Network Working Group   R. Stewart
Internet-Draft   Cisco Systems, Inc.
Expires: April 27, 2006   I. Arias-Rodriguez
                                                 Nokia Research Center
                                                            K. Poon
                                               Sun Microsystems, Inc.
                                                            A. Caro
                                              University of Delaware
                                                          M. Tuexen
                                   Muenster Univ. of Applied Sciences
                                                   October 24, 2005

   **Stream Control Transmission Protocol (SCTP) Specification Errata and
                                Issues**
                 draft-ietf-tsvwg-sctpimpguide-16.txt

Status of this Memo

Copyright Notice

Abstract

This document is a compilation of issues found during six
interoperability events and 5 years of experience with implementing,
testing, and using SCTP along with the suggested fixes. This
document provides deltas to RFC 2960 and is organized in a time based
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.

Table of Contents

1. **Introduction**

   This document contains a compilation of all defects found up until
   the publishing of this document for the Stream Control Transmission
   Protocol (SCTP) RFC2960 [7]. 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
   compliled into RFC2960's [7] BIS document. Each error will be
   detailed within this document in the form of:

   o The problem description,

   o The text quoted from RFC2960 [7],

   o The replacement text that should be placed into the 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. Each
   section should be applied in sequence to the original RFC2960 [7]
   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.

1.1. **Conventions**

   The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD,
   SHOULD NOT, RECOMMENDED, NOT RECOMMENDED, MAY, and OPTIONAL, when
   they appear in this document, are to be interpreted as described in
   RFC2119 [3].

2. Corrections to **RFC2960**

2.1. Incorrect error type during chunk processing.

2.1.1. Description of the problem

   A typo was discovered in RFC2960 [7] that incorrectly specifies an
   action to be taken when processing chunks of unknown identity.

2.1.2. Text changes to the document

   Old text: (Section 3.2)
   ----

   01 - Stop processing this SCTP packet and discard it, do not process
        any further chunks within it, and report the unrecognized
        parameter in an 'Unrecognized Parameter Type' (in either an
        ERROR or in the INIT ACK).

   New text: (Section 3.2)
   ----

   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'.

2.1.3. Solution description

   The receiver of an unrecognized Chunk should not send a 'parameter'
   error but instead the appropriate chunk error as described above.

2.2. Parameter processing issue

2.2.1. Description of the problem

   A typographical error was introduced through an improper cut and
   paste in the use of the upper two bits to describe proper handling of
   unknown parameters.

2.2.2. **Text changes to the document**

```
   ---------
   Old text: (Section 3.2.1)
   ---------
```

   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
        parameter in an 'Unrecognized Parameter Type' (in either an
        ERROR or in the INIT ACK).

```
   ---------
   New text: (Section 3.2.1)
   ---------
```

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

   01 - Stop processing this SCTP chunk and discard it, do not process
        any further parameters within this chunk, and report the
        unrecognized parameter in an 'Unrecognized Parameter Type' (in
        either an ERROR or in the INIT ACK).

2.2.3. **Solution description**

   It was always the intent to stop processing at the level one was at
   in an unknown chunk or parameter with the upper bit set to 0. Thus
   if you are processing a chunk, you should drop the packet. If you
   are processing a parameter, you should drop the chunk.

2.3. **Padding issues**

2.3.1. **Description of the problem**

   A problem was found in that when a Chunk terminated in a TLV
   parameter. If this last TLV was not on a 32 bit boundary (as
   required), there was confusion as to if the last padding was included
   in the chunk length.

2.3.2. **Text changes to the document**

```
   ---------
   Old text: (Section 3.2)
   ---------
```

   Chunk Length: 16 bits (unsigned integer)

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

   Chunk Value: variable length

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

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

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

   Chunk Length: 16 bits (unsigned integer)

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

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

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

   Chunk Value: variable length

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

      dependent on the Chunk Type.

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

2.3.3. Solution description

   The above text makes clear that the padding of the last parameter is
   not included in the Chunk Length field. It also clarifies that the
   padding of parameters that are not the last one must be counted in
   the Chunk Length field.

2.4. Parameter types across all chunk types

2.4.1. Description of the problem

   A problem was noted when multiple errors are needed to be sent
   regarding unknown or unrecognized parameters. Since often times the
   error type does not hold the chunk type field, it may become
   difficult to tell which error was associated with which chunk.

2.4.2. Text changes to the document

      Old text: (Section 3.2.1)

      The actual SCTP parameters are defined in the specific SCTP chunk
      sections. The rules for IETF-defined parameter extensions are
      defined in Section 13.2.

      New text: (Section 3.2.1)

      The actual SCTP parameters are defined in the specific SCTP chunk
      sections. The rules for IETF-defined parameter extensions are
      defined in Section 13.2. Note that a parameter type MUST be unique
      across all chunks. For example, the parameter type '5' is used to
      represent an IPv4 address (see section 3.3.2). The value '5' then is
      reserved across all chunks to represent an IPv4 address and MUST NOT
      be reused with a different meaning in any other chunk.

```
      ---------
   Old text: (Section 13.2)
      ---------
```

13.2 IETF-defined Chunk Parameter Extension

The assignment of new chunk parameter type codes is done through an
IETF Consensus action as defined in [RFC2434]. Documentation of the
chunk parameter MUST contain the following information:

a) Name of the parameter type.

b) Detailed description of the structure of the parameter field.
   This structure MUST conform to the general type-length-value
   format described in Section 3.2.1.

c) Detailed definition of each component of the parameter type.

d) Detailed description of the intended use of this parameter type,
   and an indication of whether and under what circumstances multiple
   instances of this parameter type may be found within the same
   chunk.

```
      ---------
   New text: (Section 13.2)
      ---------
```

13.2 IETF-defined Chunk Parameter Extension

The assignment of new chunk parameter type codes is done through an
IETF Consensus action as defined in [RFC2434]. Documentation of the
chunk parameter MUST contain the following information:

a) Name of the parameter type.

b) Detailed description of the structure of the parameter field. This
   structure MUST conform to the general type-length-value format
   described in Section 3.2.1.

c) Detailed definition of each component of the parameter type.

d) Detailed description of the intended use of this parameter type,
   and an indication of whether and under what circumstances multiple
   instances of this parameter type may be found within the same
   chunk.

e) Each parameter type MUST be unique across all chunks.

2.4.3. Solution description

By having all parameters unique across all chunk assignments (the
current assignment policy) no ambiguity exists as to what a parameter
means based on context. The trade off for this is a smaller
parameter space i.e. 65,536 parameters versus 65,536 * Number-of-
chunks.

2.5. Stream parameter clarification

2.5.1. Description of the problem

A problem was found where the specification is unclear on the
legality of an endpoint asking for more stream resources than were
allowed in the MIS value of the INIT. In particular the value in the
INIT ACK requested in its OS value was larger than the MIS value
received in the INIT chunk. This behavior is illegal yet it was
unspecified in RFC2960 [7]

2.5.2. Text changes to the document

```
---------
```
Old text: (Section 3.3.3)
```
---------
```

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

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

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

```
---------
```
New text: (Section 3.3.3)
```
---------
```

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

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

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

2.5.3. Solution description

   The change in wording, above, changes it so that a responder to an
   INIT chunk does not specify more streams in its OS value than was
   represented to it in the MIS value i.e. its maximum.

2.6. Restarting association security issue

2.6.1. Description of the problem

   A security problem was found when a restart occurs. It is possible
   for an intruder to send an INIT to an endpoint of an existing
   association. In the INIT the intruder would list one or more of the
   current addresses of an association and its own. The normal restart
   procedures would then occur and the intruder would have hi-jacked an
   association.

2.6.2. Text changes to the document

   Old text: (Section 3.3.10)

      Cause Code
      Value   Cause Code
      ---------   ----------------
       1   Invalid Stream Identifier
       2   Missing Mandatory Parameter
       3   Stale Cookie Error
       4   Out of Resource
       5   Unresolvable Address
       6   Unrecognized Chunk Type
       7   Invalid Mandatory Parameter
       8   Unrecognized Parameters
       9   No User Data
       10   Cookie Received While Shutting Down

   Cause Length: 16 bits (unsigned integer)

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

   Cause-specific Information: variable length

      This field carries the details of the error condition.

   Sections 3.3.10.1 - 3.3.10.10 define error causes for SCTP.

Guidelines for the IETF to define new error cause values are
discussed in Section 13.3.

New text: (Section 3.3.10)

```
   Cause Code
   Value           Cause Code
   ----------      ----------------
    1              Invalid Stream Identifier
    2              Missing Mandatory Parameter
    3              Stale Cookie Error
    4              Out of Resource
    5              Unresolvable Address
    6              Unrecognized Chunk Type
    7              Invalid Mandatory Parameter
    8              Unrecognized Parameters
    9              No User Data
   10              Cookie Received While Shutting Down
   11              Restart of an association with new addresses
```

Cause Length: 16 bits (unsigned integer)

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

Cause-specific Information: variable length

This field carries the details of the error condition.

Sections 3.3.10.1 - 3.3.10.11 define error causes for SCTP.
Guidelines for the IETF to define new error cause values are
discussed in Section 13.3.

New text: (Note no old text, new error cause added in section 3.3.10)

