Network Working Group
Internet-Draft
Intended status: Experimental
Expires: September 11, 2012

P. Amer
University of Delaware
M. Becke
T. Dreibholz
University of Duisburg-Essen
N. Ekiz
University of Delaware
J. Iyengar
Franklin and Marshall College
P. Natarajan
Cisco Systems
R. Stewart
Adara Networks
M. Tuexen
Muenster Univ. of Appl. Sciences

March 10, 2012

# **Load Sharing for the Stream Control Transmission Protocol (SCTP)**
**draft-tuexen-tsvwg-sctp-multipath-04.txt**

## Abstract

The Stream Control Transmission Protocol (SCTP) supports multi-homing for providing network fault tolerance. However, mainly one path is used for data transmission. Only timer-based retransmissions are carried over other paths as well.

This document describes how multiple paths can be used simultaneously for transmitting user messages.

## Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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

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

This Internet-Draft will expire on September 11, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors.  All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document.  Please review these documents carefully, as they describe your rights and restrictions with respect to this document.  Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

## 1. Introduction

One of the important features of the Stream Control Transmission Protocol (SCTP), which is currently specified in [RFC4960], is network fault tolerance.  This feature is for example required for Reliable Server Pooling (RSerPool, [RFC5351]).  Therefore, transmitting messages over multiple paths is supported, but only for redundancy.  So [RFC4960] does not specify how to use multiple paths simultaneously.

This document overcomes this limitation by specifying how multiple paths can be used simultaneously.  This has several benefits:

o  Improved bandwidth usage.

o  Better availability check with real user messages compared to HEARTBEAT-based information.

## 2. Conventions

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

## 3. Load Sharing

Basic requirement for applying SCTP load sharing is the Concurrent Multipath Transfer (CMT) extension of SCTP, which utilises multiple paths simultaneously.  We denote CMT-enabled SCTP as CMT-SCTP throughout this document.  CMT-SCTP is introduced in [IAS06] and in more detail in [I06], some illustrative examples of chunk handling are provided in [DBP10a].  CMT-SCTP provides three modifications to standard SCTP (split Fast Retransmissions, appropriate congestion window growth and delayed SACKs), which are described in the following subsections.

### 3.1. Split Fast Retransmissions

Paths with different latencies lead to overtaking of DATA chunks. This leads to gap reports, which are handled by Fast Retransmissions. However, due to the fact that multiple paths are used simultaneously, these Fast Retransmissions are usually useless and furthermore lead to a decreased congestion window size.

To avoid unnecessary Fast Retransmissions, the sender has to keep track of the path each DATA chunk has been sent on and consider

transmission paths before performing Fast Retransmissions. That is, on reception of a SACK, the sender MUST identify the highest acknowledged TSN on each path. A chunk SHOULD only be considered as missing if its TSN is smaller than the highest acknowledged TSN on its path. Section 3.1 of [DBP10a] contains an illustrated example.

## 3.2. Appropriate Congestion Window Growth

The congestion window adaptation algorithm for SCTP [RFC4960] allows increasing the congestion window only when a new cumulative ack (CumAck) is received by a sender. When SACKs with unchanged CumAcks are generated (due to reordering) and later arrive at a sender, the sender does not modify its congestion window. Since a CMT-SCTP receiver naturally observes reordering, many SACKs are sent containing new gap reports but not new CumAcks. When these gaps are later acked by a new CumAck, congestion window growth occurs, but only for the data newly acked in the most recent SACK. Data previously acked through gap reports will not contribute to congestion window growth, in order to prevent sudden increases in the congestion window resulting in bursts of data being sent.

To overcome the problems described above, the congestion window growth has to be handled as follows [IAS06]:

o The sender SHOULD keep track of the earliest non-retransmitted outstanding TSN per path.

o The sender SHOULD keep track of the earliest retransmitted outstanding TSN per path.

o The in-order delivery per path SHOULD be deduced.

o The congestion window of a path SHOULD be increased when the earliest non-retransmitted outstanding TSN of this path is advanced ('Pseudo CumAck') OR when the earliest retransmitted outstanding TSN of this path is advanced ('RTX Pseudo CumAck').

Section 3.2 of [DBP10a] contains an illustrated example of appropriate congestion window handling for CMT-SCTP.

## 3.3. Appropriate Delayed Acknowledgements

Standard SCTP [RFC4960] sends a SACK as soon as an out-of-sequence TSN has been received. Delayed Acknowledgements are only allowed if the received TSNs are in sequence. However, due to the load balancing of CMT-SCTP, DATA chunks may overtake each other. This leads to a high number of out-of-sequence TSNs, which have to be acknowledged immediately. Clearly, this behaviour increases the

overhead traffic (usually nearly one SACK chunk for each received packet containing a DATA chunk).

Delayed Acknowledgements for CMT-SCTP are handled as follows:

- In addition to [RFC4960], delaying of SACKs SHOULD *also* be applied for out-of-sequence TSNs.

- A receiver MUST maintain a counter for the number of DATA chunks received before sending a SACK. The value of the counter is stored into each SACK chunk (FIXME: add details; needs reservation of flags bits by IANA). After transmitting a SACK, the counter MUST be reset to 0. Its initial value MUST be 0.

