the integrity check is NOT considered an 'invalid parameter' and requires special handling see Section 5.1

After the reception of the first DATA chunk in an association the endpoint MUST immediately respond with a SACK to acknowledge the DATA chunk. Subsequent acknowledgements should be done as described in Section 6.2.

When the TCB is created, each endpoint MUST set its internal Cumulative TSN Ack Point to the value of its transmitted Initial TSN minus one.

IMPLEMENTATION NOTE: The IP addresses and SCTP port are generally used as the key to find the TCB within an SCTP instance.

5.1.1. Handle Stream Parameters

In the INIT and INIT ACK chunks, the sender of the chunk MUST indicate the number of outbound streams (OS) it wishes to have in the association, as well as the maximum inbound streams (MIS) it will accept from the other endpoint.

After receiving the stream configuration information from the other side, each endpoint MUST 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 use MIS outbound streams and MAY report any shortage to the upper layer. The upper layer can then choose to abort the association if the resource shortage is unacceptable.

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

5.1.2. Handle Address Parameters

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

A) If there are no address parameters present in the received INIT or INIT ACK chunk, the endpoint shall take the source IP address from which the chunk arrives and record it, in combination with the SCTP source port number, as the only destination transport address for this peer.

B) If there is a Host Name parameter present in the received INIT or INIT ACK chunk, the endpoint shall resolve that host name to a list of IP address(es) and derive the transport address(es) of this peer by combining the resolved IP address(es) with the SCTP source port.

   The endpoint MUST ignore any other IP address parameters if they are also present in the received INIT or INIT ACK chunk.

   The time at which the receiver of an INIT resolves the host name has potential security implications to SCTP. If the receiver of an INIT resolves the host name upon the reception of the chunk, and the mechanism the receiver uses to resolve the host name involves potential long delay (e.g. DNS query), the receiver may open itself up to resource attacks for the period of time while it is waiting for the name resolution results before it can build the State Cookie and release local resources.

   Therefore, in cases where the name translation involves potential long delay, the receiver of the INIT MUST postpone the name resolution till the reception of the COOKIE ECHO chunk from the peer. In such a case, the receiver of the INIT SHOULD build the State Cookie using the received Host Name (instead of destination

transport addresses) and send the INIT ACK to the source IP address from which the INIT was received.

The receiver of an INIT ACK shall always immediately attempt to resolve the name upon the reception of the chunk.

The receiver of the INIT or INIT ACK MUST NOT send user data (piggy-backed or stand-alone) to its peer until the host name is successfully resolved.

If the name resolution is not successful, the endpoint MUST immediately send an ABORT with "Unresolvable Address" error cause to its peer. The ABORT shall be sent to the source IP address from which the last peer packet was received.

C) If there are only IPv4/IPv6 addresses present in the received INIT or INIT ACK chunk, the receiver MUST derive and record all the transport addresses from the received chunk AND the source IP address that sent the INIT or INIT ACK. The transport addresses are derived by the combination of SCTP source port (from the common header) and the IP address parameter(s) carried in the INIT or INIT ACK chunk and the source IP address of the IP datagram. The receiver should use only these transport addresses as destination transport addresses when sending subsequent packets to its peer.

D) An INIT or INIT ACK chunk MUST be treated as belonging to an already established association (or one in the process of being established) if the use of any of the valid address parameters contained within the chunk would identify an existing TCB.

IMPLEMENTATION NOTE: In some cases (e.g., when the implementation doesn't control the source IP address that is used for transmitting), an endpoint might need to include in its INIT or INIT ACK all possible IP addresses from which packets to the peer could be transmitted.

After all transport addresses are derived from the INIT or INIT ACK chunk using the above rules, the endpoint shall select one of the transport addresses as the initial primary path.

Note: The INIT-ACK MUST be sent to the source address of the INIT.

The sender of INIT may include a 'Supported Address Types' parameter in the INIT to indicate what types of address are acceptable. When this parameter is present, the receiver of INIT (initiatee) MUST either use one of the address types indicated in the Supported Address Types parameter when responding to the INIT, or abort the association with an "Unresolvable Address" error cause if it is

unwilling or incapable of using any of the address types indicated by its peer.

IMPLEMENTATION NOTE: In the case that the receiver of an INIT ACK fails to resolve the address parameter due to an unsupported type, it can abort the initiation process and then attempt a re-initiation by using a 'Supported Address Types' parameter in the new INIT to indicate what types of address it prefers.

IMPLEMENTATION NOTE: If an SCTP endpoint that only supports either IPv4 or IPv6 receives IPv4 and IPv6 addresses in an INIT or INIT- ACK chunk from its peer, it MUST use all the addresses belonging to the supported address family. The other addresses MAY be ignored. The endpoint SHOULD NOT respond with any kind of error indication.

IMPLEMENTATION NOTE: If an SCTP endpoint lists in the 'Supported Address Types' parameter either IPv4 or IPv6, but uses the other family for sending the packet containing the INIT chunk, or if it also lists addresses of the other family in the INIT chunk, then the address family that is not listed in the 'Supported Address Types' parameter SHOULD also be considered as supported by the receiver of the INIT chunk. The receiver of the INIT chunk SHOULD NOT respond with any kind of error indication.

5.1.3. Generating State Cookie

When sending an INIT ACK as a response to an INIT chunk, the sender of INIT ACK creates a State Cookie and sends it in the State Cookie parameter of the INIT ACK. Inside this State Cookie, the sender should include a MAC (see [RFC2104] for an example), a time stamp on when the State Cookie is created, and the lifespan of the State Cookie, along with all the information necessary for it to establish the association.

The following steps SHOULD be taken to generate the State Cookie:

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

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) From the TCB, identify and collect the minimal subset of information needed to re-create the TCB, and generate a MAC using this subset of information and a secret key (see [RFC2104] for an example of generating a MAC), and

4) Generate the State Cookie by combining this subset of information and the resultant MAC.

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

The hashing method used to generate the MAC is strictly a private matter for the receiver of the INIT chunk. The use of a MAC is mandatory to prevent denial of service attacks. The secret key SHOULD be random ([RFC4086] provides some information on randomness guidelines); it SHOULD be changed reasonably frequently, and the timestamp in the State Cookie MAY be used to determine which key should be used to verify the MAC.

An implementation SHOULD make the cookie as small as possible to insure interoperability.

5.1.4. State Cookie Processing

When an endpoint (in the COOKIE WAIT state) receives an INIT ACK chunk with a State Cookie parameter, it MUST immediately send a COOKIE ECHO chunk to its peer with the received State Cookie. The sender MAY also add any pending DATA chunks to the packet after the COOKIE ECHO chunk.

The endpoint shall also start the T1-cookie timer after sending out the COOKIE ECHO chunk. If the timer expires, the endpoint shall retransmit the COOKIE ECHO chunk and restart the T1-cookie timer. This is repeated until either a COOKIE ACK is received or 'Max.Init.Retransmits' is reached causing the peer endpoint to be marked unreachable (and thus the association enters the CLOSED state).

5.1.5. State Cookie Authentication

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

1) Compute a MAC using the TCB data carried in the State Cookie and the secret key (note the timestamp in the State Cookie MAY be used to determine which secret key to use). Reference [RFC2104] can be used as a guideline for generating the MAC,

2) Authenticate the State Cookie as one that it previously generated by comparing the computed MAC against the one carried in the State Cookie. If this comparison fails, the SCTP packet, including the COOKIE ECHO and any DATA chunks, should be silently discarded,

3) Compare the port numbers and the Verification Tag contained within the COOKIE ECHO chunk to the actual port numbers and the Verification Tag within the SCTP common header of the received packet. If these values do not match, the packet MUST be silently discarded.

4) Compare the creation timestamp in the State Cookie to the current local time. If the elapsed time is longer than the lifespan carried in the State Cookie, then the packet, including the COOKIE ECHO and any attached DATA chunks, SHOULD be discarded, and the endpoint MUST transmit an ERROR chunk with a "Stale Cookie" error cause to the peer endpoint.

5) If the State Cookie is valid, create an association to the sender of the COOKIE ECHO chunk with the information in the TCB data carried in the COOKIE ECHO and enter the ESTABLISHED state.

6) Send a COOKIE ACK chunk to the peer acknowledging receipt of the COOKIE ECHO. The COOKIE ACK MAY be bundled with an outbound DATA chunk or SACK chunk; however, the COOKIE ACK MUST be the first chunk in the SCTP packet.

7) Immediately acknowledge any DATA chunk bundled with the COOKIE ECHO with a SACK (subsequent DATA chunk acknowledgement should follow the rules defined in Section 6.2). As mentioned in step 5, if the SACK is bundled with the COOKIE ACK, the COOKIE ACK MUST appear first in the SCTP packet.

If a COOKIE ECHO is received from an endpoint with which the receiver of the COOKIE ECHO has an existing association, the procedures in Section 5.2 should be followed.

5.1.6. An Example of Normal Association Establishment

In the following example, "A" initiates the association and then sends a user message to "Z", then "Z" sends two user messages to "A" later (assuming no bundling or fragmentation occurs):

```
    Endpoint A                                          Endpoint Z
    {app sets association with Z}
    (build TCB)
    INIT [I-Tag=Tag_A
          & other info]  ------\
    (Start T1-init timer)       \
    (Enter COOKIE-WAIT state)    \---> (compose temp TCB and Cookie_Z)
                                   /-- INIT ACK [Veri Tag=Tag_A,
                                  /             I-Tag=Tag_Z,
    (Cancel T1-init timer) <------/             Cookie_Z, & other info]
                                        (destroy temp TCB)
    COOKIE ECHO [Cookie_Z] ------\
    (Start T1-init timer)         \
    (Enter COOKIE-ECHOED state)    \---> (build TCB enter ESTABLISHED
                                          state)
                                  /---- COOKIE-ACK
                                 /
    (Cancel T1-init timer, <-----/
     Enter ESTABLISHED state)
    {app sends 1st user data; strm 0}
    DATA [TSN=initial TSN_A
        Strm=0,Seq=0 & user data]--\
    (Start T3-rtx timer)            \
                                     \->
                                  /----- SACK [TSN Ack=init
                                 /           TSN_A,Block=0]
    (Cancel T3-rtx timer) <------/
                                          ...
                                        {app sends 2 messages;strm 0}
                                  /---- DATA
                                 /        [TSN=init TSN_Z
                            <--/           Strm=0,Seq=0 & user data 1]
    SACK [TSN Ack=init TSN_Z,      /---- DATA
          Block=0]     --------\  /        [TSN=init TSN_Z +1,
                                \/           Strm=0,Seq=1 & user data 2]
                        <------/\
                                 \
                                  \------>
```

Figure 4: INITiation Example

If the T1-init timer expires at "A" after the INIT or COOKIE ECHO chunks are sent, the same INIT or COOKIE ECHO chunk with the same Initiate Tag (i.e., Tag_A) or State Cookie shall be retransmitted and

the timer restarted. This shall be repeated Max.Init.Retransmits times before "A" considers "Z" unreachable and reports the failure to its upper layer (and thus the association enters the CLOSED state).

When retransmitting the INIT, the endpoint MUST follow the rules defined in 6.3 to determine the proper timer value.

5.2. Handle Duplicate or Unexpected INIT, INIT ACK, COOKIE ECHO, and COOKIE ACK

During the lifetime of an association (in one of the possible states), an endpoint may receive from its peer endpoint one of the setup chunks (INIT, INIT ACK, COOKIE ECHO, and COOKIE ACK). The receiver shall treat such a setup chunk as a duplicate and process it as described in this section.

Note: An endpoint will not receive the chunk unless the chunk was sent to a SCTP transport address and is from a SCTP transport address associated with this endpoint. Therefore, the endpoint processes such a chunk as part of its current association.

The following scenarios can cause duplicated or unexpected chunks:

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

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

C) The chunk is from a stale packet that was used to establish the present association or a past association that is no longer in existence,

D) The chunk is a false packet generated by an attacker, or

E) The peer never received the COOKIE ACK and is retransmitting its COOKIE ECHO.

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

5.2.1. INIT received in COOKIE-WAIT or COOKIE-ECHOED State (Item B)

This usually indicates an initialization collision, i.e., each endpoint is attempting, at about the same time, to establish an association with the other endpoint.

Upon receipt of an INIT in the COOKIE-WAIT state, an endpoint MUST respond with an INIT ACK using the same parameters it sent in its original INIT chunk (including its Initiation Tag, unchanged). When responding, the endpoint MUST send the INIT ACK back to the same address that the original INIT (sent by this endpoint) was sent to.

Upon receipt of an INIT in the COOKIE-ECHOED state, an endpoint MUST respond with an INIT ACK using the same parameters it sent in its original INIT chunk (including its Initiation Tag, unchanged), provided that no NEW address has been added to the forming association. If the INIT message indicates that a new address has been added to the association, then the entire INIT MUST be discarded, and NO changes should be made to the existing association. An ABORT SHOULD be sent in response that MAY include the error 'Restart of an association with new addresses'. The error SHOULD list the addresses that were added to the restarting association.

When responding in either state (COOKIE-WAIT or COOKIE-ECHOED) with an INIT ACK, the original parameters are combined with those from the newly received INIT chunk. The endpoint shall also generate a State Cookie with the INIT ACK. The endpoint uses the parameters sent in its INIT to calculate the State Cookie.

After that, the endpoint MUST NOT change its state, the T1-init timer shall be left running and the corresponding TCB MUST NOT be destroyed. The normal procedures for handling State Cookies when a TCB exists will resolve the duplicate INITs to a single association.

For an endpoint that is in the COOKIE-ECHOED state it MUST populate its Tie-Tags within both the association TCB and inside the State Cookie (see Section 5.2.2 for a description of the Tie-Tags).

5.2.2. Unexpected INIT in States Other than CLOSED, COOKIE-ECHOED,COOKIE-WAIT and SHUTDOWN-ACK-SENT

Unless otherwise stated, upon receipt of an unexpected INIT for this association, the endpoint shall generate an INIT ACK with a State Cookie. Before responding, the endpoint MUST check to see if the unexpected INIT adds new addresses to the association. If new addresses are added to the association, the endpoint MUST respond with an ABORT, copying the 'Initiation Tag' of the unexpected INIT into the 'Verification Tag' of the outbound packet carrying the ABORT. In the ABORT response, the cause of error MAY be set to 'restart of an association with new addresses'. The error SHOULD list the addresses that were added to the restarting association. If no new addresses are added, when responding to the INIT in the outbound INIT ACK, the endpoint MUST copy its current Tie-Tags to a reserved place within the State Cookie and the association's TCB. We shall refer to these locations inside the cookie as the Peer's-Tie-Tag and the Local-Tie-Tag. We will refer to the copy within an association's TCB as the Local Tag and Peer's Tag. The outbound SCTP packet containing this INIT ACK MUST carry a Verification Tag value equal to the Initiation Tag found in the unexpected INIT. And the INIT ACK MUST contain a new Initiation Tag (randomly generated; see

Section 5.3.1). Other parameters for the endpoint SHOULD be copied from the existing parameters of the association (e.g., number of outbound streams) into the INIT ACK and cookie.

After sending out the INIT ACK or ABORT, the endpoint shall take no further actions; i.e., the existing association, including its current state, and the corresponding TCB MUST NOT be changed.

Note: Only when a TCB exists and the association is not in a COOKIE-WAIT or SHUTDOWN-ACK-SENT state are the Tie-Tags populated with a value other than 0. For a normal association INIT (i.e., the endpoint is in the CLOSED state), the Tie-Tags MUST be set to 0 (indicating that no previous TCB existed).

5.2.3. Unexpected 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 chunk. An unexpected INIT ACK usually indicates the processing of an old or duplicated INIT chunk.

5.2.4. Handle a COOKIE ECHO when a TCB exists

When a COOKIE ECHO chunk is received by an endpoint in any state for an existing association (i.e., not in the CLOSED state) the following rules shall be applied:

1) Compute a MAC as described in Step 1 of Section 5.1.5,

2) Authenticate the State Cookie as described in Step 2 of Section 5.1.5 (this is case C or D above).

3) Compare the timestamp in the State Cookie to the current time. If the State Cookie is older than the lifespan carried in the State Cookie and the Verification Tags contained in the State Cookie do not match the current association's Verification Tags, the packet, including the COOKIE ECHO and any DATA chunks, should be discarded. The endpoint also MUST transmit an ERROR chunk with a "Stale Cookie" error cause to the peer endpoint (this is case C or D in Section 5.2).

   If both Verification Tags in the State Cookie match the Verification Tags of the current association, consider the State Cookie valid (this is case E of section 5.2) even if the lifespan is exceeded.

4) If the State Cookie proves to be valid, unpack the TCB into a temporary TCB.
5) Refer to Table 2 to determine the correct action to be taken.

Local Tag	Peer's Tag	Local-Tie-Tag	Peer's-Tie-Tag	Action/ Description
X	X	M	M	(A)
M	X	A	A	(B)
M	0	A	A	(B)
X	M	0	0	(C)
M	M	A	A	(D)

Table 2: Handling of a COOKIE ECHO when a TCB exists

Legend:

X - Tag does not match the existing TCB
M - Tag matches the existing TCB.
0 - No Tie-Tag in Cookie (unknown).
A - All cases, i.e. M, X or 0.

Note: For any case not shown in Table 2, the cookie should be silently discarded.

Action

A) In this case, the peer may have restarted. When the endpoint recognizes this potential 'restart', the existing session is treated the same as if it received an ABORT followed by a new COOKIE ECHO with the following exceptions:

- Any SCTP DATA Chunks MAY be retained (this is an implementation specific option).

- A notification of RESTART SHOULD be sent to the ULP instead of a "COMMUNICATION LOST" notification.

All the congestion control parameters (e.g., cwnd, ssthresh) related to this peer MUST be reset to their initial values (see Section 6.2.1).

After this the endpoint shall enter the ESTABLISHED state.