3.3.10.11 Restart of an association with new addresses (11)

 Cause of error

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

```
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |          Cause Code=11          |      Cause Length=Variable    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   /                       New Address TLVs                         /
   \                                                                \
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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

   Old text: (Section 5.2.1)

   Upon receipt of an INIT in the COOKIE-WAIT or COOKIE-ECHOED state, an
   endpoint MUST respond with an INIT ACK using the same parameters it
   sent in its original INIT chunk (including its Initiation Tag,
   unchanged). These original parameters are combined with those from
   the newly received INIT chunk. The endpoint shall also generate a
   State Cookie with the INIT ACK. The endpoint uses the parameters
   sent in its INIT to calculate the State Cookie.

   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 Initiation Tag, unchanged). When
   responding the endpoint MUST send the INIT ACK back to the same
   address that the original INIT (sent by this endpoint) was sent to.

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

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

its INIT to calculate the State Cookie.

    Old text: (Section 5.2.2)

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

    Unless otherwise stated, upon reception of an unexpected INIT for
    this association, the endpoint shall generate an INIT ACK with a
    State Cookie. In the outbound INIT ACK the endpoint MUST copy its
    current Verification Tag and peer's Verification Tag into a reserved
    place within the state cookie. We shall refer to these locations as
    the Peer's-Tie-Tag and the Local-Tie-Tag. The outbound SCTP packet
    containing this INIT ACK MUST carry a Verification Tag value equal to
    the Initiation Tag found in the unexpected INIT. And the INIT ACK
    MUST contain a new Initiation Tag (randomly generated see Section
    5.3.1). Other parameters for the endpoint SHOULD be copied from the
    existing parameters of the association (e.g. number of outbound
    streams) into the INIT ACK and cookie.

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

    Note: Only when a TCB exists and the association is not in a COOKIE-
    WAIT state are the Tie-Tags populated. For a normal association INIT
    (i.e. the endpoint is in a COOKIE-WAIT state), the Tie-Tags MUST be
    set to 0 (indicating that no previous TCB existed). The INIT ACK and
    State Cookie are populated as specified in section 5.2.1.

    New text: (Section 5.2.2)

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

    Unless otherwise stated, upon reception of an unexpected INIT for
    this association, the endpoint shall generate an INIT ACK with a
    State Cookie. Before responding the endpoint MUST check to see if the
    unexpected INIT adds new addresses to the association. If new
    addresses are added to the association, the endpoint MUST respond
    with an ABORT copying the 'Initiation Tag' of the unexpected INIT
    into the 'Verification Tag' of the outbound packet carrying the
    ABORT. In the ABORT response the cause of error MAY be set to
    'restart of an association with new addresses'. The error SHOULD

list the addresses that were added to the restarting association.

If no new addresses are added, when responding to the INIT in the outbound INIT ACK the endpoint MUST copy its current Verification Tag and peer's Verification Tag into a reserved place within the state cookie. We shall refer to these locations as the Peer's-Tie-Tag and the Local-Tie-Tag. The outbound SCTP packet containing this INIT ACK MUST carry a Verification Tag value equal to the Initiation Tag found in the unexpected INIT. And the INIT ACK MUST contain a new Initiation Tag (randomly generated see Section 5.3.1). Other parameters for the endpoint SHOULD be copied from the existing parameters of the association (e.g. number of outbound streams) into the INIT ACK and cookie.

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

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

2.6.3. Solution description

A new error code is being added and specific instructions to send back an ABORT to a new association in a restart case or collision case, where new addresses have been added. The error code can be used by a legitimate restart to inform the endpoint that it has made a software error in adding a new address. The endpoint then can choose to wait until the OOTB ABORT tears down the old association, or restart without the new address.

Also the Note at the end of section 5.2.2 explaining the use of the Tie-Tags was modified to properly explain the states in which the Tie-Tags should be set to a value different than 0.

2.7. Implicit ability to exceed cwnd by PMTU-1 bytes

2.7.1. Description of the problem

Some implementations were having difficulty growing their cwnd. This was due to an improper enforcement of the congestion control rules. The rules, as written, provided for a slop over of the cwnd value. Without this slop over the sender would appear to NOT be using its full cwnd value and thus never increase it.

2.7.2. 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 cwnd or more bytes of data
      outstanding to that transport address. The sender may exceed cwnd
      by up to (PMTU-1) bytes on a new transmission if the cwnd is not
      currently exceeded.

2.7.3. Solution description

   The text changes make clear the ability to go over the cwnd value by
   no more than (PMTU-1) bytes.

2.8. Issues with Fast Retransmit

2.8.1. Description of the problem

   Several problems were found in the current specification of fast
   retransmit. The current wording did not require GAP ACK blocks to be
   sent, even though they are essential to the workings of SCTP's
   congestion control. The specification left unclear how to handle the
   fast retransmit cycle, having the implementation to wait on the cwnd
   to retransmit a TSN that was marked for fast retransmit. No limit
   was placed on how many times a TSN could be fast retransmitted. Fast
   Recovery was not specified, causing the congestion window to be
   reduced drastically when there are multiple losses in a single RTT.

2.8.2. Text changes to the document

   Old text: (Section 6.2)

   Acknowledgments MUST be sent in SACK chunks unless shutdown was
   requested by the ULP in which case an endpoint MAY send an

acknowledgment in the SHUTDOWN chunk. A SACK chunk can acknowledge
the reception of multiple DATA chunks. See Section 3.3.4 for SACK
chunk format. In particular, the SCTP endpoint MUST fill in the
Cumulative TSN Ack field to indicate the latest sequential TSN (of a
valid DATA chunk) it has received. Any received DATA chunks with TSN
greater than the value in the Cumulative TSN Ack field SHOULD also be
reported in the Gap Ack Block fields.

New text: (Section 6.2)

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

Old text: (Section 6.2.1)

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

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

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

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

      New text: (Section 6.2.1)

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

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

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

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

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

      Old text: (Section 7.2.4)

      Whenever an endpoint receives a SACK that indicates some TSN(s)
      missing, it SHOULD wait for 3 further miss indications (via
      subsequent SACK's) on the same TSN(s) before taking action with
      regard to Fast Retransmit.

      When the TSN(s) is reported as missing in the fourth consecutive
      SACK, the data sender shall:

      1) Mark the missing DATA chunk(s) for retransmission,

      2) Adjust the ssthresh and cwnd of the destination address(es) to
         which the missing DATA chunks were last sent, according to the
         formula described in Section 7.2.3.

      3) Determine how many of the earliest (i.e., lowest TSN) DATA chunks
         marked for retransmission will fit into a single packet, subject

        to constraint of the path MTU of the destination transport address
        to which the packet is being sent. Call this value K. Retransmit
        those K DATA chunks in a single packet.

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

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

   A straightforward implementation of the above keeps a counter for
   each TSN hole reported by a SACK. The counter increments for each
   consecutive SACK reporting the TSN hole. After reaching 4 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)

   Whenever an endpoint receives a SACK that indicates some TSN(s)
   missing, it SHOULD wait for 3 further miss indications (via
   subsequent SACK's) on the same TSN(s) before taking action with
   regard to Fast Retransmit.

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

   When the fourth consecutive miss indication is recieved for a TSN(s),
   the data sender shall:

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

   2) If not in Fast Recovery, adjust the ssthresh and cwnd of the

destination address(es) to which the missing DATA chunks were
last sent, according to the formula described in Section 7.2.3.

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

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

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

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

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

2.8.3. Solution description

The effect of the above wording changes are as follows:

o It requires with a MUST the sending of GAP Ack blocks instead of
   the current RFC2960 [7] SHOULD.

o It allows a TSN being Fast Retransmitted (FR) to be sent only once
   via FR.

   o It ends the delay in awaiting for the flight size to drop when a
      TSN is identified ready to FR.

   o It changes the way chunks are marked during fast retransmit, so
      that only new reports are counted.

   o It introduces a Fast Recovery period to avoid multiple congestion
      window reductions when there are multiple losses in a single RTT
      (as shown by Caro et al. [4]).

   These changes will effectively allow SCTP to follow a similar model
   as TCP+SACK in the handling of Fast Retransmit.

2.9. Missing statement about partial_bytes_acked update

2.9.1. Description of the problem

   SCTP uses four control variables to regulate its transmission rate:
   rwnd, cwnd, ssthresh and partial_bytes_acked. Upon detection of
   packet losses from SACK or when the T3-rtx timer expires on an
   address cwnd and ssthresh should be updated as stated in section
   7.2.3. However, that section should also clarify that
   partial_bytes_acked must be updated as well, having to be reset to 0.

2.9.2. **Text changes to the document**

```
---------
Old text: (Section 7.2.3)
---------
```

7.2.3 Congestion Control

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

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

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

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

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

```
---------
New text: (Section 7.2.3)
---------
```

7.2.3 Congestion Control

Upon detection of packet losses from SACK (see Section 7.2.4), an
endpoint should do the following if not in Fast Recovery:

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

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

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

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

2.9.3. **Solution description**

The missing text added solves the doubts about what to do with
partial_bytes_acked in the situations stated in section 7.2.3, making

clear that along with ssthresh and cwnd, partial_bytes_acked should
also be updated, having to be reset to 0.

2.10. Issues with Heartbeating and failure detection

2.10.1. Description of the problem

Five basic problems have been discovered with the current heartbeat
procedures:

o The current specification does not specify that you should count a
   failed heartbeat as an error against the overall association.

o The current specification is un-specific as to when you start
   sending heartbeats and when you should stop.

o The current specification is un-specific as to when you should
   respond to heartbeats.

o When responding to a Heartbeat it is unclear what to do if more
   than a single TLV is present.

o The jitter applied to a heartbeat was meant to be a small variance
   of the RTO and is currently a wide variance due to the default
   delay time and incorrect wording within the RFC.

2.10.2. Text changes to the document

```
---------
Old text: (Section 8.1)
---------

8.1 Endpoint Failure Detection

An endpoint shall keep a counter on the total number of
consecutive retransmissions to its peer (including
retransmissions to all the destination transport addresses
of the peer if it is multi-homed). If the value of this
counter exceeds the limit 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 shall report the failure to the
upper layer, and optionally report back all outstanding user data
remaining in its outbound queue. The association is automatically
closed when the peer endpoint becomes unreachable.

The counter shall be reset each time a DATA chunk sent to that
```

peer endpoint is acknowledged (by the reception of a SACK), or a
HEARTBEAT-ACK is received from the peer endpoint.

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

8.1 Endpoint Failure Detection

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

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

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

8.3 Path Heartbeat

By default, an SCTP endpoint shall 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).

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

8.3 Path Heartbeat

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

transport address(es). HEARTBEAT sending MAY begin upon
reaching the ESTABLISHED state, and is discontinued after
sending either SHUTDOWN or SHUTDOWN-ACK. A receiver of a
HEARTBEAT MUST respond to a HEARTBEAT with a HEARTBEAT-ACK
after entering the COOKIE-ECHOED state (INIT sender) or the
ESTABLISHED state (INIT receiver), up until reaching the
SHUTDOWN-SENT state (SHUTDOWN sender) or the
SHUTDOWN-ACK-SENT state (SHUTDOWN receiver).

Old text: (Section 8.3)

The receiver of the HEARTBEAT should immediately respond with a
HEARTBEAT ACK that contains the Heartbeat Information field
copied from the received HEARTBEAT chunk.

New text: (Section 8.3)

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

Old text: (Section 8.3)

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

New text: (Section 8.3)

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

2.10.3. Solution description

   The above text provides guidance as to how to respond to the five
   issues mentioned in Section 2.10.1 In particular the wording changes
   provide guidance as to when to start and stop heartbeating, how to
   respond to a heartbeat with extra parameters, and clarifies the error
   counting procedures for the association.

2.11. Security interactions with firewalls

2.11.1. Description of the problem

   When dealing with firewalls it is advantageous to the firewall to be
   able to properly determine the initial startup sequence of a reliable
   transport protocol. With this in mind the following text is to be
   added to SCTP's security section.

2.11.2. Text changes to the document

   New text: (no old text, new section added)

   11.4 SCTP interactions with firewalls

   It is helpful for some firewalls if they can inspect
   just the first fragment of a fragmented SCTP packet and unambiguously
   determine whether it corresponds to an INIT chunk (for further
   information please refer to RFC1858). Accordingly, we
   stress the requirements stated in 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.

   Old text: (Section 18)
   ----

   18. Bibliography

   [ALLMAN99] Allman, M. and Paxson, V., "On Estimating End-to-End
              Network Path Properties", Proc. SIGCOMM'99, 1999.

   [FALL96]   Fall, K. and Floyd, S., Simulation-based Comparisons of
              Tahoe, Reno, and SACK TCP, Computer Communications Review,
              V. 26 N. 3, July 1996, pp. 5-21.

   [RFC1750]   Eastlake, D. (ed.), "Randomness Recommendations for
               Security", RFC 1750, December 1994.

   [RFC1950]   Deutsch P. and J. Gailly, "ZLIB Compressed Data Format
               Specification version 3.3", RFC 1950, May 1996.

   [RFC2104]   Krawczyk, H., Bellare, M. and R. Canetti, "HMAC: Keyed-
               Hashing for Message Authentication", RFC 2104, March 1997.

   [RFC2196]   Fraser, B., "Site Security Handbook", FYI 8, RFC 2196,
               September 1997.

   [RFC2522]   Karn, P. and W. Simpson, "Photuris: Session-Key Management
               Protocol", RFC 2522, March 1999.

   [SAVAGE99] Savage, S., Cardwell, N., Wetherall, D., and Anderson, T.,
               "TCP Congestion Control with a Misbehaving Receiver", ACM
               Computer Communication Review, 29(5), October 1999.

   New text: (Section 18)
   ----

18. Bibliography

   [ALLMAN99] Allman, M. and Paxson, V., "On Estimating End-to-End
               Network Path Properties", Proc. SIGCOMM'99, 1999.

   [FALL96]   Fall, K. and Floyd, S., Simulation-based Comparisons of
               Tahoe, Reno, and SACK TCP, Computer Communications Review,
               V. 26 N. 3, July 1996, pp. 5-21.

   [RFC1750]   Eastlake, D. (ed.), "Randomness Recommendations for
               Security", RFC 1750, December 1994.

   [RFC1858]   Ziemba, G., Reed, D. and Traina P., "Security
               Considerations for IP Fragment Filtering", RFC 1858,
               October 1995.

   [RFC1950]   Deutsch P. and J. Gailly, "ZLIB Compressed Data Format
               Specification version 3.3", RFC 1950, May 1996.

   [RFC2104]   Krawczyk, H., Bellare, M. and R. Canetti, "HMAC: Keyed-
               Hashing for Message Authentication", RFC 2104, March 1997.

   [RFC2196]   Fraser, B., "Site Security Handbook", FYI 8, RFC 2196,
               September 1997.

   [RFC2522] Karn, P. and W. Simpson, "Photuris: Session-Key Management
              Protocol", RFC 2522, March 1999.

   [SAVAGE99] Savage, S., Cardwell, N., Wetherall, D., and Anderson, T.,
              "TCP Congestion Control with a Misbehaving Receiver", ACM
              Computer Communication Review, 29(5), October 1999.

2.11.3. Solution description

   The above text adding a new subsection to the Security Considerations
   section of RFC2960 [7] makes clear that, to make easier the
   interaction with firewalls, an INIT chunk must not be bundled in any
   case with any other chunk, being this rule enforced by the packet
   receiver, that will silently discard the packets that do not follow
   this rule.

2.12. Shutdown ambiguity

2.12.1. Description of the problem

   Currently there is an ambiguity between the statements in section 6.2
   and section 9.2. Section 6.2 allows the sending of a SHUTDOWN chunk
   in place of a SACK when the sender is in the process of shutting
   down, while section 9.2 requires both a SHUTDOWN chunk and a SACK
   chunk to be sent.

   Along with this ambiguity there is a problem where in an errant
   SHUTDOWN receiver may fail to stop accepting user data.

2.12.2. Text changes to the document

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

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

While in SHUTDOWN-SENT state, the SHUTDOWN sender MUST immediately
respond to each received packet containing one or more DATA chunk(s)
with a SACK, a SHUTDOWN chunk, and restart the T2-shutdown timer. If
it has no more outstanding DATA chunks, the SHUTDOWN receiver shall
send a SHUTDOWN ACK and start a T2-shutdown timer of its own,
entering the SHUTDOWN-ACK-SENT state. If the timer expires, the
endpoint must re-send the SHUTDOWN ACK.

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

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

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

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

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

2.12.3. Solution description

   The above text clarifies the use of a SACK in conjunction with a
   SHUTDOWN chunk. It also adds a guard timer to the SCTP shutdown
   sequence to protect against errant receivers of SHUTDOWN chunks.

2.13. Inconsistency in ABORT processing

2.13.1. Description of the problem

   It was noted that the wording in section 8.5.1 did not give proper
   directions in the use of the 'T bit' with the verification tags.

2.13.2. Text changes to the document