- The SACK handling procedure for a missing TSN M is extended as follows:

  * If all newly acknowledged TSNs have been transmitted over the same path:

    + If there are newly acknowledged TSNs L and H so that L <= M <= H, the missing count of TSN M SHOULD be incremented by one (like for standard SCTP according to [RFC4960]).

    + Else if all newly acknowledged TSNs N satisfy the condition M <= N, the missing count of TSN M SHOULD be incremented by the number of TSNs reported in the SACK chunk.

  * Otherwise (that is, there are newly acknowledged TSNs on different paths), the missing count of TSN M SHOULD be incremented by one (like for standard SCTP according to [RFC4960]).

Section 3.3 of [DBP10a] contains an illustrated example of Delayed Acknowledgements for CMT-SCTP.

## 4. Non-Renegable SACK

### 4.1. Negotiation

Before sending/receiving NR-SACKs, both peer endpoints MUST agree on using NR-SACKs. This agreement MUST be negotiated during association establishment. NR-SACK is an extension to the core SCTP, and SCTP extensions that an endpoint supports are reported to the peer endpoint in Supported Extensions Parameter during association establishment (see Section 4.2.7 of [RFC5061].) The Supported Extensions Parameter consists of a list of non-standard Chunk Types

that are supported by the sender.

An endpoint supporting the NR-SACK extension MUST list the NR-SACK chunk in the Supported Extensions Parameter carried in the INIT or INIT-ACK chunk, depending on whether the endpoint initiates or responds to the initiation of the association. If the NR-SACK chunk type ID is listed in the Chunk Types List of the Supported Extensions Parameter, then the receiving endpoint MUST assume that the NR-SACK chunk is supported by the sending endpoint.

Both endpoints MUST support NR-SACKs for either endpoint to send an NR-SACK. If an endpoint establishes an association with a remote endpoint that does not list NR-SACK in the Supported Extensions Parameter carried in INIT chunk, then both endpoints of the association MUST NOT use NR-SACKs. After association establishment, an endpoint MUST NOT renegotiate the use of NR-SACKs.

Once both endpoints indicate during association establishment that they support the NR-SACK extension, each endpoint SHOULD acknowledge received DATA chunks with NR-SACK chunks, and not SACK chunks. That is, throughout an SCTP association, both endpoints SHOULD send either SACK chunks or NR-SACK chunks, never a mixture of the two.

## 4.2. The New Chunk Type: Non-Renegable SACK (NR-SACK)

Table 1 illustrates a new chunk type that will be used to transfer NR-SACK information.

| Chunk Type | Chunk Name |
|---|---|
| 0x10 | Non-Renegable Selective Acknowledgment (NR-SACK) |

Table 1: NR-SACK Chunk

As the NR-SACK chunk replaces the SACK chunk, many SACK chunk fields are preserved in the NR-SACK chunk. These preserved fields have the same semantics with the corresponding SACK chunk fields, as defined in [RFC4960], Section 3.3.4. The Gap Ack fields from RFC4960 have been renamed as R Gap Ack to emphasize their renegable nature. Their semantics are unchanged. For completeness, we describe all fields of the NR-SACK chunk, including those that are identical in the SACK chunk.

Similar to the SACK chunk, the NR-SACK chunk is sent to a peer endpoint to (1) acknowledge DATA chunks received in-order, (2) acknowledge DATA chunks received out-of-order, and (3) identify DATA chunks received more than once (i.e., duplicate.) In addition, the NR-SACK chunk (4) informs the peer endpoint of non-renegable out-of-

order DATA chunks.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |   Type = 0x10 |  Chunk Flags  |         Chunk Length          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                      Cumulative TSN Ack                       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |          Advertised Receiver Window Credit (a_rwnd)           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |Number of R Gap Ack Blocks = N |Number of NR Gap Ack Blocks = M|
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | Number of Duplicate TSNs = X  |           Reserved            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | R Gap Ack Block #1 Start      |   R Gap Ack Block #1 End      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   /                                                               /
   \                              ...                              \
   /                                                               /
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  R Gap Ack Block #N Start     |  R Gap Ack Block #N End       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  NR Gap Ack Block #1 Start    |   NR Gap Ack Block #1 End     |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   /                                                               /
   \                              ...                              \
   /                                                               /
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |   NR Gap Ack Block #M Start   |  NR Gap Ack Block #M End      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                       Duplicate TSN 1                         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   /                                                               /
   \                              ...                              \
   /                                                               /
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                       Duplicate TSN X                         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type: 8 bits

This field holds the IANA defined chunk type for NR-SACK chunk.  The suggested value of this field for IANA is 0x10.

Chunk Flags: 8 bits

Currently not used. It is recommended a sender set all bits to zero on transmit, and a receiver ignore this field.

Chunk Length: 16 bits (unsigned integer) [Same as SACK chunk]

This value represents the size of the chunk in bytes including the Chunk Type, Chunk Flags, Chunk Length, and Chunk Value fields.

Cumulative TSN Ack: 32 bits (unsigned integer) [Same as SACK chunk]

The value of the Cumulative TSN Ack is the last TSN received before a break in the sequence of received TSNs occurs. The next TSN value following the Cumulative TSN Ack has not yet been received at the endpoint sending the NR-SACK.