If the endpoint is in the SHUTDOWN-ACK-SENT state and recognizes the peer has restarted (Action A), it MUST NOT setup a new association but instead resend the SHUTDOWN ACK and send an ERROR chunk with a "Cookie Received while Shutting Down" error cause to its peer.

B) In this case, both sides may be attempting to start an association at about the same time but the peer endpoint started its INIT after responding to the local endpoint's INIT. Thus it may have picked a new Verification Tag not being aware of the previous Tag it had sent this endpoint. The endpoint should stay in or enter the ESTABLISHED state but it MUST update its peer's Verification Tag from the State Cookie, stop any init or cookie timers that may running and send a COOKIE ACK.

C) In this case, the local endpoint's cookie has arrived late. Before it arrived, the local endpoint sent an INIT and received an INIT-ACK and finally sent a COOKIE ECHO with the peer's same tag but a new tag of its own. The cookie should be silently discarded. The endpoint SHOULD NOT change states and should leave any timers running.

D) When both local and remote tags match, the endpoint should enter the ESTABLISHED state, if it is in the COOKIE-ECHOED state. It should stop any cookie timer that may be running and send a COOKIE ACK.

Note: The "peer's Verification Tag" is the tag received in the Initiate Tag field of the INIT or INIT ACK chunk.

5.2.4.1. An Example of a Association Restart

In the following example, "A" initiates the association after a restart has occurred. Endpoint "Z" had no knowledge of the restart until the exchange (i.e. Heartbeats had not yet detected the failure of "A"). (assuming no bundling or fragmentation occurs):

```
   Endpoint A                                          Endpoint Z
   <-------------- Association is established---------------------->
   Tag=Tag_A                                             Tag=Tag_Z
   <---------------------------------------------------------------->
   {A crashes and restarts}
   {app sets up a association with Z}
   (build TCB)
   INIT [I-Tag=Tag_A'
         & other info]  --------\
   (Start T1-init timer)         \
   (Enter COOKIE-WAIT state)      \---> (find a existing TCB
                                         compose temp TCB and Cookie_Z
                                         with Tie-Tags to previous
                                         association)
                                   /--- INIT ACK [Veri Tag=Tag_A',
                                  /               I-Tag=Tag_Z',
   (Cancel T1-init timer) <------/                Cookie_Z[TieTags=
                                                  Tag_A,Tag_Z
                                                   & other info]
                                        (destroy temp TCB,leave original
                                         in place)
   COOKIE ECHO [Veri=Tag_Z',
                Cookie_Z
                Tie=Tag_A,
                Tag_Z]----------\
   (Start T1-init timer)         \
   (Enter COOKIE-ECHOED state)    \---> (Find existing association,
                                         Tie-Tags match old tags,
                                         Tags do not match i.e.,
                                         case X X M M above,
                                         Announce Restart to ULP
                                         and reset association).
                                  /---- COOKIE-ACK
   (Cancel T1-init timer, <------/
    Enter ESTABLISHED state)
   {app sends 1st user data; strm 0}
   DATA [TSN=initial TSN_A
       Strm=0,Seq=0 & user data]--\
   (Start T3-rtx timer)            \
                                    \->
                                 /--- SACK [TSN Ack=init TSN_A,Block=0]
   (Cancel T3-rtx timer) <------/
```

Figure 5: A Restart Example

5.2.5. Handle Duplicate COOKIE-ACK.

At any state other than COOKIE-ECHOED, an endpoint should silently discard a received COOKIE ACK chunk.

5.2.6. Handle Stale COOKIE Error

Receipt of an ERROR chunk with a "Stale Cookie" error cause indicates one of a number of possible events:

A) That the association failed to completely setup before the State Cookie issued by the sender was processed.

B) An old State Cookie was processed after setup completed.

C) An old State Cookie is received from someone that the receiver is not interested in having an association with and the ABORT chunk was lost.

When processing an ERROR chunk with a "Stale Cookie" error cause an endpoint should first examine if an association is in the process of being setup, i.e. the association is in the COOKIE-ECHOED state. In all cases if the association is not in the COOKIE-ECHOED state, the ERROR chunk should be silently discarded.

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

1) Send a new INIT chunk to the endpoint to generate a new State 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 chunk to the endpoint, adding a Cookie Preservative parameter requesting an extension to the lifetime of the State Cookie. When calculating the time extension, an implementation SHOULD use the RTT information measured based on the previous COOKIE ECHO / ERROR exchange, and should add no more than 1 second beyond the measured RTT, due to long State Cookie lifetimes making the endpoint more subject to a replay attack.

5.3. Other Initialization Issues

5.3.1. Selection of Tag Value

Initiate Tag values should be selected from the range of 1 to 2**32 - 1. It is very important that the Initiate Tag value be randomized to

help protect against "man in the middle" and "sequence number" attacks. The methods described in [RFC4086] can be used for the Initiate Tag randomization. Careful selection of Initiate Tags is also necessary to prevent old duplicate packets from previous associations being mistakenly processed as belonging to the current association.

Moreover, the Verification Tag value used by either endpoint in a given association MUST NOT change during the lifetime of an association. A new Verification Tag value MUST be used each time the endpoint tears-down and then re-establishes an association to the same peer.

5.4. Path Verification

During association establishment, the two peers exchange a list of addresses. In the predominant case, these lists accurately represent the addresses owned by each peer. However, it is possible that a misbehaving peer may supply addresses that it does not own. To prevent this, the following rules are applied to all addresses of the new association:

1) Any address passed to the sender of the INIT by its upper layer is automatically considered to be CONFIRMED.
2) For the receiver of the COOKIE-ECHO the only CONFIRMED address is the one that the INIT-ACK was sent to.
3) All other addresses not covered by rules 1 and 2 are considered UNCONFIRMED and are subject to probing for verification.

To probe an address for verification, an endpoint will send HEARTBEATs including a 64-bit random nonce and a path indicator (to identify the address that the HEARTBEAT is sent to) within the HEARTBEAT parameter.

Upon receipt of the HEARTBEAT-ACK, a verification is made that the nonce included in the HEARTBEAT parameter is the one sent to the address indicated inside the HEARTBEAT parameter. When this match occurs, the address that the original HEARTBEAT was sent to is now considered CONFIRMED and available for normal data transfer.

These probing procedures are started when an association moves to the ESTABLISHED state and are ended when all paths are confirmed.

Each RTO a probe may be sent on an active UNCONFIRMED path in an attempt to move it to the CONFIRMED state. If during this probing the path becomes inactive, this rate is lowered to the normal HEARTBEAT rate. At the expiration of the RTO timer, the error counter of any path that was probed but not CONFIRMED is incremented by one and subjected to path failure detection, as defined in

Section 8.2. When probing UNCONFIRMED addresses, however, the association overall error count is NOT incremented.

The number of HEARTBEATS sent at each RTO SHOULD be limited by the HB.Max.Burst parameter. It is an implementation decision as to how to distribute HEARTBEATS to the peer's addresses for path verification.

Whenever a path is confirmed, an indication MAY be given to the upper layer.

An endpoint MUST NOT send any chunks to an UNCONFIRMED address, with the following exceptions:

- A HEARTBEAT including a nonce MAY be sent to an UNCONFIRMED address.
- A HEARTBEAT-ACK MAY be sent to an UNCONFIRMED address.
- A COOKIE-ACK MAY be sent to an UNCONFIRMED address, but it MUST be bundled with a HEARTBEAT including a nonce. An implementation that does NOT support bundling MUST NOT send a COOKIE-ACK to an UNCONFIRMED address.
- A COOKE-ECHO MAY be sent to an UNCONFIRMED address, but it MUST be bundled with a HEARTBEAT including a nonce, and the packet MUST NOT exceed the path MTU. If the implementation does NOT support bundling or if the bundled COOKIE-ECHO plus HEARTBEAT (including nonce) would exceed the path MTU, then the implementation MUST NOT send a COOKIE-ECHO to an UNCONFIRMED address.

6. User Data Transfer

Data transmission MUST only happen in the ESTABLISHED, SHUTDOWN-PENDING, and SHUTDOWN-RECEIVED states. The only exception to this is that DATA chunks are allowed to be bundled with an outbound COOKIE ECHO chunk when in COOKIE-WAIT state.

DATA chunks MUST only be received according to the rules below in ESTABLISHED, SHUTDOWN-PENDING, SHUTDOWN-SENT. A DATA chunk received in CLOSED is out of the blue and SHOULD be handled per 8.4. A DATA chunk received in any other state SHOULD be discarded.

A SACK MUST be processed in ESTABLISHED, SHUTDOWN-PENDING, and SHUTDOWN-RECEIVED. An incoming SACK MAY be processed in COOKIE-ECHOED. A SACK in the CLOSED state is out of the blue and SHOULD be processed according to the rules in 8.4. A SACK chunk received in any other state SHOULD be discarded.

A SCTP receiver MUST be able to receive a minimum of 1500 bytes in one SCTP packet. This means that a SCTP endpoint MUST NOT indicate

less than 1500 bytes in its Initial a_rwnd sent in the INIT or INIT ACK.

For transmission efficiency, SCTP defines mechanisms for bundling of small user messages and fragmentation of large user messages. The following diagram depicts the flow of user messages through SCTP.

In this section the term "data sender" refers to the endpoint that transmits a DATA chunk and the term "data receiver" refers to the endpoint that receives a DATA chunk. A data receiver will transmit SACK chunks.

```
                +--------------------------+
                |      User Messages       |
                +--------------------------+
     SCTP user        ^  |
    ==================|==|=====================================
                      |  v (1)
            +------------------+    +--------------------+
            | SCTP DATA Chunks |    |SCTP Control Chunks |
            +------------------+    +--------------------+
                      ^  |             ^  |
                      |  v (2)         |  v (2)
                   +--------------------------+
                   |      SCTP packets        |
                   +--------------------------+
     SCTP                         ^  |
    ==========================|==|==========================
                              |  v
            Connectionless Packet Transfer Service (e.g., IP)
```

Notes:

1) When converting user messages into DATA chunks, an endpoint will fragment user messages larger than the current association path MTU into multiple DATA chunks. The data receiver will normally reassemble the fragmented message from DATA chunks before delivery to the user (see Section 6.9 for details).

2) Multiple DATA and control chunks may be bundled by the sender into a single SCTP packet for transmission, as long as the final size of the packet does not exceed the current path MTU. The receiver will unbundle the packet back into the original chunks. Control chunks MUST come before DATA chunks in the packet.

Figure 6: Illustration of User Data Transfer

The fragmentation and bundling mechanisms, as detailed in Section 6.9 and Section 6.10, are OPTIONAL to implement by the data sender, but they MUST be implemented by the data receiver, i.e., an endpoint MUST properly receive and process bundled or fragmented data.

6.1. Transmission of DATA Chunks

This document is specified as if there is a single retransmission timer per destination transport address, but implementations MAY have a retransmission timer for each DATA chunk.

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

A) At any given time, the data sender MUST NOT transmit new data to any destination transport address if its peer's rwnd indicates that the peer has no buffer space (i.e., rwnd is 0; see Section 6.2.1). However, regardless of the value of rwnd (including if it is 0), the data sender can always have one DATA chunk in flight to the receiver if allowed by cwnd (see rule B, below). This rule allows the sender to probe for a change in rwnd that the sender missed due to the SACK's having been lost in transit from the data receiver to the data sender.

   When the receiver's advertised window is zero, this probe is called a zero window probe. Note that a zero window probe SHOULD only be sent when all outstanding DATA chunks have been cumulatively acknowledged and no DATA chunks are in flight. Zero window probing MUST be supported.

   If the sender continues to receive new packets from the receiver while doing zero window probing, the unacknowledged window probes should not increment the error counter for the association or any destination transport address. This is because the receiver MAY keep its window closed for an indefinite time. Refer to Section 6.2 on the receiver behavior when it advertises a zero window. The sender SHOULD send the first zero window probe after 1 RTO when it detects that the receiver has closed its window and SHOULD increase the probe interval exponentially afterwards. Also note that the cwnd SHOULD be adjusted according to Section 7.2.1. Zero window probing does not affect the calculation of cwnd.

   The sender MUST also have an algorithm for sending new DATA chunks to avoid silly window syndrome (SWS) as described in [RFC0813]. The algorithm can be similar to the one described in Section 4.2.3.4 of [RFC1122].

   that the peer has no buffer space (i.e. rwnd is 0, see).

However, regardless of the value of rwnd (including if it is 0), the data sender can always have one DATA chunk in flight to the receiver if allowed by cwnd (see rule B below). This rule allows the sender to probe for a change in rwnd that the sender missed due to the SACK having been lost in transit from the data receiver to the data sender.

B) At any given time, the sender MUST NOT transmit new data to a given transport address if it has cwnd or more bytes of data outstanding to 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 (limited by the current cwnd).

D) When the time comes for the sender to transmit new DATA chunks, the protocol parameter Max.Burst SHOULD be used to limit the number of packets sent. The limit MAY be applied by adjusting cwnd as follows:

if((flightsize + Max.Burst*MTU) < cwnd) cwnd = flightsize + Max.Burst*MTU

Or it MAY be applied by strictly limiting the number of packets emitted by the output routine.

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

Multiple DATA chunks committed for transmission MAY be bundled in a single packet. Furthermore, DATA chunks being retransmitted MAY be bundled with new DATA chunks, as long as the resulting packet size does not exceed the path MTU. A ULP may request that no bundling is performed but this should only turn off any delays that a SCTP implementation may be using to increase bundling efficiency. It does not in itself stop all bundling from occurring (i.e. in case of congestion or retransmission).

Before an endpoint transmits a DATA chunk, if any received DATA chunks have not been acknowledged (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 packet does not exceed the current MTU. See Section 6.2.

IMPLEMENTATION NOTE: When 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 MUST 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.

Whenever a transmission or retransmission is made to any address, if the T3-rtx timer of that address is not currently running, the sender MUST start that timer. If the timer for that address is already running, the sender MUST restart the timer if the earliest (i.e., lowest TSN) outstanding DATA chunk sent to that address is being retransmitted. Otherwise, the data sender MUST NOT restart the timer.

When starting or restarting the T3-rtx timer, the timer value must be adjusted according to the timer rules defined in Section 6.3.2, and Section 6.3.3.

Note: The data sender SHOULD NOT use a TSN that is more than 2**31 - 1 above the beginning TSN of the current send window.

6.2. Acknowledgement on Reception of DATA Chunks

The SCTP endpoint MUST always acknowledge the reception of each valid DATA chunk when the DATA chunk received is inside its receive window.

When the receiver's advertised window is 0, the receiver MUST drop any new incoming DATA chunk with a TSN larger than the largest TSN received so far. If the new incoming DATA chunk holds a TSN value less than the largest TSN received so far, then the receiver SHOULD drop the largest TSN held for reordering and accept the new incoming DATA chunk. In either case, if such a DATA chunk is dropped, the receiver MUST immediately send back a SACK with the current receive window showing only DATA chunks received and accepted so far. The dropped DATA chunk(s) MUST NOT be included in the SACK, as they were not accepted. The receiver MUST also have an algorithm for advertising its receive window to avoid receiver silly window syndrome (SWS), as described in [RFC0813]. The algorithm can be similar to the one described in Section 4.2.3.3 of [RFC1122].

The guidelines on delayed acknowledgement algorithm specified in Section 4.2 of [RFC2581] SHOULD be followed. Specifically, an acknowledgement SHOULD be generated for at least every second packet (not every second DATA chunk) received, and SHOULD be generated within 200 ms of the arrival of any unacknowledged DATA chunk. In some situations it may be beneficial for an SCTP transmitter to be more conservative than the algorithms detailed in this document allow. However, an SCTP transmitter MUST NOT be more aggressive than the following algorithms allow.

A SCTP receiver MUST NOT generate more than one SACK for every incoming packet, other than to update the offered window as the receiving application consumes new data.

IMPLEMENTATION NOTE: The maximum delay for generating an acknowledgement may be configured by the SCTP administrator, either statically or dynamically, in order to meet the specific timing requirement of the protocol being carried.

An implementation MUST NOT allow the maximum delay to be configured to be more than 500 ms. In other words an implementation MAY lower this value below 500ms but MUST NOT raise it above 500ms.

Acknowledgments MUST be sent in SACK chunks unless shutdown was requested by the ULP, in which case an endpoint MAY send an acknowledgement in the SHUTDOWN chunk. A SACK chunk can acknowledge the reception of multiple DATA chunks. See Section 3.3.4 for SACK chunk format. In particular, the SCTP endpoint MUST fill in the Cumulative TSN Ack field to indicate the latest sequential TSN (of a valid DATA chunk) it has received. Any received DATA chunks with TSN greater than the value in the Cumulative TSN Ack field are reported in the Gap Ack Block fields. The SCTP endpoint MUST report as many Gap Ack Blocks as can fit in a single SACK chunk limited by the current path MTU.

Note: The SHUTDOWN chunk does not contain Gap Ack Block fields. Therefore, the endpoint should use a SACK instead of the SHUTDOWN chunk to acknowledge DATA chunks received out of order .

When a packet arrives with duplicate DATA chunk(s) and with no new DATA chunk(s), the endpoint MUST immediately send a SACK with no delay. If a packet arrives with duplicate DATA chunk(s) bundled with new DATA chunks, the endpoint MAY immediately send a SACK. Normally receipt of duplicate DATA chunks will occur when the original SACK chunk was lost and the peer's RTO has expired. The duplicate TSN number(s) SHOULD be reported in the SACK as duplicate.

When an endpoint receives a SACK, it MAY use the Duplicate TSN information to determine if SACK loss is occurring. Further use of this data is for future study.

The data receiver is responsible for maintaining its receive buffers. The data receiver SHOULD notify the data sender in a timely manner of changes in its ability to receive data. How an implementation manages its receive buffers is dependent on many factors (e.g., Operating System, memory management system, amount of memory, etc.). However, the data sender strategy defined in Section 6.2.1 is based on the assumption of receiver operation similar to the following:

A) At initialization of the association, the endpoint tells the peer how much receive buffer space it has allocated to the association in the INIT or INIT ACK. The endpoint sets a_rwnd to this value.