```
---------
Old text: (Section 8.5.1)
---------
```

B) Rules for packet carrying ABORT:

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

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

   - The receiver MUST accept the packet if the Verification Tag
      matches either its own tag, OR the tag of its peer. Otherwise,
      the receiver MUST silently discard the packet and take no
      further action.

```
---------
New text: (Section 8.5.1)
---------
```

B) Rules for packet carrying ABORT:

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

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

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

2.13.3. Solution description

The above text change clarifies that the T bit must be set before an
implementation looks for the peers tag.

2.14. Cwnd gated by its full use

2.14.1. Description of the problem

   A problem was found with the current specification of the growth and
   decay of cwnd. The cwnd should only be increased if it is being
   fully utilized, and after periods of under utilization, the cwnd
   should be decreased. In some sections, the current wording is weak
   and is not clearly defined. Also, the current specification
   unnecessarily introduces the need for special case code to ensure
   cwnd degradation. Plus, the cwnd should not be increased during Fast
   Recovery since a full cwnd during Fast Recovery does not qualify the
   cwnd as being fully utilized. Additionally, multiple loss scenarios
   in a single window may cause the cwnd to grow more rapidly as the
   number of losses in a window increases [4].

2.14.2. Text changes to the document

   Old text: (Section 6.1)

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

   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 as follows:

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

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

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

   Old text: (Section 7.2.1)
   ----

   o When cwnd is less than or equal to ssthresh an SCTP endpoint MUST

   use the slow start algorithm to increase cwnd (assuming the
   current congestion window is being fully utilized). If an
   incoming SACK advances the Cumulative TSN Ack Point, cwnd MUST be
   increased by at most the lesser of 1) the total size of the
   previously outstanding DATA chunk(s) acknowledged, and 2) the
   destination's path MTU. This protects against the ACK-Splitting
   attack outlined in [SAVAGE99].

   New text: (Section 7.2.1)
   ----

o When cwnd is less than or equal to ssthresh an SCTP endpoint MUST
   use the slow start algorithm to increase cwnd only if the current
   congestion window is being fully utilized, an incoming SACK
   advances the Cumulative TSN Ack Point, and the data sender is not
   in Fast Recovery. Only when these three conditions are met, can
   the cwnd be increased; otherwise the cwnd MUST not be increased.
   If these conditions are met then cwnd MUST be increased by at
   most the lesser of 1) the total size of the previously outstanding
   DATA chunk(s) acknowledged, and 2) the destination's path MTU.
   This upper bound protects against the ACK-Splitting attack
   outlined in [SAVAGE99].

   Old text: (Section 14)
   ----

14. Suggested SCTP Protocol Parameter Values

The following protocol parameters are RECOMMENDED:

RTO.Initial   - 3 seconds
RTO.Min   - 1 second
RTO.Max   - 60 seconds
RTO.Alpha   - 1/8
RTO.Beta   - 1/4
Valid.Cookie.Life   - 60   seconds
Association.Max.Retrans   - 10 attempts
Path.Max.Retrans   - 5 attempts (per destination address)
Max.Init.Retransmits   - 8 attempts
HB.interval   - 30 seconds

   New text: (Section 14)
   ----

14. Suggested SCTP Protocol Parameter Values

The following protocol parameters are RECOMMENDED:

```
RTO.Initial              - 3  seconds
RTO.Min                  - 1  second
RTO.Max                  - 60 seconds
Max.Burst                - 4
RTO.Alpha                - 1/8
RTO.Beta                 - 1/4
Valid.Cookie.Life        - 60 seconds
Association.Max.Retrans  - 10 attempts
Path.Max.Retrans         - 5  attempts (per destination address)
Max.Init.Retransmits     - 8  attempts
HB.Interval              - 30 seconds
```

2.14.3. Solution description

The above changes strengthens the rules and makes it much more
apparent as to the need to block cwnd growth when the full cwnd is
not being utilized. The changes also applies cwnd degradation
without introducing the need for complex special case code.

2.15. Window probes in SCTP

2.15.1. Description of the problem

When a receiver clamps its rwnd to 0 to flow control the peer, the
specification implies that one must continue to accept data from the
remote peer. This is incorrect and needs clarification.

2.15.2. Text changes to the document

```
---------
```
Old text: (Section 6.2)
```
---------
```

The SCTP endpoint MUST always acknowledge the reception of each valid
DATA chunk.

```
---------
```
New text: (Section 6.2)
```
---------
```

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

When the receiver's advertised window is 0, the receiver MUST drop
any new incoming DATA chunk with a TSN larger than the largest TSN
received so far. If the new incoming DATA chunk holds a TSN value
less than the largest TSN received so far, then the receiver SHOULD
drop the largest TSN held for reordering, and accept the new
incoming DATA chunk. In either case, if such a DATA chunk is dropped,
the receiver MUST immediately send back a SACK with the current
receive window showing only DATA chunks received and accepted so
far. The dropped DATA chunk(s) MUST NOT be included in the SACK as
they were not accepted. The receiver MUST also have an algorithm
for advertising its receive window to avoid receiver silly window
syndrome (SWS) as described in RFC 813. The algorithm can be
similar to the one described in Section 4.2.3.3 of RFC 1122.

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 having been lost in transit from the data
   receiver to the data sender.

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 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 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 chunk(s) are in
   flight. Zero window probing MUST be supported.

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

The sender MUST also have an algorithm for sending new DATA chunks
to avoid silly window syndrome (SWS) as described in RFC 813. The
algorithm can be similar to the one described in Section 4.2.3.4
of RFC 1122.

2.15.3. Solution description

The above allows a receiver to drop new data that arrives and yet
still requires the receiver to send a SACK showing the conditions
unchanged (with the possible exception of a new a_rwnd) and the
dropped chunk as missing. This will allow the association to
continue until the rwnd condition clears.

2.16. Fragmentation and Path MTU issues

2.16.1. Description of the problem

The current wording of the Fragmentation and Reassembly forces an
implementation that supports fragmentation to always fragment. This
prohibits an implementation from offering its users an option to
disable sends that exceed the SCTP fragmentation point.

The restriction in RFC2960 [7] section 6.9 was never meant to
restrict an implementations API from this behavior.

2.16.2. **Text changes to the document**

    Old text: (Section 6.1)

    6.9 Fragmentation and Reassembly

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

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

    New text: (Section 6.1)

    6.9 Fragmentation and Reassembly

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

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

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

2.16.3. **Solution description**

   The above wording will allow an implementation to offer the option of
   rejecting sends that exceed the P-MTU size even when the
   implementation supports fragmentation.

2.17. **Initial value of the cumulative TSN Ack**

2.17.1. **Description of the problem**

   The current description of the SACK chunk within the RFC does not
   clearly state the value that would be put within a SACK when no DATA
   chunk has been received.

2.17.2. **Text changes to the document**

   Old text: (Section 3.3.4)

   Cumulative TSN Ack: 32 bits (unsigned integer)

      This parameter contains the TSN of the last DATA chunk received in
      sequence before a gap.

   New text: (Section 3.3.4)

   Cumulative TSN Ack: 32 bits (unsigned integer)

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

2.17.3. **Solution description**

   This change clearly states what the initial value will be for a SACK
   sender.

2.18. **Handling of address parameters within the INIT or INIT-ACK**

2.18.1. **Description of the problem**

   The current description on handling address parameters contained
   within the INIT and INIT-ACK do not fully describe a requirement for

their handling.

2.18.2. Text changes to the document

```
---------
Old text: (Section 5.1.2)
---------
```

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

```
---------
New text: (Section 5.1.2)
---------
```

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

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

2.18.3. Solution description

This new text clearly specifies to an implementor the need to look
within the INIT or INIT ACK. Any implementation that does not do
this, may for example not be able to recognize an INIT chunk coming
from an already established association that adds new addresses (see

section 2.6), or an incoming INIT ACK chunk sent from a source
address different than the destination address used to send the INIT
chunk.

2.19. Handling of stream shortages

2.19.1. Description of the problem

The current wording in the RFC places the choice of sending an ABORT
upon the SCTP stack when a stream shortage occurs. This decision
should really be made by the upper layer not the SCTP stack.

2.19.2. **Text changes to the document**

    Old text:

    5.1.1 Handle Stream Parameters

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

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

    New text: (Section 5.1.2)

    5.1.1 Handle Stream Parameters

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

    After receiving the stream configuration information from the other
    side, each endpoint MUST perform the following check: If the peer's
    MIS is less than the endpoint's OS, meaning that the peer is
    incapable of supporting all the outbound streams the endpoint wants
    to configure, the endpoint MUST use MIS outbound streams and MAY
    report any shortage to the upper layer. The upper layer can then
    choose to abort the association if the resource shortage
    is unacceptable.

2.19.3. **Solution description**

    The above changes take the decision to ABORT out of the realm of the
    SCTP stack and places it into the users hands.

2.20. Indefinite postponement

2.20.1. Description of the problem

   The current RFC does not provide any guidance on the assignment of
   TSN sequence numbers to outbound message nor reception of these
   message. This could lead to a possible indefinite postponement.

2.20.2. 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.

   6.2 Acknowledgment on Reception of DATA Chunks

   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.

   The algorithm by which an implementation assigns sequential TSNs to
   messages on a particular association MUST ensure that no user
   message that has been accepted by SCTP is indefinitely postponed
   from being assigned a TSN. Acceptable algorithms for assigning TSNs
   include

   (a) assigning TSNs in round-robin order over all streams with
       pending data

   (b) preserving the linear order in which the user messages were
       submitted to the SCTP association.

   When an upper layer requests to read data on an SCTP association,
   the SCTP receiver SHOULD choose the message with the lowest TSN from
   among all deliverable messages. In SCTP implementations that allow a
   user to request data on a specific stream, this operation SHOULD NOT
   block if data is not available, since this can lead to a deadlock
   under certain conditions.

   6.2 Acknowledgment on Reception of DATA Chunks

2.20.3. Solution description

   The above wording clarifies how TSNs SHOULD be assigned by the
   sender.

2.21. User initiated abort of an association

2.21.1. Description of the problem

   It is not possible for an upper layer to abort the association and
   provide the peer with an indication why the association is aborted.

2.21.2. Text changes to the document

   Some of the changes given here already include changes suggested in
   section Section 2.6 of this document.

   Old text: (Section 3.3.10)