Advertised Receiver Window Credit (a_rwnd): 32 bits (unsigned integer) [Same as SACK chunk]

Indicates the updated receive buffer space in bytes of the sender of this NR-SACK, see Section 6.2.1 of [RFC4960] for details.

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

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

Number of (N)on(R)enegable Gap Ack Blocks (M): 16 bits (unsigned integer)

Indicates the number of Non-Renegable Gap Ack Blocks included in this NR-SACK.

Number of Duplicate TSNs (X): 16 bits [Same as SACK chunk]

Contains the number of duplicate TSNs the endpoint has received. Each duplicate TSN is listed following the NR Gap Ack Block list.

Reserved : 16 bits

Currently not used. It is recommended a sender set all bits to zero on transmit, and a receiver ignore this field.

(R)enegable Gap Ack Blocks:

The NR-SACK contains zero or more R Gap Ack Blocks. Each R Gap Ack Block acknowledges a subsequence of renegable out-of-order TSNs. By definition, all TSNs acknowledged by R Gap Ack Blocks are "greater than" the value of the Cumulative TSN Ack.

Because of TSN numbering wraparound, comparisons and all arithmetic operations discussed in this document are based on "Serial Number Arithmetic" as described in Section 1.6 of [RFC4960].

R Gap Ack Blocks are repeated for each R Gap Ack Block up to 'N' defined in the Number of R Gap Ack Blocks field. All DATA chunks with TSNs >= (Cumulative TSN Ack + R Gap Ack Block Start) and <= (Cumulative TSN Ack + R Gap Ack Block End) of each R Gap Ack Block are assumed to have been received correctly, and are renegable.

R Gap Ack Block Start: 16 bits (unsigned integer)

Indicates the Start offset TSN for this R Gap Ack Block. This number is set relative to the cumulative TSN number defined in Cumulative TSN Ack field. To calculate the actual start TSN number, the Cumulative TSN Ack is added to this offset number. The calculated TSN identifies the first TSN in this R Gap Ack Block that has been received.

R Gap Ack Block End: 16 bits (unsigned integer)

Indicates the End offset TSN for this R Gap Ack Block. This number is set relative to the cumulative TSN number defined in the Cumulative TSN Ack field. To calculate the actual TSN number, the Cumulative TSN Ack is added to this offset number. The calculated TSN identifies the TSN of the last DATA chunk received in this R Gap Ack Block.

N(on)R(enegable) Gap Ack Blocks:

The NR-SACK contains zero or more NR Gap Ack Blocks. Each NR Gap Ack Block acknowledges a continuous subsequence of non-renegable out-of-order DATA chunks. If a TSN is nr-gap-acked in any NR-SACK chunk, then all subsequently transmitted NR-SACKs with a smaller cum-ack value than that TSN SHOULD also nr-gap-ack that TSN.

NR Gap Ack Blocks are repeated for each NR Gap Ack Block up to 'M' defined in the Number of NR Gap Ack Blocks field. All DATA chunks with TSNs >= (Cumulative TSN Ack + NR Gap Ack Block Start) and <= (Cumulative TSN Ack + NR Gap Ack Block End) of each NR Gap Ack Block are assumed to be received correctly, and are Non-Renegable.

NR Gap Ack Block Start: 16 bits (unsigned integer)

Indicates the Start offset TSN for this NR Gap Ack Block. This number is set relative to the cumulative TSN number defined in Cumulative TSN Ack field. To calculate the actual TSN number, the Cumulative TSN Ack is added to this offset number. The calculated

TSN identifies the first TSN in this NR Gap Ack Block that has been received.

NR Gap Ack Block End: 16 bits (unsigned integer)

Indicates the End offset TSN for this NR Gap Ack Block. This number is set relative to the cumulative TSN number defined in Cumulative TSN Ack field. To calculate the actual TSN number, the Cumulative TSN Ack is added to this offset number. The calculated TSN identifies the TSN of the last DATA chunk received in this NR Gap Ack Block.

Note:

NR Gap Ack Blocks and R Gap Ack Blocks in an NR-SACK chunk SHOULD acknowledge disjoint sets of TSNs. That is, an out-of-order TSN SHOULD be listed in either an R Gap Ack Block or an NR Gap Ack Block, but not the both. R Gap Ack Blocks and NR Gap Ack Blocks together provide the information as do the Gap Ack Block of a SACK chunk, plus additional information about non-renegability.

If all out-of-order data acked by an NR-SACK are renegable, then the Number of NR Gap Ack Blocks MUST be set to 0. If all out-of-order data acked by an NR-SACK are non-renegable, then the Number of R Gap Ack Blocks SHOULD be set to 0. TSNs listed in R Gap Ack Block will be referred as r-gap-acked.

Duplicate TSN: 32 bits (unsigned integer) [Same as SACK chunk]

Indicates a duplicate TSN received since the last NR-SACK was sent. Exactly 'X' duplicate TSNs SHOULD be reported where 'X' was defined in Number of Duplicate TSNs field.

Each duplicate TSN is listed in this field as many times as the TSN was received since the previous NR-SACK was sent. For example, if a data receiver were to get the TSN 19 three times, the data receiver would list 19 twice in the outbound NR-SACK. After sending the NR-SACK if the receiver received one more TSN 19, the receiver would list 19 as a duplicate once in the next outgoing NR-SACK.