B) As DATA chunks are received and buffered, decrement a_rwnd by the number of bytes received and buffered. This is, in effect, closing rwnd at the data sender and restricting the amount of data it can transmit.

C) As DATA chunks are delivered to the ULP and released from the receive buffers, increment a_rwnd by the number of bytes delivered to the upper layer. This is, in effect, opening up rwnd on the data sender and allowing it to send more data. The data receiver SHOULD NOT increment a_rwnd unless it has released bytes from its receive buffer. For example, if the receiver is holding fragmented DATA chunks in a reassembly queue, it should not increment a_rwnd.

D) When sending a SACK, the data receiver SHOULD place the current value of a_rwnd into the a_rwnd field. The data receiver SHOULD take into account that the data sender will not retransmit DATA chunks that are acked via the Cumulative TSN Ack (i.e., will drop from its retransmit queue).

Under certain circumstances, the data receiver may need to drop DATA chunks that it has received but hasn't released from its receive buffers (i.e., delivered to the ULP). These DATA chunks may have been acked in Gap Ack Blocks. For example, the data receiver may be holding data in its receive buffers while reassembling a fragmented user message from its peer when it runs out of receive buffer space. It may drop these DATA chunks even though it has acknowledged them in Gap Ack Blocks. If a data receiver drops DATA chunks, it MUST NOT include them in Gap Ack Blocks in subsequent SACKs until they are received again via retransmission. In addition, the endpoint should take into account the dropped data when calculating its a_rwnd.

An endpoint SHOULD NOT revoke a SACK and discard data. Only in extreme circumstance should an endpoint use this procedure (such as out of buffer space). The data receiver should take into account that dropping data that has been acked in Gap Ack Blocks can result in suboptimal retransmission strategies in the data sender and thus in suboptimal performance.

The following example illustrates the use of delayed acknowledgements:

```
    Endpoint A                                      Endpoint Z

    {App sends 3 messages; strm 0}
    DATA [TSN=7,Strm=0,Seq=3] ------------> (ack delayed)
    (Start T3-rtx timer)

    DATA [TSN=8,Strm=0,Seq=4] ------------> (send ack)
                                /------- SACK [TSN Ack=8,block=0]
    (cancel T3-rtx timer)  <-----/

    DATA [TSN=9,Strm=0,Seq=5] ------------> (ack delayed)
    (Start T3-rtx timer)
                                          ...
                                          {App sends 1 message; strm 1}
                                          (bundle SACK with DATA)
                                  /----- SACK [TSN Ack=9,block=0] \
                                 /         DATA [TSN=6,Strm=1,Seq=2]
    (cancel T3-rtx timer)  <------/       (Start T3-rtx timer)

    (ack delayed)
    (send ack)
    SACK [TSN Ack=6,block=0] -------------> (cancel T3-rtx timer)
```

Figure 7: Delayed Acknowledgment Example

If an endpoint receives a DATA chunk with no user data (i.e., the Length field is set to 16) it MUST send an ABORT with error cause set to "No User Data".

An endpoint SHOULD NOT send a DATA chunk with no user data part.

6.2.1. Processing a Received SACK

Each SACK an endpoint receives contains an a_rwnd value. This value represents the amount of buffer space the data receiver, at the time of transmitting the SACK, has left of its total receive buffer space (as specified in the INIT/INIT ACK). Using a_rwnd, Cumulative TSN Ack and Gap Ack Blocks, the data sender can develop a representation of the peer's receive buffer space.

One of the problems the data sender must take into account when processing a SACK is that a SACK can be received out of order. That is, a SACK sent by the data receiver can pass an earlier SACK and be received first by the data sender. If a SACK is received out of order, the data sender can develop an incorrect view of the peer's receive buffer space.

Since there is no explicit identifier that can be used to detect out-of-order SACKs, the data sender must use heuristics to determine if a SACK is new.

An endpoint SHOULD use the following rules to calculate the rwnd, using the a_rwnd value, the Cumulative TSN Ack and Gap Ack Blocks in a received SACK.

A) At the establishment of the association, the endpoint initializes the rwnd to the Advertised Receiver Window Credit (a_rwnd) the peer specified in the INIT or INIT ACK.

B) Any time a DATA chunk is transmitted (or retransmitted) to a peer, the endpoint subtracts the data size of the chunk from the rwnd of that peer.

C) Any time a DATA chunk is marked for retransmission (via either T3-rtx timer expiration (Section 6.3.3) or via fast retransmit (Section 7.2.4), add the data size of those chunks to the rwnd.

Note: If the implementation is maintaining a timer on each DATA chunk then only DATA chunks whose timer expired would be marked for retransmission.

D) Any time a SACK arrives, the endpoint performs the following:

i) If Cumulative TSN Ack is less than the Cumulative TSN Ack Point, then drop the SACK. Since Cumulative TSN Ack is monotonically increasing, a SACK whose Cumulative TSN Ack is less than the Cumulative TSN Ack Point indicates an out-of-order SACK.

ii) Set rwnd equal to the newly received a_rwnd minus the number of bytes still outstanding after processing the Cumulative TSN Ack and the Gap Ack Blocks.

iii) If the SACK is missing a TSN that was previously acknowledged via a Gap Ack Block (e.g., the data receiver reneged on the data), then consider the corresponding DATA that might be possibly missing: Count one miss indication towards fast retransmit as described in Section 7.2.4 , and if no retransmit timer is running for the destination address to which the DATA chunk was originally transmitted, then T3-rtx is started for that destination address.

iv) If the Cumulative TSN Ack matches or exceeds the Fast Recovery exitpoint (Section 7.2.4), Fast Recovery is exited.

6.3. Management of Retransmission Timer

An SCTP endpoint uses a retransmission timer T3-rtx to ensure data delivery in the absence of any feedback from its peer. The duration of this timer is referred to as RTO (retransmission timeout).

When an endpoint's peer is multi-homed, the endpoint will calculate a separate RTO for each different destination transport address of its peer endpoint.

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

6.3.1. 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 + 4 * RTTVAR.

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

RTTVAR <- (1 - RTO.Beta) * RTTVAR + RTO.Beta * |SRTT - R'|

and

SRTT <- (1 - RTO.Alpha) * SRTT + RTO.Alpha * R'

Note: The value of SRTT used in the update to RTTVAR is its value before updating SRTT itself using the second assignment.

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

C4) When data is in flight and when allowed by rule C5 below, a new RTT measurement MUST be made each round trip. Furthermore, 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 frequently often does not in practice yield any significant benefit [ALLMAN99]; second, if measurements are made more often, then the values of RTO.Alpha and RTO.Beta in rule C3 above should 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 values) as they would if making only one measurement per round-trip and using RTO.Alpha and RTO.Beta as given in rule C3. However, the exact nature of these adjustments remains a research issue.

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 the chunk or for a later instance)

IMPLEMENTATION NOTE: RTT measurements should only be made using a chunk with TSN r if no chunk with TSN less than or equal to r is retransmitted since r is first sent.

C6) Whenever RTO is computed, if it is less than RTO.Min seconds then it is rounded up to RTO.Min seconds. The reason for this rule is that RTOs that do not have a high minimum value are susceptible to unnecessary timeouts [ALLMAN99].

C7) A maximum value may be placed on RTO provided it is at least RTO.max 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 [ALLMAN99] has shown that finer clock granularities (<= 100 msec) perform somewhat better than more coarse granularities.

6.3.2. Retransmission Timer Rules

The rules for managing the retransmission timer are as follows:

R1) Every time a DATA chunk is sent to any address (including a retransmission), if the T3-rtx timer of that address is not running, start it running so that it will expire after the RTO of that address. The RTO used here is that obtained after any doubling due to previous T3-rtx timer expirations on the corresponding destination address as discussed in rule E2 below.

R2) Whenever all outstanding data sent to an address have been acknowledged, turn off the T3-rtx timer of that address.

R3) Whenever a SACK is received that acknowledges the DATA chunk with the earliest outstanding TSN for that address, restart T3-rtx timer for that address with its current RTO (if there is still outstanding data on that address).

R4) Whenever a SACK is received missing a TSN that was previously acknowledged via a Gap Ack Block, start T3-rtx for the destination address to which the DATA chunk was originally transmitted if it is not already running.

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

```
Endpoint A                                          Endpoint Z
{App begins to send}
Data [TSN=7,Strm=0,Seq=3] ------------> (ack delayed)
(Start T3-rtx timer)
                                         {App sends 1 message; strm 1}
                                         (bundle ack with data)
DATA [TSN=8,Strm=0,Seq=4] ----\     /-- SACK [TSN Ack=7,Block=0]
                               \   /      DATA [TSN=6,Strm=1,Seq=2]
                                \ /     (Start T3-rtx timer)
                                 \
                                / \
(Re-start T3-rtx timer) <------/   \--> (ack delayed)
(ack delayed)
{send ack}
SACK [TSN Ack=6,Block=0] --------------> (Cancel T3-rtx timer)
                                         ..
                                         (send ack)
(Cancel T3-rtx timer)  <-------------- SACK [TSN Ack=8,Block=0]
```

Figure 8 - Timer Rule Examples

6.3.3. Handle T3-rtx Expiration

Whenever the retransmission timer T3-rtx expires for a destination address, do the following:

E1) For the destination address for which the timer expires, adjust its ssthresh with rules defined in Section 7.2.3 and set the cwnd <- MTU.

E2) For the destination address for which the timer expires, set RTO <- RTO * 2 ("back off the timer"). The maximum value discussed in rule C7 above (RTO.max) may be used to provide an upper bound to this doubling operation.

E3) Determine how many of the earliest (i.e., lowest TSN) outstanding DATA chunks for the address for which the T3-rtx has expired will fit into a single packet, subject to the MTU constraint for the path corresponding to the destination transport address to which the retransmission is being sent (this may be different from the address for which the timer expires [see Section 6.4]). Call this value K. Bundle and retransmit those K DATA chunks in a single packet to the destination endpoint.

E4) Start the retransmission timer T3-rtx on the destination address to which the retransmission is sent, if rule R1 above indicates to do so. The RTO to be used for starting T3-rtx should be the one for the destination address to which the retransmission is sent, which, when the receiver is multi-homed, may be different from the destination address for which the timer expired (see Section 6.4 below).

After retransmitting, once a new RTT measurement is obtained (which can happen only when new data has been sent and acknowledged, per rule C5, or for a measurement made from a HEARTBEAT [see Section 8.3), 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 E2).

Note: Any DATA chunks that were sent to the address for which the T3-rtx timer expired but did not fit in one MTU (rule E3 above), should be marked for retransmission and sent as soon as cwnd allows (normally when a SACK arrives).

The final rule for managing the retransmission timer concerns failover (see Section 6.4.1):

F1) Whenever an endpoint switches from the current destination transport address to a different one, the current retransmission timers are left running. As soon as the endpoint transmits a packet containing DATA chunk(s) to the new transport address, start the timer on that transport address, using the RTO value of the destination address to which the data is being sent, if rule R1 indicates to do so.

6.4. Multi-homed SCTP Endpoints

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

Moreover, the ULP of an endpoint shall select one of the multiple destination addresses of a multi-homed peer endpoint as the primary path (see Section 5.1.2 and Section 10.1 for details).

By default, an endpoint SHOULD always transmit to the primary path, unless the SCTP user explicitly specifies the destination transport address (and possibly source transport address) to use.

An endpoint SHOULD transmit reply chunks (e.g., SACK, HEARTBEAT ACK, etc.) to the same destination transport address from which it received the DATA or control chunk to which it is replying. This rule should also be followed if the endpoint is bundling DATA chunks together with the reply chunk.

However, when acknowledging multiple DATA chunks received in packets from different source addresses in a single SACK, the SACK chunk may be transmitted to one of the destination transport addresses from which the DATA or control chunks being acknowledged were received.

When a receiver of a duplicate DATA chunk sends a SACK to a multi-homed endpoint it MAY be beneficial to vary the destination address and not use the source address of the DATA chunk. The reason being that receiving a duplicate from a multi-homed endpoint might indicate that the return path (as specified in the source address of the DATA chunk) for the SACK is broken.

Furthermore, when its peer is multi-homed, an endpoint SHOULD try to retransmit a chunk that timed out to an active destination transport address that is different from the last destination address to which the DATA chunk was sent.

Retransmissions do not affect the total outstanding data count. However, if the DATA chunk is retransmitted onto a different destination address, both the outstanding data counts on the new

destination address and the old destination address to which the data chunk was last sent shall be adjusted accordingly.

6.4.1. Failover from Inactive Destination Address

Some of the transport addresses of a multi-homed SCTP endpoint may become inactive due to either the occurrence of certain error conditions (see Section 8.2) or adjustments from SCTP user.

When there is outbound data to send and the primary path becomes inactive (e.g., due to failures), or where the SCTP user explicitly requests to send data to an inactive destination transport address, before reporting an error to its ULP, the SCTP endpoint should try to send the data to an alternate active destination transport address if one exists.

When retransmitting data that timed out, if the endpoint is multi-homed, it should consider each source-destination address pair in its retransmission selection policy. When retransmitting timed out data, the endpoint should attempt to pick the most divergent source-destination pair from the original source-destination pair to which the packet was transmitted.

Note: Rules for picking the most divergent source-destination pair are an implementation decision and is not specified within this document.

6.5. Stream Identifier and Stream Sequence Number

Every DATA chunk MUST carry a valid stream identifier. If an endpoint receives a DATA chunk with an invalid stream identifier, it shall acknowledge the reception of the DATA chunk following the normal procedure, immediately send an ERROR chunk with cause set to "Invalid Stream Identifier" (see Section 3.3.10) and discard the DATA chunk. The endpoint may bundle the ERROR chunk in the same packet as the SACK as long as the ERROR follows the SACK.

The stream sequence number in all the streams MUST start from 0 when the association is established. Also, when the stream sequence number reaches the value 65535 the next stream sequence number MUST be set to 0.

6.6. Ordered and Unordered Delivery

Within a stream, an endpoint MUST deliver DATA chunks received with the U flag set to 0 to the upper layer according to the order of their stream sequence number. If DATA chunks arrive out of order of their stream sequence number, the endpoint MUST hold the received

DATA chunks from delivery to the ULP until they are re-ordered.

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

When an endpoint receives a DATA chunk with the U flag set to 1, it must bypass the ordering mechanism and immediately deliver the data to the upper layer (after re-assembly if the user data is fragmented by the data 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 all DATA chunks sent through that stream.

IMPLEMENTATION NOTE: When sending an unordered DATA chunk, an implementation may choose to place the DATA chunk in an outbound packet that is at the head of the outbound transmission queue if possible.

The 'Stream Sequence Number' field in a DATA chunk with U flag set to 1 has no significance. The sender can fill it with arbitrary value, but the receiver MUST ignore the field.

Note: When transmitting ordered and unordered data, an endpoint does not increment its Stream Sequence Number when transmitting a DATA chunk with U flag set to 1.

6.7. Report Gaps in Received DATA TSNs

Upon the reception of a new DATA chunk, an endpoint shall examine the continuity of the TSNs received. If the endpoint detects a gap in the received DATA chunk sequence, it SHOULD send a SACK with Gap Ack Blocks immediately. The data receiver continues sending a SACK after receipt of each SCTP packet that doesn't fill the gap.

Based on the Gap Ack Block from the received SACK, the endpoint can calculate the missing DATA chunks and make decisions on whether to retransmit them (see Section 6.2.1 for details).

Multiple gaps can be reported in one single SACK (see Section 3.3.4).

When its peer is multi-homed, the SCTP endpoint SHOULD always try to send the SACK to the same destination address from which the last DATA chunk was received.

Upon the reception of a SACK, the endpoint MUST remove all DATA

chunks which have been acknowledged by the SACK's Cumulative TSN Ack from its transmit queue. The endpoint MUST also treat all the DATA chunks with TSNs not included in the Gap Ack Blocks 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 decisions. See Section 7.2.4 for details.

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

```
    Endpoint A                                      Endpoint Z
    {App sends 3 messages; strm 0}
    DATA [TSN=6,Strm=0,Seq=2] ---------------> (ack delayed)
    (Start T3-rtx timer)

    DATA [TSN=7,Strm=0,Seq=3] --------> X (lost)

    DATA [TSN=8,Strm=0,Seq=4] ---------------> (gap detected,
                                                immediately send ack)
                                  /----- SACK [TSN Ack=6,Block=1,
                                 /             Strt=2,End=2]
                            <-----/
    (remove 6 from out-queue,
     and mark 7 as "1" missing report)
```

Figure 9 - Reporting a Gap using SACK

The maximum number of Gap Ack Blocks that can be reported within a single SACK chunk is limited by the current path MTU. When a single SACK can not cover all the Gap Ack Blocks needed to be reported due to the MTU limitation, the endpoint MUST send only one SACK, reporting the Gap Ack Blocks from the lowest to highest TSNs, within the size limit set by the MTU, and leave the remaining highest TSN numbers unacknowledged.

6.8. CRC32c Checksum Calculation

When sending an SCTP packet, the endpoint MUST strengthen the data integrity of the transmission by including the CRC32c checksum value calculated on the packet, as described below.

After the packet is constructed (containing the SCTP common header and one or more control or DATA chunks), the transmitter MUST

1) fill in the proper Verification Tag in the SCTP common header and initialize the checksum field to '0's,

2) calculate the CRC32c checksum of the whole packet, including the SCTP common header and all the chunks (refer to appendix B for details of the CRC32c algorithm); and

3) put the resultant value into the checksum field in the common header, and leave the rest of the bits unchanged.

When an SCTP packet is received, the receiver MUST first check the CRC32c checksum as follows:

1) Store the received CRC32c checksum value aside.

2) Replace the 32 bits of the checksum field in the received SCTP packet with all '0's and calculate a CRC32c checksum value of the whole received packet.