```
   Cause Code
   Value           Cause Code
   ---------       ----------------
    1               Invalid Stream Identifier
    2               Missing Mandatory Parameter
    3               Stale Cookie Error
    4               Out of Resource
    5               Unresolvable Address
    6               Unrecognized Chunk Type
    7               Invalid Mandatory Parameter
    8               Unrecognized Parameters
    9               No User Data
    10              Cookie Received While Shutting Down
```

   Cause Length: 16 bits (unsigned integer)

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

   Cause-specific Information: variable length

      This field carries the details of the error condition.

   Sections 3.3.10.1 - 3.3.10.10 define error causes for SCTP.
   Guidelines for the IETF to define new error cause values are
   discussed in Section 13.3.

   New text: ([Section 3.3.10](#))

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

   Cause Length: 16 bits (unsigned integer)

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

   Cause-specific Information: variable length

      This field carries the details of the error condition.

   Sections [3.3.10.1](#) - [3.3.10.12](#) define error causes for SCTP.
   Guidelines for the IETF to define new error cause values are
   discussed in [Section 13.3](#).

   New text: (Note no old text, new error added in [section 3.3.10](#))

   3.3.10.12 User Initiated Abort (12)

    Cause of error

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

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Cause Code=12         |      Cause Length=Variable    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                    Upper Layer Abort Reason                   /
\\                                                             \\
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Old text: ([Section 9.1](#))

9.1 Abort of an Association

   When an endpoint decides to abort an existing association,
   it shall send an ABORT chunk to its peer endpoint. The
   sender MUST fill in the peer's Verification Tag in the
   outbound packet and MUST NOT bundle any DATA chunk
   with the ABORT.

   An endpoint MUST NOT respond to any received packet that contains
   an ABORT chunk (also see [Section 8.4](#)).

   An endpoint receiving an ABORT shall apply the special
   Verification Tag check rules described in [Section 8.5.1](#).

   After checking the Verification Tag, the receiving endpoint shall
   remove the association from its record, and shall report the
   termination to its upper layer.

New text: ([Section 9.1](#))

9.1 Abort of an Association

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

   An endpoint MUST NOT respond to any received packet that contains
   an ABORT chunk (also see [Section 8.4](#)).

   An endpoint receiving an ABORT MUST apply the special Verification
   Tag check rules described in [Section 8.5.1](#).

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

```
---------
```
Old text: (Section 10.1)
```
---------
```

D) Abort

Format: ABORT(association id [, cause code])
-> result

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

Mandatory attributes:

o association id - local handle to the SCTP association

Optional attributes:

o cause code - reason of the abort to be passed to the peer.


```
---------
```
New text: (Section 10.1)
```
---------
```

D) Abort

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

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

Mandatory attributes:

o association id - local handle to the SCTP association

   Optional attributes:

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

   None.

   Old text: ([Section 10.2](#))

   E) COMMUNICATION LOST notification

   When SCTP loses communication to an endpoint completely (e.g., via
   Heartbeats) or detects that the endpoint has performed an abort
   operation, it shall invoke this notification on the ULP.

   The following shall be passed with the notification:

   o association id - local handle to the SCTP association

   o status - This indicates what type of event has occurred; The
              status may indicate a failure OR a normal termination
              event occurred in response to a shutdown or abort
              request.

   The following may be passed with the notification:

   o data retrieval id - an identification used to retrieve
      unsent and unacknowledged data.

   o last-acked - the TSN last acked by that peer endpoint;

   o last-sent - the TSN last sent to that peer endpoint;

   New text: ([Section 10.2](#))

   E) COMMUNICATION LOST notification

   When SCTP loses communication to an endpoint completely (e.g., via
   Heartbeats) or detects that the endpoint has performed an abort
   operation, it shall invoke this notification on the ULP.

   The following shall be passed with the notification:

   o association id - local handle to the SCTP association

   o status - This indicates what type of event has occurred; The
              status may indicate a failure OR a normal termination
              event occurred in response to a shutdown or
              abort request.

   The following may be passed with the notification:

   o data retrieval id - an identification used to retrieve unsent
      and unacknowledged data.

   o last-acked - the TSN last acked by that peer endpoint;

   o last-sent - the TSN last sent to that peer endpoint;

   o Upper Layer Abort Reason - the abort reason specified if
                                 case of an user initiated abort.

2.21.3. Solution description

   The above allows an upper layer to provide its peer with an
   indication why the association was aborted. Therefore an addition
   error cause was introduced.

2.22. Handling of invalid Initiate Tag of INIT-ACK

2.22.1. Description of the problem

   RFC 2960 requires that the receiver of an INIT-ACK with the Initiate
   Tag set to zero handles this as an error and sends back an ABORT.
   But the sender of the INIT-ACK normally has no TCB and so the ABORT
   is useless.

2.22.2. **Text changes to the document**

   Old text: (Section 3.3.3)

      Initiate Tag: 32 bits (unsigned integer)

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

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

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

   New text: (Section 3.3.3)

      Initiate Tag: 32 bits (unsigned integer)

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

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

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

2.22.3. **Solution description**

   The new text does not require the receiver of the invalid INIT-ACK to
   send the ABORT. This behavior is in tune with the error case of
   invalid stream numbers in the INIT-ACK. However it is allowed to
   send an ABORT for debugging purposes.

2.23. **ABORT sending in response to an INIT**

2.23.1. **Description of the problem**

   Whenever the receiver of an INIT chunk has to send an ABORT chunk in
   response for whatever reason it is not stated clearly which
   Verification Tag and value of the T-bit should be used.

2.23.2. **Text changes to the document**

   Old text: (Section 8.4)
   ----

      3) If the packet contains an INIT chunk with a Verification Tag
         set to '0', process it as described in Section 5.1.
         Otherwise,

   New text: (Section 8.4)
   ----

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

2.23.3. **Solution description**

   The new text stated clearly which value of the Verification Tag and
   T-bit have to be used.

2.24. **Stream Sequence Number (SSN) Initialization**

2.24.1. **Description of the problem**

   RFC 2960 does not describe the fact that the SSN have to be
   initialized to 0 in the way it is required by RFC2119.

2.24.2. **Text changes to the document**

      Old text: (Section 6.5)

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

      New text: (Section 6.5)

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

2.24.3. **Solution description**

   The 'shall' in the text is replaced by a 'MUST' to clearly state the
   required behavior.

2.25. **SACK packet format**

2.25.1. **Description of the problem**

   It is not clear in RFC 2960 whether a SACK must contain the fields
   Number of Gap Ack Blocks and Number of Duplicate TSNs or not.

2.25.2. **Text changes to the document**

```
   ---------
   Old text: (Section 3.3.4)
   ---------
```

```
       The SACK MUST contain the Cumulative TSN Ack and
       Advertised Receiver Window Credit (a_rwnd) parameters.
```

```
   ---------
   New text: (Section 3.3.4)
   ---------
```

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

2.25.3. **Solution description**

The text has been modified. It is now clear that a SACK always contains the fields Number of Gap Ack Blocks and Number of Duplicate TSNs.

2.26. **Protocol Violation Error Cause**

2.26.1. **Description of the problem**

There are many situations where an SCTP endpoint may detect that its peer violates the protocol. The result of such detection often results in the association being destroyed by the sending of an ABORT. Currently there are only some error causes which could be used to indicate the reason of the abort but these do not cover all cases.

2.26.2. **Text changes to the document**

Some of the changes given here already include changes suggested in section Section 2.6 and Section 2.21 of this document.

```
   ---------
   Old text: (Section 3.3.10)
   ---------
```

```
       Cause Code
       Value           Cause Code
       ---------       ----------------
```

         1   Invalid Stream Identifier
         2   Missing Mandatory Parameter
         3   Stale Cookie Error
         4   Out of Resource
         5   Unresolvable Address
         6   Unrecognized Chunk Type
         7   Invalid Mandatory Parameter
         8   Unrecognized Parameters
         9   No User Data
         10   Cookie Received While Shutting Down

   Cause Length: 16 bits (unsigned integer)

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

   Cause-specific Information: variable length

      This field carries the details of the error condition.

   Sections 3.3.10.1 - 3.3.10.10 define error causes for SCTP.
   Guidelines for the IETF to define new error cause values are
   discussed in Section 13.3.

   New text: (Section 3.3.10)

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

   Cause Length: 16 bits (unsigned integer)

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

   Cause-specific Information: variable length

      This field carries the details of the error condition.

   Sections 3.3.10.1 - 3.3.10.13 define error causes for SCTP.
   Guidelines for the IETF to define new error cause values are
   discussed in Section 13.3.

   New text: (Note no old text, new error added in section 3.3.10)

   3.3.10.13 Protocol Violation (13)

    Cause of error

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

       +-+
       | Cause Code=13 | Cause Length=Variable |
       +-+
       /   Additional Information   /
       \\   \\
       +-+

2.26.3. Solution description

   An additional error cause which can be used by an endpoint to
   indicate a protocol violation of the peer has been defined.

2.27. Reporting of Unrecognized Parameters

2.27.1. Description of the problem

   It is not stated clearly in RFC2960 [7] how unrecognized parameters
   should be reported. Unrecognized parameters in an INIT chunk could
   be reported in the INIT-ACK chunk or in a separate ERROR chunk which
   can get lost. Unrecognized parameters in an INIT-ACK chunk have to

be reported in an ERROR-chunk. This can be bundled with the COOKIE-
ERROR chunk or sent separately. If it is sent separately and
received before the COOKIE-ECHO it will be handled as an OOTB packet
resulting in sending out an ABORT chunk. Therefore the association
would not be established.

2.27.2. Text changes to the document

Some of the changes given here already include changes suggested in
section Section 2.2 of this document.

```
---------
Old text: (Section 3.2.1)
---------
```

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
     parameter in an 'Unrecognized Parameter Type' (in either an
     ERROR or in the INIT ACK).

10 - Skip this parameter and continue processing.

11 - Skip this parameter and continue processing but report the
     unrecognized parameter in an 'Unrecognized Parameter Type' (in
     either an ERROR or in the INIT ACK).

```
---------
New text: (Section 3.2.1)
---------
```

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

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

10 - Skip this parameter and continue processing.

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

      New text: (Note no old text, clarification added in section 3.2)

      3.2.2 Reporting of Unrecognized Parameters

         If the receiver of an INIT chunk detects unrecognized parameters
         and has to report them according to section 3.2.1 it MUST put
         the 'Unrecognized Parameter' parameter(s) in the INIT-ACK chunk
         sent in response to the INIT-chunk. Note that if the receiver
         of the INIT chunk is NOT going to establish an association (e.g.
         due to lack of resources) then no report would be sent back.

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

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

2.27.3. Solution description

   The procedure of reporting unrecognized parameters has been described
   clearly.

2.28. Handling of IP Address Parameters

2.28.1. Description of the problem

   It is not stated clearly in RFC2960 [7] how a SCTP endpoint which
   supports either IPv4 addresses or IPv6 addresses should respond if
   IPv4 and IPv6 addresses are presented by the peer in the INIT or
   INIT-ACK chunk.

2.28.2. **Text changes to the document**

```
   ---------
   Old text: (Section 5.1.2)
   ---------
```

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

```
   ---------
   New text: (Section 5.1.2)
   ---------
```

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

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

2.28.3. **Solution description**

   The procedure of handling IP address parameters has been described
   clearly.

2.29. **Handling of** COOKIE ECHO chunks when a TCB exists

2.29.1. **Description of the problem**

   The description of the behavior in RFC2960 [7] when a COOKIE ECHO
   chunk and a TCB exists could be misunderstood. When a COOKIE ECHO is
   received, a TCB exist and the local and peer's tag match it is stated
   that the endpoint should enter the ESTABLISHED state if it has not
   already done so and send a COOKIE ACK. It was not clear that in case
   the endpoint has already left again the ESTABLISHED state then it
   should not go back to established. In case D the endpoint can only
   enter state ESTABLISHED from COOKIE-ECHOED because in state CLOSED it

has no TCB and in state COOKIE-WAIT it has a TCB but knows nothing
about the peer's tag which is requested to match in this case.

2.29.2. Text changes to the document

```
---------
```
Old text: (Section 5.2.4)
```
---------
```
```
    D) When both local and remote tags match the endpoint should
       always enter the ESTABLISHED state, if it has not already
       done so. It should stop any init or cookie timers that may
       be running and send a COOKIE ACK.
```

```
---------
```
New text: (Section 5.2.4)
```
---------
```
```
    D) When both local and remote tags match the endpoint should
       enter the ESTABLISHED state, if it is in the COOKIE-ECHOED
       state. It should stop any cookie timer that may
       be running and send a COOKIE ACK.
```

2.29.3. Solution description

The procedure of handling of COOKIE-ECHO chunks when a TCB exists has
been described clearly.

2.30. The Initial Congestion Window Size

2.30.1. Description of the problem

RFC2960 was published with the intention of having the same
congestion control properties as TCP. Since the publication of
RFC2960, TCP's initial congestion window size as been increased via
RFC3390. This same update will be needed for SCTP to keep SCTP's
congestion control properties equivilant to that of TCP.

2.30.2. 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 <= 2*MTU.
```

    New 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 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(cwnd/2, 2*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(cwnd/2, 4*MTU) per RTO.