## 4.3. An Illustrative Example

Assume the following DATA chunks have arrived at the receiver.

```
                     --------------------------------
                     | TSN=16| SID=2 | SSN=N/A| U=1 |
                     --------------------------------
                     | TSN=15| SID=1 | SSN= 4 | U=0 |
                     --------------------------------
                     | TSN=14| SID=0 | SSN= 4 | U=0 |
                     --------------------------------
                     | TSN=13| SID=2 | SSN=N/A| U=1 |
                     --------------------------------
                     |                              |
                     --------------------------------
                     | TSN=11| SID=0 | SSN= 3 | U=0 |
                     -------------------------------
                     |                              |
                     --------------------------------
                     |                              |
                     --------------------------------
                     | TSN=8 | SID=2 | SSN=N/A| U=1 |
                     --------------------------------
                     | TSN=7 | SID=1 | SSN= 2 | U=0 |
                     --------------------------------
                     | TSN=6 | SID=1 | SSN= 1 | U=0 |
                     --------------------------------
                     | TSN=5 | SID=0 | SSN= 1 | U=0 |
                     --------------------------------
                     |                              |
                     --------------------------------
                     | TSN=3 | SID=1 | SSN= 0 | U=0 |
                     --------------------------------
                     | TSN=2 | SID=0 | SSN= 0 | U=0 |
                     --------------------------------
```

The above figure shows the list of DATA chunks at the receiver. TSN denotes the transmission sequence number of the DATA chunk, SID denotes the stream id to which the DATA chunk belongs, SSN denotes the sequence number of the DATA chunk within its stream, and the U bit denotes whether the DATA chunk requires ordered(=0) or unordered(=1) delivery [RFC4960]. Note that TSNs 4,9,10, and 12 have not arrived.

This data can be viewed as three separate streams as follows (assume each stream begins with SSN=0.) Note that in this example, the application uses stream 2 for unordered data transfer. By definition, SSN fields of unordered DATA chunks are ignored.

Stream-0:

```
   SSN:          0     1     2     3     4
   TSN:       |  2  |  5  |     |  11 |  14 |
   U-Bit:     |  0  |  0  |     |  0  |   0 |

   Stream-1:

   SSN:          0     1     2     3     4
   TSN:       |  3  |  6  |  7  |     |  15 |
   U-Bit:     |  0  |  0  |  0  |     |   0 |

   Stream-2:

   SSN:         N/A   N/A   N/A
   TSN:       |  8  |  13 |  16 |
   U-Bit:     |  1  |   1 |   1 |
```

The NR-SACK to acknowledge the above data SHOULD be constructed as follows for each of the three cases described below (the a_rwnd is arbitrarily set to 4000):

CASE-1: Minimal Data Receiver Responsibility - no out-of-order deliverable data yet delivered

None of the deliverable out-of-order DATA chunks have been delivered, and the receiver of the above data does not take responsibility for any of the received out-of-order DATA chunks.  The receiver reserves the right to renege any or all of the out-of-order DATA chunks.

```
       +------------------------------+------------------------------+
       | Type = 0x10  |   00000000    |      Chunk Length = 32       |
       +------------------------------+------------------------------+
       |                      Cumulative TSN Ack = 3                 |
       +------------------------------+------------------------------+
       |                          a_rwnd = 4000                      |
       +------------------------------+------------------------------+
       | Num of R Gap Ack Blocks = 3 |Num of NR Gap Ack Blocks = 0 |
       +------------------------------+------------------------------+
       |    Num of Duplicates = 0    |            0x00             |
       +------------------------------+------------------------------+
       |R Gap Ack Block #1 Start = 2 | R Gap Ack Block #1 End = 5  |
       +------------------------------+------------------------------+
       |R Gap Ack Block #2 Start = 8 | R Gap Ack Block #2 End = 8  |
       +------------------------------+------------------------------+
       |R Gap Ack Block #3 Start = 10| R Gap Ack Block #3 End = 13 |
       +------------------------------+------------------------------+
```

CASE-2: Minimal Data Receiver Responsibility - all out-of-order

deliverable data delivered

In this case, the NR-SACK chunk is being sent after the data receiver has delivered all deliverable out-of-order DATA chunks to its receiving application(i.e., TSNs 5,6,7,8,13, and 16.)  The receiver reserves the right to renege on all undelivered out-of-order DATA chunks(i.e., TSNs 11,14, and 15.)

```
      +------------------------------+------------------------------+
      | Type = 0x10  |     0x00      |       Chunk Length = 40      |
      +------------------------------+------------------------------+
      |                     Cumulative TSN Ack = 3                  |
      +------------------------------+------------------------------+
      |                        a_rwnd = 4000                        |
      +------------------------------+------------------------------+
      | Num of R Gap Ack Blocks = 2  | Num of NR Gap Ack Blocks = 3 |
      +------------------------------+------------------------------+
      |    Num of Duplicates = 0     |              0x00            |
      +------------------------------+------------------------------+
      | R Gap Ack Block #1 Start = 8 | R Gap Ack Block #1 End = 8   |
      +------------------------------+------------------------------+
      | R Gap Ack Block #2 Start = 11| R Gap Ack Block #2 End = 12  |
      +------------------------------+------------------------------+
      |NR Gap Ack Block #1 Start = 2 | NR Gap Ack Block #1 End = 5  |
      +------------------------------+------------------------------+
      |NR Gap Ack Block #2 Start = 10| NR Gap Ack Block #2 End = 10 |
      +------------------------------+------------------------------+
      |NR Gap Ack Block #3 Start = 13| NR Gap Ack Block #3 End = 13 |
      +------------------------------+------------------------------+
```