3) Verify that the calculated CRC32c checksum is the same as the received CRC32c checksum. If it is not, the receiver MUST treat the packet as an invalid SCTP packet.

The default procedure for handling invalid SCTP packets is to silently discard them.

Any hardware implementation SHOULD be done in a way that is verifiable by the software.

6.9. Fragmentation and Reassembly

An endpoint MAY support fragmentation when sending DATA chunks, but it MUST support reassembly when receiving DATA chunks. If an endpoint supports fragmentation, it MUST fragment a user message if the size of the user message to be sent causes the outbound SCTP packet size to exceed the current MTU. If an implementation does not support fragmentation of outbound user messages, the endpoint MUST return an error to its upper layer and not attempt to send the user message.

Note: If an implementation that supports fragmentation makes available to its upper layer a mechanism to turn off fragmentation it may do so. However, in so doing, it MUST react just like an implementation that does NOT support fragmentation, i.e., it MUST reject sends that exceed the current P-MTU.

IMPLEMENTATION NOTE: In this error case, the Send primitive discussed in Section 10.1 would need to return an error to the upper layer.

If its peer is multi-homed, the endpoint shall choose a size no larger than the association Path MTU. The association Path MTU is the smallest Path MTU of all destination addresses.

Note: Once a message is fragmented it cannot be re-fragmented. Instead if the PMTU has been reduced, then IP fragmentation must be used. Please see Section 7.3 for details of PMTU discovery.

When determining when to fragment, the SCTP implementation MUST take into account the SCTP packet header as well as the DATA chunk header(s). The implementation MUST also take into account the space required for a SACK chunk if bundling a SACK chunk with the DATA chunk.

Fragmentation takes the following steps:

1) The data sender MUST break the user message into a series of DATA chunks such that each chunk plus SCTP overhead fits into an IP datagram smaller than or equal to the association Path MTU.

2) The transmitter MUST then assign, in sequence, a separate TSN to each of the DATA chunks in the series. The transmitter assigns the same SSN to each of the DATA chunks. If the user indicates that the user message is to be delivered using unordered delivery, then the U flag of each DATA chunk of the user message MUST be set to 1.

3) The transmitter 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'.

An endpoint MUST recognize fragmented DATA chunks by examining the B/E bits in each of the received DATA chunks, and queue the fragmented DATA chunks for re-assembly. Once the user message is reassembled, SCTP shall pass the re-assembled user message to the specific stream for possible re-ordering and final dispatching.

Note: If the data receiver runs out of buffer space while still waiting for more fragments to complete the re-assembly of the message, it should dispatch part of its inbound message through a partial delivery API (see Section 10), freeing some of its receive buffer space so that the rest of the message may be received.

6.10. Bundling

An endpoint bundles chunks by simply including multiple chunks in one outbound SCTP packet. The total size of the resultant IP datagram,

including the SCTP packet and IP headers, MUST be less or equal to the current Path MTU.

If its peer endpoint is multi-homed, the sending endpoint shall choose a size no larger than the latest MTU of the current primary path.

When bundling control chunks with DATA chunks, an endpoint MUST place control chunks first in the outbound SCTP packet. The transmitter MUST transmit DATA chunks within a SCTP packet in increasing order of TSN.

Note: Since control chunks must be placed first in a packet and since DATA chunks must be transmitted before SHUTDOWN or SHUTDOWN ACK chunks, DATA chunks cannot be bundled with SHUTDOWN or SHUTDOWN ACK chunks.

Partial chunks MUST NOT be placed in an SCTP packet. A partial chunk is a chunk that is not completely contained in the SCTP packet; i.e., the SCTP packet is too short to contain all the bytes of the chunk as indicated by the chunk length.

An endpoint MUST process received chunks in their order in the packet. 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 4 byte boundary. If the receiver detects a partial chunk, it MUST drop the chunk.

An endpoint MUST NOT bundle INIT, INIT ACK or SHUTDOWN COMPLETE with any other chunks.

7. Congestion control

Congestion control is one of the basic functions in SCTP. For some applications, it may be likely that adequate resources will be allocated to SCTP traffic to assure prompt delivery of time-critical data - thus it would appear to be unlikely, during normal operations, that transmissions encounter severe congestion conditions. However SCTP must operate under adverse operational conditions, which can develop upon partial network failures or unexpected traffic surges. 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 should show no impact on the protocol performance.

IMPLEMENTATION NOTE: As far as its specific performance requirements

are met, an implementation is always allowed to adopt a more conservative congestion control algorithm than the one defined below.

The congestion control algorithms used by SCTP are based on [RFC2581]. This section describes how the algorithms defined in [RFC2581] are adapted 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.

SCTP congestion control is always applied to the entire association, and not to individual streams.

7.1. SCTP Differences from TCP Congestion control

Gap Ack Blocks in the SCTP SACK carry the same semantic meaning as the TCP SACK. TCP considers the information carried in the SACK as advisory information only. SCTP considers the information carried in the Gap Ack Blocks in the SACK chunk as advisory. In SCTP, any DATA chunk that has been acknowledged by SACK, including DATA that arrived at the receiving end out of order, are not considered fully delivered until the Cumulative TSN Ack Point passes the TSN of the DATA chunk (i.e., the DATA chunk has been acknowledged by the Cumulative TSN Ack field in the SACK). Consequently, the value of cwnd controls the amount of outstanding data, rather than (as in the case of non-SACK TCP) the upper bound between the highest acknowledged sequence number and the latest DATA chunk that can be sent within the congestion window. SCTP SACK leads to different implementations of fast-retransmit and fast-recovery than non-SACK TCP. As an example see [FALL96].

The biggest difference between SCTP and TCP, however, is multi-homing. SCTP is designed to establish robust communication associations between two endpoints each of which may be reachable by more than one transport address. Potentially different addresses may lead to different data paths between the two endpoints, thus ideally one may need a separate set of congestion control parameters for each of the paths. The treatment here of congestion control for multi-homed receivers is new with SCTP and may require refinement in the future. The current algorithms make the following assumptions:

- o The sender usually uses the same destination address until being instructed by the upper layer otherwise; however, SCTP may change to an alternate destination in the event an address is marked inactive (see Section 8.2). Also, SCTP may retransmit to a different transport address than the original transmission.

- o The sender keeps a separate congestion control parameter set for each of the destination addresses it can send to (not each source-destination pair but for each destination). The parameters should decay if the address is not used for a long enough time period.

- o For each of the destination addresses, an endpoint does slow-start upon the first transmission to that address.

Note: TCP guarantees in-sequence delivery of data to its upper-layer protocol within a single TCP session. This means that when TCP notices a gap in the received sequence number, it waits until the gap is filled before delivering the data that was received with sequence numbers higher than that of the missing data. On the other hand, SCTP can deliver data to its upper-layer protocol even if there is a gap in TSN if the Stream Sequence Numbers are in sequence for a particular stream (i.e., the missing DATA chunks are for a different stream) or if unordered delivery is indicated. Although this does not affect cwnd, it might affect rwnd calculation.

7.2. SCTP Slow-Start and Congestion Avoidance

The slow start and congestion avoidance algorithms MUST be used by an endpoint to control the amount of 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 an SCTP sender to be more conservative than the algorithms allow; however, an SCTP sender MUST NOT be more aggressive than the following algorithms allow.

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

- o Receiver advertised window size (rwnd, in bytes), which is set by the receiver based on its available buffer space for incoming packets.

  Note: This variable is kept on the entire association.

- o Congestion control window (cwnd, in bytes), which is adjusted by the sender based on observed network conditions.

  Note: This variable is maintained on a per-destination address basis.

- o Slow-start threshold (ssthresh, in bytes), which is used by the sender to distinguish slow start and congestion avoidance phases.

  Note: This variable is maintained on a per-destination address basis.

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

Unlike TCP, an SCTP sender MUST keep a set of these control variables cwnd, ssthresh and partial_bytes_acked for EACH destination address of its peer (when its peer is multi-homed). Only one rwnd is kept for the whole association (no matter if the peer is multi-homed or has a single address).

7.2.1. Slow-Start

Beginning data transmission into a network with unknown conditions or after a sufficiently long idle period 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 cwnd before DATA transmission or after a sufficiently long idle period MUST be set to min(4*MTU, max (2*MTU, 4380 bytes)).

- o The initial cwnd after a retransmission timeout MUST be no more than 1*MTU.

- o The initial value of ssthresh MAY be arbitrarily high (for example, implementations MAY use the size of the receiver advertised window).

- o Whenever cwnd is greater than zero, the endpoint is allowed to have cwnd bytes of data outstanding on that transport address.

- o When cwnd is less than or equal to ssthresh, an SCTP endpoint MUST use the slow start algorithm to increase cwnd only if the current congestion window is being fully utilized, an incoming SACK advances the Cumulative TSN Ack Point, and the data sender is not in Fast Recovery. Only when these three conditions are met can the cwnd be increased; otherwise, the cwnd MUST not be increased. If these conditions are met, then cwnd MUST be increased by, at most, the lesser of 1) the total size of the previously outstanding DATA chunk(s) acknowledged, and 2) the destination's path MTU. This upper bound protects against the ACK-Splitting

attack outlined in [SAVAGE99].

In instances where its peer endpoint is multi-homed, if an endpoint receives a SACK that advances its Cumulative TSN Ack Point, then it should update its cwnd (or cwnds) apportioned to the destination addresses to which it transmitted the acknowledged data. However if the received SACK does not advance the Cumulative TSN Ack Point, the endpoint MUST NOT adjust the cwnd of any of the destination addresses.

Because an endpoint's cwnd is not tied to its Cumulative TSN Ack Point, as duplicate SACKs come in, even though they may not advance the Cumulative TSN Ack Point an endpoint can still use them to clock out new data. That is, the data newly acknowledged by the SACK diminishes the amount of data now in flight to less than cwnd; and so the current, unchanged value of cwnd now allows new data to be sent. On the other hand, the increase of cwnd must be tied to the Cumulative TSN Ack Point advancement as specified above. Otherwise the duplicate SACKs will not only clock out new data, but also will adversely clock out more new data than what has just left the network, during a time of possible congestion.

- o When the endpoint does not transmit data on a given transport address, the cwnd of the transport address should be adjusted to max(cwnd/2, 4*MTU) per RTO.

7.2.2. Congestion Avoidance

When cwnd is greater than ssthresh, cwnd should be incremented by 1*MTU per RTT if the sender has cwnd or more bytes of data outstanding for the corresponding transport address.

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

- o partial_bytes_acked is initialized to 0.

- o Whenever cwnd is greater than ssthresh, upon each SACK arrival that advances the Cumulative TSN Ack Point, increase partial_bytes_acked by the total number of bytes of all new chunks acknowledged in that SACK including chunks acknowledged by the new Cumulative TSN Ack and by Gap Ack Blocks.

- o When partial_bytes_acked is equal to or greater than cwnd and before the arrival of the SACK the sender had cwnd or more bytes of data outstanding (i.e., before arrival of the SACK, flightsize was greater than or equal to cwnd), increase cwnd by MTU, and reset partial_bytes_acked to (partial_bytes_acked - cwnd).

- o Same as in the slow start, 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, 4*MTU) per RTO.

- o When all of the data transmitted by the sender has been acknowledged by the receiver, partial_bytes_acked is initialized to 0.

7.2.3. Congestion Control

Upon detection of packet losses from SACK (see Section 7.2.4), An endpoint should do the following:

```
ssthresh = max(cwnd/2, 4*MTU)
cwnd = ssthresh
partial_bytes_acked = 0
```

Basically, a packet loss causes cwnd to be cut in half.

When the T3-rtx timer expires on an address, SCTP should perform slow start by:

```
ssthresh = max(cwnd/2, 4*MTU)
cwnd = 1*MTU
```

and assure that no more than one SCTP packet will be in flight for that address until the endpoint receives acknowledgement for successful delivery of data to that address.

7.2.4. Fast Retransmit on Gap Reports

In the absence of data loss, an endpoint performs delayed acknowledgement. However, whenever an endpoint notices a hole in the arriving TSN sequence, it SHOULD start sending a SACK back every time a packet arrives carrying data until the hole is filled.

Whenever an endpoint receives a SACK that indicates that some TSNs are missing, it SHOULD wait for 2 further miss indications (via subsequent SACKs for a total of 3 missing reports) on the same TSNs before taking action with regard to Fast Retransmit.

Miss indications SHOULD follow the HTNA (Highest TSN Newly Acknowledged) algorithm. For each incoming SACK, miss indications are incremented only for missing TSNs prior to the highest TSN newly acknowledged in the SACK. A newly acknowledged DATA chunk is one not previously acknowledged in a SACK. If an endpoint is in Fast Recovery and a SACK arrives that advances the Cumulative TSN Ack Point, the miss indications are incremented for all TSNs reported missing in the SACK.

When the third consecutive miss indication is received for a TSN(s), the data sender shall do the following:

1) Mark the DATA chunk(s) with three miss indications for retransmission.

2) If not in Fast Recovery, adjust the ssthresh and cwnd of the destination address(es) to which the missing DATA chunks were last sent, according to the formula described in Section 7.2.3.

3) Determine how many of the earliest (i.e., lowest TSN) DATA chunks marked for retransmission will fit into a single packet, subject to constraint of the path MTU of the destination transport address to which the packet is being sent. Call this value K. Retransmit those K DATA chunks in a single packet. When a Fast Retransmit is being performed, the sender SHOULD ignore the value of cwnd and SHOULD NOT delay retransmission for this single packet.

4) Restart T3-rtx timer only if the last SACK acknowledged the lowest outstanding TSN number sent to that address, or the endpoint is retransmitting the first outstanding DATA chunk sent to that address.

5) Mark the DATA chunk(s) as being fast retransmitted and thus ineligible for a subsequent fast retransmit. Those TSNs marked for retransmission due to the Fast Retransmit algorithm that did not fit in the sent datagram carrying K other TSNs are also marked as ineligible for a subsequent fast retransmit. However, as they are marked for retransmission they will be retransmitted later on as soon as cwnd allows.

6) If not in Fast Recovery, enter Fast Recovery and mark the highest outstanding TSN as the Fast Recovery exit point. When a SACK acknowledges all TSNs up to and including this exit point, Fast Recovery is exited. While in Fast Recovery, the ssthresh and cwnd SHOULD NOT change for any destinations due to a subsequent Fast Recovery event (i.e., one SHOULD NOT reduce the cwnd further due to a subsequent fast retransmit).

Note: Before the above adjustments, if the received SACK also acknowledges new DATA chunks and advances the Cumulative TSN Ack Point, the cwnd adjustment rules defined in Section 7.2.1 and Section 7.2.2 must be applied first.

A straightforward implementation of the above keeps a counter for each TSN hole reported by a SACK. The counter increments for each consecutive SACK reporting the TSN hole. After reaching 3 and

starting the fast retransmit procedure, the counter resets to 0.

Because cwnd in SCTP indirectly bounds the number of outstanding TSN's, the effect of TCP fast-recovery is achieved automatically with no adjustment to the congestion control window size.

7.3. Path MTU Discovery

[RFC1191] specifies "Path MTU Discovery", whereby an endpoint maintains an estimate of the maximum transmission unit (MTU) along a given Internet path and refrains from sending packets along that path which exceed the MTU, other than occasional attempts to probe for a change in the Path MTU (PMTU). [RFC1191] is thorough in its discussion of the MTU discovery mechanism and strategies for determining the current end-to-end MTU setting as well as detecting changes in this value. [RFC1981] specifies the same mechanisms for IPv6. An SCTP sender using IPv6 MUST use Path MTU Discovery unless all packets are less than the minimum IPv6 MTU [RFC2460].

An endpoint SHOULD apply these techniques, and SHOULD do so on a per-destination-address basis.

There are 4 ways in which SCTP differs from the description in [RFC1191] of applying MTU discovery to TCP:

1) SCTP associations can span multiple addresses. An endpoint MUST maintain separate MTU estimates for each destination address of its peer.

2) Elsewhere in this document, when the term "MTU" is discussed, it refers to the MTU associated with the destination address corresponding to the context of the discussion.

3) Unlike TCP, SCTP does not have a notion of "Maximum Segment Size". Accordingly, the MTU for each destination address SHOULD be initialized to a value no larger than the link MTU for the local interface to which packets for that remote destination address will be routed.

4) Since data transmission in SCTP is naturally structured in terms of TSNs rather than bytes (as is the case for TCP), the discussion in Section 6.5 of [RFC1191] applies: When retransmitting an IP datagram to a remote address for which the IP datagram appears too large for the path MTU to that address, the IP datagram SHOULD be retransmitted without the DF bit set, allowing it to possibly be fragmented. Transmissions of new IP datagrams MUST have DF set.

5) The sender should track an association PMTU which will be the smallest PMTU discovered for all of the peer's destination addresses. When fragmenting messages into multiple parts this association PMTU should be used to calculate the size of each fragment. This will allow retransmissions to be seamlessly sent to an alternate address without encountering IP fragmentation.

Other than these differences, the discussion of TCP's use of MTU discovery in [RFC1191] and [RFC1981] applies to SCTP on a per-destination-address basis.

Note: For IPv6 destination addresses the DF bit does not exist, instead the IP datagram must be fragmented as described in [RFC2460].

8. Fault Management

8.1. Endpoint Failure Detection

An endpoint shall keep a counter on the total number of consecutive retransmissions to its peer (this includes retransmissions to all the destination transport addresses of the peer if it is multi-homed), including unacknowledged HEARTBEAT Chunks. If the value of this counter exceeds the limit indicated in the protocol parameter 'Association.Max.Retrans', the endpoint shall consider the peer endpoint unreachable and shall stop transmitting any more data to it (and thus the association enters the CLOSED state). In addition, the endpoint MAY report the failure to the upper layer and optionally report back all outstanding user data remaining in its outbound queue. The association is automatically closed when the peer endpoint becomes unreachable.

