

Network Working Group
Internet-Draft
Intended status: Informational
Expires: November 22, 2018

R. Stewart
Netflix, Inc.
M. Tuexen
Muenster Univ. of Appl. Sciences
M. Proshin
Ericsson
May 21, 2018

RFC 4960 Errata and Issues
draft-ietf-tsvwg-rfc4960-errata-06.txt

Abstract

This document is a compilation of issues found since the publication of [RFC4960](#) in September 2007 based on experience with implementing, testing, and using SCTP along with the suggested fixes. This document provides deltas to [RFC4960](#) and is organized in a time ordered way. The issues are listed in the order they were brought up. Because some text is changed several times the last delta in the text is the one which should be applied. In addition to the delta a description of the problem and the details of the solution are also provided.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

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."

This Internet-Draft will expire on November 22, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents

(<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	3
2.	Conventions	4
3.	Corrections to RFC 4960	4
3.1.	Path Error Counter Threshold Handling	4
3.2.	Upper Layer Protocol Shutdown Request Handling	5
3.3.	Registration of New Chunk Types	6
3.4.	Variable Parameters for INIT Chunks	7
3.5.	CRC32c Sample Code on 64-bit Platforms	8
3.6.	Endpoint Failure Detection	9
3.7.	Data Transmission Rules	10
3.8.	T1-Cookie Timer	11
3.9.	Miscellaneous Typos	12
3.10.	CRC32c Sample Code	19
3.11.	partial_bytes_acked after T3-rtx Expiration	20
3.12.	Order of Adjustments of partial_bytes_acked and cwnd	21
3.13.	HEARTBEAT ACK and the association error counter	21
3.14.	Path for Fast Retransmission	23
3.15.	Transmittal in Fast Recovery	24
3.16.	Initial Value of ssthresh	24
3.17.	Automatically Confirmed Addresses	25
3.18.	Only One Packet after Retransmission Timeout	26
3.19.	INIT ACK Path for INIT in COOKIE-WAIT State	27
3.20.	Zero Window Probing and Unreachable Primary Path	28
3.21.	Normative Language in Section 10	29
3.22.	Increase of partial_bytes_acked in Congestion Avoidance	33
3.23.	Inconsistency in Notifications Handling	34
3.24.	SACK.Delay Not Listed as a Protocol Parameter	38
3.25.	Processing of Chunks in an Incoming SCTP Packet	40
3.26.	CWND Increase in Congestion Avoidance Phase	41
3.27.	Refresh of cwnd and ssthresh after Idle Period	43
3.28.	Window Updates After Receiver Window Opens Up	44
3.29.	Path of DATA and Reply Chunks	45
3.30.	Outstanding Data, Flightsize and Data In Flight Key Terms	47
3.31.	CWND Degradation due to Max.Burst	48
3.32.	Reduction of RTO.Initial	49
3.33.	Ordering of Bundled SACK and ERROR Chunks	51
3.34.	Undefined Parameter Returned by RECEIVE Primitive	51
3.35.	DSCP Changes	52

3.36 . Inconsistent Handling of ICMPv4 and ICMPv6 Messages . . .	54
3.37 . Handling of Soft Errors	55
3.38 . Honoring CWND	56
3.39 . Zero Window Probing	57
3.40 . Updating References Regarding ECN	59
3.41 . Host Name Address Parameter Deprecated	60
3.42 . Conflicting Text Regarding the Supported Address Types Parameter	64
3.43 . Integration of RFC 6096	65
3.44 . Integration of RFC 6335	67
3.45 . Integration of RFC 7053	69
3.46 . CRC32c Code Improvements	72
3.47 . Clarification of Gap Ack Blocks in SACK Chunks	82
3.48 . Handling of SSN Wrap Arounds	84
3.49 . Update RFC 2119 Boilerplate	85
3.50 . Missed Text Removal	85
4 . IANA Considerations	86
5 . Security Considerations	86
6 . Acknowledgments	86
7 . References	86
7.1 . Normative References	87
7.2 . Informative References	87
Authors' Addresses	88

[1](#). Introduction

This document contains a compilation of all defects found up until the publication of this document for [\[RFC4960\]](#) specifying the Stream Control Transmission Protocol (SCTP). These defects may be of an editorial or technical nature. This document may be thought of as a companion document to be used in the implementation of SCTP to clarify errors in the original SCTP document.

This document provides a history of the changes that will be compiled into a BIS document for [\[RFC4960\]](#). It is structured similar to [\[RFC4460\]](#).

Each error will be detailed within this document in the form of:

- o The problem description,
- o The text quoted from [\[RFC4960\]](#),
- o The replacement text that should be placed into an upcoming BIS document,
- o A description of the solution.

Note that when reading this document one must use care to assure that a field or item is not updated further on within the document. Since this document is a historical record of the sequential changes that

have been found necessary at various inter-op events and through discussion on the list, the last delta in the text is the one which should be applied.

2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [BCP 14](#) [[RFC2119](#)] [[RFC8174](#)] when, and only when, they appear in all capitals, as shown here.

3. Corrections to [RFC 4960](#)

[NOTE to RFC-Editor:

References to obsoleted RFCs are in OLD TEXT sections and have the corresponding references to the obsoleting RFCs in the NEW TEXT sections. In addition to this, there are some references to the obsoleted [[RFC2960](#)], which are intended.

]

3.1. Path Error Counter Threshold Handling

3.1.1. Description of the Problem

The handling of the 'Path.Max.Retrans' parameter is described in [Section 8.2](#) and [Section 8.3 of \[RFC4960\]](#) in an inconsistent way. Whereas [Section 8.2](#) describes that a path is marked inactive when the path error counter exceeds the threshold, [Section 8.3](#) says the path is marked inactive when the path error counter reaches the threshold.

This issue was reported as an Errata for [[RFC4960](#)] with Errata ID 1440.

3.1.2. Text Changes to the Document

Old text: ([Section 8.3](#))

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.

New text: ([Section 8.3](#))

When the value of this counter exceeds 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.

[3.1.3.](#) Solution Description

The intended state change should happen when the threshold is exceeded.

[3.2.](#) Upper Layer Protocol Shutdown Request Handling

[3.2.1.](#) Description of the Problem

[Section 9.2 of \[RFC4960\]](#) describes the handling of received SHUTDOWN chunks in the SHUTDOWN-RECEIVED state instead of the handling of shutdown requests from its upper layer in this state.

This issue was reported as an Errata for [\[RFC4960\]](#) with Errata ID 1574.

[3.2.2.](#) Text Changes to the Document

Old text: ([Section 9.2](#))

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

New text: ([Section 9.2](#))

Once an endpoint has reached the SHUTDOWN-RECEIVED state, it MUST ignore ULP shutdown requests, but MUST continue responding to SHUTDOWN chunks from its peer.

[3.2.3.](#) Solution Description

The text never intended the SCTP endpoint to ignore SHUTDOWN chunks from its peer. If it did, the endpoints could never gracefully terminate associations in some cases.

[3.3.](#) Registration of New Chunk Types

[3.3.1.](#) Description of the Problem

[Section 14.1 of \[RFC4960\]](#) should deal with new chunk types, however, the text refers to parameter types.

This issue was reported as an Errata for [\[RFC4960\]](#) with Errata ID 2592.

[3.3.2.](#) Text Changes to the Document

Old text: ([Section 14.1](#))

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:

New text: ([Section 14.1](#))

The assignment of new chunk type codes is done through an IETF Consensus action, as defined in [[RFC8126](#)]. Documentation of the chunk type MUST contain the following information:

[3.3.3](#). Solution Description

Refer to chunk types as intended and change reference to [[RFC8126](#)].

[3.4](#). Variable Parameters for INIT Chunks

[3.4.1](#). Description of the Problem

Newlines in wrong places break the layout of the table of variable parameters for the INIT chunk in [Section 3.3.2 of \[\[RFC4960\]\(#\)\]](#).

This issue was reported as an Errata for [[RFC4960](#)] with Errata ID 3291 and Errata ID 3804.

[3.4.2](#). Text Changes to the Document

 Old text: ([Section 3.3.2](#))

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

 New text: ([Section 3.3.2](#))

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	

[3.4.3.](#) Solution Description

Fix the formatting of the table.

[3.5.](#) CRC32c Sample Code on 64-bit Platforms

[3.5.1.](#) Description of the Problem

The sample code for computing the CRC32c provided in [[RFC4960](#)] assumes that a variable of type unsigned long uses 32 bits. This is not true on some 64-bit platforms (for example the ones using LP64).

This issue was reported as an Errata for [[RFC4960](#)] with Errata ID 3423.

[3.5.2.](#) Text Changes to the Document

Old text: (Appendix C)

```
unsigned long
generate_crc32c(unsigned char *buffer, unsigned int length)
{
    unsigned int i;
    unsigned long crc32 = ~0L;
```

New text: (Appendix C)

```
unsigned long
generate_crc32c(unsigned char *buffer, unsigned int length)
{
    unsigned int i;
    unsigned long crc32 = 0xffffffffL;
```

[3.5.3.](#) Solution Description

Use 0xffffffffL instead of ~0L which gives the same value on platforms using 32 bits or 64 bits for variables of type unsigned long.

[3.6.](#) Endpoint Failure Detection

[3.6.1.](#) Description of the Problem

The handling of the association error counter defined in [Section 8.1 of \[RFC4960\]](#) can result in an association failure even if the path used for data transmission is available, but idle.

This issue was reported as an Errata for [\[RFC4960\]](#) with Errata ID 3788.

[3.6.2.](#) Text Changes to the Document

Old text: ([Section 8.1](#))

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.

New text: ([Section 8.1](#))

An endpoint SHOULD keep a counter on the total number of consecutive retransmissions to its peer (this includes data retransmissions to all the destination transport addresses of the peer if it is multi-homed), including the number of unacknowledged HEARTBEAT chunks observed on the path which is currently used for data transfer. Unacknowledged HEARTBEAT chunks observed on paths different from the path currently used for data transfer SHOULD NOT increment the association error counter, as this could lead to association closure even if the path which is currently used for data transfer is available (but idle).