CASE-3: Maximal Data Receiver Responsibility

In this special case, all out-of-order data blocks acknowledged are non-renegable.  This case would occur when the data receiver is programmed never to renege, and takes responsibility to deliver all DATA chunks that arrive out-of-order.  In this case Num of R Gap Ack Blocks is zero indicating all reported out-of-order TSNs are nr-gap-acked.

```
     +--------------------------------+--------------------------------+
     |  Type = 0x10   |     0x00      |     Chunk Length = 32          |
     +--------------------------------+--------------------------------+
     |                     Cumulative TSN Ack = 3                      |
     +--------------------------------+--------------------------------+
     |                         a_rwnd = 4000                           |
     +--------------------------------+--------------------------------+
     |  Num of R Gap Ack Blocks = 0   |  Num of NR Gap Ack Blocks = 3 |
     +--------------------------------+--------------------------------+
     |     Num of Duplicates = 0      |             0x00               |
     +--------------------------------+--------------------------------+
     | NR Gap Ack Block #1 Start = 2  | NR Gap Ack Block #1 End = 5    |
     +--------------------------------+--------------------------------+
     | NR Gap Ack Block #2 Start = 8  | NR Gap Ack Block #2 End = 8    |
     +--------------------------------+--------------------------------+
     | NR Gap Ack Block #3 Start = 10 | NR Gap Ack Block #3 End = 13   |
     +--------------------------------+--------------------------------+
```

## 4.4. Procedures

The procedures regarding "when" to send an NR-SACK chunk are identical to the procedures regarding when to send a SACK chunk, as outlined in Section 6.2 of [RFC4960].

### 4.4.1. Sending an NR-SACK chunk

All of the NR-SACK chunk fields identical to the SACK chunk MUST be formed as described in Section 6.2 of [RFC4960].

It is up to the data receiver whether or not to take responsibility for delivery of each out-of-order DATA chunk.  An out-of-order DATA chunk that has already been delivered, or that the receiver takes responsibility to deliver (i.e., guarantees not to renege) is Non Renegable(NR), and SHOULD be included in an NR Gap Ack Block field of the outgoing NR-SACK.  All other out-of-order data is (R)enegable, and SHOULD be included in R Gap Ack Block field of the outgoing NR-SACK.

Consider three types of data receiver:

CASE-1:  Data receiver takes no responsibility for delivery of any out-of-order DATA chunks

CASE-2:  Data receiver takes responsibility for all out-of-order DATA chunks that are "deliverable" (i.e., DATA chunks in-sequence within the stream they belong to, or DATA chunks whose (U)nordered bit is 1)

CASE-3:  Data receiver takes responsibility for delivery of all out-of-order DATA chunks, whether deliverable or not deliverable

The data receiver SHOULD follow the procedures outlined below for building the NR-SACK.

CASE-1:

1A)  Identify the TSNs received out-of-order.

1B)  For these out-of-order TSNs, identify the R Gap Ack Blocks. Fill the Number of R Gap Ack Blocks (N) field, R Gap Ack Block #i Start, and R Gap Ack Block #i End where i goes from 1 to N.

1C)  Set the Number of NR Gap Ack Blocks (M) field to 0.

CASE-2:

2A)  Identify the TSNs received out-of-order.

2B)  For the received out-of-order TSNs, check the (U)nordered bit of each TSN.  Tag unordered TSNs as NR.

2C)  For each stream, also identify the TSNs received out-of-order but are in-sequence within that stream.  Tag those in-sequence TSNs as NR.

2D)  Tag all out-of-order data that is not NR as (R)enegable.

2E)  For those TSNs tagged as (R)enegable, identify the (R)enegable Blocks.  Fill the Number of R Gap Ack Blocks(N) field, R Gap Ack Block #i Start, and R Gap Ack Block #i End where i goes from 1 to N.

2F)  For those TSNs tagged as NR, identify the NR Blocks.  Fill the Number of NR Gap Ack Blocks(M) field, NR Gap Ack Block #i Start, and NR Gap Ack Block #i End where i goes from 1 to M.

CASE-3:

3A) Identify the TSNs received out-of-order. All of these TSNs SHOULD be nr-gap-acked.

3B) Set the Number of R Gap Ack Blocks (N) field to 0.

3C) For these out-of-order TSNs, identify the NR Gap Ack Blocks. Fill the Number of NR Gap Ack Blocks (M) field, NR Gap Ack Block #i Start, and NR Gap Ack Block #i End where i goes from 1 to M.