    Old text: (Section 7.2.2)
         ----

       o Same as in the slow start, when the sender does not transmit
          DATA on a given transport address, the cwnd of the transport
          address should be adjusted to max(cwnd / 2, 4*MTU) per RTO.

    New text: (Section 7.2.2)
         ----

       o Same as in the slow start, when the sender does not transmit
          DATA on a given transport address, the cwnd of the transport
          address should be adjusted to max(cwnd / 2, 4*MTU) per RTO.

    Old text: (Section 7.2.3)

    7.2.3 Congestion Control

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

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

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

       When the T3-rtx timer expires on an address, SCTP should perform

      slow start by:

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

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

    7.2.3 Congestion Control

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

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

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

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

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

[2.30.3](#). Solution description

   The change to SCTP's initial congestion window will allow it to
   continue to maintain the same congestion control properties as TCP.

[2.31](#). Stream Sequence Numbers in Figures

[2.31.1](#). Description of the problem

   In [Section 2.24](#) of this document it is clarified that the SSN are
   initialized with 0. Two figures in [RFC2960](#) [[7](#)] illustrate that they
   start with 1.

[2.31.2](#). Text changes to the document

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

    Endpoint A   Endpoint Z
     {app sets association with Z}

```
    (build TCB)
    INIT [I-Tag=Tag_A
          & other info]  ------\
    (Start T1-init timer)       \
    (Enter COOKIE-WAIT state)    \---> (compose temp TCB and Cookie_Z)
                                  /-- INIT ACK [Veri Tag=Tag_A,
                                 /            I-Tag=Tag_Z,
    (Cancel T1-init timer) <-----/             Cookie_Z, & other info]
                                      (destroy temp TCB)

    COOKIE ECHO [Cookie_Z] ------\
    (Start T1-init timer)        \
    (Enter COOKIE-ECHOED state)   \---> (build TCB enter ESTABLISHED
                                           state)
                                  /---- COOKIE-ACK
                                 /
   (Cancel T1-init timer, <-----/
    Enter ESTABLISHED state)
   {app sends 1st user data; strm 0}
    DATA [TSN=initial TSN_A
        Strm=0,Seq=1 & user data]--\
     (Start T3-rtx timer)           \
                                      \->
                                  /----- SACK [TSN Ack=init
                                 /            TSN_A,Block=0]
    (Cancel T3-rtx timer) <------/

                                      ...
                                   {app sends 2 messages;strm 0}
                                  /---- DATA
                                 /     [TSN=init TSN_Z
                             <--/       Strm=0,Seq=1 & user data 1]
    SACK [TSN Ack=init TSN_Z,     /   ---- DATA
          Block=0]     --------\  /      [TSN=init TSN_Z +1,
                               \/        Strm=0,Seq=2 & user data 2]
                        <------/\
                                \
                                 \------>
```

                   Figure 4: INITiation Example

    New text: (Section 7.2.1)

    Endpoint A   Endpoint Z
    {app sets association with Z}
    (build TCB)

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

                     Figure 4: INITiation Example


   ---------
   Old text: (Section 5.2.4.1)
   ---------

   Endpoint A                                          Endpoint Z
   <------------ Association is established--------------------->
   Tag=Tag_A                                           Tag=Tag_Z
   <----------------------------------------------------------->
   {A crashes and restarts}
```

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

                   Figure 5: A Restart Example

     New text: ([Section 5.2.4.1](#))

```
     Endpoint A                                       Endpoint Z
     <-------------- Association is established--------------------->
     Tag=Tag_A                                         Tag=Tag_Z
     <------------------------------------------------------------->
```

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

                     Figure 5: A Restart Example

2.31.3. Solution description

   Figure 4 and figure 5 were changed such that the SSN start with 0
   instead of 1.

2.32. **Unrecognized Parameters**

2.32.1. **Description of the problem**

   The RFC does not state clearly in section 3.3.3.1 if one or multiple
   unrecognized parameters are included in the 'Unrecognized Parameter'
   parameter.

2.32.2. **Text changes to the document**

```
   ---------
   Old text: (Section 3.3.3)
   ---------
         Variable Parameters                 Status    Type Value
         -------------------------------------------------------------
         State Cookie                        Mandatory  7
         IPv4 Address (Note 1)               Optional   5
         IPv6 Address (Note 1)               Optional   6
         Unrecognized Parameters             Optional   8
         Reserved for ECN Capable (Note 2)   Optional   32768 (0x8000)
         Host Name Address (Note 3)          Optional   11


   ---------
   New text: (Section 3.3.3)
   ---------
         Variable Parameters                 Status    Type Value
         -------------------------------------------------------------
         State Cookie                        Mandatory  7
         IPv4 Address (Note 1)               Optional   5
         IPv6 Address (Note 1)               Optional   6
         Unrecognized Parameter              Optional   8
         Reserved for ECN Capable (Note 2)   Optional   32768 (0x8000)
         Host Name Address (Note 3)          Optional   11


   ---------
   Old text: (Section 3.3.3.1)
   ---------
      Unrecognized Parameters:

         Parameter Type Value: 8

         Parameter Length:  Variable Size.

         Parameter Value:
            This parameter is returned to the originator of the INIT
            chunk when the INIT contains an unrecognized parameter
            which has a value that indicates that it should be reported
            to the sender. This parameter value field will contain
```

          unrecognized parameters copied from the INIT chunk complete
          with Parameter Type, Length and Value fields.

   New text: (Section 3.3.3.1)

      Unrecognized Parameter:

          Parameter Type Value: 8

          Parameter Length: Variable Size.

          Parameter Value:

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

2.32.3. Solution description

   The new text states clearly that only one unrecognized parameter is
   reported per parameter.

2.33. Handling of unrecognized parameters

2.33.1. Description of the problem

   It is not stated clearly in RFC2960 [7] how unrecognized parameters
   should be handled. The problem came up when an INIT contains an
   unrecognized parameter with highest bits 00. It was not clear if an
   INIT-ACK should be sent or not.

2.33.2. Text changes to the document

   Some of the changes given here already include changes suggested in
   section Section 2.27 of this document.

   Old text: (Section 3.2.1)
   ----

   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
        parameter in an 'Unrecognized Parameter Type' (in either an
        ERROR or in the INIT ACK).

   10 - Skip this parameter and continue processing.

   11 - Skip this parameter and continue processing but report the
        unrecognized parameter in an 'Unrecognized Parameter Type' (in
        either an ERROR or in the INIT ACK).

   New text: (Section 3.2.1)
   ----

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

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

   10 - Skip this parameter and continue processing.

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

   New text: (Note no old text, clarification added in section 3.2)

   3.2.2 Reporting of Unrecognized Parameters

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

      If the receiver of an INIT-ACK chunk detects unrecognized
      parameters and has to report them according to section 3.2.1 it
      SHOULD bundle the ERROR chunk containing the 'Unrecognized
      Parameters' error cause with the COOKIE-ECHO chunk sent in
      response to the INIT-ACK chunk. If the receiver of the

INIT-ACK can not bundle the COOKIE-ECHO chunk with the ERROR
chunk the ERROR chunk MAY be sent separately but not
before the COOKIE-ACK has been received.

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

2.33.3. Solution description

The procedure of handling unrecognized parameters has been described
clearly.

2.34. Tie Tags

2.34.1. Description of the problem

RFC2960 requires Tie-Tags to be included in the COOKIE. The cookie
may not be encrypted. An attacker could discover the value of the
verification tags by analyzing cookies received after sending an
INIT.

2.34.2. Text changes to the document

```
---------
Old text: (Section 1.4)
---------
   o  Tie-Tags: Verification Tags from a previous association. These
      Tags are used within a State Cookie so that the newly
      restarting association can be linked to the original
      association within the endpoint that did not restart.


---------
New text: (Section 1.4)
---------

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


---------
Old text: (Section 5.2.1)
---------

   For an endpoint that is in the COOKIE-ECHOED state it MUST
```

populate its Tie-Tags with the Tag information of itself and
its peer (see section 5.2.2 for a description of the Tie-Tags).

New text: (Section 5.2.1)

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

Old text: (Section 5.2.2)

Unless otherwise stated, upon reception of an unexpected INIT for
this association, the endpoint shall generate an INIT ACK with a
State Cookie. In the outbound INIT ACK the endpoint MUST copy its
current Verification Tag and peer's Verification Tag into a
reserved place within the state cookie. We shall refer to these
locations as the Peer's-Tie-Tag and the Local-Tie-Tag. The
outbound SCTP packet containing this INIT ACK MUST carry a
Verification Tag value equal to the Initiation Tag found in the
unexpected INIT. And the INIT ACK MUST contain a new Initiation
Tag (randomly generated see Section 5.3.1). Other parameters
for the endpoint SHOULD be copied from the existing parameters
of the association (e.g. number of outbound streams) into the
INIT ACK and cookie.

New text: (Section 5.2.2)

Unless otherwise stated, upon reception of an unexpected INIT for
this association, the endpoint MUST generate an INIT ACK with a
State Cookie. In the outbound INIT ACK the endpoint MUST copy its
current Tie-Tags to a reserved place within the State Cookie and
the association's TCB. We shall refer to these locations inside
the cookie as the Peer's-Tie-Tag and the Local-Tie-Tag. We will
refer to the copy within an association's TCB as the Local Tag
and Peer's Tag. The outbound SCTP packet containing this
INIT ACK MUST carry a Verification Tag value equal to the
Initiation Tag found in the unexpected INIT. And the INIT ACK
MUST contain a new Initiation Tag (randomly generated see
Section 5.3.1). Other parameters for the endpoint SHOULD
be copied from the existing parameters of the association
(e.g. number of outbound streams) into the INIT ACK and cookie.

2.34.3. Solution description

   The solution to this problem is not to use the real verification tags
   within the State Cookie as tie-tags. Instead two 32 bit random
   numbers are created to form one 64 bit nonces and stored both in the
   State Cookie and the existing association TCB. This prevents
   exposing the verification tags inadvertently.

2.35. Port number verification in the COOKIE-ECHO

2.35.1. Description of the problem

   The State Cookie sent by a listening SCTP endpoint may not contain
   the original port numbers or the local verification tag. It is then
   possible that the endpoint on reception of the COOKIE-ECHO will not
   be able to verify that these values match the original values found
   in the INIT and INIT-ACK that began the association setup.

2.35.2. Text changes to the document

   Old text: (Section 5.1.5)
   ----
       3) Compare the creation timestamp in the State Cookie to the
          current local time. If the elapsed time is longer than the
          lifespan carried in the State Cookie, then the packet,
          including the COOKIE ECHO and any attached DATA chunks,
          SHOULD be discarded and the endpoint MUST transmit an ERROR
          chunk with a "Stale Cookie" error cause to the peer endpoint,

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

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

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

   New text: (Section 5.1.5)

         ----

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

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

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

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

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

2.35.3. Solution description

   By including both port numbers and the local verification tag within
   the State Cookie and verifying these during COOKIE-ECHO processing
   this issue is resolved.

2.36. Path Initialization

2.36.1. Description of the problem

   When an association enters the ESTABLISHED state the endpoint has no
   verification that all of the addresses presented by the peer are in
   fact belonging to the peer. This could cause various forms of denial
   of service attacks.

2.36.2. **Text changes to the document**

```
---------
Old text: None
---------


---------
New text: (Section 5.4)
---------
5.4 Path Verification

During association estabilishment the two peers
exchange a list of addresses. In the predominant case
these lists accurately represent the addresses owned
by each peer. However there exists the possibility that
a mis-behaving peer may supply addresses that it does
not own. To prevent this the following rules are applied
to all addresses of the new association:

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

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

3) All other addresses not covered by rules 1 and 2 are considered
   UNCONFIRMED and are subject to probing for verification.