The counter shall be reset each time a DATA chunk sent to that peer endpoint is acknowledged (by the reception of a SACK), or a HEARTBEAT-ACK is received from the peer endpoint.

8.2. Path Failure Detection

When its peer endpoint is multi-homed, an endpoint should keep a error counter for each of the destination transport addresses of the peer endpoint.

Each time the T3-rtx timer expires on any address, or when a HEARTBEAT sent to an idle address is not acknowledged within a RTO, the error counter of that destination address will be incremented. When the value in the error counter exceeds the protocol parameter 'Path.Max.Retrans' of that destination address, the endpoint should mark the destination transport address as inactive, and a

notification SHOULD be sent to the upper layer.

When an outstanding TSN is acknowledged or a HEARTBEAT sent to that address is acknowledged with a HEARTBEAT ACK, the endpoint shall clear the error counter of the destination transport address to which the DATA chunk was last sent (or HEARTBEAT was sent). When the peer endpoint is multi-homed and the last chunk sent to it was a retransmission to an alternate address, there exists an ambiguity as to whether or not the acknowledgement should be credited to the address of the last chunk sent. However, this ambiguity does not seem to bear any significant consequence to SCTP behavior. If this ambiguity is undesirable, the transmitter may choose not to clear the error counter if the last chunk sent was a retransmission.

Note: When configuring the SCTP endpoint, the user should avoid having the value of 'Association.Max.Retrans' larger than the summation of the 'Path.Max.Retrans' of all the destination addresses for the remote endpoint. Otherwise, all the destination addresses may become inactive while the endpoint still considers the peer endpoint reachable. When this condition occurs, how the SCTP chooses to function is implementation specific.

When the primary path is marked inactive (due to excessive retransmissions, for instance), the sender MAY automatically transmit new packets to an alternate destination address if one exists and is active. If more than one alternate address is active when the primary path is marked inactive only ONE transport address SHOULD be chosen and used as the new destination transport address.

8.3. Path Heartbeat

By default, an SCTP endpoint SHOULD monitor the reachability of the idle destination transport address(es) of its peer by sending a HEARTBEAT chunk periodically to the destination transport address(es). HEARTBEAT sending MAY begin upon reaching the ESTABLISHED state and is discontinued after sending either SHUTDOWN or SHUTDOWN-ACK. A receiver of a HEARTBEAT MUST respond to a HEARTBEAT with a HEARTBEAT-ACK after entering the COOKIE-ECHOED state (INIT sender) or the ESTABLISHED state (INIT receiver), up until reaching the SHUTDOWN-SENT state (SHUTDOWN sender) or the SHUTDOWN-ACK-SENT state (SHUTDOWN receiver).

A destination transport address is considered "idle" if no new chunk which can be used for updating path RTT (usually including first transmission DATA, INIT, COOKIE ECHO, HEARTBEAT etc.) and no HEARTBEAT has been sent to it within the current heartbeat period of that address. This applies to both active and inactive destination addresses.

The upper layer can optionally initiate the following functions:

A) Disable heartbeat on a specific destination transport address of a given association,

B) Change the HB.interval,

C) Re-enable heartbeat on a specific destination transport address of a given association, and,

D) Request an on-demand HEARTBEAT on a specific destination transport address of a given association.

The endpoint should increment the respective error counter of the destination transport address each time a HEARTBEAT is sent to that address and not acknowledged within one RTO.

When the value of this counter reaches the protocol parameter 'Path.Max.Retrans', the endpoint should mark the corresponding destination address as inactive if it is not so marked, and may also optionally report to the upper layer the change of reachability of this destination address. After this, the endpoint should continue HEARTBEAT on this destination address but should stop increasing the counter.

The sender of the HEARTBEAT chunk should include in the Heartbeat Information field of the chunk the current time when the packet is sent out and the destination address to which the packet is sent.

IMPLEMENTATION NOTE: An alternative implementation of the heartbeat mechanism that can be used is to increment the error counter variable every time a HEARTBEAT is sent to a destination. Whenever a HEARTBEAT ACK arrives, the sender SHOULD clear the error counter of the destination that the HEARTBEAT was sent to. This in effect would clear the previously stroked error (and any other error counts as well).

The receiver of the HEARTBEAT should immediately respond with a HEARTBEAT ACK that contains the Heartbeat Information TLV, together with any other received TLVs, copied unchanged from the received HEARTBEAT chunk.

Upon the receipt of the HEARTBEAT ACK, the sender of the HEARTBEAT should clear the error counter 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 optionally report to the upper layer when an inactive destination address is marked as active due to the reception of the latest

HEARTBEAT ACK. The receiver of the HEARTBEAT ACK must also clear the association overall error count as well (as defined in Section 8.1).

The receiver of the HEARTBEAT ACK should also perform an RTT measurement for that destination transport address using the time value carried in the HEARTBEAT ACK chunk.

On an idle destination address that is allowed to heartbeat, it is recommended that a HEARTBEAT chunk is sent once per RTO of that destination address plus the protocol parameter 'HB.interval', with jittering of +/- 50% of the RTO value, and exponential back-off of the RTO if the previous HEARTBEAT is unanswered.

A primitive is provided for the SCTP user to change the HB.interval and turn on or off the heartbeat on a given destination address. The heartbeat interval set by the SCTP user is added to the RTO of that destination (including any exponential backoff). Only one heartbeat should be sent each time the heartbeat timer expires (if multiple destinations are idle). It is a implementation decision on how to choose which of the candidate idle destinations to heartbeat to (if more than one destination is idle).

Note: When tuning the heartbeat interval, there is a side effect that SHOULD be taken into account. When this value is increased, i.e. the HEARTBEAT takes longer, the detection of lost ABORT messages takes longer as well. If a peer endpoint ABORTs the association for any reason and the ABORT chunk is lost, the local endpoint will only discover the lost ABORT by sending a DATA chunk or HEARTBEAT chunk (thus causing the peer to send another ABORT). This must be considered when tuning the HEARTBEAT timer. If the HEARTBEAT is disabled only sending DATA to the association will discover a lost ABORT from the peer.

8.4. Handle "Out of the blue" Packets

An SCTP packet is called an "out of the blue" (OOTB) packet if it is correctly formed (i.e., passed the receiver's CRC32c check; see Section 6.8), but the receiver is not able to identify the association to which this packet belongs.

The receiver of an OOTB packet MUST do the following:

1) If the OOTB packet is to or from a non-unicast address, a receiver SHOULD silently discard the packet. Otherwise,

2) If the OOTB packet contains an ABORT chunk, the receiver MUST silently discard the OOTB packet and take no further action. Otherwise,

3) If the packet contains an INIT chunk with a Verification Tag set to '0', process it as described in Section 5.1. If, for whatever reason, the INIT cannot be processed normally and an ABORT has to be sent in response, the Verification Tag of the packet containing the ABORT chunk MUST be the Initiate tag of the received INIT chunk, and the T-Bit of the ABORT chunk has to be set to 0, indicating that the Verification Tag is NOT reflected.

4) If the packet contains a COOKIE ECHO in the first chunk, process it as described in Section 5.1. Otherwise,

5) If the packet contains a SHUTDOWN ACK chunk, the receiver should respond to the sender of the OOTB packet with a SHUTDOWN COMPLETE. When sending the SHUTDOWN COMPLETE, the receiver of the OOTB packet must fill in the Verification Tag field of the outbound packet with the Verification Tag received in the SHUTDOWN ACK and set the T-bit in the Chunk Flags to indicate that the Verification Tag is reflected. Otherwise,

6) If the packet contains a SHUTDOWN COMPLETE chunk, the receiver should silently discard the packet and take no further action. Otherwise,

7) If the packet contains a "Stale cookie" ERROR or a COOKIE ACK the SCTP Packet should be silently discarded. Otherwise,

8) The receiver should respond to the sender of the OOTB packet with an ABORT. When sending the ABORT, the receiver of the OOTB packet MUST fill in the Verification Tag field of the outbound packet with the value found in the Verification Tag field of the OOTB packet and set the T-bit in the Chunk Flags to indicate that the Verification Tag is reflected. After sending this ABORT, the receiver of the OOTB packet shall discard the OOTB packet and take no further action.

8.5. Verification Tag

The Verification Tag rules defined in this section apply when sending or receiving SCTP packets which do not contain an INIT, SHUTDOWN COMPLETE, COOKIE ECHO (see Section 5.1), ABORT or SHUTDOWN ACK chunk. The rules for sending and receiving SCTP packets containing one of these chunk types are discussed separately in Section 8.5.1.

When sending an SCTP packet, the endpoint MUST fill in the

Verification Tag field of the outbound packet with the tag value in the Initiate Tag parameter of the INIT or INIT ACK received from its peer.

When receiving an SCTP packet, the endpoint MUST ensure that the value in the Verification Tag field of the received SCTP packet matches its own Tag. If the received Verification Tag value does not match the receiver's own tag value, the receiver shall silently discard the packet and shall not process it any further except for those cases listed in Section 8.5.1 below.

8.5.1. Exceptions in Verification Tag Rules

A) Rules for packet carrying INIT:

- The sender MUST set the Verification Tag of the packet to 0.

- When an endpoint receives an SCTP packet with the Verification Tag set to 0, it should verify that the packet contains only an INIT chunk. Otherwise, the receiver MUST silently discard the packet.

B) Rules for packet carrying ABORT:

- The endpoint MUST always fill in the Verification Tag field of the outbound packet with the destination endpoint's tag value, if it is known.

- If the ABORT is sent in response to an OOTB packet, the endpoint MUST follow the procedure described in Section 8.4

- The receiver of an ABORT MUST accept the packet if the Verification Tag field of the packet matches its own tag and the T bit is not set OR if it is set to its peer's tag and the T bit is set in the Chunk Flags. Otherwise, the receiver MUST silently discard the packet and take no further action.

C) Rules for packet carrying SHUTDOWN COMPLETE:

- When sending a SHUTDOWN COMPLETE, if the receiver of the SHUTDOWN ACK has a TCB, then the destination endpoint's tag MUST be used, and the T-bit MUST NOT be set. Only where no TCB exists should the sender use the Verification Tag from the SHUTDOWN ACK, and MUST set the T-bit.

- The receiver of a SHUTDOWN COMPLETE shall accept the packet if the Verification Tag field of the packet matches its own tag and the T bit is not set OR if it is set to its peer's tag and the T bit is set in the Chunk Flags. Otherwise, the receiver MUST silently discard the packet and take no further action. An endpoint MUST ignore the SHUTDOWN COMPLETE if it is not in the SHUTDOWN-ACK-SENT state.

D) Rules for packet carrying a COOKIE ECHO

- When sending a COOKIE ECHO, the endpoint MUST use the value of the Initial Tag received in the INIT ACK.

- The receiver of a COOKIE ECHO follows the procedures in Section 5.

E) Rules for packet carrying a SHUTDOWN ACK

- If the receiver is in COOKIE-ECHOED or COOKIE-WAIT state the procedures in Section 8.4 SHOULD be followed, in other words it should be treated as an Out Of The Blue packet.

9. Termination of Association

An endpoint should terminate its association when it exits from service. An association can be terminated by either abort or shutdown. An abort of an association is abortive by definition in that any data pending on either end of the association is discarded and not delivered to the peer. A shutdown of an association is considered a graceful close where all data in queue by either endpoint is delivered to the respective peers. However, in the case of a shutdown, SCTP does not support a half-open state (like TCP) wherein one side may continue sending data while the other end is closed. When either endpoint performs a shutdown, the association on each peer will stop accepting new data from its user and only deliver data in queue at the time of sending or receiving the SHUTDOWN chunk.

9.1. Abort of an Association

When an endpoint decides to abort an existing association, it MUST send an ABORT chunk to its peer endpoint. The sender MUST fill in the peer's Verification Tag in the outbound packet and MUST NOT bundle any DATA chunk with the ABORT. If the association is aborted on request of the upper layer, a User-Initiated Abort error cause (see Section 3.3.10.12) SHOULD be present in the ABORT chunk.

An endpoint MUST NOT respond to any received packet that contains an

ABORT chunk (also see Section 8.4).

An endpoint receiving an ABORT MUST apply the special Verification Tag check rules described in Section 8.5.1.

After checking the Verification Tag, the receiving endpoint MUST remove the association from its record and SHOULD report the termination to its upper layer. If a User-Initiated Abort error cause is present in the ABORT chunk, the Upper Layer Abort Reason SHOULD be made available to the upper layer.

9.2. Shutdown of an Association

Using the SHUTDOWN primitive (see Section 10.1), the upper layer of an endpoint in an association can gracefully close the association. This will allow all outstanding DATA chunks from the peer of the shutdown initiator to be delivered before the association terminates.

Upon receipt of the SHUTDOWN primitive from its upper layer, the endpoint enters SHUTDOWN-PENDING state and remains there until all outstanding data has been acknowledged by its peer. The endpoint accepts no new data from its upper layer, but retransmits data to the far end if necessary to fill gaps.

Once all its outstanding data has been acknowledged, the endpoint shall send a SHUTDOWN chunk to its peer including in the Cumulative TSN Ack field the last sequential TSN it has received from the peer. It shall then start the T2-shutdown timer and enter the SHUTDOWN-SENT state. If the timer expires, the endpoint must re-send the SHUTDOWN with the updated last sequential TSN received from its peer.

The rules in Section 6.3 MUST be followed to determine the proper timer value for T2-shutdown. To indicate any gaps in TSN, the endpoint may also bundle a SACK with the SHUTDOWN chunk in the same SCTP packet.

An endpoint should limit the number of retransmissions of the SHUTDOWN chunk to the protocol parameter 'Association.Max.Retrans'. If this threshold is exceeded the endpoint should destroy the TCB and MUST report the peer endpoint unreachable to the upper layer (and thus the association enters the CLOSED state). The reception of any packet from its peer (i.e. as the peer sends all of its queued DATA chunks) should clear the endpoint's retransmission count and restart the T2-Shutdown timer, giving its peer ample opportunity to transmit all of its queued DATA chunks that have not yet been sent.

Upon the reception of the SHUTDOWN, the peer endpoint shall

- enter the SHUTDOWN-RECEIVED state,

- stop accepting new data from its SCTP user

- verify, by checking the Cumulative TSN Ack field of the chunk, that all its outstanding DATA chunks have been received by the SHUTDOWN sender.

Once an endpoint as reached the SHUTDOWN-RECEIVED state it MUST NOT send a SHUTDOWN in response to a ULP request, and should discard subsequent SHUTDOWN chunks.

If there are still outstanding DATA chunks left, the SHUTDOWN receiver MUST continue to follow normal data transmission procedures defined in Section 6, until all outstanding DATA chunks are acknowledged; however, the SHUTDOWN receiver MUST NOT accept new data from its SCTP user.

While in SHUTDOWN-SENT state, the SHUTDOWN sender MUST immediately respond to each received packet containing one or more DATA chunks with a SHUTDOWN chunk and restart the T2-shutdown timer. If a SHUTDOWN chunk by itself cannot acknowledge all of the received DATA chunks (i.e., there are TSNs that can be acknowledged that are larger than the cumulative TSN, and thus gaps exist in the TSN sequence), or if duplicate TSNs have been received, then a SACK chunk MUST also be sent.

The sender of the SHUTDOWN MAY also start an overall guard timer 'T5-shutdown-guard' to bound the overall time for the shutdown sequence. At the expiration of this timer, the sender SHOULD abort the association by sending an ABORT chunk. If the 'T5-shutdown-guard' timer is used, it SHOULD be set to the recommended value of 5 times 'RTO.Max'.

If the receiver of the SHUTDOWN has no more outstanding DATA chunks, the SHUTDOWN receiver MUST send a SHUTDOWN ACK and start a T2-shutdown timer of its own, entering the SHUTDOWN-ACK-SENT state. If the timer expires, the endpoint must re-send the SHUTDOWN ACK.

The sender of the SHUTDOWN ACK should limit the number of retransmissions of the SHUTDOWN ACK chunk to the protocol parameter 'Association.Max.Retrans'. If this threshold is exceeded, the endpoint should destroy the TCB and may report the peer endpoint unreachable to the upper layer (and thus the association enters the CLOSED state).

Upon the receipt of the SHUTDOWN ACK, the SHUTDOWN sender shall stop the T2-shutdown timer, send a SHUTDOWN COMPLETE chunk to its peer,

and remove all record of the association.

Upon reception of the SHUTDOWN COMPLETE chunk the endpoint will verify that it is in SHUTDOWN-ACK-SENT state, if it is not the chunk should be discarded. If the endpoint is in the SHUTDOWN-ACK-SENT state the endpoint should stop the T2-shutdown timer and remove all knowledge of the association (and thus the association enters the CLOSED state).

An endpoint SHOULD assure that all its outstanding DATA chunks have been acknowledged before initiating the shutdown procedure.

An endpoint should reject any new data request from its upper layer if it is in SHUTDOWN-PENDING, SHUTDOWN-SENT, SHUTDOWN-RECEIVED, or SHUTDOWN-ACK-SENT state.

If an endpoint is in SHUTDOWN-ACK-SENT state and receives an INIT chunk (e.g., if the SHUTDOWN COMPLETE was lost) with source and destination transport addresses (either in the IP addresses or in the INIT chunk) that belong to this association, it should discard the INIT chunk and retransmit the SHUTDOWN ACK chunk.

Note: Receipt of an INIT with the same source and destination IP addresses as used in transport addresses assigned to an endpoint but with a different port number indicates the initialization of a separate association.

The sender of the INIT or COOKIE ECHO should respond to the receipt of a SHUTDOWN-ACK with a stand-alone SHUTDOWN COMPLETE in an SCTP packet with the Verification Tag field of its common header set to the same tag that was received in the SHUTDOWN ACK packet. This is considered an Out of the Blue packet as defined in Section 8.4. The sender of the INIT lets T1-init continue running and remains in the COOKIE-WAIT or COOKIE-ECHOED state. Normal T1-init timer expiration will cause the INIT or COOKIE chunk to be retransmitted and thus start a new association.