RFC4960 states that the SCTP endpoint MUST report as many Gap Ack Blocks as can fit in a single SACK chunk limited by the current path MTU. When using NR-SACKs, the SCTP endpoint SHOULD fill as many R Gap Ack Blocks and NR Gap Ack Blocks starting from the Cumulative TSN Ack value as can fit in a single NR-SACK chunk limited by the current path MTU. If space remains, the SCTP endpoint SHOULD fill as many Duplicate TSNs as possible starting from Cumulative TSN Ack value.

### 4.4.2. Receiving an NR-SACK Chunk

When an NR-SACK chunk is received, all of the NR-SACK fields identical to a SACK chunk SHOULD be processed and handled as in SACK chunk handling outlined in Section 6.2.1 of [RFC4960].

The NR Gap Ack Block Start(s) and NR Gap Ack Block End(s) are offsets relative to the cum-ack. To calculate the actual range of nr-gap-acked TSNs, the cum-ack MUST be added to the Start and End.

For example, assume an incoming NR-SACK chunk's cum-ack is 12 and an NR Gap Ack Block defines the NR Gap Ack Block Start=5, and the NR Gap Ack Block End=7. This NR Gap Ack block nr-gap-acks TSNs 17 through 19 inclusive.

Upon reception of an NR-SACK chunk, all TSNs listed in either R Gap Ack Block(s) or NR Gap Ack Block(s) SHOULD be processed as would be TSNs included in Gap Ack Block(s) of a SACK chunk. All TSNs in all NR Gap Ack Blocks SHOULD be removed from the data sender's retransmission queue as their delivery to the receiving application has either already occurred, or is guaranteed by the data receiver.

Although R Gap Ack Blocks and NR Gap Ack Blocks SHOULD be disjoint sets, NR-SACK processing SHOULD work if an NR-SACK chunk has a TSN listed in both an R Gap Ack Block and an NR Gap Ack Block. In this case, the TSN SHOULD be treated as Non-Renegable.

Implementation Note:

Most of NR-SACK processing at the data sender can be implemented by using the same routines as in SACK that process the cum ack and the gap ack(s), followed by removal of nr-gap-acked DATA chunks from the retransmission queue. However, with NR-SACKs, as out-of-order DATA is sometimes removed from the retransmission queue, the gap ack processing routine should recognize that the data sender's retransmission queue has some transmitted data removed. For example, while calculating missing reports, the gap ack processing routine cannot assume that the highest TSN transmitted is always at the tail (right edge) of the retransmission queue.

## 5. Buffer Blocking Mitigation

TBD. See [ADB11], [DBR10].

### 5.1. Sender Buffer Splitting

TBD. See [ADB11], [DBR10].

### 5.2. Receiver Buffer Splitting

TBD. See [ADB11], [DBR10].

### 5.3. Problems during Path Failure

This section discusses CMT's receive buffer related problems during path failure, and proposes a solution for the same.

#### 5.3.1. Problem Description

Link failures arise when a router or a link connecting two routers fails due to link disconnection, hardware malfunction, or software error. Overloaded links caused by flash crowds and denial-of-service (DoS) attacks also degrade end-to-end communication between peer hosts. Ideally, the routing system detects link failures, and in response, reconfigures the routing tables and avoids routing traffic via the failed link. However, existing research highlights problems with Internet backbone routing that result in long route convergence times. The pervasiveness of path failures motivated us to study their impact on CMT, since CMT achieves better throughput via simultaneous data transmission over multiple end-to-end paths.

CMT is an extension to SCTP, and therefore retains SCTP's failure detection process. A CMT sender uses a tunable failure detection threshold called Path.Max.Retrans (PMR). When a sender experiences

more than PMR consecutive timeouts while trying to reach an active destination, the destination is marked as failed. With PMR=5, the failure detection takes 6 consecutive timeouts or 63s. After every timeout, the CMT sender continues to transmit new data on the failed path increasing the chances of receive buffer (rbuf) blocking and degrading CMT performance during permanent and short-term path failures [NEA08].

### 5.3.2. Solution: Potentially-failed Destination State

To mitigate the rbuf blocking, we introduce a new destination state called 'potentially-failed' state in SCTP (and CMT's) failure detection process [I-D.nishida-tsvwg-sctp-failover]. This solution is based on the rationale that loss detected by a timeout implies either severe congestion or failure en route. After a single timeout on a path, a sender is unsure, and marks the corresponding destination as 'potentially-failed' (PF). A PF destination is not used for data transmission or retransmission. CMT's retransmission policies are augmented to include the PF state. Performance evaluations prove that the PF state significantly reduces rbuf blocking during failure detection [NEA08].

## 5.4. Non-Renegable SACK

This section discusses problems with SCTP's SACK mechanism and how it affects the send buffer and CMT performance.

### 5.4.1. Problem Description

Gap-acks acknowledge DATA chunks that arrive out-of-order to a transport layer data receiver. A gap-ack in SCTP is advisory, in that, while it notifies a data sender about the reception of indicated DATA chunks, the data receiver is permitted to later discard DATA chunks that it previously had gap-acked. Discarding a previously gap-acked DATA chunk is known as 'reneging'. Because of the possibility of reneging in SCTP, any gap-acked DATA chunk MUST NOT be removed from the data sender's retransmission queue until the DATA chunk is later CumAcked.