[3.6.3.](#) Solution Description

A more refined handling for the association error counter is defined.

[3.7.](#) Data Transmission Rules

[3.7.1.](#) Description of the Problem

When integrating the changes to [Section 6.1](#) A) of [[RFC2960](#)] as described in [Section 2.15.2 of \[RFC4460\]](#) some text was duplicated and became the final paragraph of [Section 6.1](#) A) of [[RFC4960](#)].

This issue was reported as an Errata for [[RFC4960](#)] with Errata ID 4071.

[3.7.2.](#) Text Changes to the Document

Old text: ([Section 6.1](#) A))

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 \[RFC1122\]](#).

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.

New text: ([Section 6.1](#) A))

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

[3.7.3.](#) Solution Description

Last paragraph of [Section 6.1](#) A) removed as intended in [Section 2.15.2 of \[RFC4460\]](#).

[3.8.](#) T1-Cookie Timer

[3.8.1.](#) Description of the Problem

Figure 4 of [[RFC4960](#)] illustrates the SCTP association setup. However, it incorrectly shows that the T1-init timer is used in the COOKIE-ECHOED state whereas the T1-cookie timer should have been used instead.

This issue was reported as an Errata for [[RFC4960](#)] with Errata ID 4400.

[3.8.2.](#) Text Changes to the Document

Old text: ([Section 5.1.6](#), Figure 4)

```
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)
```

New text: ([Section 5.1.6](#), Figure 4)

```
COOKIE ECHO [Cookie_Z] -----\
(Start T1-cookie timer)         \
(Enter COOKIE-ECHOED state)     \---> (build TCB enter ESTABLISHED
                                     state)
                                     /---- COOKIE-ACK
                                     /
(Cancel T1-cookie timer, <---/
Enter ESTABLISHED state)
```

[3.8.3.](#) Solution Description

Change the figure such that the T1-cookie timer is used instead of the T1-init timer.

[3.9.](#) Miscellaneous Typos

[3.9.1.](#) Description of the Problem

While processing [[RFC4960](#)] some typos were not caught.

[3.9.2.](#) Text Changes to the Document

Old text: ([Section 1.6](#))

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.

New text: ([Section 1.6](#))

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.

Old text: ([Section 3.3.10.9](#))

No User Data: This error cause is returned to the originator of a DATA chunk if a received DATA chunk has no user data.

New text: ([Section 3.3.10.9](#))

No User Data: This error cause is returned to the originator of a DATA chunk if a received DATA chunk has no user data.

Old text: ([Section 6.7](#), Figure 9)

```

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,
                                   /           Start=2,End=2]
                                   <-----/ (remove 6 from out-queue,
                                   and mark 7 as "1" missing report)

```

New text: ([Section 6.7](#), Figure 9)

```

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,
                                   /           Start=2,End=2]
                                   <-----/
(remove 6 from out-queue,
 and mark 7 as "1" missing report)

```

Old text: ([Section 6.10](#))

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 than or equal to the current Path MTU.

New text: ([Section 6.10](#))

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 than or equal to the current PMTU.

Old text: ([Section 10.1](#))

o Receive Unacknowledged Message

Format: RECEIVE_UNACKED(data retrieval id, buffer address, buffer size, [,stream id] [, stream sequence number] [,partial flag] [,payload protocol-id])

New text: ([Section 10.1](#))

0) Receive Unacknowledged Message

Format: RECEIVE_UNACKED(data retrieval id, buffer address, buffer size [,stream id] [,stream sequence number] [,partial flag] [,payload protocol-id])

Old text: ([Section 10.1](#))

M) Set Protocol Parameters

Format: SETPROTOCOLPARAMETERS(association id,
[,destination transport address,]
protocol parameter list)

New text: ([Section 10.1](#))

M) Set Protocol Parameters

Format: SETPROTOCOLPARAMETERS(association id,
[destination transport address,]
protocol parameter list)

Old text: (Appendix C)

ICMP2) An implementation MAY ignore all ICMPv6 messages where the
type field is not "Destination Unreachable", "Parameter
Problem", or "Packet Too Big".

New text: (Appendix C)

ICMP2) An implementation MAY ignore all ICMPv6 messages where the
type field is not "Destination Unreachable", "Parameter
Problem", or "Packet Too Big".

Old text: (Appendix C)

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.

New text: (Appendix C)

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 PMTU discovery.

Old text: ([Section 5.4](#))

2) For the receiver of the COOKIE ECHO, the only CONFIRMED address is the one to which the INIT-ACK was sent.

New text: ([Section 5.4](#))

2) For the receiver of the COOKIE ECHO, the only CONFIRMED address is the one to which the INIT ACK was sent.

Old text: ([Section 5.1.6](#), Figure 4)

```
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)
```

New text: ([Section 5.1.6](#), Figure 4)

```
COOKIE ECHO [Cookie_Z] -----\  
(Start T1-cookie timer)        \  
(Enter COOKIE-ECHOED state)    \---> (build TCB enter ESTABLISHED  
                                     state)  
                                /---- COOKIE ACK  
                                /  
(Cancel T1-cookie timer, <---/  
  Enter ESTABLISHED state)
```

Old text: ([Section 5.2.5](#))

5.2.5. Handle Duplicate COOKIE-ACK.

New text: ([Section 5.2.5](#))

5.2.5. Handle Duplicate COOKIE ACK.

Old text: ([Section 8.3](#))

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).

New text: ([Section 8.3](#))

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).

[3.9.3.](#) Solution Description

Typos fixed.

[3.10.](#) CRC32c Sample Code

[3.10.1.](#) Description of the Problem

The CRC32c computation is described in [Appendix B of \[RFC4960\]](#). However, the corresponding sample code and its explanation appears at the end of [Appendix C](#), which deals with ICMP handling.

[3.10.2.](#) Text Changes to the Document

Move all of [Appendix C](#) starting with the following sentence to the end of [Appendix B](#).

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 CRC32c with 256 entries, each 32 bits wide.

[3.10.3.](#) Solution Description

Text moved to the appropriate location.

[3.11.](#) partial_bytes_acked after T3-rtx Expiration

[3.11.1.](#) Description of the Problem

[Section 7.2.3 of \[RFC4960\]](#) explicitly states that partial_bytes_acked should be reset to 0 after packet loss detection from SACK but the same is missed for T3-rtx timer expiration.

[3.11.2.](#) Text Changes to the Document

Old text: ([Section 7.2.3](#))

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

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

New text: ([Section 7.2.3](#))

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

```
ssthresh = max(cwnd/2, 4*MTU)
cwnd = 1*MTU
partial_bytes_acked = 0
```

[3.11.3.](#) Solution Description

Specify that partial_bytes_acked should be reset to 0 after T3-rtx timer expiration.

[3.12.](#) Order of Adjustments of `partial_bytes_acked` and `cwnd`

[3.12.1.](#) Description of the Problem

[Section 7.2.2 of \[RFC4960\]](#) likely implies the wrong order of adjustments applied to `partial_bytes_acked` and `cwnd` in the congestion avoidance phase.

[3.12.2.](#) Text Changes to the Document

Old text: ([Section 7.2.2](#))

- 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`).

New text: ([Section 7.2.2](#))

- 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`), `partial_bytes_acked` is reset to (`partial_bytes_acked` - `cwnd`). Next, `cwnd` is increased by `1*MTU`.

[3.12.3.](#) Solution Description

The new text defines the exact order of adjustments of `partial_bytes_acked` and `cwnd` in the congestion avoidance phase.

[3.13.](#) HEARTBEAT ACK and the association error counter

[3.13.1.](#) Description of the Problem

[Section 8.1](#) and [Section 8.3 of \[RFC4960\]](#) prescribe that the receiver of a HEARTBEAT ACK must reset the association overall error counter. In some circumstances, e.g. when a router discards DATA chunks but not HEARTBEAT chunks due to the larger size of the DATA chunk, it might be better to not clear the association error counter on reception of the HEARTBEAT ACK and reset it only on reception of the SACK to avoid stalling the association.

[3.13.2](#). Text Changes to the Document

Old text: ([Section 8.1](#))

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.

New text: ([Section 8.1](#))

The counter MUST be reset each time a DATA chunk sent to that peer endpoint is acknowledged (by the reception of a SACK). When a HEARTBEAT ACK is received from the peer endpoint, the counter SHOULD also be reset. The receiver of the HEARTBEAT ACK MAY choose not to clear the counter if there is outstanding data on the association. This allows for handling the possible difference in reachability based on DATA chunks and HEARTBEAT chunks.

Old text: ([Section 8.3](#))

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](#)).

New text: ([Section 8.3](#))

Upon the receipt of the HEARTBEAT ACK, the sender of the HEARTBEAT MUST 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 SHOULD also clear the association overall error counter (as defined in [Section 8.1](#)).

[3.13.3.](#) Solution Description

The new text provides a possibility to not reset the association overall error counter when a HEARTBEAT ACK is received if there are valid reasons for it.

[3.14.](#) Path for Fast Retransmission

[3.14.1.](#) Description of the Problem

[RFC4960] clearly describes where to retransmit data that is timed out when the peer is multi-homed but the same is not stated for fast retransmissions.

[3.14.2.](#) Text Changes to the Document

Old text: ([Section 6.4](#))

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.

New text: ([Section 6.4](#))

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.

When its peer is multi-homed, an endpoint SHOULD send fast retransmissions to the same destination transport address where the original data was sent to. If the primary path has been changed and the original data was sent there before the fast retransmit, the implementation MAY send it to the new primary path.

[3.14.3.](#) Solution Description

The new text clarifies where to send fast retransmissions.

[3.15.](#) Transmittal in Fast Recovery

[3.15.1.](#) Description of the Problem

The Fast Retransmit on Gap Reports algorithm intends that only the very first packet may be sent regardless of cwnd in the Fast Recovery phase but rule 3) of [\[RFC4960\], Section 7.2.4](#), misses this clarification.