If a SHUTDOWN is received in COOKIE WAIT or COOKIE ECHOED states the SHUTDOWN chunk SHOULD be silently discarded.

If an endpoint is in SHUTDOWN-SENT state and receives a SHUTDOWN chunk from its peer, the endpoint shall respond immediately with a SHUTDOWN ACK to its peer, and move into a SHUTDOWN-ACK-SENT state restarting its T2-shutdown timer.

If an endpoint is in the SHUTDOWN-ACK-SENT state and receives a SHUTDOWN ACK, it shall stop the T2-shutdown timer, send a SHUTDOWN COMPLETE chunk to its peer, and remove all record of the association.

10. 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 shown for illustrative purposes. 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.

10.1. 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 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. Once SCTP is initialized, ULP can communicate directly with other endpoints without re-invoking this primitive.

SCTP will return a local SCTP instance name to the ULP.

Mandatory attributes:

None.

Optional attributes:

The following types of attributes may be passed along with the primitive:

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

- o local eligible address list - An address list that the local SCTP endpoint should bind. By default, if an address list is not included, all IP addresses assigned to the host should be used by the local endpoint.

IMPLEMENTATION NOTE: If this optional attribute is supported by an implementation, it will be the responsibility of the implementation to enforce that the IP source address field of any SCTP packets sent out by this endpoint contains one of the IP addresses indicated in the local eligible address list.

B) Associate

```
Format: ASSOCIATE(local SCTP instance name, destination transport addr,
        outbound stream count)
-> association id [,destination transport addr 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 defines the endpoint (see Section 1.4). If the local SCTP instance has not been initialized, the ASSOCIATE is considered an error.

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.

Other association parameters may be returned, including the complete destination transport addresses of the peer as well as the outbound stream count of the local endpoint. One of the transport address from the returned destination addresses will be selected by the local endpoint as default primary path for sending SCTP packets to this peer. The returned "destination transport addr list" can be used by the ULP to change the default primary path or to force sending a packet to a specific transport address.

IMPLEMENTATION 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.

Mandatory attributes:

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

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

o outbound stream count - the number of outbound streams the ULP would like to open towards this peer endpoint.

Optional attributes:

None.

C) Shutdown

```
Format: SHUTDOWN(association id)
-> result
```

Gracefully closes an association. Any locally queued user data will be delivered to the peer. The association will be terminated only after the peer acknowledges all the SCTP packets 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:

o association id - local handle to the SCTP association

Optional attributes:

None.

D) Abort

```
Format: ABORT(association id [, Upper Layer Abort Reason])
-> result
```

Ungracefully closes an association. Any locally queued user data will be discarded, and an ABORT chunk 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:

o association id - local handle to the SCTP association

Optional attributes:

o Upper Layer Abort Reason - Reason of the abort to be passed to the peer.

None.

E) Send

```
Format: SEND(association id, buffer address, byte count [,context]
        [,stream id] [,life time] [,destination transport address]
        [,unorder flag] [,no-bundle flag] [,payload protocol-id] )
-> result
```

This is the main method to send user data via SCTP.

Mandatory attributes:

o association id - local handle to the SCTP association

o buffer address - the location where the user message to be transmitted is stored;

o byte count - The size of the user data in number of bytes;

Optional attributes:

o context - an optional 32 bit integer that will be carried in the sending failure notification to the ULP if the transportation of this User Message fails.

o stream id - to indicate which stream to send the data on. If not specified, stream 0 will be used.

o life time - specifies the life time of the user data. The user data will not be sent by SCTP after the life time expires. This parameter can be used to avoid efforts to transmit stale user messages. SCTP notifies the ULP if the data cannot be initiated to transport (i.e. sent to the destination via SCTP's send primitive) within the life time variable. However, the user data

will be transmitted if SCTP has attempted to transmit a chunk before the life time expired.

IMPLEMENTATION NOTE: In order to better support the data lifetime option, the transmitter may hold back the assigning of the TSN number to an outbound DATA chunk to the last moment. And, for implementation simplicity, once a TSN number has been assigned the sender should consider the send of this DATA chunk as committed, overriding any lifetime option attached to the DATA chunk.

- o destination transport address - specified as one of the destination transport addresses of the peer endpoint to which this packet should be sent. Whenever possible, SCTP should use this destination transport address for sending the packets, instead of the current primary path.

- o unorder flag - this flag, if present, indicates that the user would like the data delivered in an unordered fashion to the peer (i.e., the U flag is set to 1 on all DATA chunks carrying this message).

- o no-bundle flag - instructs SCTP not to bundle this user data with other outbound DATA chunks. SCTP MAY still bundle even when this flag is present, when faced with network congestion.

- o payload protocol-id - A 32 bit unsigned integer that is to be passed to the peer indicating the type of payload protocol data being transmitted. This value is passed as opaque data by SCTP.

F) Set Primary

```
Format: SETPRIMARY(association id, destination transport address,
                   [source transport address] )
-> result
```

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

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 packets. This overrides the current primary address information maintained by the local SCTP endpoint.

Optional attributes:

o source transport address - optionally, some implementations may allow you to set the default source address placed in all outgoing IP datagrams.

G) Receive

```
 Format: RECEIVE(association id, buffer address, buffer size
         [,stream id])
 -> byte count [,transport address] [,stream id] [,stream sequence
    number] [,partial flag] [,delivery number] [,payload protocol-id]
```

This primitive shall read the first user message in the SCTP in-queue into the buffer specified by ULP, if there is one available. The size of the message read, in bytes, 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 messages available for retrieval, etc. For ordered messages, their stream sequence number may also be returned.

Depending upon the implementation, if this primitive is invoked when no message 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 message.

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

Optional attributes:

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

- o stream sequence number - the stream sequence number assigned by the sending SCTP peer.

- o partial flag - if this returned flag is set to 1, then this Receive contains a partial delivery of the whole message. When this flag is set, the stream id and stream sequence number MUST accompany this receive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.

- o payload protocol-id - A 32 bit unsigned integer that is received from the peer indicating the type of payload protocol of the received data. This value is passed as opaque data by SCTP.

H) Status

```
Format: STATUS(association id)
-> status data
```

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

```
association connection state,
destination transport address list,
destination transport address reachability states,
current receiver window size,
current congestion window sizes,
number of unacknowledged DATA chunks,
number of DATA chunks pending receipt,
primary path,
most recent SRTT on primary path,
RTO on primary path,
SRTT and RTO on other destination addresses, etc.
```

Mandatory attributes:

o association id - local handle to the SCTP association

Optional attributes:

None.

I) Change Heartbeat

```
Format: CHANGEHEARTBEAT(association id, destination transport address,
        new state [,interval])
-> result
```

Instructs the local endpoint to enable or disable heartbeat on the specified destination transport address.

The result of attempting this operation shall be returned.

Note: Even when enabled, heartbeat will not take place if the destination transport address is not idle.

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.

o new state - the new state of heartbeat for this destination transport address (either enabled or disabled).

Optional attributes:

o interval - if present, indicates the frequency of the heartbeat if this is to enable heartbeat on a destination transport address. This value is added to the RTO of the destination transport address. This value, if present, effects all destinations.

J) Request HeartBeat

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

Instructs the local endpoint to perform a HeartBeat on the specified destination transport address of the given association. The returned result should indicate whether the transmission of the HEARTBEAT chunk to the destination address is successful.

Mandatory attributes:

o association id - local handle to the SCTP association

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

K) Get SRTT Report

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

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

Mandatory attributes:

o association id - local handle to the SCTP association

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

L) Set Failure Threshold

Format: SETFAILURETHRESHOLD(association id, destination transport address, failure threshold) -> result

This primitive allows the local SCTP to customize the reachability failure detection threshold 'Path.Max.Retrans' for the specified destination address.

Mandatory attributes:

o association id - local handle to the SCTP association

o destination transport address - the transport address of the association on which the failure detection threshold is to be set.

o failure threshold - the new value of 'Path.Max.Retrans' for the destination address.

M) Set Protocol Parameters

Format: SETPROTOCOLPARAMETERS(association id, [,destination transport address,] protocol parameter list) -> result

This primitive allows the local SCTP to customize the protocol parameters.

Mandatory attributes:

o association id - local handle to the SCTP association

o protocol parameter list - The specific names and values of the protocol parameters (e.g., Association.Max.Retrans [see Section 15) that the SCTP user wishes to customize.

Optional attributes:

o destination transport address - some of the protocol parameters may be set on a per destination transport address basis.

N) Receive unsent message

Format: RECEIVE_UNSENT(data retrieval id, buffer address, buffer size [,stream id] [, stream sequence number] [,partial flag] [,payload protocol-id])

o data retrieval id - The identification passed to the ULP in the failure notification.

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

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

Optional attributes:

o stream id - this is a return value that is set to indicate which stream the data was sent to.

o stream sequence number - this value is returned indicating the stream sequence number that was associated with the message.

o partial flag - if this returned flag is set to 1, then this message is a partial delivery of the whole message. When this flag is set, the stream id and stream sequence number MUST accompany this receive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.

- o payload protocol-id - The 32 bit unsigned integer that was sent to be sent to the peer indicating the type of payload protocol of the received data.

- o Receive unacknowledged message

```
 Format: RECEIVE_UNACKED(data retrieval id, buffer address, buffer size,
         [,stream id] [, stream sequence number] [,partial flag]
         [,payload protocol-id])
```

- o data retrieval id - The identification passed to the ULP in the failure notification.

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

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

Optional attributes:

- o stream id - this is a return value that is set to indicate which stream the data was sent to.

- o stream sequence number - this value is returned indicating the stream sequence number that was associated with the message.

- o partial flag - if this returned flag is set to 1, then this message is a partial delivery of the whole message. When this flag is set, the stream id and stream sequence number MUST accompany this receive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.

- o payload protocol-id - The 32 bit unsigned integer that was sent to be sent to the peer indicating the type of payload protocol of the received data.

P) Destroy SCTP instance

```
Format: DESTROY(local SCTP instance name)
```

- o local SCTP instance name - this is the value that was passed to the application in the initialize primitive and it indicates which SCTP instance to be destroyed.

10.2. 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.

IMPLEMENTATION NOTE: In some cases this may be done through a separate socket or error channel.

A) DATA ARRIVE notification

SCTP shall invoke this notification on the ULP when a user message 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 message can not be delivered SCTP shall invoke this notification on the ULP.

The following may be optionally be passed with the notification:

- o association id - local handle to the SCTP association
- o data retrieval id - an identification used to retrieve unsent and unacknowledged data.
- o cause code - indicating the reason of the failure, e.g., size too large, message life-time expiration, etc.
- o context - optional information associated with this message (see D in Section 10.1).

C) NETWORK STATUS CHANGE notification

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

The following shall be passed with the notification:

o association id - local handle to the SCTP association

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 user messages, 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 association id - local handle to the SCTP association

o status - This indicates what type of event has occurred

o destination transport address list - the complete 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

o inbound stream count - the number of streams the peer endpoint has requested with this association (this may not be the same number as 'outbound stream count').

E) COMMUNICATION LOST notification

When SCTP loses communication to an endpoint completely (e.g., via Heartbeats) or detects that the endpoint has performed an abort operation, it shall invoke this notification on the ULP.

The following shall be passed with the notification:

o association id - local handle to the SCTP association

o status - This indicates what type of event has occurred; the status may indicate that a failure OR a normal termination event occurred in response to a shutdown or abort request.

The following may be passed with the notification:

o data retrieval id - an identification used to retrieve unsent and unacknowledged data.

o last-acked - the TSN last acked by that peer endpoint.

o last-sent - the TSN last sent to that peer endpoint.

o Upper Layer Abort Reason - The abort reason specified in case of a user-initiated abort.

F) COMMUNICATION ERROR notification

When SCTP receives an ERROR chunk from its peer and decides to notify its ULP, it can invoke this notification on the ULP.

The following can be passed with the notification:

o association id - local handle to the SCTP association

o error info - this indicates the type of error and optionally some additional information received through the ERROR chunk.

G) RESTART notification

When SCTP detects that the peer has restarted, it may send this notification to its ULP.

The following can be passed with the notification:

o association id - local handle to the SCTP association

H) SHUTDOWN COMPLETE notification

When SCTP completes the shutdown procedures (section 9.2) this notification is passed to the upper layer.

The following can be passed with the notification:

o association id - local handle to the SCTP association

11. Security Considerations

11.1. Security Objectives

As a common transport protocol designed to reliably carry time-sensitive user messages, such as billing or signaling 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

11.2. SCTP Responses To Potential Threats

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

Operators of systems running SCTP should consult [RFC2196] for guidance in securing their site.

11.2.1. Countering Insider Attacks

The principles of [RFC2196] should be applied to minimize the risk of theft of information or sabotage by insiders. Such procedures include publication of security policies, control of access at the physical, software, and network levels, and separation of services.

11.2.2. 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 integrity protection is required. If this additional protection were provided in the application-layer, the SCTP header would remain vulnerable to deliberate integrity attacks. While the existing SCTP mechanisms for detection of packet replays are considered sufficient for normal operation, stronger protections are needed to protect SCTP when the operating environment contains significant risk of deliberate attacks from a sophisticated adversary.

In order to promote software code-reuse, to avoid re-inventing the wheel, and to avoid gratuitous complexity to SCTP, the IP

Authentication Header [RFC4302] SHOULD be used when the threat environment requires stronger integrity protections, but does not require confidentiality.

A widely implemented BSD Sockets API extension exists for applications to request IP security services, such as AH or ESP from an operating system kernel. Applications can use such an API to request AH whenever AH use is appropriate.

11.2.3. Protecting Confidentiality

In most cases, the risk of breach of confidentiality applies to the signaling data payload, not to the SCTP or lower-layer protocol overheads. If that is true, encryption of the SCTP user data only might be considered. As with the supplementary checksum service, user data encryption MAY be performed by the SCTP user application. Alternately, the user application MAY use an implementation-specific API to request that the IP Encapsulating Security Payload (ESP) [RFC4303] be used to provide confidentiality and integrity.

Particularly for mobile users, the requirement for confidentiality might include the masking of IP addresses and ports. In this case ESP SHOULD be used instead of application-level confidentiality. If ESP is used to protect confidentiality of SCTP traffic, an ESP cryptographic transform that includes cryptographic integrity protection MUST be used, because if there is a confidentiality threat there will also be a strong integrity threat.

Whenever ESP is in use, application-level encryption is not generally required.

Regardless of where confidentiality is provided, the IKEv2 [RFC4306] SHOULD be used for key management.

Operators should consult [RFC4301] for more information on the security services available at and immediately above the Internet Protocol layer.

11.2.4. 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 node. Blind denial of service attacks may take the form of flooding, masquerade, or improper monopolization of services.

11.2.4.1. Flooding

The objective of flooding is to cause loss of service and incorrect behavior 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 node or at resources in the intervening IP Access Links or the Internet. 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.

- not responding to unexpected packets sent to non-unicast addresses.

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 system 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 node until the handshake is completed, and its use of a Verification Tag to prevent insertion of extraneous packets into the flow of an established association.

The IP Authentication Header and Encapsulating Security Payload might be useful in reducing the risk of certain kinds of denial of service attacks.

The use of the Host Name feature in the INIT chunk could be used to flood a target DNS server. A large backlog of DNS queries, resolving the Host Name received in the INIT chunk to IP addresses, could be accomplished by sending INIT's to multiple hosts in a given domain.

In addition, an attacker could use the Host Name feature in an indirect attack on a third party by sending large numbers of INITs to random hosts containing the host name of the target. In addition to the strain on DNS resources, this could also result in large numbers of INIT ACKs being sent to the target. One method to protect against this type of attack is to verify that the IP addresses received from DNS include the source IP address of the original INIT. If the list of IP addresses received from DNS does not include the source IP address of the INIT, the endpoint MAY silently discard the INIT. This last option will not protect against the attack against the DNS.

11.2.4.2. Blind Masquerade

Masquerade can be used to deny service in several ways:

- by tying up resources at the target SCTP node to which the impersonated node has limited access. For example, the target node may by policy permit a maximum of one SCTP association with the impersonated SCTP node. The masquerading attacker may attempt to establish an association purporting to come from the impersonated node 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 node to be locked out of the target SCTP node.

- by interfering with an established association by inserting extraneous content such as a SHUTDOWN request.

SCTP reduces the risk of blind masquerade attacks through IP spoofing by use of the four-way startup handshake. Man-in-the-middle masquerade attacks are discussed in Section 11.3 below. Because the initial exchange is memoryless, no lockout mechanism is triggered by blind masquerade attacks. In addition, the INIT ACK containing the State Cookie is transmitted back to the IP address from which it received the INIT. Thus the attacker would not receive the INIT ACK containing the State Cookie. SCTP protects against insertion of extraneous packets 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 node more vulnerable to flooding attacks. Logging is pointless without the establishment of operating procedures to review and analyze the logs on a routine basis.

11.2.4.3. 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 node or of the shared resources between the attacker and the target node. Possible attacks include the opening of a large number of associations between the attacker's node and the target, or transfer of large volumes of information within a legitimately-established association.

Policy limits should be placed on the number of associations per adjoining SCTP node. 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.

11.3. 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 node 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 packets passing through a target SCTP association in real time.

The interception attack is countered by the confidentiality measures discussed in Section 11.2.3 above.

Section 11.2.4.2 describes how SCTP is resistant to blind masquerade attacks, as a result of the four-way startup handshake and the Verification Tag. The Verification 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 packets sent and received in an association. For example, the INIT ACK will have sufficient information sent on the wire for an adversary in the middle to hijack an existing SCTP 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 SCTP packets passed.

SCTP also provides no protection against attacks originating at or beyond the SCTP node 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 11.2.1.

11.4. SCTP Interactions with Firewalls

It is helpful for some firewalls if they can inspect just the first fragment of a fragmented SCTP packet and unambiguously determine whether it corresponds to an INIT chunk (for further information, please refer to [RFC1858]). Accordingly, we stress the requirements, stated in Section 3.1, that (1) an INIT chunk MUST NOT be bundled with any other chunk in a packet, and (2) a packet containing an INIT chunk MUST have a zero Verification Tag. Furthermore, we require that the receiver of an INIT chunk MUST enforce these rules by silently discarding an arriving packet with an INIT chunk that is bundled with other chunks.

11.5. Protection of Non-SCTP Capable Hosts.

To provide a non-SCTP capable host with the same level of protection against attacks as for SCTP-capable ones, all SCTP stacks MUST implement the ICMP handling described in Appendix C.

When an SCTP stack receives a packet containing multiple control or DATA chunks and the processing of the packet requires the sending of multiple chunks in response, the sender of the response chunk(s) MUST NOT send more than one packet. If bundling is supported, multiple response chunks that fit into a single packet MAY be bundled together into one single response packet. If bundling is not supported, then the sender MUST NOT send more than one response chunk and MUST discard all other responses. Note that this rule does NOT apply to a SACK chunk, since a SACK chunk is, in itself, a response to DATA and a SACK does not require a response of more DATA.

An SCTP implementation SHOULD abort the association if it receives a SACK acknowledging a TSN that has not been sent.

An SCTP implementation that receives an INIT that would require a large packet in response, due to the inclusion of multiple ERROR parameters, MAY (at its discretion) elect to omit some or all of the ERROR parameters to reduce the size of the INIT-ACK. Due to a combination of the size of the COOKIE parameter and the number of addresses a receiver of an INIT may be indicating to a peer, it is

always possible that the INIT-ACK will be larger than the original INIT. An SCTP implementation SHOULD attempt to make the INIT-ACK as small as possible to reduce the possibility of byte amplification attacks.

12. Network Managment Considerations

A MIB for SCTP is defined in [RFC3873].

13. Recommended Transmission Control Block (TCB) Parameters

This section details a recommended set of parameters that should be contained within the TCB for an implementation. This section is for illustrative purposes and should not be deemed as requirements on an implementation or as an exhaustive list of all parameters inside an SCTP TCB. Each implementation may need its own additional parameters for optimization.

13.1. Parameters necessary for the SCTP instance

Associations: A list of current associations and mappings to the data consumers for each association. This may be in the form of a hash table or other implementation dependent structure. The data consumers may be process identification information such as file descriptors, named pipe pointer, or table pointers dependent on how SCTP is implemented.

Secret Key: A secret key used by this endpoint to compute the MAC. This SHOULD be a cryptographic quality random number with a sufficient length. Discussion in RFC4086 can be helpful in selection of the key.

Address List: The list of IP addresses that this instance has bound. This information is passed to one's peer(s) in INIT and INIT ACK chunks.

SCTP Port: The local SCTP port number the endpoint is bound to.

13.2. Parameters necessary per association (i.e. the TCB)

Peer Verification Tag : Tag value to be sent in every packet and is received in the INIT or INIT ACK chunk.

My : Tag expected in every inbound packet and sent in the