Situations exist when a data receiver knows that reneging on a particular out-of-order DATA chunk will never take place, such as (but not limited to) after an out-of-order DATA chunk is delivered to the receiving application. With current SACKs in SCTP, it is not possible for a data receiver to inform a data sender if or when a particular out-of-order 'deliverable' DATA chunk has been 'delivered' to the receiving application. Thus the data sender MUST keep a copy of every gap-acked out-of-order DATA chunk(s) in the data sender's retransmission queue until the DATA chunk is CumAcked. This use of

the data sender's retransmission queue is wasteful. The wasted buffer often degrades CMT performance; the degradation increases when a CMT flow traverses via paths with disparate end-to-end properties [NEY08].

### 5.4.2. Solution: Non-Renegable SACKs

Non-Renegable Selective Acknowledgments (NR-SACKs) Section 4 are a new kind of acknowledgements, extending SCTP's SACK chunk functionalities. The NR-SACK chunk is an extension of the existing SACK chunk. Several fields are identical, including the Cumulative TSN Ack, the Advertised Receiver Window Credit (a_rwnd), and Duplicate TSNs. These fields have the same semantics as described in [RFC4960].

NR-SACKs also identify out-of-order DATA chunks that a receiver either: (1) has delivered to its receiving application, or (2) takes full responsibility to eventually deliver to its receiving application. These out-of-order DATA chunks are 'non-renegable.' Non-Renegable data are reported in the NR Gap Ack Block field of the NR-SACK chunk as described Section 4. We refer to non-renegable selective acknowledgements as 'nr-gap-acks.'

When an out-of-order DATA chunk is nr-gap-acked, the data sender no longer needs to keep that particular DATA chunk in its retransmission queue, thus allowing the data sender to free up its buffer space sooner than if the DATA chunk were only gap-acked. NR-SACKs improve send buffer utilization and throughput for CMT flows [NEY08].

## 6. Handling of Shared Bottlenecks

### 6.1. Introduction

CMT-SCTP assumes all paths to be disjoint. Since each path independently uses a TCP-like congestion control, an SCTP association using N paths over the same bottleneck acquires N times the bandwidth of a concurrent TCP flow. This is clearly unfair. A reliable detection of shared bottlenecks is impossible in arbitrary networks like the Internet. Therefore, [DBA11], [DBP10b] apply the idea of Resource Pooling to CMT-SCTP. Resource Pooling (RP) denotes 'making a collection of resources behave like a single pooled resource' [WHB09]. The modifications of RP-enabled CMT-SCTP, further denoted as CMT/RP-SCTP, are described in the following subsections. A detailed description of CMT/RP-SCTP, including congestion control examples, can be found in [DBA11], [DBP10b].

### 6.2. Initial Values

TDB.

### 6.3. Congestion Window Growth

TDB. See [DBA11].

### 6.4. Congestion Window Decrease

TDB. See [DBA11].

## 7. Chunk Scheduling

TDB. See [DST10].

## 8. Socket API Considerations

See [I-D.dreibholz-tsvwg-sctpsocket-multipath] and [I-D.dreibholz-tsvwg-sctpsocket-sqinfo].

## 9. IANA Considerations

[NOTE to RFC-Editor:

"RFCXXXX" is to be replaced by the RFC number you assign this document.

]

[NOTE to RFC-Editor:

The suggested values for the chunk type and the chunk parameter types are tentative and to be confirmed by IANA.

]

This document (RFCXXXX) is the reference for all registrations described in this section. The suggested changes are described below.

### 9.1. A New Chunk Type

A chunk type has to be assigned by IANA. It is suggested to use the values given in Section 4. IANA should assign this value from the

pool of chunks with the upper two bits set to '00'.

This requires an additional line in the "Chunk Types" registry for SCTP:

Chunk Types

| ID Value | Chunk Type | Reference |
|---|---|---|
| 16 | Non-Renegable SACK (NR-SACK) | [RFCXXXX] |

The registration table as defined in [RFC6096] for the chunk flags of this chunk type is empty.

## 10. Security Considerations

This document does not add any additional security considerations in addition to the ones given in [RFC4960].

## 11. Acknowledgments

The authors wish to thank Phillip Conrad, Jonathan Leighton, and Ertugrul Yilmaz for their invaluable comments.

## 12. References

### 12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.

[RFC5061] Stewart, R., Xie, Q., Tuexen, M., Maruyama, S., and M. Kozuka, "Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration", RFC 5061, September 2007.

[RFC5351] Lei, P., Ong, L., Tuexen, M., and T. Dreibholz, "An Overview of Reliable Server Pooling Protocols", RFC 5351, September 2008.

[RFC6096] Tuexen, M. and R. Stewart, "Stream Control Transmission Protocol (SCTP) Chunk Flags Registration", RFC 6096,

January 2011.

[I-D.nishida-tsvwg-sctp-failover]
Nishida, Y., Natarajan, P., and A. Caro, "Quick Failover Algorithm in SCTP", draft-nishida-tsvwg-sctp-failover-04 (work in progress), September 2011.

[I-D.dreibholz-tsvwg-sctpsocket-multipath]
Dreibholz, T., Becke, M., and H. Adhari, "SCTP Socket API Extensions for Concurrent Multipath Transfer", draft-dreibholz-tsvwg-sctpsocket-multipath-03 (work in progress), March 2012.