[3.15.2.](#) Text Changes to the Document

Old text: ([Section 7.2.4](#))

- 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.

New text: ([Section 7.2.4](#))

- 3) If not in Fast Recovery, 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 PMTU 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.

[3.15.3.](#) Solution Description

The new text explicitly specifies to send only the first packet in the Fast Recovery phase disregarding cwnd limitations.

[3.16.](#) Initial Value of ssthresh

[3.16.1.](#) Description of the Problem

The initial value of ssthresh should be set arbitrarily high. Using the advertised receiver window of the peer is inappropriate if the peer increases its window after the handshake. Furthermore, use a higher requirements level, since not following the advice may result in performance problems.

[3.16.2.](#) Text Changes to the Document

Old text: ([Section 7.2.1](#))

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

New text: ([Section 7.2.1](#))

- o The initial value of ssthresh SHOULD be arbitrarily high (e.g., the size of the largest possible advertised window).

[3.16.3.](#) Solution Description

Use the same value as suggested in [[RFC5681](#)], [Section 3.1](#), as an appropriate initial value. Furthermore, use the same requirements level.

[3.17.](#) Automatically Confirmed Addresses

[3.17.1.](#) Description of the Problem

The Path Verification procedure of [[RFC4960](#)] prescribes that any address passed to the sender of the INIT by its upper layer is automatically CONFIRMED. This, however, is unclear if only addresses in the request to initiate association establishment are considered or any addresses provided by the upper layer in any requests (e.g. in 'Set Primary').

[3.17.2.](#) Text Changes to the Document

Old text: ([Section 5.4](#))

- 1) Any address passed to the sender of the INIT by its upper layer is automatically considered to be CONFIRMED.

New text: ([Section 5.4](#))

- 1) Any addresses passed to the sender of the INIT by its upper layer in the request to initialize an association are automatically considered to be CONFIRMED.

[3.17.3.](#) Solution Description

The new text clarifies that only addresses provided by the upper layer in the request to initialize an association are automatically confirmed.

[3.18.](#) Only One Packet after Retransmission Timeout

[3.18.1.](#) Description of the Problem

[RFC4960] is not completely clear when it describes data transmission after T3-rtx timer expiration. [Section 7.2.1](#) does not specify how many packets are allowed to be sent after T3-rtx timer expiration if more than one packet fit into cwnd. At the same time, [Section 7.2.3](#) has the text without normative language saying that SCTP should ensure that no more than one packet will be in flight after T3-rtx timer expiration until successful acknowledgment. It makes the text inconsistent.

[3.18.2.](#) Text Changes to the Document

Old text: ([Section 7.2.1](#))

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

New text: ([Section 7.2.1](#))

- o The initial cwnd after a retransmission timeout MUST be no more than 1*MTU and only one packet is allowed to be in flight until successful acknowledgement.

[3.18.3.](#) Solution Description

The new text clearly specifies that only one packet is allowed to be sent after T3-rtx timer expiration until successful acknowledgement.

[3.19.](#) INIT ACK Path for INIT in COOKIE-WAIT State

[3.19.1.](#) Description of the Problem

In case of an INIT received in the COOKIE-WAIT state [[RFC4960](#)] prescribes to send an INIT ACK to the same destination address to which the original INIT has been sent. This text does not address the possibility of the upper layer to provide multiple remote IP addresses while requesting the association establishment. If the upper layer has provided multiple IP addresses and only a subset of these addresses are supported by the peer then the destination address of the original INIT may be absent in the incoming INIT and sending INIT ACK to that address is useless.

[3.19.2.](#) Text Changes to the Document

Old text: ([Section 5.2.1](#))

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 Initiate 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.

New text: ([Section 5.2.1](#))

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 Initiate Tag, unchanged). When responding, the following rules MUST be applied:

- 1) The INIT ACK MUST only be sent to an address passed by the upper layer in the request to initialize the association.
- 2) The INIT ACK MUST only be sent to an address reported in the incoming INIT.
- 3) The INIT ACK SHOULD be sent to the source address of the received INIT.

[3.19.3.](#) Solution Description

The new text requires sending INIT ACK to a destination address that is passed by the upper layer and reported in the incoming INIT. If the source address of the INIT meets these conditions, sending the INIT ACK to the source address of the INIT is the preferred behavior.

[3.20.](#) Zero Window Probing and Unreachable Primary Path

[3.20.1.](#) Description of the Problem

[Section 6.1 of \[RFC4960\]](#) states that when sending zero window probes, SCTP should neither increment the association counter nor increment the destination address error counter if it continues to receive new packets from the peer. However, the reception of new packets from the peer does not guarantee the peer's reachability and, if the destination address becomes unreachable during zero window probing, SCTP cannot get an updated rwnd until it switches the destination address for probes.

[3.20.2.](#) Text Changes to the Document

Old text: ([Section 6.1](#))

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.

New text: ([Section 6.1](#))

If the sender continues to receive SACKs from the peer 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 could keep its window closed for an indefinite time. [Section 6.2](#) describes the receiver behavior when it advertises a zero window.

[3.20.3.](#) Solution Description

The new text clarifies that if the receiver continues to send SACKs, the sender of probes should not increment the error counter of the association and the destination address even if the SACKs do not acknowledge the probes.

[3.21.](#) Normative Language in [Section 10](#)

[3.21.1.](#) Description of the Problem

[Section 10 of \[RFC4960\]](#) is informative and, therefore, normative language such as MUST and MAY cannot be used there. However, there are several places in [Section 10](#) where MUST and MAY are used.

[3.21.2.](#) Text Changes to the Document

Old text: ([Section 10.1](#))

E) Send

Format: SEND(association id, buffer address, byte count [,context])


```
    [,stream id] [,life time] [,destination transport address]
    [,unordered flag] [,no-bundle flag] [,payload protocol-id] )
-> result
```

...

- 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.

New text: ([Section 10.1](#))

E) Send

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

...

- 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.

Old text: ([Section 10.1](#))