```
   Verification: INIT or INIT ACK chunk.
   Tag         :

   State       : A state variable indicating what state the association
               : is in, i.e. COOKIE-WAIT, COOKIE-ECHOED, ESTABLISHED,
               : SHUTDOWN-PENDING, SHUTDOWN-SENT, SHUTDOWN-RECEIVED,
               : SHUTDOWN-ACK-SENT.

                 Note: No "CLOSED" state is illustrated since if a
                 association is "CLOSED" its TCB SHOULD be removed.

   Peer        : A list of SCTP transport addresses that the peer is
   Transport   : bound to.  This information is derived from the INIT or
   Address     : INIT ACK and is used to associate an inbound packet
   List        : with a given association.  Normally this information is
               : hashed or keyed for quick lookup and access of the TCB.

   Primary     : This is the current primary destination transport
   Path        : address of the peer endpoint.  It may also specify a
               : source transport address on this endpoint.

   Overall     : The overall association error count.
   Error Count :

   Overall     : The threshold for this association that if the Overall
   Error       : Error Count reaches will cause this association to be
   Threshold   : torn down.

   Peer Rwnd   : Current calculated value of the peer's rwnd.

   Next TSN    : The next TSN number to be assigned to a new DATA chunk.
               : This is sent in the INIT or INIT ACK chunk to the peer
               : and incremented each time a DATA chunk is assigned a
               : TSN (normally just prior to transmit or during
               : fragmentation).

   Last Rcvd   : This is the last TSN received in sequence.  This value
   TSN         : is set initially by taking the peer's Initial TSN,
               : received in the INIT or INIT ACK chunk, and
               : subtracting one from it.

   Mapping     : An array of bits or bytes indicating which out of
   Array       : order TSN's have been received (relative to the
               : Last Rcvd TSN).  If no gaps exist, i.e. no out of order
               : packets have been received, this array will be set to
               : all zero.  This structure may be in the form of a
               : circular buffer or bit array.
```

```
Ack State   : This flag indicates if the next received packet
            : is to be responded to with a SACK.  This is initialized
            : to 0.  When a packet is received it is incremented.
            : If this value reaches 2 or more, a SACK is sent and the
            : value is reset to 0.  Note: This is used only when no
            : DATA chunks are received out of order.  When DATA chunks
            : are out of order, SACK's are not delayed (see Section
            : 6).

Inbound     : An array of structures to track the inbound streams.
Streams     : Normally including the next sequence number expected
            : and possibly the stream number.

Outbound    : An array of structures to track the outbound streams.
Streams     : Normally including the next sequence number to
            : be sent on the stream.

Reasm Queue : A re-assembly queue.

Local       : The list of local IP addresses bound in to this
Transport   : association.
Address     :
List        :

Association : The smallest PMTU discovered for all of the
PMTU        : peer's transport addresses.
```

13.3. Per Transport Address Data

For each destination transport address in the peer's address list derived from the INIT or INIT ACK chunk, a number of data elements needs to be maintained including:

```
   Error count : The current error count for this destination.

   Error       : Current error threshold for this destination i.e.
   Threshold   : what value marks the destination down if Error count
               : reaches this value.

   cwnd        : The current congestion window.

   ssthresh    : The current ssthresh value.

   RTO         : The current retransmission timeout value.

   SRTT        : The current smoothed round trip time.

   RTTVAR      : The current RTT variation.

   partial     : The tracking method for increase of cwnd when in
   bytes acked : congestion avoidance mode (see  Section 7.2.2)

   state       : The current state of this destination, i.e. DOWN, UP,
               : ALLOW-HB, NO-HEARTBEAT, etc.

   PMTU        : The current known path MTU.

   Per         : A timer used by each destination.
   Destination :
   Timer       :

   RTO-Pending : A flag used to track if one of the DATA chunks sent to
                 this address is currently being used to compute a
                 RTT.  If this flag is 0, the next DATA chunk sent to this
                 destination should be used to compute a RTT and this
                 flag should be set.  Every time the RTT calculation
                 completes (i.e. the DATA chunk is SACK'd) clear this
                 flag.

   last-time   : The time this destination was last sent to.  This can be
   used        : used to determine if a HEARTBEAT is needed.
```

13.4. General Parameters Needed

```
   Out Queue : A queue of outbound DATA chunks.

   In Queue : A queue of inbound DATA chunks.
```

14. IANA Considerations

This protocol will require port reservation like TCP for the use of "well known" servers within the Internet. All current TCP ports shall be automatically reserved in the SCTP port address space. New requests should follow IANA's current mechanisms for TCP.

This protocol may also 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 ERROR chunks

In the case where a particular ULP using SCTP desires to have its own ports, the ULP should be responsible for registering with IANA for getting its ports assigned.

14.1. IETF-defined Chunk Extension

The assignment of new chunk parameter type codes is done through an IETF Consensus action, as defined in [RFC2434]. Documentation of the chunk parameter MUST contain 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 3.2;

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.

The last chunk type (255) is reserved for future extension if necessary.

14.2. IETF-defined Chunk Parameter Extension

The assignment of new chunk parameter type codes is done through an IETF Consensus action as defined in [RFC2434]. Documentation of the chunk parameter 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 3.2.1.

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.

e) Each parameter type MUST be unique across all chunks.

14.3. IETF-defined Additional Error Causes

Additional cause codes may be allocated in the range 11 to 65535 through a Specification Required action as defined in [RFC2434]. Provided documentation must include the following information:

a) Name of the error condition.

b) Detailed description of the conditions under which an SCTP endpoint should issue an ERROR (or ABORT) with this cause code.

c) Expected action by the SCTP endpoint which receives an ERROR (or ABORT) 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 3.3.10, i.e.:

```
--   first two bytes contain the cause code value
--   last two bytes contain length of the Cause Parameter.
```

14.4. Payload Protocol Identifiers

Except for value 0 which is reserved by SCTP to indicate an unspecified payload protocol identifier in a DATA chunk, SCTP will not be responsible for standardizing or verifying any payload protocol identifiers; SCTP simply receives the identifier from the upper layer and carries it with the corresponding payload data.

The upper layer, i.e., the SCTP user, SHOULD standardize any specific protocol identifier with IANA if it is so desired. The use of any specific payload protocol identifier is out of the scope of SCTP.

15. Suggested SCTP Protocol Parameter Values

The following protocol parameters are RECOMMENDED:

```
RTO.Initial - 3 seconds
RTO.Min - 1 second
RTO.Max - 60 seconds
Max.Burst - 4
RTO.Alpha - 1/8
RTO.Beta - 1/4
Valid.Cookie.Life - 60 seconds
Association.Max.Retrans - 10 attempts
Path.Max.Retrans - 5 attempts (per destination address)
Max.Init.Retransmits - 8 attempts
HB.interval - 30 seconds
HB.Max.Burst - 1
```

IMPLEMENTATION NOTE: The SCTP implementation may allow ULP to customize some of these protocol parameters (see Section 10).

Note: RTO.Min SHOULD be set as recommended above.

16. Acknowledgements

An undertaking represented by this updated document is not a small feat and represents the summation of the initial authors of RFC2960,

Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer T. Taylor, I. Rytina, M. Kalla, L. Zhang, V. Paxson ,

add to that the comments from every one that contributed to the original RFC:

Mark Allman, R.J. Atkinson, Richard Band, Scott Bradner, Steve Bellovin, Peter Butler, Ram Dantu, R. Ezhirpavai, Mike Fisk, Sally Floyd, Atsushi Fukumoto, Matt Holdrege, Henry Houh, Christian Huitema, Gary Lehecka, Jonathan Lee, David Lehmann, John Loughney, Daniel Luan, Barry Nagelberg, Thomas Narten, Erik Nordmark, Lyndon Ong, Shyamal Prasad, Kelvin Porter, Heinz Prantner, Jarno Rajahalme, Raymond E. Reeves, Renee Revis, Ivan Arias Rodriguez, A. Sankar, Greg Sidebottom, Brian Wyld, La Monte Yarroll, and many others for their invaluable comments.

then add the authors of the SCTP implementors guide, I. Arias-Rodriguez, K. Poon, A. Caro, M. Tuexen,

Add to these the efforts of all the subsequent seven SCTP inter-

operability tests and those who commented on the RFC4460 as shown in its acknowledgments:

Barry Zuckerman, La Monte Yarroll, Qiaobing Xie, Wang Xiaopeng, Jonathan Wood, Jeff Waskow, Mike Turner, John Townsend, Sabina Torrente, Cliff Thomas, Yuji Suzuki, Manoj Solanki, Sverre Slotte, Keyur Shah, Jan Rovins, Ben Robinson, Renee Revis, Ian Periam, RC Monee, Sanjay Rao, Sujith Radhakrishnan, Heinz Prantner, Biren Patel, Nathalie Mouellic, Mitch Miers, Bernward Meyknecht, Stan McClellan, Oliver Mayor, Tomas Orti Martin, Sandeep Mahajan, David Lehmann, Jonathan Lee, Philippe Langlois, Karl Knutson, Joe Keller, Gareth Keily, Andreas Jungmaier, Janardhan Iyengar, Mutsuya Irie, John Hebert, Kausar Hassan, Fred Hasle, Dan Harrison, Jon Grim, Laurent Glaude, Steven Furniss, Atsushi Fukumoto, Ken Fujita, Steve Dimig, Thomas Curran, Serkan Cil, Melissa Campbell, Peter Butler, Rob Brennan, Harsh Bhondwe, Brian Bidulock, Caitlin Bestler, Jon Berger, Robby Benedyk, Stephen Baucke, Sandeep Balani, and Ronnie Sellar.

A special thanks to Mark Allman, who should actually be a co-author for his work on the max-burst, but managed to wiggle out due to a technicality. Also, we would like to acknowledge Lyndon Ong and Phil Conrad for their valuable input and many contributions.

And finally you have this document, and those that have commented upon that including, Alfred Hines and Ronnie Sellars.

My thanks cannot be adequately expressed for all of you who have participated in the coding, testing and updating process of this document all I can say is Thank You!

Randall Stewart - Editor

Appendix A. Explicit Congestion Notification

ECN (Ramakrishnan, K., Floyd, S., "Explicit Congestion Notification", [RFC3168], January 1999) describes a proposed extension to IP that details a method to become aware of congestion outside of datagram loss. This is an optional feature that an implementation MAY choose to add to SCTP. This appendix details the minor differences implementers will need to be aware of if they choose to implement this feature. In general [RFC3168] should be followed with the following exceptions.

Negotiation:

[RFC3168] details negotiation of ECN during the SYN and SYN-ACK

stages of a TCP connection. The sender of the SYN sets two bits in the TCP flags, and the sender of the SYN-ACK sets only 1 bit. The reasoning behind this is to assure both sides are truly ECN capable. For SCTP this is not necessary. To indicate that an endpoint is ECN capable an endpoint SHOULD add to the INIT and or INIT ACK chunk the TLV reserved for ECN. This TLV contains no parameters, and thus has the following format:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Parameter Type = 32768      |     Parameter Length = 4      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

ECN-Echo:

[RFC3168] details a specific bit for a receiver to send back in its TCP acknowledgements to notify the sender of the Congestion Experienced (CE) bit having arrived from the network. For SCTP this same indication is made by including the ECNE chunk. This chunk contains one data element, i.e. the lowest TSN associated with the IP datagram marked with the CE bit, and looks as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Chunk Type=12 | Flags=00000000|    Chunk Length = 8           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Lowest TSN Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Note: The ECNE is considered a Control chunk.

CWR:

[RFC3168] details a specific bit for a sender to send in the header of its next outbound TCP segment to indicate to its peer that it has reduced its congestion window. This is termed the CWR bit. For SCTP the same indication is made by including the CWR chunk. This chunk contains one data element, i.e. the TSN number that was sent in the ECNE chunk. This element represents the lowest TSN number in the datagram that was originally marked with the CE bit.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Chunk Type=13 | Flags=00000000|    Chunk Length = 8           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Lowest TSN Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Note: The CWR is considered a Control chunk.

Appendix B. CRC32c Checksum Calculation

We define a 'reflected value' as one that is the opposite of the normal bit order of the machine. The 32-bit CRC is calculated as described for CRC-32c and uses the polynomial code 0x11EDC6F41 (Castagnoli93) or x^32+x^28+x^27+x^26+x^25 +x^23+x^22+x^20+x^19+x^18+x^14+x^13+x^11+x^10+x^9+x^8+x^6+x^0. The CRC is computed using a procedure similar to ETHERNET CRC [ITU32], modified to reflect transport level usage.

CRC computation uses polynomial division. A message bit-string M is transformed to a polynomial, M(X), and the CRC is calculated from M(X) using polynomial arithmetic.

When CRCs are used at the link layer, the polynomial is derived from on-the-wire bit ordering: the first bit 'on the wire' is the high-order coefficient. Since SCTP is a transport-level protocol, it cannot know the actual serial-media bit ordering. Moreover, different links in the path between SCTP endpoints may use different link-level bit orders.

A convention must therefore be established for mapping SCTP transport messages to polynomials for purposes of CRC computation. The bit-ordering for mapping SCTP messages to polynomials is that bytes are taken most-significant first; but within each byte, bits are taken least-significant first. The first byte of the message provides the eight highest coefficients. Within each byte, the least-significant SCTP bit gives the most significant polynomial coefficient within that byte, and the most-significant SCTP bit is the least significant polynomial coefficient in that byte. (This bit ordering is sometimes called 'mirrored' or 'reflected' [WILLIAMS93]) CRC polynomials are to be transformed back into SCTP transport-level byte values, using a consistent mapping.

The SCTP transport-level CRC value should be calculated as follows:

- CRC input data are assigned to a byte stream, numbered from 0 to N-1.
- The transport-level byte-stream is mapped to a polynomial value. An N-byte PDU with j bytes numbered 0 to N-1 is considered as coefficients of a polynomial M(x) of order 8N-1, with bit 0 of byte j being coefficient x^(8(N-j)-8), and bit 7 of byte j being coefficient x^(8(N-j)-1).
- The CRC remainder register is initialized with all 1s and the CRC is computed with an algorithm that simultaneously multiplies by x^32 and divides by the CRC polynomial.
- The polynomial is multiplied by x^32 and divided by G(x), the generator polynomial, producing a remainder R(x) of degree less than or equal to 31.
- The coefficients of R(x) are considered a 32-bit sequence.
- The bit sequence is complemented. The result is the CRC polynomial.
- The CRC polynomial is mapped back into SCTP transport-level bytes. The coefficient of x^31 gives the value of bit 7 of SCTP byte 0, and the coefficient of x^24 gives the value of bit 0 of byte 0. The coefficient of x^7 gives bit 7 of byte 3, and the coefficient of x^0 gives bit 0 of byte 3. The resulting four-byte transport-level sequence is the 32-bit SCTP checksum value.

IMPLEMENTATION NOTE: Standards documents, textbooks, and vendor literature on CRCs often follow an alternative formulation, in which the register used to hold the remainder of the long-division algorithm is initialized to zero rather than all-1s, and instead the first 32 bits of the message are complemented. The long-division algorithm used in our formulation is specified such that the initial multiplication by 2^32 and the long-division are combined into one simultaneous operation. For such algorithms, and for messages longer than 64 bits, the two specifications are precisely equivalent. That equivalence is the intent of this document.

Implementors of SCTP are warned that both specifications are to be found in the literature, sometimes with no restriction on the long-division algorithm. The choice of formulation in this document is to permit non-SCTP usage, where the same CRC algorithm may be used to protect messages shorter than 64 bits.

There may be a computational advantage in validating the Association against the Verification Tag, prior to performing a checksum, as invalid tags will result in the same action as a bad checksum in most cases. The exceptions for this technique would be INIT and some SHUTDOWN-COMPLETE exchanges, as well as a stale COOKIE-ECHO. These special case exchanges must represent small packets and will minimize the effect of the checksum calculation.

Appendix C. ICMP Handling

Whenever an ICMP message is received by an SCTP endpoint the following procedures MUST be followed to ensure proper utilization of the information being provided by layer 3.

ICMP1) An implementation MAY ignore all ICMPv4 messages where the type field is not set to "Destination Unreachable".

ICMP2) An implementation MAY ignore all ICMPv6 messages where the type field is not "Destination Unreachable, "Parameter Problem" or "Packet Too Big".

ICMP3) An implementation MAY ignore any ICMPv4 messages where the code does not indicate "Protocol Unreachable" or "Fragmentation Needed".

ICMP4) An implementation MAY ignore all ICMPv6 messages of type "Parameter Problem" if the code is not "Unrecognized next header type encountered".

ICMP5) An implementation MUST use the payload of the ICMP message (V4 or V6) to locate the association that sent the message that ICMP is responding to. If the association cannot be found, an implementation SHOULD ignore the ICMP message.

ICMP6) An implementation MUST validate that the Verification Tag contained in the ICMP message matches the verification tag of the peer. If the Verification Tag is not 0 and does NOT match, discard the ICMP message. If it is 0 and the ICMP message contains enough bytes to verify that the chunk type is an INIT chunk and that the initiate tag matches the tag of the peer, continue with ICMP7. If the ICMP message is too short or the chunk type or the initiate tag does not match, silently discard the packet.

ICMP7) If the ICMP message is either a V6 "Packet Too Big" or a V4 "Fragmentation Needed", an implementation MAY process this information as defined for PATH MTU discovery.

ICMP8) If the ICMP code is a "Unrecognized next header type encountered" or a "Protocol Unreachable", an implementation MUST treat this message as an abort with the T bit set if it does not contain an INIT chunk. If it does contain an INIT chunk and the association is in COOKIE-WAIT state, handle the ICMP message like an ABORT.

ICMP9) If the ICMPv6 code is "Destination Unreachable", the implementation MAY mark the destination into the unreachable state or alternatively increment the path error counter.

Note that these procedures differ from [RFC1122] and from its requirements for processing of port-unreachable messages and the requirements that an implementation MUST abort associations in response to a "protocol unreachable" message. Port unreachable messages are not processed, since an implementation will send an ABORT, not a port unreachable. The stricter handling of the

"protocol unreachable" message is due to security concerns for hosts that do NOT support SCTP.

The following non-normative sample code is taken from an open-source CRC generator [WILLIAMS93], using the "mirroring" technique and yielding a lookup table for SCTP CRC32-c with 256 entries, each 32 bits wide. While neither especially slow nor especially fast, as software table-lookup CRCs go, it has the advantage of working on both big-endian and little-endian CPUs, using the same (host-order) lookup tables, and using only the pre-defined ntohl() and htonl() operations. The code is somewhat modified from [WILLIAMS93], to ensure portability between big-endian and little-endian architectures. (Note that if the byte endian-ness of the target architecture is known to be little-endian the final bit-reversal and byte-reversal steps can be folded into a single operation.)