[I-D.dreibholz-tsvwg-sctpsocket-sqinfo]
Dreibholz, T., Seggelmann, R., and M. Becke, "Sender Queue Info Option for the SCTP Socket API", draft-dreibholz-tsvwg-sctpsocket-sqinfo-03 (work in progress), March 2012.

## 12.2. Informative References

[I06] Iyengar, J., "End-to-End Concurrent Multipath Transfer Using Transport Layer Multihoming", PhD Dissertation Computer Science Dept., University of Delaware, April 2006.

[IAS06] Iyengar, J., Amer, P., and R. Stewart, "Concurrent Multipath Transfer Using SCTP Multihoming Over Independent End-to-End Paths", Journal IEEE/ACM Transactions on Networking, October 2006.

[NEA08] Natarajan, P., Ekiz, N., Iyengar, J., Amer, P., and R. Stewart, "Concurrent Multipath Transfer Using Transport Layer Multihoming: Introducing the Potentially-failed Destination State", Proceedings of the IFIP Networking, May 2008.

[NEY08] Natarajan, P., Ekiz, N., Yilmaz, E., Amer, P., Iyengar, J., and R. Stewart, "Non-Renegable Selective Acknowledgments (NR-SACKs) for SCTP", Proceedings of the 16th IEEE International Conference on Network Protocols (ICNP) , October 2008.

[WHB09] Wischik, D., Handley, M., and M. Braun, "The Resource Pooling Principle", Journal ACM SIGCOMM Computer Communication Review, October 2009.

[DBP10a] Dreibholz, T., Becke, M., Pulinthanath, J., and E.

Rathgeb, "Implementation and Evaluation of Concurrent Multipath Transfer for SCTP in the INET Framework", Proceedings of the 3rd ACM/ICST OMNeT++ Workshop, March 2010.

[DBP10b] Dreibholz, T., Becke, M., Pulinthanath, J., and E. Rathgeb, "Applying TCP-Friendly Congestion Control to Concurrent Multipath Transfer", Proceedings of the IEEE 24th International Conference on Advanced Information Networking and Applications (AINA), April 2010.

[YEN10] Yilmaz, E., Ekiz, N., Natarajan, P., Amer, P., Leighton, J., Baker, F., and R. Stewart, "Throughput analysis of Non-Renegable Selective Acknowledgments (NR-SACKs) for SCTP", Comput. Commun. (2010), doi:10.1016/ j.comcom.2010.06.028 , 2010.

[DST10] Dreibholz, T., Seggelmann, R., Tuexen, M., and E. Rathgeb, "Transmission Scheduling Optimizations for Concurrent Multipath Transfer", Proceedings of the 8th International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNeT) , November 2010.

[DBR10] Dreibholz, T., Becke, M., Rathgeb, E., and M. Tuexen, "On the Use of Concurrent Multipath Transfer over Asymmetric Paths", Proceedings of the IEEE Global Communications Conference (GLOBECOM), December 2010.

[ADB11] Adhari, H., Dreibholz, T., Becke, M., Rathgeb, E., and M. Tuexen, "Evaluation of Concurrent Multipath Transfer over Dissimilar Paths", Proceedings of the 1st International Workshop on Protocols and Applications with Multi-Homing Support (PAMS), March 2011.

[DBA11] Dreibholz, T., Becke, M., Adhari, H., and E. Rathgeb, "On the Impact of Congestion Control for Concurrent Multipath Transfer on the Transport Layer", Proceedings of the 11th IEEE International Conference on Telecommunications (ConTEL), June 2011.

## Authors' Addresses

Paul D. Amer
University of Delaware, Computer and Information Sciences Department
Newark, DE  19716
US

Phone: +1-302-831-1944
Email: amer@cis.udel.edu

Martin Becke
University of Duisburg-Essen, Institute for Experimental Mathematics
Ellernstrasse 29
45326 Essen, Nordrhein-Westfalen
DE

Phone: +49-201-183-7667
Fax:   +49-201-183-7673
Email: martin.becke@uni-due.de

Thomas Dreibholz
University of Duisburg-Essen, Institute for Experimental Mathematics
Ellernstrasse 29
45326 Essen, Nordrhein-Westfalen
DE

Phone: +49-201-183-7637
Fax:   +49-201-183-7673
Email: dreibh@iem.uni-due.de
URI:   http://www.iem.uni-due.de/~dreibh/

Nasif Ekiz
University of Delaware, Computer and Information Sciences Department
Newark, DE  19716
US

Email: nekiz@udel.edu

Janardhan Iyengar
Franklin and Marshall College, Mathematics and Computer Science
PO Box 3003
Lancaster, Pennsylvania 17604-3003
US

Phone: +1-717-358-4774
Email: jiyengar@fandm.edu
URI: http://www.fandm.edu/jiyengar/

Preethi Natarajan
Cisco Systems
425 East Tasman Drive
San Jose, California 95134
US

Email: prenatar@cisco.com

Randall R. Stewart
Adara Networks
Chapin, SC 29036
US

Email: randall@lakerest.net

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

Email: tuexen@fh-muenster.de