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]
```

...

- 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.

New text: ([Section 10.1](#))

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]

...

- 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 primitive contains a partial delivery of the whole message. When this flag is set, the stream id and stream sequence number must accompany this primitive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.

Old text: ([Section 10.1](#))

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 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.

New text: ([Section 10.1](#))

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 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 primitive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.

Old text: ([Section 10.1](#))

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 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.

New text: ([Section 10.1](#))

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 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 primitive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.

[3.21.3.](#) Solution Description

The normative language is removed from [Section 10](#). In addition, the consistency of the text has been improved.

[3.22.](#) Increase of partial_bytes_acked in Congestion Avoidance

[3.22.1.](#) Description of the Problem

Two issues have been discovered with the partial_bytes_acked handling described in [Section 7.2.2 of \[RFC4960\]](#):

- o If the Cumulative TSN Ack Point is not advanced but the SACK chunk acknowledges new TSNs in the Gap Ack Blocks, these newly acknowledged TSNs are not considered for partial_bytes_acked although these TSNs were successfully received by the peer.
- o Duplicate TSNs are not considered in partial_bytes_acked although they confirm that the DATA chunks were successfully received by the peer.

[3.22.2.](#) Text Changes to the Document

Old text: ([Section 7.2.2](#))

- 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.

New text: ([Section 7.2.2](#))

- o Whenever cwnd is greater than ssthresh, upon each SACK arrival, 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, by Gap Ack Blocks and by the number of bytes of duplicated chunks reported in Duplicate TSNS.

[3.22.3.](#) Solution Description

Now partial_bytes_acked is increased by TSNS reported as duplicated as well as TSNS newly acknowledged in Gap Ack Blocks even if the Cumulative TSN Ack Point is not advanced.

[3.23.](#) Inconsistency in Notifications Handling

[3.23.1.](#) Description of the Problem

[RFC4960] uses inconsistent normative and non-normative language when describing rules for sending notifications to the upper layer. E.g. [Section 8.2 of \[RFC4960\]](#) says that when a destination address becomes inactive due to an unacknowledged DATA chunk or HEARTBEAT chunk, SCTP SHOULD send a notification to the upper layer while [Section 8.3 of \[RFC4960\]](#) says that when a destination address becomes inactive due to an unacknowledged HEARTBEAT chunk, SCTP may send a notification to the upper layer.

This makes the text inconsistent.

[3.23.2.](#) Text Changes to the Document

The following change is based on the change described in [Section 3.6](#).

Old text: ([Section 8.1](#))

An endpoint shall keep a counter on the total number of consecutive retransmissions to its peer (this includes data retransmissions to all the destination transport addresses of the peer if it is multi-homed), including the number of unacknowledged HEARTBEAT chunks observed on the path which currently is used for data transfer. Unacknowledged HEARTBEAT chunks observed on paths different from the path currently used for data transfer shall not increment the association error counter, as this could lead to association closure even if the path which currently is used for data transfer is available (but idle). 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.

New text: ([Section 8.1](#))

An endpoint SHOULD keep a counter on the total number of consecutive retransmissions to its peer (this includes data retransmissions to all the destination transport addresses of the peer if it is multi-homed), including the number of unacknowledged HEARTBEAT chunks observed on the path which currently is used for data transfer. Unacknowledged HEARTBEAT chunks observed on paths different from the path currently used for data transfer SHOULD NOT increment the association error counter, as this could lead to association closure even if the path which currently is used for data transfer is available (but idle). If the value of this counter exceeds the limit indicated in the protocol parameter 'Association.Max.Retrans', the endpoint SHOULD 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 SHOULD 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 following changes are based on [[RFC4960](#)].

Old text: ([Section 8.2](#))

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.

New text: ([Section 8.2](#))

When an outstanding TSN is acknowledged or a HEARTBEAT sent to that address is acknowledged with a HEARTBEAT ACK, the endpoint SHOULD clear the error counter of the destination transport address to which the DATA chunk was last sent (or HEARTBEAT was sent), and SHOULD also report to the upper layer when an inactive destination address is marked as active. 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 could 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.

Old text: ([Section 8.3](#))

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.

New text: ([Section 8.3](#))

When the value of this counter exceeds the protocol parameter 'Path.Max.Retrans', the endpoint SHOULD mark the corresponding destination address as inactive if it is not so marked, and SHOULD also 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.

Old text: ([Section 8.3](#))

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](#)).

New text: ([Section 8.3](#))

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 SHOULD 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 SHOULD also clear the association overall error counter (as defined in [Section 8.1](#)).

Old text: ([Section 9.2](#))

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).

New text: ([Section 9.2](#))

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 SHOULD report the peer endpoint unreachable to the upper layer (and thus the association enters the CLOSED state).

Old text: ([Section 9.2](#))

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).

New text: ([Section 9.2](#))

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 SHOULD report the peer endpoint unreachable to the upper layer (and thus the association enters the CLOSED state).

[3.23.3.](#) Solution Description

The inconsistencies are removed by using consistently SHOULD.

[3.24.](#) SACK.Delay Not Listed as a Protocol Parameter

[3.24.1.](#) Description of the Problem

SCTP as specified in [[RFC4960](#)] supports delaying SACKs. The timer value for this is a parameter and [Section 6.2 of \[RFC4960\]](#) specifies a default and maximum value for it. However, defining a name for this parameter and listing it in the table of protocol parameters in [Section 15 of \[RFC4960\]](#) is missing.

This issue was reported as an Errata for [[RFC4960](#)] with Errata ID 4656.

[3.24.2.](#) Text Changes to the Document

Old text: ([Section 6.2](#))

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 500 ms but MUST NOT raise it above 500 ms.

New text: ([Section 6.2](#))

An implementation MUST NOT allow the maximum delay (protocol parameter 'SACK.Delay') to be configured to be more than 500 ms. In other words, an implementation MAY lower the value of SACK.Delay below 500 ms but MUST NOT raise it above 500 ms.

Old text: ([Section 15](#))

The following protocol parameters are RECOMMENDED:

RT0.Initial - 3 seconds
RT0.Min - 1 second
RT0.Max - 60 seconds
Max.Burst - 4
RT0.Alpha - 1/8
RT0.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

New text: ([Section 15](#))

The following protocol parameters are RECOMMENDED:

RT0.Initial - 3 seconds
RT0.Min - 1 second
RT0.Max - 60 seconds
Max.Burst - 4

RT0.Alpha - 1/8
RT0.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
SACK.Delay - 200 milliseconds

[3.24.3.](#) Solution Description

The parameter was given a name and added to the list of protocol parameters.

[3.25.](#) Processing of Chunks in an Incoming SCTP Packet

[3.25.1.](#) Description of the Problem

There are a few places in [[RFC4960](#)] where the receiver of a packet must discard it while processing the chunks of the packet. It is unclear whether the receiver has to rollback state changes already performed while processing the packet or not.

The intention of [[RFC4960](#)] is to process an incoming packet chunk by chunk and not to perform any prescreening of chunks in the received packet. Thus, by discarding one chunk the receiver also causes discarding of all further chunks.

[3.25.2.](#) Text Changes to the Document

Old text: ([Section 3.2](#))

- 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'.

New text: ([Section 3.2](#))

- 00 - Stop processing this SCTP packet, discard the unrecognized chunk and all further chunks.

- 01 - Stop processing this SCTP packet, discard the unrecognized chunk and all further chunks, and report the unrecognized chunk in an 'Unrecognized Chunk Type'.

Old text: ([Section 11.3](#))

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 or has a non-zero verification tag and contains an INIT-chunk.

New text: ([Section 11.3](#))

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.
The receiver of an INIT chunk MUST silently discard the INIT chunk and all further chunks if the INIT chunk is bundled with other chunks or the packet has a non-zero verification tag.

[3.25.3.](#) Solution Description

The new text makes it clear that chunks can be processed from the beginning to the end and no rollback or pre-screening is required.

[3.26.](#) CWND Increase in Congestion Avoidance Phase

[3.26.1.](#) Description of the Problem

[RFC4960] in [Section 7.2.2](#) prescribes to increase cwnd by 1*MTU per RTT if the sender has cwnd or more bytes of data outstanding to the corresponding address in the Congestion Avoidance phase. However, this is described without normative language. Moreover,

[Section 7.2.2](#) includes an algorithm how an implementation can achieve this but this algorithm is underspecified and actually allows increasing cwnd by more than 1*MTU per RTT.

[3.26.2](#). Text Changes to the Document

Old text: ([Section 7.2.2](#))

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.

New text: ([Section 7.2.2](#))

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. The basic guidelines for incrementing cwnd during congestion avoidance are:

- o SCTP MAY increment cwnd by 1*MTU.
- o SCTP SHOULD increment cwnd by one 1*MTU once per RTT when the sender has cwnd or more bytes of data outstanding for the corresponding transport address.
- o SCTP MUST NOT increment cwnd by more than 1*MTU per RTT.

Old text: ([Section 7.2.2](#))

- 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).

New text: ([Section 7.2.2](#))

- o Whenever cwnd is greater than ssthresh, upon each SACK arrival, 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, by Gap Ack Blocks and by the number of bytes of duplicated chunks reported in Duplicate TSNS.
- o When partial_bytes_acked is greater than cwnd and before the arrival of the SACK the sender had less than cwnd bytes of data outstanding (i.e., before arrival of the SACK, flightsize was less than cwnd), reset partial_bytes_acked to cwnd.
- 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), partial_bytes_acked is reset to (partial_bytes_acked - cwnd). Next, cwnd is increased by 1*MTU.

[3.26.3.](#) Solution Description

The basic guidelines for incrementing cwnd during the congestion avoidance phase are added into [Section 7.2.2](#). The guidelines include the normative language and are aligned with [[RFC5681](#)].

The algorithm from [Section 7.2.2](#) is improved to not allow increasing cwnd by more than 1*MTU per RTT.

[3.27.](#) Refresh of cwnd and ssthresh after Idle Period

[3.27.1.](#) Description of the Problem

[RFC4960] prescribes to adjust cwnd per RTO if the endpoint does not transmit data on a given transport address. In addition to that, it prescribes to set cwnd to the initial value after a sufficiently long idle period. The latter is excessive. Moreover, it is unclear what is a sufficiently long idle period.

[RFC4960] doesn't specify the handling of ssthresh in the idle case. If ssthresh is reduced due to a packet loss, ssthresh is never recovered. So traffic can end up in Congestion Avoidance all the time, resulting in a low sending rate and bad performance. The problem is even more serious for SCTP because in a multi-homed SCTP association traffic that switches back to the previously failed primary path will also lead to the situation where traffic ends up in Congestion Avoidance.

[3.27.2.](#) Text Changes to the Document

Old text: ([Section 7.2.1](#))

- o The initial cwnd before DATA transmission or after a sufficiently long idle period MUST be set to $\min(4*MTU, \max(2*MTU, 4380 \text{ bytes}))$.

New text: ([Section 7.2.1](#))

- o The initial cwnd before DATA transmission MUST be set to $\min(4*MTU, \max(2*MTU, 4380 \text{ bytes}))$.

Old text: ([Section 7.2.1](#))

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

New text: ([Section 7.2.1](#))

- o When the endpoint does not transmit data on a given transport address, the cwnd of the transport address SHOULD be adjusted to $\max(\text{cwnd}/2, 4*MTU)$ per RTO. At the first cwnd adjustment, the ssthresh of the transport address SHOULD be adjusted to the cwnd.

[3.27.3.](#) Solution Description

A rule about cwnd adjustment after a sufficiently long idle period is removed.

The text is updated to refresh ssthresh after the idle period. When the idle period is detected, the cwnd value is stored to the ssthresh value.

[3.28.](#) Window Updates After Receiver Window Opens Up