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

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

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

Each RTO a probe may be sent on an active UNCONFIRMED path
in an attempt to move it to to the CONFIRMED state.
If during this probing the path becomes inactive this rate
is lowered to the normal HEARTBEAT rate. At the expiration
```

of the RTO timer the error counter of any path that was
probed but not CONFIRMED is incremented by one and subjected
to path failure detection defined in section 8.2. When probing
UNCONFIRMED addresses, however, the association overall error count
is NOT incremented.

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

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

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

- A HEARTBEAT including a nonce MAY be sent to an UNCONFIRMED
  address.

- A HEARTBEAT-ACK MAY be sent to an UNCONFIRMED address.

- A COOKIE-ACK MAY be sent to an UNCONFIRMED address but
  it MUST be bundled with a HEARTBEAT including a nonce.
  An implementation that does NOT support bundling MUST
  NOT send a COOKIE-ACK to an UNCONFIRMED address.

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

Old text: (Section 14)
----

14. Suggested SCTP Protocol Parameter Values

The following protocol parameters are RECOMMENDED:

RTO.Initial   - 3 seconds
RTO.Min   - 1 second
RTO.Max   - 60 seconds
RTO.Alpha   - 1/8

```
   RTO.Beta                 - 1/4
   Valid.Cookie.Life        - 60  seconds
   Association.Max.Retrans  - 10 attempts
   Path.Max.Retrans         - 5  attempts (per destination address)
   Max.Init.Retransmits     - 8  attempts
   HB.interval              - 30 seconds


   ---------
   New text: (Section 14)
   ---------


   14. Suggested SCTP Protocol Parameter Values

   The following protocol parameters are RECOMMENDED:

   RTO.Initial              - 3  seconds
   RTO.Min                  - 1  second
   RTO.Max                  - 60 seconds
   Max.Burst                - 4
   RTO.Alpha                - 1/8
   RTO.Beta                 - 1/4
   Valid.Cookie.Life        - 60 seconds
   Association.Max.Retrans  - 10 attempts
   Path.Max.Retrans         - 5  attempts (per destination address)
   Max.Init.Retransmits     - 8  attempts
   HB.Interval              - 30 seconds
   HB.Max.Burst             - 1
```

2.36.3. Solution description

   By properly setting up initial path state and accelerated probing via
   HEARTBEAT's an new association can verify that all addresses
   presented by a peer belong to that peer.

2.37. ICMP handling procedures

2.37.1. Description of the problem

   RFC2960 does not describe how ICMP messages should be processed by an
   SCTP endpoint.

2.37.2. Text changes to the document

   Old text: None

         New text

         11.5 Protection of non-SCTP capable hosts.

         To provide non-SCTP capable host with the same level of
         protection against attacks as SCTP capable ones all SCTP
         stacks MUST implement the ICMP handling described in Appendix C.

         When an SCTP stack receives a packet containing multiple
         control or DATA chunks and the processing of the packet
         requires the sending of multiple chunks in response,
         the sender of the response chunk('s) MUST NOT send
         more than one packet. If bundling is supported multiple
         response chunks that fit into a single packet MAY be
         bundled together into one single response packet. If bundling
         is not supported then the sender MUST NOT send more
         than one response chunk and MUST discard all other
         responses. Note that this rule does NOT apply to a
         SACK chunk since a SACK chunk is, in itself, a response
         to DATA and a SACK does not require a response of more DATA.

         An SCTP implementation SHOULD abort the association if
         it receives a SACK acknowledging a TSN which has not been sent.

         An SCTP implementation that receives an INIT that would
         require a large packet in response, due to the inclusion
         of multiple ERROR parameters, MAY (at its discrection)
         elect to omit some or all of the ERROR parameters
         to reduce the size of the INIT-ACK. Due to a combination
         of the size of the COOKIE parameter and the number
         of addresses a receiver of an INIT may be indicating
         to a peer, it is always possible that the INIT-ACK will
         be larger than the original INIT. An SCTP implementation
         SHOULD attempt to make the INIT-ACK as small as possible
         to reduce the possibility of byte amplification attacks.

         Old text: None

         New text: (Appendix C)

         Appendix C ICMP Handling

Whenever an ICMP message is received by an SCTP endpoint the
following procedures MUST be followed to assure proper
utilization of the information being provided by layer 3.

ICMP1) An implementation MAY ignore all ICMPv4 messages
       where the type field is not set to "Destination
       Unreachable".

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

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

ICMP4) An implementation MAY ignore all ICMPv6 messages
       of type "Parameter Problem" if the code is not
       "Unrecognized next header type encountered".

ICMP5) An implementation MUST use the payload of the
       ICMP message (V4 or V6) to locate the association
       which sent the message that ICMP is responding to.
       If the association cannot be found, An implementation SHOULD
       ignore the ICMP message.

ICMP6) An implementation MUST validate that the verification tag
       contained in the ICMP message matches the verification
       tag of the peer. If the verification tag is not 0 and does
       NOT match, discard the ICMP message. If it is 0 and the
       ICMP message contains enough bytes to verify that the chunk
       type is an INIT chunk and that the initiate tag matches
       the tag of the peer continue with ICMP7. If the ICMP
       message is too short or, the chunk type or the
       initiate tag does not match, silently discard the packet.

ICMP7) If the ICMP message is either a V6 "Packet Too Big" or a V4
       "Fragmentation Needed" an implemenation MAY process this
       information as defined for PATH MTU discovery.

ICMP8) If the ICMP code is a "Unrecognized next header type
       encountered" or a "Protocol Unreachable" an implementation
       MUST treat this message as an abort with the T bit set
       if it does not contain an INIT chunk. If it does contain
       an INIT chunk and the association is in COOKIE-WAIT state,
       handle the ICMP message like an ABORT.

ICMP9) If the ICMPv6 code is "Destination Unreachable" the

          implementation MAY mark the destination into the unreachable
          state or alternatively increment the path error counter.

   Note that these procedures differ with RFC1122 [1] and its
   requirements for processing of port unreachable messages and the
   requirements that an implemenation MUST abort associations in
   response to a "protocol unreachable" message. Port unreachable
   messages are not processed since an implementation will send an ABORT
   not a port unreachable. The stricter handling of the "protocol
   unreachable" message is due to security concerns for hosts that do
   NOT support SCTP.

2.37.3. Solution description

   The new appendix now describes proper handling of ICMP messages in
   conjunction with SCTP.

2.38. Checksum

2.38.1. Description of the problem

   RFC3309 [8] changes the SCTP checksum due to weaknesses in the
   original Adler 32 checksum for small messages. This document, being
   used as a guide for a cut and paste replacement to update RFC2960,
   thus needs to also incorporate the checksum changes. The idea being
   that one could apply all changes found in this guide to a copy of
   RFC2960 and have a "new" document that has ALL changes (including
   RFC3309).

2.38.2. Text changes to the document

   Old text:

   6.8 Adler-32 Checksum Calculation

      When sending an SCTP packet, the endpoint MUST strengthen the data
      integrity of the transmission by including the Adler-32 checksum
      value calculated on the packet, as described below.

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

      1) Fill in the proper Verification Tag in the SCTP common header

          and initialize the checksum field to 0's.

     2) Calculate the Adler-32 checksum of the whole packet, including
        the SCTP common header and all the chunks. Refer to
        [appendix B](#) for details of the Adler-32 algorithm. And,

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

     When an SCTP packet is received, the receiver MUST first check the
     Adler-32 checksum:

     1) Store the received Adler-32 checksum value aside,

     2) Replace the 32 bits of the checksum field in the received SCTP
        packet with all '0's and calculate an Adler-32 checksum value
        of the whole received packet. And,

     3) Verify that the calculated Adler-32 checksum is the same as the
        received Adler-32 checksum. If not, the receiver MUST treat
        the packet as an invalid SCTP packet.

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

   New text:

     6.8 CRC-32c Checksum Calculation

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

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

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

     2) Calculate the CRC32c checksum of the whole packet, including
        the SCTP common header and all the chunks. Refer to
        [appendix B](#) for details of the CRC32c algorithm. And,

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

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

1) Store the received CRC32c checksum value aside,

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

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

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

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

Old text:

Appendix B Alder 32 bit checksum calculation

The Adler-32 checksum calculation given in this appendix is
copied from [RFC1950].

Adler-32 is composed of two sums accumulated per byte: s1 is the
sum of all bytes, s2 is the sum of all s1 values. Both sums are
done modulo 65521. s1 is initialized to 1, s2 to zero. The
Adler-32 checksum is stored as s2*65536 + s1 in network byte
order.

The following C code computes the Adler-32 checksum of a data
buffer. It is written for clarity, not for speed. The sample
code is in the ANSI C programming language. Non C users may
find it easier to read with these hints:

```
&       Bitwise AND operator.
>>      Bitwise right shift operator.  When applied to an
        unsigned quantity, as here, right shift inserts zero bit(s)
        at the left.
<<      Bitwise left shift operator.  Left shift inserts zero
        bit(s) at the right.
++      "n++" increments the variable n.
%       modulo operator: a % b is the remainder of a divided by b.
 #define BASE 65521 /* largest prime smaller than 65536 */
```

```
   /*
     Update a running Adler-32 checksum with the bytes buf[0..len-1]
     and return the updated checksum.  The Adler-32 checksum should
     be initialized to 1.

      Usage example:

        unsigned long adler = 1L;

        while (read_buffer(buffer, length) != EOF) {
          adler = update_adler32(adler, buffer, length);
        }
        if (adler != original_adler) error();
   */
   unsigned long update_adler32(unsigned long adler,
      unsigned char *buf, int len)
   {
     unsigned long s1 = adler & 0xffff;
     unsigned long s2 = (adler >> 16) & 0xffff;
     int n;

     for (n = 0; n < len; n++) {
       s1 = (s1 + buf[n]) % BASE;
       s2 = (s2 + s1)     % BASE;
     }
     return (s2 << 16) + s1;
   }