```
/*************************************************************/
/* Note Definition for Ross Williams table generator would   */
/* be: TB_WIDTH=4, TB_POLLY=0x1EDC6F41, TB_REVER=TRUE        */
/* For Mr. Williams direct calculation code use the settings */
/* cm_width=32, cm_poly=0x1EDC6F41, cm_init=0xFFFFFFFF,      */
/* cm_refin=TRUE, cm_refot=TRUE, cm_xorort=0x00000000        */
/*************************************************************/

/* Example of the crc table file */
#ifndef __crc32cr_table_h__
#define __crc32cr_table_h__

#define CRC32C_POLY 0x1EDC6F41
#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])

unsigned long  crc_c[256] =
{
0x00000000L, 0xF26B8303L, 0xE13B70F7L, 0x1350F3F4L,
0xC79A971FL, 0x35F1141CL, 0x26A1E7E8L, 0xD4CA64EBL,
0x8AD958CFL, 0x78B2DBCCL, 0x6BE22838L, 0x9989AB3BL,
0x4D43CFD0L, 0xBF284CD3L, 0xAC78BF27L, 0x5E133C24L,
0x105EC76FL, 0xE235446CL, 0xF165B798L, 0x030E349BL,
0xD7C45070L, 0x25AFD373L, 0x36FF2087L, 0xC494A384L,
0x9A879FA0L, 0x68EC1CA3L, 0x7BBCEF57L, 0x89D76C54L,
0x5D1D08BFL, 0xAF768BBCL, 0xBC267848L, 0x4E4DFB4BL,
0x20BD8EDEL, 0xD2D60DDDL, 0xC186FE29L, 0x33ED7D2AL,
0xE72719C1L, 0x154C9AC2L, 0x061C6936L, 0xF477EA35L,
0xAA64D611L, 0x580F5512L, 0x4B5FA6E6L, 0xB93425E5L,
0x6DFE410EL, 0x9F95C20DL, 0x8CC531F9L, 0x7EAEB2FAL,
0x30E349B1L, 0xC288CAB2L, 0xD1D83946L, 0x23B3BA45L,
```

```
   0xF779DEAEL, 0x05125DADL, 0x1642AE59L, 0xE4292D5AL,
   0xBA3A117EL, 0x4851927DL, 0x5B016189L, 0xA96AE28AL,
   0x7DA08661L, 0x8FCB0562L, 0x9C9BF696L, 0x6EF07595L,
   0x417B1DBCL, 0xB3109EBFL, 0xA0406D4BL, 0x522BEE48L,
   0x86E18AA3L, 0x748A09A0L, 0x67DAFA54L, 0x95B17957L,
   0xCBA24573L, 0x39C9C670L, 0x2A993584L, 0xD8F2B687L,
   0x0C38D26CL, 0xFE53516FL, 0xED03A29BL, 0x1F682198L,
   0x5125DAD3L, 0xA34E59D0L, 0xB01EAA24L, 0x42752927L,
   0x96BF4DCCL, 0x64D4CECFL, 0x77843D3BL, 0x85EFBE38L,
   0xDBFC821CL, 0x2997011FL, 0x3AC7F2EBL, 0xC8AC71E8L,
   0x1C661503L, 0xEE0D9600L, 0xFD5D65F4L, 0x0F36E6F7L,
   0x61C69362L, 0x93AD1061L, 0x80FDE395L, 0x72966096L,
   0xA65C047DL, 0x5437877EL, 0x4767748AL, 0xB50CF789L,
   0xEB1FCBADL, 0x197448AEL, 0x0A24BB5AL, 0xF84F3859L,
   0x2C855CB2L, 0xDEEEDFB1L, 0xCDBE2C45L, 0x3FD5AF46L,
   0x7198540DL, 0x83F3D70EL, 0x90A324FAL, 0x62C8A7F9L,
   0xB602C312L, 0x44694011L, 0x5739B3E5L, 0xA55230E6L,
   0xFB410CC2L, 0x092A8FC1L, 0x1A7A7C35L, 0xE811FF36L,
   0x3CDB9BDDL, 0xCEB018DEL, 0xDDE0EB2AL, 0x2F8B6829L,
   0x82F63B78L, 0x709DB87BL, 0x63CD4B8FL, 0x91A6C88CL,
   0x456CAC67L, 0xB7072F64L, 0xA457DC90L, 0x563C5F93L,
   0x082F63B7L, 0xFA44E0B4L, 0xE9141340L, 0x1B7F9043L,
   0xCFB5F4A8L, 0x3DDE77ABL, 0x2E8E845FL, 0xDCE5075CL,
   0x92A8FC17L, 0x60C37F14L, 0x73938CE0L, 0x81F80FE3L,
   0x55326B08L, 0xA759E80BL, 0xB4091BFFL, 0x466298FCL,
   0x1871A4D8L, 0xEA1A27DBL, 0xF94AD42FL, 0x0B21572CL,
   0xDFEB33C7L, 0x2D80B0C4L, 0x3ED04330L, 0xCCBBC033L,
   0xA24BB5A6L, 0x502036A5L, 0x4370C551L, 0xB11B4652L,
   0x65D122B9L, 0x97BAA1BAL, 0x84EA524EL, 0x7681D14DL,
   0x2892ED69L, 0xDAF96E6AL, 0xC9A99D9EL, 0x3BC21E9DL,
   0xEF087A76L, 0x1D63F975L, 0x0E330A81L, 0xFC588982L,
   0xB21572C9L, 0x407EF1CAL, 0x532E023EL, 0xA145813DL,
   0x758FE5D6L, 0x87E466D5L, 0x94B49521L, 0x66DF1622L,
   0x38CC2A06L, 0xCAA7A905L, 0xD9F75AF1L, 0x2B9CD9F2L,
   0xFF56BD19L, 0x0D3D3E1AL, 0x1E6DCDEEL, 0xEC064EEDL,
   0xC38D26C4L, 0x31E6A5C7L, 0x22B65633L, 0xD0DDD530L,
   0x0417B1DBL, 0xF67C32D8L, 0xE52CC12CL, 0x1747422FL,
   0x49547E0BL, 0xBB3FFD08L, 0xA86F0EFCL, 0x5A048DFFL,
   0x8ECEE914L, 0x7CA56A17L, 0x6FF599E3L, 0x9D9E1AE0L,
   0xD3D3E1ABL, 0x21B862A8L, 0x32E8915CL, 0xC083125FL,
   0x144976B4L, 0xE622F5B7L, 0xF5720643L, 0x07198540L,
   0x590AB964L, 0xAB613A67L, 0xB831C993L, 0x4A5A4A90L,
   0x9E902E7BL, 0x6CFBAD78L, 0x7FAB5E8CL, 0x8DC0DD8FL,
   0xE330A81AL, 0x115B2B19L, 0x020BD8EDL, 0xF0605BEEL,
   0x24AA3F05L, 0xD6C1BC06L, 0xC5914FF2L, 0x37FACCF1L,
   0x69E9F0D5L, 0x9B8273D6L, 0x88D28022L, 0x7AB90321L,
   0xAE7367CAL, 0x5C18E4C9L, 0x4F48173DL, 0xBD23943EL,
   0xF36E6F75L, 0x0105EC76L, 0x12551F82L, 0xE03E9C81L,
```

```
0x34F4F86AL, 0xC69F7B69L, 0xD5CF889DL, 0x27A40B9EL,
0x79B737BAL, 0x8BDCB4B9L, 0x988C474DL, 0x6AE7C44EL,
0xBE2DA0A5L, 0x4C4623A6L, 0x5F16D052L, 0xAD7D5351L,
};

#endif

 /* Example of table build routine */



#include <stdio.h>
#include <stdlib.h>

#define OUTPUT_FILE   "crc32cr.h"
#define CRC32C_POLY    0x1EDC6F41L
FILE *tf;
unsigned long
reflect_32 (unsigned long b)
{
  int i;
  unsigned long rw = 0L;

  for (i = 0; i < 32; i++){
      if (b & 1)
        rw |= 1 << (31 - i);
      b >>= 1;
  }
  return (rw);
}

unsigned long
build_crc_table (int index)
{
  int i;
  unsigned long rb;

  rb = reflect_32 (index);

  for (i = 0; i < 8; i++){
      if (rb & 0x80000000L)
       rb = (rb << 1) ^ CRC32C_POLY;
      else
       rb <<= 1;
  }
  return (reflect_32 (rb));
}
```

```
main ()
{
  int i;

  printf ("\nGenerating CRC-32c table file <%s>\n",
  OUTPUT_FILE);
  if ((tf = fopen (OUTPUT_FILE, "w")) == NULL){
      printf ("Unable to open %s\n", OUTPUT_FILE);
      exit (1);
  }
  fprintf (tf, "#ifndef __crc32cr_table_h__\n");
  fprintf (tf, "#define __crc32cr_table_h__\n\n");
  fprintf (tf, "#define CRC32C_POLY 0x%08lX\n",
  CRC32C_POLY);
  fprintf (tf,
  "#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])\n");
  fprintf (tf, "\nunsigned long  crc_c[256] =\n{\n");
  for (i = 0; i < 256; i++){
      fprintf (tf, "0x%08lXL, ", build_crc_table (i));
      if ((i & 3) == 3)
        fprintf (tf, "\n");
  }
  fprintf (tf, "};\n\n#endif\n");

  if (fclose (tf) != 0)
    printf ("Unable to close <%s>." OUTPUT_FILE);
  else
    printf ("\nThe CRC-32c table has been written to <%s>.\n",
      OUTPUT_FILE);
}



/* Example of crc insertion */

#include "crc32cr.h"

unsigned long
generate_crc32c(unsigned char *buffer, unsigned int length)
{
  unsigned int i;
  unsigned long crc32 = ~0L;
  unsigned long result;
  unsigned char byte0,byte1,byte2,byte3;

  for (i = 0; i < length; i++){
      CRC32C(crc32, buffer[i]);
  }
```

```
    result = ~crc32;

    /*  result  now holds the negated polynomial remainder;
     *  since the table and algorithm is "reflected" [williams95].
     *  That is,  result has the same value as if we mapped the message
     *  to a polynomial, computed the host-bit-order polynomial
     *  remainder, performed final negation, then did an end-for-end
     *  bit-reversal.
     *  Note that a 32-bit bit-reversal is identical to four inplace
     *  8-bit reversals followed by an end-for-end byteswap.
     *  In other words, the bytes of each bit are in the right order,
     *  but the bytes have been byteswapped.  So we now do an explicit
     *  byteswap.  On a little-endian machine, this byteswap and
     *  the final ntohl cancel out and could be elided.
     */

    byte0 = result & 0xff;
    byte1 = (result>>8) & 0xff;
    byte2 = (result>>16) & 0xff;
    byte3 = (result>>24) & 0xff;
    crc32 = ((byte0 << 24) |
             (byte1 << 16) |
             (byte2 << 8)  |
             byte3);
    return ( crc32 );
  }

  int
  insert_crc32(unsigned char *buffer, unsigned int length)
  {
    SCTP_message *message;
    unsigned long crc32;
    message = (SCTP_message *) buffer;
    message->common_header.checksum = 0L;
    crc32 = generate_crc32c(buffer,length);
    /* and insert it into the message */
    message->common_header.checksum = htonl(crc32);
    return 1;
  }

  int
  validate_crc32(unsigned char *buffer, unsigned int length)
  {
    SCTP_message *message;
    unsigned int i;
    unsigned long original_crc32;
    unsigned long crc32 = ~0L;
```

```
    /* save and zero checksum */
    message = (SCTP_message *) buffer;
    original_crc32 = ntohl(message->common_header.checksum);
    message->common_header.checksum = 0L;
    crc32 = generate_crc32c(buffer,length);
    return ((original_crc32 == crc32)? 1 : -1);
  }
```

17. References

17.1. Normative references

[ITU32] "ITU-T Recommendation V.42, "Error-correcting procedures for DCEs using asynchronous-to-synchronous conversion".", ITU-T section 8.1.1.6.2.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC0813] Clark, D., "Window and Acknowledgement Strategy in TCP", RFC 813, July 1982.

[RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[RFC1123] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.

[RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, November 1990.

[RFC1858] Ziemba, G., Reed, D., and P. Traina, "Security Considerations for IP Fragment Filtering", RFC 1858, October 1995.

[RFC1981] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, August 1996.

[RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, August 1996.

[RFC2026] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2373] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 2373, July 1998.

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[RFC2581] Allman, M., Paxson, V., and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

[RFC3873] Pastor, J. and M. Belinchon, "Stream Control Transmission Protocol (SCTP) Management Information Base (MIB)", RFC 3873, September 2004.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

[RFC4302] Kent, S., "IP Authentication Header", RFC 4302, December 2005.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, December 2005.

[RFC4306] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005.

17.2. Informative References

[FALL96] Fall, K. and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", SIGCOMM'99 V. 26 N. 3 pp 5-21, July 1996.

[SAVAGE99]
Savage, S., Cardwell, N., Wetherall, D., and T. Anderson, "TCP Congestion Control with a Misbehaving Receiver", ACM Computer Commuinications Review 29(5), October 1999.

[ALLMAN99]
Allman, M. and V. Paxson, "On Estimating End-to-End Network Path Properties", SIGCOMM'99 , 1999.

[WILLIAMS93]

Williams, R., "A PAINLESS GUIDE TO CRC ERROR DETECTION ALGORITHMS" - Internet publication http://www.geocities.com/SiliconValley/Pines/8659/crc.htm.", August 1993.

[RFC1950] Deutsch, L. and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, May 1996.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[RFC2196] Fraser, B., "Site Security Handbook", RFC 2196, September 1997.

[RFC2522] Karn, P. and W. Simpson, "Photuris: Session-Key Management Protocol", RFC 2522, March 1999.

[RFC2960] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, October 2000.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC4460] Stewart, R., Arias-Rodriguez, I., Poon, K., Caro, A., and M. Tuexen, "Stream Control Transmission Protocol (SCTP) Specification Errata and Issues", RFC 4460, April 2006.

Author's Address

Randall R. Stewart
Editor
4875 Forest Drive
Suite 200
Columbia, SC 29206
US

Email: rrs@cisco.com

Full Copyright Statement

Copyright (C) The IETF Trust (2006).

This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).