[3.28.1.](#) Description of the Problem

The sending of SACK chunks for window updates is only indirectly referenced in [\[RFC4960\]](#), [Section 6.2](#), where it is stated that an SCTP receiver must not generate more than one SACK for every incoming packet, other than to update the offered window.

However, the sending of window updates when the receiver window opens up is necessary to avoid performance problems.

[3.28.2.](#) Text Changes to the Document

Old text: ([Section 6.2](#))

An 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.

New text: ([Section 6.2](#))

An 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. When the window opens up, an SCTP receiver SHOULD send additional SACK chunks to update the window even if no new data is received. The receiver MUST avoid sending large bursts of window updates.

[3.28.3.](#) Solution Description

The new text makes clear that additional SACK chunks for window updates should be sent as long as excessive bursts are avoided.

[3.29.](#) Path of DATA and Reply Chunks

[3.29.1.](#) Description of the Problem

[Section 6.4 of \[RFC4960\]](#) describes the transmission policy for multi-homed SCTP endpoints. However, there are the following issues with it:

- o It states that a SACK should be sent to the source address of an incoming DATA. However, it is known that other SACK policies

(e.g. sending SACKs always to the primary path) may be more beneficial in some situations.

- o Initially it states that an endpoint should always transmit DATA chunks to the primary path. Then it states that the rule for transmittal of reply chunks should also be followed if the endpoint is bundling DATA chunks together with the reply chunk which contradicts with the first statement to always transmit DATA chunks to the primary path. Some implementations were having problems with it and sent DATA chunks bundled with reply chunks to a different destination address than the primary path that caused many gaps.

[3.29.2.](#) Text Changes to the Document

Old text: ([Section 6.4](#))

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.

New text: ([Section 6.4](#))

An endpoint SHOULD transmit reply chunks (e.g., INIT ACK, COOKIE ACK, HEARTBEAT ACK, etc.) in response to control chunks to the same destination transport address from which it received the control chunk to which it is replying.

The selection of the destination transport address for packets containing SACK chunks is implementation dependent. However, an endpoint SHOULD NOT vary the destination transport address of a SACK when it receives DATA chunks coming from the same source address.

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.

[3.29.3.](#) Solution Description

The SACK transmission policy is left implementation dependent but it is specified to not vary the destination address of a packet containing a SACK chunk unless there are reasons for it as it may negatively impact RTT measurement.

A confusing statement that prescribes to follow the rule for transmittal of reply chunks when the endpoint is bundling DATA chunks together with the reply chunk is removed.

[3.30.](#) Outstanding Data, Flightsize and Data In Flight Key Terms

[3.30.1.](#) Description of the Problem

[RFC4960] uses outstanding data, flightsize and data in flight key terms in formulas and statements but their definitions are not provided in [Section 1.3](#). Furthermore, outstanding data does not include DATA chunks which are classified as lost but which have not been retransmitted yet and there is a paragraph in [Section 6.1 of \[RFC4960\]](#) where this statement is broken.

[3.30.2.](#) Text Changes to the Document

Old text: ([Section 1.3](#))

- o 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.

...

- 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.

New text: ([Section 1.3](#))

- 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 Outstanding data (or Data outstanding or Data in flight): The total amount of the DATA chunks associated with outstanding TSNs.

A retransmitted DATA chunk is counted once in outstanding data.
A DATA chunk which is classified as lost but which has not been retransmitted yet is not in outstanding data.

- o Flightsize: The amount of bytes of outstanding data to a particular destination transport address at any given time.
- o Congestion window (cwnd): An SCTP variable that limits outstanding data, in number of bytes, a sender can send to a particular destination transport address before receiving an acknowledgement.

Old text: ([Section 6.1](#))

- C) When the time comes for the sender to transmit, before sending new DATA chunks, the sender MUST first transmit any outstanding DATA chunks that are marked for retransmission (limited by the current cwnd).

New text: ([Section 6.1](#))

- C) When the time comes for the sender to transmit, before sending new DATA chunks, the sender MUST first transmit any DATA chunks that are marked for retransmission (limited by the current cwnd).

[3.30.3.](#) Solution Description

Now [Section 1.3](#), Key Terms, includes explanations of outstanding data, data in flight and flightsize key terms. [Section 6.1](#) is corrected to properly use the outstanding data term.

[3.31.](#) CWND Degradation due to Max.Burst

[3.31.1.](#) Description of the Problem

Some implementations were experiencing a degradation of cwnd because of the Max.Burst limit. This was due to misinterpretation of the suggestion in [\[RFC4960\]](#), [Section 6.1](#), on how to use the Max.Burst parameter when calculating the number of packets to transmit.

[3.31.2.](#) Text Changes to the Document

Old text: ([Section 6.1](#))

- 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.

New text: ([Section 6.1](#))

- 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 temporarily 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. When calculating the number of packets to transmit and particularly using the formula above, cwnd SHOULD NOT be changed permanently.

[3.31.3.](#) Solution Description

The new text clarifies that cwnd should not be changed when applying the Max.Burst limit. This mitigates packet bursts related to the reception of SACK chunks, but not bursts related to an application sending a burst of user messages.

[3.32.](#) Reduction of RT0.Initial

[3.32.1.](#) Description of the Problem

[RFC4960] uses 3 seconds as the default value for RT0.Initial in accordance with [Section 4.3.2.1 of \[RFC1122\]](#). [\[RFC6298\]](#) updates [\[RFC1122\]](#) and lowers the initial value of the retransmission timer from 3 seconds to 1 second.

3.32.2. Text Changes to the Document

Old text: ([Section 15](#))

The following protocol parameters are RECOMMENDED:

RT0.Initial - 3 seconds
RT0.Min - 1 second
RT0.Max - 60 seconds
Max.Burst - 4
RT0.Alpha - 1/8
RT0.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
SACK.Delay - 200 milliseconds

New text: ([Section 15](#))

The following protocol parameters are RECOMMENDED:

RT0.Initial - 1 second
RT0.Min - 1 second
RT0.Max - 60 seconds
Max.Burst - 4
RT0.Alpha - 1/8
RT0.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
SACK.Delay - 200 milliseconds

3.32.3. Solution Description

The value RT0.Initial has been lowered to 1 second to be in tune with [[RFC6298](#)].

[3.33.](#) Ordering of Bundled SACK and ERROR Chunks

[3.33.1.](#) Description of the Problem

When an SCTP endpoint receives a DATA chunk with an invalid stream identifier it shall acknowledge it by sending a SACK chunk and indicate that the stream identifier was invalid by sending an ERROR chunk. These two chunks may be bundled. However, [[RFC4960](#)] requires in case of bundling that the ERROR chunk follows the SACK chunk. This restriction of the ordering is not necessary and might only limit interoperability.

[3.33.2.](#) Text Changes to the Document

Old text: ([Section 6.5](#))

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.

New text: ([Section 6.5](#))

Every DATA chunk MUST carry a valid stream identifier. If an endpoint receives a DATA chunk with an invalid stream identifier, it SHOULD 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 and the SACK Chunk in the same packet.

[3.33.3.](#) Solution Description

The unnecessary restriction regarding the ordering of the SACK and ERROR chunk has been removed.

[3.34.](#) Undefined Parameter Returned by RECEIVE Primitive

[3.34.1.](#) Description of the Problem

[RFC4960] provides a description of an abstract API. In the definition of the RECEIVE primitive an optional parameter with name "delivery number" is mentioned. However, no definition of this parameter is given in [[RFC4960](#)] and the parameter is unnecessary.

[3.34.2.](#) Text Changes to the Document

Old text: ([Section 10.1](#))

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]

New text: ([Section 10.1](#))

G) Receive

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

[3.34.3.](#) Solution Description

The undefined parameter has been removed.

[3.35.](#) DSCP Changes

[3.35.1.](#) Description of the Problem

The upper layer can change the Differentiated Services Code Point (DSCP) used for packets being sent. A change of the DSCP can result in packets hitting different queues on the path and, therefore, the congestion control should be initialized when the DSCP is changed by the upper layer. This is not described in [[RFC4960](#)].

[3.35.2](#). Text Changes to the Document

New text: ([Section 7.2.5](#))

SCTP implementations MAY allow an application to configure the Differentiated Services Code Point (DSCP) used for sending packets. If a DSCP change might result in outgoing packets being queued in different queues, the congestion control parameters for all affected destination addresses MUST be reset to their initial values.

Old text: ([Section 10.1](#))

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.

Old text: ([Section 10.1](#))

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](#), or other parameters like the DSCP) that the SCTP user wishes to customize.

[3.35.3](#). Solution Description

Text describing the required action on DSCP changes has been added.

[3.36](#). Inconsistent Handling of ICMPv4 and ICMPv6 Messages

[3.36.1](#). Description of the Problem

[Appendix C of \[RFC4960\]](#) describes the handling of ICMPv4 and ICMPv6 messages. The text explicitly describes the handling of ICMPv6 packets indicating reachability problems, but does not do the same for the corresponding ICMPv4 packets.

[3.36.2](#). Text Changes to the Document

Old text: (Appendix C)

ICMP3) An implementation MAY ignore any ICMPv4 messages where the code does not indicate "Protocol Unreachable" or "Fragmentation Needed".

New text:

ICMP3) An implementation SHOULD ignore any ICMPv4 messages where the code indicates "Port Unreachable".

Old text: (Appendix C)

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.

New text:

ICMP9) If the ICMP type is "Destination Unreachable", the implementation MAY mark the destination into the unreachable state or alternatively increment the path error counter.

[3.36.3.](#) Solution Description

The text has been changed to not limit the processing of ICMPv4 packets with type "Destination Unreachable" by rewording the third rule. Furthermore, remove the limitation to ICMPv6 in the ninth rule.

[3.37.](#) Handling of Soft Errors

[3.37.1.](#) Description of the Problem

[RFC1122] defines the handling of soft errors and hard errors for TCP. [Appendix C of \[RFC4960\]](#) only deals with hard errors.

[3.37.2.](#) Text Changes to the Document

Old text: (Appendix C)

ICMP8) If the ICMP type is "Destination Unreachable", the implementation MAY mark the destination into the unreachable state or alternatively increment the path error counter.

New text: (Appendix C)

ICMP8) If the ICMP type is "Destination Unreachable", the implementation MAY mark the destination into the unreachable state or alternatively increment the path error counter. SCTP MAY provide information to the upper layer indicating the reception of ICMP messages when reporting a network status change.

[3.37.3.](#) Solution Description

Text has been added allowing SCTP to notify the application in case of soft errors.

[3.38.](#) Honoring CWND

[3.38.1.](#) Description of the Problem

When using the slow start algorithm, SCTP increases the congestion window only when it is being fully utilized. Since SCTP uses DATA chunks and does not use the congestion window to fragment user messages, this requires that some overbooking of the congestion window is allowed.

[3.38.2.](#) Text Changes to the Document

Old text: ([Section 6.1](#))

- 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.

New text: ([Section 6.1](#))

- B) At any given time, the sender MUST NOT transmit new data to a given transport address if it has $\text{cwnd} + (\text{PMTU} - 1)$ or more bytes of data outstanding to that transport address. If data is available the sender SHOULD exceed cwnd by up to $(\text{PMTU}-1)$ bytes on a new data transmission if the flightsize does not currently reach cwnd. The breach of cwnd MUST constitute one packet only.

Old text: ([Section 7.2.1](#))

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

New text: ([Section 7.2.1](#))

- o Whenever cwnd is greater than zero, the endpoint is allowed to have cwnd bytes of data outstanding on that transport address. A limited overbooking as described in B) of [Section 6.1](#) SHOULD be supported.

[3.38.3](#). Solution Description

Text was added to clarify how the cwnd limit should be handled.

[3.39](#). Zero Window Probing

[3.39.1](#). Description of the Problem

The text describing zero window probing was not clearly handling the case where the window was not zero, but too small for the next DATA chunk to be transmitted. Even in this case, zero window probing has to be performed to avoid deadlocks.

3.39.2. Text Changes to the Document

Old text: ([Section 6.1](#))

- 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.

New text: ([Section 6.1](#))

- 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 smaller than the size of the next DATA chunk; 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 has no buffer space, 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.