   /* Return the adler32 of the bytes buf[0..len-1] */
   unsigned long adler32(unsigned char *buf, int len)
   {
     return update_adler32(1L, buf, len);
   }
```

New text:

Appendix B CRC32c checksum calculation

   We define a 'reflected value' as one that is the opposite of the
   normal bit order of the machine. The 32 bit CRC is calculated as
   described for CRC-32c and uses the polynomial code 0x11EDC6F41
   (Castagnoli93) or x^32+x^28+x^27+x^26+x^25
   +x^23+x^22+x^20+x^19+x^18+x^14+x^13+x^11+x^10+x^9+x^8+x^6+x^0.
   The CRC is computed using a procedure similar to ETHERNET CRC
   [ITU32], modified to reflect transport level usage.
   CRC computation uses polynomial division. A message
   bit-string M is transformed to a polynomial, M(X), and the CRC

is calculated from M(X) using polynomial arithmetic [PETERSON 72].

When CRCs are used at the link layer, the polynomial is derived
from on-the-wire bit ordering: the first bit 'on the wire' is the
high-order coefficient. Since SCTP is a transport-level protocol,
it cannot know the actual serial-media bit ordering. Moreover,
different links in the path between SCTP endpoints may use
different link-level bit orders.

A convention must therefore be established for mapping SCTP
transport messages to polynomials for purposes of CRC computation.
The bit-ordering for mapping SCTP messages to polynomials is that
bytes are taken most-significant first; but within each byte, bits
are taken least-significant first. The first byte of the message
provides the eight highest coefficients. Within each byte,
the least-significant SCTP bit gives the most significant
polynomial coefficient within that byte, and the most-significant
SCTP bit is the least significant polynomial coefficient in that
byte. (This bit ordering is sometimes called 'mirrored' or
'reflected' [WILLIAMS93].) CRC polynomials are to be transformed
back into SCTP transport-level byte values, using a consistent
mapping.

The SCTP transport-level CRC value should be calculated as
follows:

   - CRC input data are assigned to a byte stream, numbered from
      0 to N-1.

   - the transport-level byte-stream is mapped to a polynomial
      value. An N-byte PDU with j bytes numbered 0 to N-1, is
      considered as coefficients of a polynomial M(x) of order
      8N-1, with bit 0 of byte j being coefficient $x^{(8(N-j)-8)}$,
      bit 7 of byte j being coefficient $x^{(8(N-j)-1)}$.

   - the CRC remainder register is initialized with all 1s and
      the CRC is computed with an algorithm that simultaneously
      multiplies by x^{32} and divides by the CRC polynomial.

   - the polynomial is multiplied by x^{32} and divided by G(x),
      the generator polynomial, producing a remainder R(x) of
      degree less than or equal to 31.

   - the coefficients of R(x) are considered a 32 bit sequence.

   - the bit sequence is complemented. The result is the CRC
      polynomial.

- The CRC polynomial is mapped back into SCTP transport-level
  bytes. Coefficient of x^31 gives the value of bit 7 of
  SCTP byte 0, the coefficient of x^24 gives the value of
  bit 0 of byte 0. The coefficient of x^7 gives bit 7 of
  byte 3 and the coefficient of x^0 gives bit 0 of byte 3.
  The resulting four-byte transport-level sequence is the
  32-bit SCTP checksum value.

IMPLEMENTATION NOTE: Standards documents, textbooks, and vendor
literature on CRCs often follow an alternative formulation, in
which the register used to hold the remainder of the
long-division algorithm is initialized to zero rather than
all-1s, and instead the first 32 bits of the message are
complemented. The long-division algorithm used in our
formulation is specified, such that the the initial
multiplication by 2^32 and the long-division are combined into
one simultaneous operation. For such algorithms, and for
messages longer than 64 bits, the two specifications are
precisely equivalent. That equivalence is the intent of
this document.

Implementors of SCTP are warned that both specifications are to be
found in the literature, sometimes with no restriction on the
long-division algorithm. The choice of formulation in this
document is to permit non-SCTP usage, where the same CRC
algorithm may be used to protect messages shorter than 64 bits.

There may be a computational advantage in validating the
Association against the Verification Tag, prior to performing a
checksum, as invalid tags will result in the same action as a bad
checksum in most cases. The exceptions for this technique would
be INIT and some SHUTDOWN-COMPLETE exchanges, as well as a stale
COOKIE-ECHO. These special case exchanges must represent small
packets and will minimize the effect of the checksum calculation.

Old text: (Section 18)
----
18. Bibliography

[ALLMAN99] Allman, M. and Paxson, V., "On Estimating End-to-End
          Network Path Properties", Proc. SIGCOMM'99, 1999.

[FALL96]   Fall, K. and Floyd, S., Simulation-based Comparisons of
          Tahoe, Reno, and SACK TCP, Computer Communications Review,
          V. 26 N. 3, July 1996, pp. 5-21.

   [RFC1750]   Eastlake, D. (ed.), "Randomness Recommendations for
               Security", RFC 1750, December 1994.

   [RFC1950]   Deutsch P. and J. Gailly, "ZLIB Compressed Data Format
               Specification version 3.3", RFC 1950, May 1996.

   [RFC2104]   Krawczyk, H., Bellare, M. and R. Canetti, "HMAC: Keyed-
               Hashing for Message Authentication", RFC 2104, March 1997.

   [RFC2196]   Fraser, B., "Site Security Handbook", FYI 8, RFC 2196,
               September 1997.

   [RFC2522]   Karn, P. and W. Simpson, "Photuris: Session-Key Management
               Protocol", RFC 2522, March 1999.

   [SAVAGE99] Savage, S., Cardwell, N., Wetherall, D., and Anderson, T.,
               "TCP Congestion Control with a Misbehaving Receiver", ACM
               Computer Communication Review, 29(5), October 1999.

   New text: (Section 18, including changes from 2.11)
   ----

18. Bibliography

   [ALLMAN99] Allman, M. and Paxson, V., "On Estimating End-to-End
               Network Path Properties", Proc. SIGCOMM'99, 1999.

   [FALL96]   Fall, K. and Floyd, S., Simulation-based Comparisons of
               Tahoe, Reno, and SACK TCP, Computer Communications Review,
               V. 26 N. 3, July 1996, pp. 5-21.

   [ITU32]   ITU-T Recommendation V.42, "Error-correcting
                   procedures for DCEs using asynchronous-to-synchronous
                   conversion", section 8.1.1.6.2, October 1996.

   [PETERSON 1972] W. W. Peterson and E.J Weldon, Error Correcting
                   Codes, 2nd. edition, MIT Press, Cambridge,
                   Massachusetts.

   [RFC1750]   Eastlake, D. (ed.), "Randomness Recommendations for
               Security", RFC 1750, December 1994.

   [RFC1858]   Ziemba, G., Reed, D. and Traina P., "Security
               Considerations for IP Fragment Filtering", RFC 1858,
               October 1995.

   [RFC1950]   Deutsch P. and J. Gailly, "ZLIB Compressed Data Format

              Specification version 3.3", RFC 1950, May 1996.

   [RFC2104] Krawczyk, H., Bellare, M. and R. Canetti, "HMAC: Keyed-
              Hashing for Message Authentication", RFC 2104, March 1997.

   [RFC2196] Fraser, B., "Site Security Handbook", FYI 8, RFC 2196,
              September 1997.

   [RFC2522] Karn, P. and W. Simpson, "Photuris: Session-Key Management
              Protocol", RFC 2522, March 1999.

   [SAVAGE99] Savage, S., Cardwell, N., Wetherall, D., and Anderson, T.,
              "TCP Congestion Control with a Misbehaving Receiver", ACM
              Computer Communication Review, 29(5), October 1999.

   [WILLIAMS93]   Williams, R., "A PAINLESS GUIDE TO CRC ERROR
                   DETECTION ALGORITHMS" - Internet publication, August
                   1993,
                   http://www.geocities.com/SiliconValley/Pines/
                   8659/crc.htm.

2.38.3. Solution description

   This change adds the implementors guide the complete set of changes
   that when combined with RFC2960 [7] encompasses the changes from
   RFC3309 [8].

2.39. Retransmission Policy

2.39.1. Description of the problem

   The current retransmission policy (send all retransmissions an
   alternate destination) in the specification has performance issues
   under certain loss conditions with multihomed endpoints. Instead,
   fast retransmissions should be sent to the same destination, and only
   timeout retransmissions should be sent to an alternate destination
   [5].

2.39.2. 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 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.

    Old text: (Section 6.4.1)

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

    New text: (Section 6.4.1)

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

2.39.3. Solution description

    The above wording changes clarifies that only timeout retransmissions
    should be sent to an alternate active destination.

2.40. **Port Number 0**

2.40.1. **Description of the problem**

   The port number 0 has a special semantic in various APIs. For
   example in the socket API, if the user specifies 0, the SCTP
   implementation choses an appropriate port number for the user.
   Therefore the port number 0 should not be used on the wire.

2.40.2. **Text changes to the document**

   Old text: (Section 3.1)

      Source Port Number: 16 bits (unsigned integer)

         This is the SCTP sender's port number. It can be used by the
         receiver in combination with the source IP address, the SCTP
         destination port and possibly the destination IP address to
         identify the association to which this packet belongs.

      Destination Port Number: 16 bits (unsigned integer)

         This is the SCTP port number to which this packet is destined.
         The receiving host will use this port number to de-multiplex
         the SCTP packet to the correct receiving endpoint/application.

   New text: (Section 3.1)

      Source Port Number: 16 bits (unsigned integer)

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

      Destination Port Number: 16 bits (unsigned integer)

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

2.40.3. Solution description

It is clearly stated that the port number 0 is an invalid value on
the wire.

2.41. T Bit

2.41.1. Description of the problem

The description of the T bit as the bit describing whether a TCB has
been destroyed or not is misleading. In addition, the procedure
described in Section 2.13 is not as precise as needed.

2.41.2. Text changes to the document

```
---------
Old text: (Section 3.3.7)
---------

   T bit:  1 bit
      The T bit is set to 0 if the sender had a TCB that it
      destroyed. If the sender did not have a TCB it should set
      this bit to 1.


---------
New text: (Section 3.3.7)
---------

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

---------
Old text: (Section 3.3.13)
---------

   T bit:  1 bit
      The T bit is set to 0 if the sender had a TCB that it
      destroyed. If the sender did not have a TCB it should set
      this bit to 1.

---------
New text: (Section 3.3.13)
```

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

     Old text: (Section 8.4)
     ----

        3) If the packet contains an INIT chunk with a Verification Tag
           set to '0', process it as described in Section 5.1.
           Otherwise,

     New text: (Section 8.4)
     ----

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

     Old text: (Section 8.4)
     ----
       5) If the packet contains a SHUTDOWN ACK chunk, the receiver
          should respond to the sender of the OOTB packet with a
          SHUTDOWN COMPLETE. When sending the SHUTDOWN COMPLETE, the
          receiver of the OOTB packet must fill in the Verification
          Tag field of the outbound packet with the Verification Tag
          received in the SHUTDOWN ACK and set the T-bit in the Chunk
          Flags to indicate that no TCB was found. Otherwise,
     ----
     New text: (Section 8.4)
     ----

       5) If the packet contains a SHUTDOWN ACK chunk, the receiver
          should respond to the sender of the OOTB packet with a
          SHUTDOWN COMPLETE. When sending the SHUTDOWN COMPLETE, the
          receiver of the OOTB packet must fill in the Verification

        Tag field of the outbound packet with the Verification Tag
        received in the SHUTDOWN ACK and set the T-bit in the
        Chunk Flags to indicate that the Verification Tag is
        reflected. Otherwise,

   Old text: ([Section 8.4](#))
   ----

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

   New text: ([Section 8.4](#))
   ----

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

   Old text: ([Section 8.5.1](#))

      B) Rules for packet carrying ABORT:

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

         - If the ABORT is sent in response to an OOTB packet, the
            endpoint MUST follow the procedure described in
            [Section 8.4](#).

         - The receiver MUST accept the packet if the Verification
            Tag matches either its own tag, OR the tag of its peer.
            Otherwise, the receiver MUST silently discard the packet
            and take no further action.

```
      ---------
   New text: (Section 8.5.1)
      ---------
```

      B) Rules for packet carrying ABORT:

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

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

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

```
      ---------
   Old text: (Section 8.5.1)
      ---------
```

      C) Rules for packet carrying SHUTDOWN COMPLETE:

            - When sending a SHUTDOWN COMPLETE, if the receiver of the
               SHUTDOWN ACK has a TCB then the destination endpoint's
               tag MUST be used. Only where no TCB exists should the
               sender use the Verification Tag from the SHUTDOWN ACK.

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


```
      ---------
   New text: (Section 8.5.1)
      ---------
```

      C) Rules for packet carrying SHUTDOWN COMPLETE:

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

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

2.41.3. Solution description

The description of the T bit now clearly describes the semantic of
the bit. The procedures for the reception of the T bit have been
clarified.

2.42. Unknown Parameter Handling

2.42.1. Description of the problem

The description given in Section 2.33 does not state clearly if an
INIT-ACK or COOKIE-ECHO is sent.

2.42.2. Text changes to the document

The changes given here already include changes suggested in sections
Section 2.2, Section 2.27, and Section 2.33 of this document.