3.39.3. Solution Description

The terminology is used in a cleaner way.

[3.40.](#) Updating References Regarding ECN

[3.40.1.](#) Description of the Problem

[RFC4960] refers for ECN only to [\[RFC3168\]](#), which will be updated by [\[RFC8311\]](#). This needs to be reflected when referring to ECN.

[3.40.2.](#) Text Changes to the Document

Old text: (Appendix A)

ECN [\[RFC3168\]](#) describes a proposed extension to IP that details a method to become aware of congestion outside of datagram loss.

New text: (Appendix A)

ECN as specified in [\[RFC3168\]](#) updated by [\[RFC8311\]](#) describes a proposed extension to IP that details a method to become aware of congestion outside of datagram loss.

Old text: (Appendix A)

In general, [\[RFC3168\]](#) should be followed with the following exceptions.

New text: (Appendix A)

In general, [\[RFC3168\]](#) updated by [\[RFC8311\]](#) SHOULD be followed with the following exceptions.

Old text: (Appendix A)

[\[RFC3168\]](#) details negotiation of ECN during the SYN and SYN-ACK stages of a TCP connection.

New text: (Appendix A)

[RFC3168] updated by [\[RFC8311\]](#) details negotiation of ECN during the SYN and SYN-ACK stages of a TCP connection.

Old text: (Appendix A)

[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.

New text: (Appendix A)

[RFC3168] updated by [\[RFC8311\]](#) 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.

Old text: (Appendix A)

[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.

New text: (Appendix A)

[RFC3168] updated by [\[RFC8311\]](#) 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.

[3.40.3.](#) Solution Description

References to [\[RFC8311\]](#) have been added.

[3.41.](#) Host Name Address Parameter Deprecated

[3.41.1.](#) Description of the Problem

[RFC4960] defines three types of address parameters to be used with INIT and INIT ACK chunks:

1. IPv4 Address parameters.
2. IPv6 Address parameters.
3. Host Name Address parameters.

The first two are supported by the SCTP kernel implementations of FreeBSD, Linux and Solaris, but the third one is not. In addition, the first two were successfully tested in all nine interoperability tests for SCTP, but the third one has never been successfully tested. Therefore, the Host Name Address parameter should be deprecated.

[3.41.2.](#) Text Changes to the Document

Old text: ([Section 3.3.2](#))

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.

New text: ([Section 3.3.2](#))

Note 3: An INIT chunk MUST NOT contain the Host Name Address parameter. The receiver of an INIT chunk containing a Host Name Address parameter MUST send an ABORT and MAY include an Error Cause indicating an Unresolvable Address.

Old text: ([Section 3.3.2.1](#))

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.

New text: ([Section 3.3.2.1](#))

The sender of an INIT chunk MUST NOT include this parameter. The usage of the Host Name Address parameter is deprecated.

Old text: ([Section 3.3.2.1](#))

Address Type: 16 bits (unsigned integer)

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

New text: ([Section 3.3.2.1](#))

Address Type: 16 bits (unsigned integer)

This is filled with the type value of the corresponding address TLV (e.g., IPv4 = 5, IPv6 = 6). The value indicating the Host Name Address parameter (Host name = 11) MUST NOT be used.

Old text: ([Section 3.3.3](#))

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.

New text: ([Section 3.3.3](#))

Note 3: An INIT ACK chunk MUST NOT contain the Host Name Address parameter. The receiver of INIT ACK chunks containing a Host Name Address parameter MUST send an ABORT and MAY include an Error Cause indicating an Unresolvable Address.

Old text: ([Section 5.1.2](#))

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.

New text: ([Section 5.1.2](#))

- B) If there is a Host Name parameter present in the received INIT or INIT ACK chunk, the endpoint MUST immediately send an ABORT and MAY include an Error Cause indicating an Unresolvable Address to its peer. The ABORT SHALL be sent to the source IP address from which the last peer packet was received.

Old text: ([Section 11.2.4.1](#))

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 INITs 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.

New text: ([Section 11.2.4.1](#))

The support of the Host Name Address parameter has been removed from the protocol. Endpoints receiving INIT or INIT ACK chunks containing the Host Name Address parameter MUST send an ABORT chunk in response and MAY include an Error Cause indicating an Unresolvable Address.

[3.41.3.](#) Solution Description

The usage of the Host Name Address parameter has been deprecated.

[3.42.](#) Conflicting Text Regarding the Supported Address Types Parameter

[3.42.1.](#) Description of the Problem

When receiving an SCTP packet containing an INIT chunk sent from an address for which the corresponding address type is not listed in the Supported Address Types, there is conflicting text in [Section 5.1.2 of \[RFC4960\]](#). It is stated that the association MUST be aborted and also that the association SHOULD be established and there SHOULD NOT be any error indication.

[3.42.2.](#) Text Changes to the Document

Old text: ([Section 5.1.2](#))

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 (initiate) 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.

New text: ([Section 5.1.2](#))

The sender of INIT chunks MAY include a 'Supported Address Types' parameter in the INIT to indicate what types of addresses are acceptable.

[3.42.3.](#) Solution Description

The conflicting text has been removed.

[3.43.](#) Integration of [RFC 6096](#)

[3.43.1.](#) Description of the Problem

[RFC6096] updates [[RFC4960](#)] by adding a Chunk Flags Registry. This should be integrated into the base specification.

[3.43.2.](#) Text Changes to the Document

Old text: ([Section 14.1](#))

[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 the 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.

New text: ([Section 14.1](#))

[14.1](#). IETF-Defined Chunk Extension

The assignment of new chunk type codes is done through an IETF Review action, as defined in [[RFC8126](#)]. Documentation of a new chunk 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 of \[RFC4960\]](#);
- c) A detailed definition and description of the intended use of each field within the chunk, including the chunk flags if any. Defined chunk flags will be used as initial entries in the chunk flags table for the new chunk type;
- 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.

For each new chunk type, IANA creates a registration table for the chunk flags of that type. The procedure for registering particular chunk flags is described in the following [Section 14.2](#).

New text: ([Section 14.2](#))

[14.2](#). New IETF Chunk Flags Registration

The assignment of new chunk flags is done through an RFC required action, as defined in [[RFC8126](#)]. Documentation of the chunk flags MUST contain the following information:

- a) A name for the new chunk flag;
- b) A detailed procedural description of the use of the new chunk flag within the operation of the protocol. It MUST be considered that implementations not supporting the flag will send '0' on transmit and just ignore it on receipt.

IANA selects a chunk flags value. This MUST be one of 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, or 0x80, which MUST be unique within the chunk flag values for the specific chunk type.

Please note that Sections [14.2](#), [14.3](#), [14.4](#), and [14.5](#) need to be renumbered.

[3.43.3](#). Solution Description

[RFC6096] was integrated and the reference updated to [[RFC8126](#)].

[3.44](#). Integration of [RFC 6335](#)

[3.44.1](#). Description of the Problem

[RFC6335] updates [[RFC4960](#)] by updating Procedures for the Port Numbers Registry. This should be integrated into the base specification. While there, update the reference to the RFC giving guidelines for writing IANA sections to [[RFC8126](#)].

[3.44.2](#). Text Changes to the Document

Old text: ([Section 14.5](#))

SCTP services may use contact port numbers to provide service to unknown callers, as in TCP and UDP. IANA is therefore requested to open the existing Port Numbers registry for SCTP using the following rules, which we intend to mesh well with existing Port Numbers registration procedures. An IESG-appointed Expert Reviewer supports IANA in evaluating SCTP port allocation requests, according to the procedure defined in [[RFC2434](#)].

Port numbers are divided into three ranges. The Well Known Ports are those from 0 through 1023, the Registered Ports are those from 1024 through 49151, and the Dynamic and/or Private Ports are those from

49152 through 65535. Well Known and Registered Ports are intended for use by server applications that desire a default contact point on a system. On most systems, Well Known Ports can only be used by system (or root) processes or by programs executed by privileged users, while Registered Ports can be used by ordinary user processes or programs executed by ordinary users. Dynamic and/or Private Ports are intended for temporary use, including client-side ports, out-of-band negotiated ports, and application testing prior to registration of a dedicated port; they MUST NOT be registered.

The Port Numbers registry should accept registrations for SCTP ports in the Well Known Ports and Registered Ports ranges. Well Known and Registered Ports SHOULD NOT be used without registration. Although in some cases -- such as porting an application from TCP to SCTP -- it may seem natural to use an SCTP port before registration completes, we emphasize that IANA will not guarantee registration of particular Well Known and Registered Ports. Registrations should be requested as early as possible.

Each port registration SHALL include the following information:

- o A short port name, consisting entirely of letters (A-Z and a-z), digits (0-9), and punctuation characters from "-_+./*" (not including the quotes).
- o The port number that is requested for registration.
- o A short English phrase describing the port's purpose.
- o Name and contact information for the person or entity performing the registration, and possibly a reference to a document defining the port's use. Registrations coming from IETF working groups need only name the working group, but indicating a contact person is recommended.

Registrants are encouraged to follow these guidelines when submitting a registration.

- o A port name SHOULD NOT be registered for more than one SCTP port number.
- o A port name registered for TCP MAY be registered for SCTP as well. Any such registration SHOULD use the same port number as the existing TCP registration.
- o Concrete intent to use a port SHOULD precede port registration. For example, existing TCP ports SHOULD NOT be registered in advance of any intent to use those ports for SCTP.

 New text: ([Section 14.5](#))

SCTP services can use contact port numbers to provide service to unknown callers, as in TCP and UDP. IANA is therefore requested to open the existing Port Numbers registry for SCTP using the following rules, which we intend to mesh well with existing Port Numbers registration procedures. An IESG-appointed Expert Reviewer supports IANA in evaluating SCTP port allocation requests, according to the procedure defined in [[RFC8126](#)]. The details of this process are defined in [[RFC6335](#)].

[3.44.3. Solution Description](#)

[RFC6335] was integrated and the reference was updated to [[RFC8126](#)].

[3.45. Integration of \[RFC 7053\]\(#\)](#)

[3.45.1. Description of the Problem](#)

[RFC7053] updates [[RFC4960](#)] by adding the I bit to the DATA chunk. This should be integrated into the base specification.

[3.45.2. Text Changes to the Document](#)

 Old text: ([Section 3.3.1](#))

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	2	3	4	5	6	7	8	9
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.

New text: ([Section 3.3.1](#))

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								
Type = 0										Res										I U B E										Length									
TSN																																							
Stream Identifier S															Stream Sequence Number n																								
Payload Protocol Identifier																																							
User Data (seq n of Stream S)																																							

Res: 4 bits

SHOULD be set to all '0's and ignored by the receiver.

I bit: 1 bit

The (I)mmediate Bit MAY be set by the sender, whenever the sender of a DATA chunk can benefit from the corresponding SACK chunk being sent back without delay. See [[RFC7053](#)] for a discussion about

New text: (Append to [Section 6.1](#))

Whenever the sender of a DATA chunk can benefit from the corresponding SACK chunk being sent back without delay, the sender MAY set the I bit in the DATA chunk header. Please note that why the sender has set the I bit is irrelevant to the receiver.

Reasons for setting the I bit include, but are not limited to (see [Section 4 of \[RFC7053\]](#) for the benefits):

- o The application requests to set the I bit of the last DATA chunk of a user message when providing the user message to the SCTP

implementation (see [Section 7](#)).

- o The sender is in the SHUTDOWN-PENDING state.
- o The sending of a DATA chunk fills the congestion or receiver window.

Old text: ([Section 6.2](#))

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.

New text: ([Section 6.2](#))

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.

Upon receipt of an SCTP packet containing a DATA chunk with the I bit set, the receiver SHOULD NOT delay the sending of the corresponding SACK chunk, i.e., the receiver SHOULD immediately respond with the corresponding SACK chunk.

Old text: ([Section 10.1](#))

E) Send

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

New text: ([Section 10.1](#))

E) Send

Format: SEND(association id, buffer address, byte count [,context]
 [,stream id] [,life time] [,destination transport address]


```

        [,unordered flag] [,no-bundle flag] [,payload protocol-id]
        [,sack immediately] )
-> result

```

```

-----
New text: (Append optional parameter in Subsection E of Section 10.1)
-----

```

- o sack immediately - set the I bit on the last DATA chunk used for sending buffer.

Please note that the change in [Section 6.2](#) is only about adding a paragraph.

[3.45.3. Solution Description](#)

[RFC7053] was integrated.

[3.46. CRC32c Code Improvements](#)

[3.46.1. Description of the Problem](#)

The code given for the CRC32c computations uses types like long which may have different length on different operating systems or processors. Therefore, the code is changed to use specific types like uint32_t.

While there, fix also some syntax errors and a comment.

[3.46.2. Text Changes to the Document](#)

```

-----
Old text: (Appendix B)
-----
/*****
/* 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,
0x96BF4DCCCL, 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, 0xCA7A905L, 0xD9F75AF1L, 0x2B9CD9F2L,
0xFF56BD19L, 0x0D3D3E1AL, 0x1E6DCDEEL, 0xEC064EEDL,
0xC38D26C4L, 0x31E6A5C7L, 0x22B65633L, 0xD0DDD530L,
0x0417B1DBL, 0xF67C32D8L, 0xE52CC12CL, 0x1747422FL,
0x49547E0BL, 0xBB3FFD08L, 0xA86F0EFCL, 0x5A048DFFL,
0x8ECEEE914L, 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

```

```

-----

```

```

New text: (Appendix B)

```

```

-----

```

```

<CODE BEGINS>

```

```

/*****
/* 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_h__
#define __crc32cr_h__

```

```

#define CRC32C_POLY 0x1EDC6F41UL
#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])

```

```

uint32_t crc_c[256] =
{

```


0x00000000UL, 0xF26B8303UL, 0xE13B70F7UL, 0x1350F3F4UL,
0xC79A971FUL, 0x35F1141CUL, 0x26A1E7E8UL, 0xD4CA64EBUL,
0x8AD958CFUL, 0x78B2DBCCUL, 0x6BE22838UL, 0x9989AB3BUL,
0x4D43CFD0UL, 0xBF284CD3UL, 0xAC78BF27UL, 0x5E133C24UL,
0x105EC76FUL, 0xE235446CUL, 0xF165B798UL, 0x030E349BUL,
0xD7C45070UL, 0x25AFD373UL, 0x36FF2087UL, 0xC494A384UL,
0x9A879FA0UL, 0x68EC1CA3UL, 0x7BBCEF57UL, 0x89D76C54UL,
0x5D1D08BFUL, 0xAF768BBCUL, 0xBC267848UL, 0x4E4DFB4BUL,
0x20BD8EDEUL, 0xD2D60DDUL, 0xC186FE29UL, 0x33ED7D2AUL,
0xE72719C1UL, 0x154C9AC2UL, 0x061C6936UL, 0xF477EA35UL,
0xAA64D611UL, 0x580F5512UL, 0x4B5FA6E6UL, 0xB93425E5UL,
0x6DFE410EUL, 0x9F95C20DUL, 0x8CC531F9UL, 0x7EAE2FAUL,
0x30E349B1UL, 0xC288CAB2UL, 0xD1D83946UL, 0x23B3BA45UL,
0xF779DEAEUL, 0x05125DADUL, 0x1642AE59UL, 0xE4292D5AUL,
0xBA3A117EUL, 0x4851927DUL, 0x5B016189UL, 0xA96AE28AUL,
0x7DA08661UL, 0x8FCB0562UL, 0x9C9BF696UL, 0x6EF07595UL,
0x417B1DBCUL, 0xB3109EBFUL, 0xA0406D4BUL, 0x522BEE48UL,
0x86E18AA3UL, 0x748A09A0UL, 0x67DAFA54UL, 0x95B17957UL,
0xCBA24573UL, 0x39C9C670UL, 0x2A993584UL, 0xD8F2B687UL,
0x0C38D26CUL, 0xFE53516FUL, 0xED03A29BUL, 0x1F682198UL,
0x5125DAD3UL, 0xA34E59D0UL, 0xB01EAA24UL, 0x42752927UL,
0x96BF4DCCUL, 0x64D4CECFUL, 0x77843D3BUL, 0x85EFBE38UL,
0xDBFC821CUL, 0x2997011FUL, 0x3AC7F2EBUL, 0xC8AC71E8UL,
0x1C661503UL, 0xEE0D9600UL, 0xFD5D65F4UL, 0x0F36E6F7UL,
0x61C69362UL, 0x93AD1061UL, 0x80FDE395UL, 0x72966096UL,
0xA65C047DUL, 0x5437877EUL, 0x4767748AUL, 0xB50CF789UL,
0xEB1FCBADUL, 0x197448AEUL, 0x0A24BB5AUL, 0xF84F3859UL,
0x2C855CB2UL, 0xDEEEDFB1UL, 0xCDBE2C45UL, 0x3FD5AF46UL,
0x7198540DUL, 0x83F3D70EUL, 0x90A324FAUL, 0x62C8A7F9UL,
0xB602C312UL, 0x44694011UL, 0x5739B3E5UL, 0xA55230E6UL,
0xFB410CC2UL, 0x092A8FC1UL, 0x1A7A7C35UL, 0xE811FF36UL,
0x3CDB9BDDUL, 0xCEB018DEUL, 0xDDE0EB2AUL, 0x2F8B6829UL,
0x82F63B78UL, 0x709DB87BUL, 0x63CD4B8FUL, 0x91A6C88CUL,
0x456CAC67UL, 0xB7072F64UL, 0xA457DC90UL, 0x563C5F93UL,
0x082F63B7UL, 0xFA44E0B4UL, 0xE9141340UL, 0x1B7F9043UL,
0xCFB5F4A8UL, 0x3DDE77ABUL, 0x2E8E845FUL, 0xDCE5075CUL,
0x92A8FC17UL, 0x60C37F14UL, 0x73938CE0UL, 0x81F80FE3UL,
0x55326B08UL, 0xA759E80BUL, 0xB4091BFFUL, 0x466298FCUL,
0x1871A4D8UL, 0xEA1A27DBUL, 0xF94AD42FUL, 0x0B21572CUL,
0xDFEB33C7UL, 0x2D80B0C4UL, 0x3ED04330UL, 0xCCBBC033UL,
0xA24BB5A6UL, 0x502036A5UL, 0x4370C551UL, 0xB11B4652UL,
0x65D122B9UL, 0x97BAA1BAUL, 0x84EA524EUL, 0x7681D14DUL,
0x2892ED69UL, 0xDAF96E6AUL, 0xC9A99D9EUL, 0x3BC21E9DUL,
0xEF087A76UL, 0x1D63F975UL, 0x0E330A81UL, 0xFC588982UL,
0xB21572C9UL, 0x407EF1CAUL, 0x532E023EUL, 0xA145813DUL,
0x758FE5D6UL, 0x87E466D5UL, 0x94B49521UL, 0x66DF1622UL,
0x38CC2A06UL, 0xCA7A905UL, 0xD9F75AF1UL, 0x2B9CD9F2UL,
0xFF56BD19UL, 0x0D3D3E1AUL, 0x1E6DCDEEUL, 0xEC064EEDUL,