```
---------
```
Old text: (Section 3.2.1)
```
---------
```

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
     parameter in an 'Unrecognized Parameter Type' (in either an
     ERROR or in the INIT ACK).

   10 - Skip this parameter and continue processing.

   11 - Skip this parameter and continue processing but report the
        unrecognized parameter in an 'Unrecognized Parameter Type' (in
        either an ERROR or in the INIT ACK).

   New text: (Section 3.2.1)
   ----

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

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

   10 - Skip this parameter and continue processing.

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

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

   New text: (Note no old text, clarification added in section 3.2)

   3.2.2 Reporting of Unrecognized Parameters

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

      If the receiver of an INIT-ACK chunk detects unrecognized
      parameters and has to report them according to section 3.2.1 it
      SHOULD bundle the ERROR chunk containing the 'Unrecognized
      Parameters' error cause with the COOKIE-ECHO chunk sent in

   response to the INIT-ACK chunk. If the receiver of the INIT-ACK
   can not bundle the COOKIE-ECHO chunk with the ERROR chunk the
   ERROR chunk MAY be sent separately but not before the COOKIE-ACK
   has been received.

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

2.42.3. Solution description

   The new text clearly states that an INIT-ACK or COOKIE-ECHO has to be
   sent.

2.43. Cookie Echo Chunk

2.43.1. Description of the problem

   The description given in section 3.3.11 of RFC2960 [7] is unclear as
   to how the COOKIE-ECHO is composed.

2.43.2. Text changes to the document

```
   ---------
Old text: (Section 3.3.11)
   ---------
   Cookie: variable size

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

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

   ---------
New text: (Section 3.3.11)
   ---------
   Cookie: variable size

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

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

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

2.43.3. Solution description

The new text adds a note that helps clarify that a Cookie Echo chunk
is nothing more than the State Cookie parameter with only two bytes
modified.

2.44. Partial Chunks

2.44.1. Description of the problem

Section 6.10 of RFC2960 [7] uses the notion of 'partial chunks'
without defining it.

2.44.2. Text changes to the document

```
   ---------
```
   Old text: ([Section 6.10](#))
```
   ---------
```
   Partial chunks MUST NOT be placed in an SCTP packet.


```
   ---------
```
   New text: ([Section 6.10](#))
```
   ---------
```
   Partial chunks MUST NOT be placed in an SCTP packet. A partial
   chunk is a chunk which is not completely contained in the SCTP
   packet, i.e. the SCTP packet is too short to contain all the bytes
   of the chunk as indicated by the chunk length.

2.44.3. Solution description

   The new text adds a definition of 'partial chunks'.

2.45. Non-unicast addresses

2.45.1. Description of the problem

   [Section 8.4 of RFC2960](#) [7] forces the OOTB handling to discard all
   non-unicast addresses. This MUST, leaves future use of any-cast
   addresses in question. With the addition of the add-ip feature SCTP
   should be able to easily handle any-cast INIT's that can be followed,
   after association setup, with a delete of the any-cast address from
   the association.

2.45.2. Text changes to the document

   Old text: (Section 8.4)

   8.4 Handle "Out of the blue" Packets

      An SCTP packet is called an "out of the blue" (OOTB) packet if
      it is correctly formed, i.e., passed the receiver's Adler-32
      check (see Section 6.8), but the receiver is not able to
      identify the association to which this packet belongs.

      The receiver of an OOTB packet MUST do the following:

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

   New text: (Section 8.4)

   8.4 Handle "Out of the blue" Packets

      An SCTP packet is called an "out of the blue" (OOTB) packet if
      it is correctly formed, i.e., passed the receiver's Adler-32 check
      (see Section 6.8), but the receiver is not able to identify the
      association to which this packet belongs.

      The receiver of an OOTB packet MUST do the following:

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

2.45.3. Solution description

   The loosening of the wording to a SHOULD will now allow future use of
   anycast addresses. Note that no changes is made to section 11.2.4.1
   since responding to broadcast addresses could lead to flooding
   attacks and implementors should pay careful attention to these words.

2.46. Processing of ABORT chunks

2.46.1. Description of the problem

   Section 3.3.7 of RFC2960 [7] requires an SCTP endpoint to silently
   discard ABORT chunks received for associations that do not exist. It
   is not clear what this means in the COOKIE-WAIT state, for example.
   Therefore it was not clear if an ABORT sent in response to an INIT

should be processed or silently discarded.

2.46.2. Text changes to the document


```
   ---------
   Old text: (Section 3.3.7)
   ---------
```
      If an endpoint receives an ABORT with a format error or for an
      association that doesn't exist, it MUST silently discard it.


```
   ---------
   New text: (Section 3.3.7)
   ---------
```
      If an endpoint receives an ABORT with a format error or no
      TCB is found, it MUST silently discard it.

2.46.3. Solution description

   It is now clearly stated that an ABORT chunk should be processed
   whenever a TCB is found.

2.47. Sending of ABORT chunks

2.47.1. Description of the problem

   Section 5.1 of RFC2960 [7] requires that an ABORT chunk is sent in
   response to an INIT chunk when there is no listening end point. To
   make port scanning harder someone might not want these ABORTs to be
   received by the sender of the INIT chunks. Currently the only way to
   enforce this is by using a firewall discarding the packets containing
   the INIT chunks or the packets containing the ABORT chunks. It is
   desirable that the same can be done without a middle box.

2.47.2. Text changes to the document

```
     ---------
   Old text: (Section 5.1)
     ---------
```

       If an endpoint receives an INIT, INIT ACK, or COOKIE ECHO chunk
but decides not to establish the new association due to missing
mandatory parameters in the received INIT or INIT ACK, invalid
parameter values, or lack of local resources, it MUST respond with
an ABORT chunk.

```
     ---------
   New text: (Section 5.1)
     ---------
```

       If an endpoint receives an INIT, INIT ACK, or COOKIE ECHO chunk
but decides not to establish the new association due to missing
mandatory parameters in the received INIT or INIT ACK, invalid
parameter values, or lack of local resources, it SHOULD respond
with an ABORT chunk.

2.47.3. Solution description

The requirement of sending ABORT chunks is relaxed such that an
implementation can decide not to send ABORT chunks.

2.48. Handling of Supported Address Types parameter

2.48.1. Description of the problem

The sender of the INIT chunk can include a 'Supported Address Types'
parameter to indicate which address families are supported. It is
unclear how an INIT chunk should be processed where the source
address of the packet containing the INIT chunk or listed addresses
within the INIT chunk indicate that more address types are supported
than listed in the 'Supported Address Types' parameter.

2.48.2. Text changes to the document

The changes given here already include changes suggested in
Section 2.28 of this document.

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

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

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

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

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

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

[2.48.3](#). Solution description

   It is now clearly described how these Supported Address Types
   parameters with incorrect data should be handled.

[2.49](#). Handling of unexpected parameters

[2.49.1](#). Description of the problem

   [RFC2960](#) [[7](#)] clearly describes how unknown parameters in the INIT and
   INIT-ACK chunk should be processed. But it is not described how
   unexpected parameters should be processed. A parameter is unexpected
   if it is known and an optional parameter in either the INIT or INIT-

   ACK chunk but received in the chunk for which it is not an optional
   parameter. For exmaple, the 'Supported Address Types' parameter
   would be an unexpected parameter if contained in an INIT-ACK chunk.

2.49.2. Text changes to the document

   Old text: (Section 3.3.2)

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

   New text: (Section 3.3.2)

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

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

   Old text: (Section 3.3.3)

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

   New text: (Section 3.3.3)

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

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

2.49.3. Solution description

It is now stated how unexpected parameters should be processed.

2.50. Payload Protocol Identifier

2.50.1. Description of the problem

The current description of the payload protocol identifier does NOT
highlight the fact that the field is NOT necessarily in network byte
order.

2.50.2. Text changes to the document

    Old text: (Section 3.3.1)

       Payload Protocol Identifier: 32 bits (unsigned integer)

          This value represents an application (or upper layer) specified
          protocol identifier. This value is passed to SCTP by its upper
          layer and sent to its peer. This identifier is not used by
          SCTP but can be used by certain network entities as well as
          the peer application to identify the type of information being
          carried in this DATA chunk. This field must be sent even in
          fragmented DATA chunks (to make sure it is available for agents
          in the middle of the network).

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

    New text: (Section 3.3.1)

       Payload Protocol Identifier: 32 bits (unsigned integer)

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

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

2.50.3. Solution description

   It is now explicited stated that the upper layer is responsible for
   the byte order of this field.

2.51. Karns Algorithm

2.51.1. Description of the problem

   The current wording of the use of KARN's algorithm is not descriptive

    enough to assure that an implementation in a multi-homed association
    does not incorrectly mis-measure the RTT.

2.51.2. Text changes to the document

    Old text: (Section 6.3.1)

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

    New text: (Section 6.3.1)

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

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

2.51.3. Solution description

    The above clarification adds an implementation note that will provide
    additional guidance in the application of KARN's algorithm.

2.52. Fast Retransmit algorithm

2.52.1. Description of the problem

    The original SCTP specification is overly conservative in requiring 4
    missing reports before fast retransmitting a segment. TCP uses 3
    missing reports or 4 acknowledgments indicating the same segment was
    received.

2.52.2. Text changes to the document

```
     ---------
     Old text:
     ---------


     7.2.4 Fast Retransmit on Gap Reports

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

        Whenever an endpoint receives a SACK that indicates some TSN(s)
        missing, it SHOULD wait for 3 further miss indications (via
        subsequent SACK's) on the same TSN(s) before taking action with
        regard to Fast Retransmit.


     ---------
     New text:
     ---------


     7.2.4 Fast Retransmit on Gap Reports

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

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

2.52.3. Solution description

   The above changes will make SCTP and TCP behave similarly in terms of
   how fast they engage the Fast Retansmission algorithm upon receiving
   missing reports.

[3](). Acknowledgments

4. IANA considerations

This document recommends changes for the RFC2960 [7] BIS document.
As such, even though it lists new error cause code, this document in
itself does NOT define those new codes. Instead, the BIS document
will make the needed changes to RFC2960 [7] and thus it's IANA
section will require changes to be made.

5. Security considerations

This document should add no additional security risks to SCTP and in
fact SHOULD correct some original security flaws within the original
document once incorporated into a RFC2960 [7] BIS document .

6. References

6.1. Normative references

   [1]   Braden, R., "Requirements for Internet Hosts - Communication
         Layers", STD 3, RFC 1122, October 1989.

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

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

   [4]   Caro, A., Shah, K., Iyengar, J., Amer, P., and R. Stewart, "SCTP
         and TCP Variants: Congestion Control Under Multiple Losses",
          Technical Report TR2003-04, Computer and Information Sciences
         Department, University of Delaware, February 2003,
         <http://www.armandocaro.net/papers>.

   [5]   Caro, A., Amer, P., and R. Stewart, "Retransmission Schemes for
         End-to-end Failover with Transport Layer Multihoming",
          GLOBECOM, November 2004., <http://www.armandocaro.net/papers>.

   [6]   Handley, M., Padhye, J., and S. Floyd, "TCP Congestion Window
         Validation", RFC 2861, June 2000.

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

   [8]   Stone, J., Stewart, R., and D. Otis, "Stream Control
         Transmission Protocol (SCTP) Checksum Change", RFC 3309,
         September 2002.

6.2. Informational References

Authors' Addresses

    Randall R. Stewart
    Cisco Systems, Inc.
    4875 Forest Drive
    Suite 200
    Columbia, SC 29206
    USA

    Email: rrs@cisco.com

    Ivan Arias-Rodriguez
    Nokia Research Center
    PO Box 407
    FIN-00045 Nokia Group
    Finland

    Email: ivan.arias-rodriguez@nokia.com

    Kacheong Poon
    Sun Microsystems, Inc.
    3571 N. First St.
    San Jose, CA 95134
    USA

    Email: kacheong.poon@sun.com

    Armando L. Caro Jr.
    University of Delaware
    Department of Computer & Information Sciences
    103 Smith Hall
    Newark, DE 19716
    USA

    Email: me @ armandocaro . net
    URI: http://www.armandocaro.net

Michael Tuexen
Muenster Univ. of Applied Sciences
Stegerwaldstr. 39
48565 Steinfurt
Germany

Email: tuexen@fh-muenster.de

Intellectual Property Statement

Acknowledgment