```
0xC38D26C4UL, 0x31E6A5C7UL, 0x22B65633UL, 0xD0DDD530UL,  
0x0417B1DBUL, 0xF67C32D8UL, 0xE52CC12CUL, 0x1747422FUL,  
0x49547E0BUL, 0xBB3FFD08UL, 0xA86F0EFCUL, 0x5A048DFFUL,  
0x8ECE914UL, 0x7CA56A17UL, 0x6FF599E3UL, 0x9D9E1AE0UL,  
0xD3D3E1ABUL, 0x21B862A8UL, 0x32E8915CUL, 0xC083125FUL,  
0x144976B4UL, 0xE622F5B7UL, 0xF5720643UL, 0x07198540UL,  
0x590AB964UL, 0xAB613A67UL, 0xB831C993UL, 0x4A5A4A90UL,  
0x9E902E7BUL, 0x6CFBAD78UL, 0x7FAB5E8CUL, 0x8DC0DD8FUL,  
0xE330A81AUL, 0x115B2B19UL, 0x020BD8EDUL, 0xF0605BEEUL,  
0x24AA3F05UL, 0xD6C1BC06UL, 0xC5914FF2UL, 0x37FACCF1UL,  
0x69E9F0D5UL, 0x9B8273D6UL, 0x88D28022UL, 0x7AB90321UL,  
0xAE7367CAUL, 0x5C18E4C9UL, 0x4F48173DUL, 0xBD23943EUL,  
0xF36E6F75UL, 0x0105EC76UL, 0x12551F82UL, 0xE03E9C81UL,  
0x34F4F86AUL, 0xC69F7B69UL, 0xD5CF889DUL, 0x27A40B9EUL,  
0x79B737BAUL, 0x8BDCB4B9UL, 0x988C474DUL, 0x6AE7C44EUL,  
0xBE2DA0A5UL, 0x4C4623A6UL, 0x5F16D052UL, 0xAD7D5351UL,  
};
```

```
#endif
```

```
-----
```

```
Old text: (Appendix B)
```

```
-----
```

```
/* 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);
}

```

New text: (Appendix B)

```
/* Example of table build routine */

#include <stdio.h>
#include <stdlib.h>

#define OUTPUT_FILE    "crc32cr.h"
#define CRC32C_POLY    0x1EDC6F41UL

static FILE *tf;

static uint32_t
reflect_32(uint32_t b)
{
    int i;
    uint32_t rw = 0UL;

    for (i = 0; i < 32; i++) {
        if (b & 1)
            rw |= 1 << (31 - i);
        b >>= 1;
    }
    return (rw);
}

static uint32_t
build_crc_table(int index)
{
    int i;
    uint32_t rb;

    rb = reflect_32(index);

    for (i = 0; i < 8; i++) {
        if (rb & 0x80000000UL)
            rb = (rb << 1) ^ (uint32_t)CRC32C_POLY;
        else
            rb <<= 1;
    }
    return (reflect_32(rb));
}

int
main (void)
{
    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_h__\n");
fprintf(tf, "#define __crc32cr_h__\n\n");
fprintf(tf, "#define CRC32C_POLY 0x%08XUL\n",
    (uint32_t)CRC32C_POLY);
fprintf(tf,
    "#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])\n");
fprintf(tf, "\nuint32_t crc_c[256] =\n{\n");
for (i = 0; i < 256; i++) {
    fprintf(tf, "0x%08XUL", build_crc_table (i));
    if ((i & 3) == 3)
        fprintf(tf, "\n");
    else
        fprintf(tf, " ");
}
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);
}

```

Old text: (Appendix B)

```

/* 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);
}
```

New text: (Appendix B)

```
/* Example of crc insertion */

#include "crc32cr.h"

uint32_t
generate_crc32c(unsigned char *buffer, unsigned int length)
{
    unsigned int i;
    uint32_t crc32 = 0xffffffffUL;
    uint32_t result;
    uint8_t 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 bits of each byte 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;
    uint32_t crc32;
    message = (SCTP_message *) buffer;
    message->common_header.checksum = 0UL;
    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;
    uint32_t original_crc32;
    uint32_t crc32;

    /* 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);
}
<CODE ENDS>
```

[3.46.3.](#) Solution Description

The code was changed to use platform independent types.

[3.47.](#) Clarification of Gap Ack Blocks in SACK Chunks

[3.47.1.](#) Description of the Problem

The Gap Ack Blocks in the SACK chunk are intended to be isolated. However, this is not mentioned with normative text.

[3.47.2.](#) Text Changes to the Document

Old text: ([Section 3.3.4](#))

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.

New text: ([Section 3.3.4](#))

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. The Gap Ack Blocks SHOULD be isolated. This means that the TSN just before each Gap Ack Block and the TSN just after each Gap Ack Block has not been received. By definition, all TSNs acknowledged by Gap Ack Blocks are greater than the value of the Cumulative TSN Ack.

Old text: ([Section 3.3.4](#))

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.

New text: ([Section 3.3.4](#))

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 Blocks SHOULD be isolated. That means that the DATA chunks with TSN equal to (Cumulative TSN Ack + Gap Ack Block Start - 1) and (Cumulative TSN Ack + Gap Ack Block End + 1) have not been received.

[3.47.3.](#) Solution Description

Normative text describing the intended usage of Gap Ack Blocks has been added.

[3.48.](#) Handling of SSN Wrap Arouns

[3.48.1.](#) Description of the Problem

The Stream Sequence Number (SSN) is used for preserving the ordering of user messages within each SCTP stream. The SSN is limited to 16 bits. Therefore, multiple wrap arounds of the SSN might happen within the current send window. To allow the receiver to deliver ordered user messages in the correct sequence, the sender should limit the number of user messages per stream.

[3.48.2.](#) Text Changes to the Document

Old text: ([Section 6.1](#))

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.

New text: ([Section 6.1](#))

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.

Note: For each stream, the data sender SHOULD NOT have more than $2^{16} - 1$ ordered user messages in the current send window.

[3.48.3.](#) Solution Description

The data sender is required to limit the number of ordered user messages within the current send window.

[3.49.](#) Update [RFC 2119](#) Boilerplate

[3.49.1.](#) Description of the Problem

The text to be used to refer to the [[RFC2119](#)] terms has been updated by [[RFC8174](#)].

[3.49.2.](#) Text Changes to the Document

Old text: ([Section 2](#))

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](#) [[RFC2119](#)].

New text: ([Section 2](#))

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [BCP 14](#) [[RFC2119](#)] [[RFC8174](#)] when, and only when, they appear in all capitals, as shown here.

[3.49.3.](#) Solution Description

The text has been updated to the one specified in [[RFC8174](#)].

[3.50.](#) Missed Text Removal

[3.50.1.](#) Description of the Problem

When integrating the changes to [Section 7.2.4 of \[RFC2960\]](#) as described in [Section 2.8.2 of \[RFC4460\]](#) some text was not removed and is therefore still in [[RFC4960](#)].

[3.50.2.](#) Text Changes to the Document

Old text: ([Section 7.2.4](#))

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.

New text: ([Section 7.2.4](#))

[3.50.3.](#) Solution Description

The text has finally been removed.

[4.](#) IANA Considerations

[Section 3.44](#) of this document updates the port number registry for SCTP to be consistent with [[RFC6335](#)]. IANA is requested to review [Section 3.44](#).

[5.](#) Security Considerations

This document does not add any security considerations to those given in [[RFC4960](#)].

[6.](#) Acknowledgments

The authors wish to thank Pontus Andersson, Eric W. Biederman, Cedric Bonnet, Spencer Dawkins, Gorrry Fairhurst, Peter Lei, Gyula Marosi, Lionel Morand, Jeff Morriss, Karen E. E. Nielsen, Tom Petch, Kacheong Poon, Julien Pourtet, Irene Ruengeler, Michael Welzl, and Qiaobing Xie for their invaluable comments.

[7.](#) References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", [RFC 4960](#), DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/info/rfc4960>>.

7.2. Informative References

- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, [RFC 1122](#), DOI 10.17487/RFC1122, October 1989, <<https://www.rfc-editor.org/info/rfc1122>>.
- [RFC1858] Ziemba, G., Reed, D., and P. Traina, "Security Considerations for IP Fragment Filtering", [RFC 1858](#), DOI 10.17487/RFC1858, October 1995, <<https://www.rfc-editor.org/info/rfc1858>>.
- [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](#), DOI 10.17487/RFC2960, October 2000, <<https://www.rfc-editor.org/info/rfc2960>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", [RFC 3168](#), DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.
- [RFC4460] Stewart, R., Arias-Rodriguez, I., Poon, K., Caro, A., and M. Tuexen, "Stream Control Transmission Protocol (SCTP) Specification Errata and Issues", [RFC 4460](#), DOI 10.17487/RFC4460, April 2006, <<https://www.rfc-editor.org/info/rfc4460>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", [RFC 5681](#), DOI 10.17487/RFC5681, September 2009, <<https://www.rfc-editor.org/info/rfc5681>>.
- [RFC6096] Tuexen, M. and R. Stewart, "Stream Control Transmission Protocol (SCTP) Chunk Flags Registration", [RFC 6096](#), DOI 10.17487/RFC6096, January 2011, <<https://www.rfc-editor.org/info/rfc6096>>.

- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", [RFC 6298](#), DOI 10.17487/RFC6298, June 2011, <<https://www.rfc-editor.org/info/rfc6298>>.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", [BCP 165](#), [RFC 6335](#), DOI 10.17487/RFC6335, August 2011, <<https://www.rfc-editor.org/info/rfc6335>>.
- [RFC7053] Tuexen, M., Ruengeler, I., and R. Stewart, "SACK-IMMEDIATELY Extension for the Stream Control Transmission Protocol", [RFC 7053](#), DOI 10.17487/RFC7053, November 2013, <<https://www.rfc-editor.org/info/rfc7053>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", [BCP 26](#), [RFC 8126](#), DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in [RFC 2119](#) Key Words", [BCP 14](#), [RFC 8174](#), DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8311] Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", [RFC 8311](#), DOI 10.17487/RFC8311, January 2018, <<https://www.rfc-editor.org/info/rfc8311>>.

Authors' Addresses

Randall R. Stewart
Netflix, Inc.
Chapin, SC 29036
United States

Email: randall@lakerest.net

Michael Tuexen
Muenster University of Applied Sciences
Stegerwaldstrasse 39
48565 Steinfurt
Germany

Email: tuexen@fh-muenster.de

Maksim Proshin
Ericsson
Kistavaegen 25
Stockholm 164 80
Sweden

Email: mproshin@tieto.mera.ru