Network Working Group
Internet-Draft
Intended status: Best Current Practice
Expires: March 13, 2014

R. Stewart
Adara Networks
M. Tuexen
I. Ruengeler
Muenster Univ. of Appl. Sciences
September 09, 2013

Stream Control Transmission Protocol (SCTP) Network Address Translation
draft-ietf-behave-sctpnat-09.txt

Abstract

Stream Control Transmission Protocol [RFC4960] provides a reliable communications channel between two end-hosts in many ways similar to TCP [RFC0793]. With the widespread deployment of Network Address Translators (NAT), specialized code has been added to NAT for TCP that allows multiple hosts to reside behind a NAT and yet use only a single globally unique IPv4 address, even when two hosts (behind a NAT) choose the same port numbers for their connection. This additional code is sometimes classified as Network Address and Port Translation or NAPT. To date, specialized code for SCTP has NOT yet been added to most NATs so that only pure NAT is available. The end result of this is that only one SCTP capable host can be behind a NAT.

This document describes an SCTP specific variant of NAT which provides similar features of NAPT in the single point and multi-point traversal scenario.

Table of Contents

1. Introduction

Stream Control Transmission Protocol [RFC4960] provides a reliable communications channel between two end-hosts in many ways similar to TCP [RFC0793]. With the widespread deployment of Network Address Translators (NAT), specialized code has been added to NAT for TCP that allows multiple hosts to reside behind a NAT and use private

addresses (see [RFC5735]) and yet use only a single globally unique IPv4 address, even when two hosts (behind a NAT) choose the same port numbers for their connection. This additional code is sometimes classified as Network Address and Port Translation or NAPT. To date, specialized code for SCTP has not yet been added to most NATs so that only true NAT is available. The end result of this is that only one SCTP capable host can be behind a NAT.

This document proposes an SCTP specific variant NAT that provides the NAPT functionality without changing SCTP port numbers. The authors feel it is possible and desirable to make these changes for a number of reasons.

- It is desirable for SCTP internal end-hosts on multiple platforms to be able to share a NAT's public IP address, much as TCP does today.
- If a NAT does not need to change any data within an SCTP packet it will reduce the processing burden of NAT'ing SCTP by NOT needing to execute the CRC32c checksum required by SCTP.
- Not having to touch the IP payload makes the processing of ICMP messages in NATs easier.

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. Terminology

For this discussion we will use several terms, which we will define and point out in Figure 1.

Private-Address (Priv-Addr): The private address that is known to the internal host.

Internal-Port (Int-Port): The port number that is in use by the host holding the Private-Address.

Internal-VTag (Int-VTag): The Verification Tag that the internal host has chosen for its communication. The VTag is a unique 32 bit tag that must accompany any incoming SCTP packet for this association to the Private-Address.

External-Address (Ext-Addr): The address that an internal host is attempting to contact.

External-Port (Ext-Port): The port number of the peer process at the External-Address.

External-VTag (Ext-VTag): The Verification Tag that the host holding the External-Address has chosen for its communication. The VTag is a unique 32 bit tag that must accompany any incoming SCTP packet for this association to the External-Address.

Public-Address (Pub-Addr): The public address assigned to the NAT box which it uses as a source address when sending packets towards the External-Address.

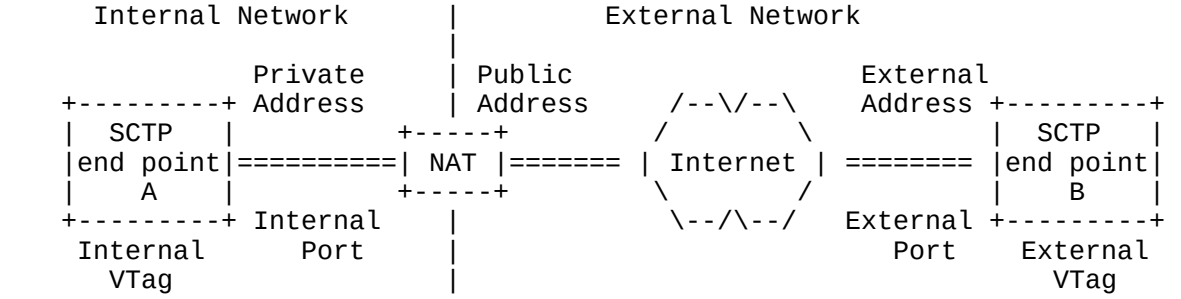


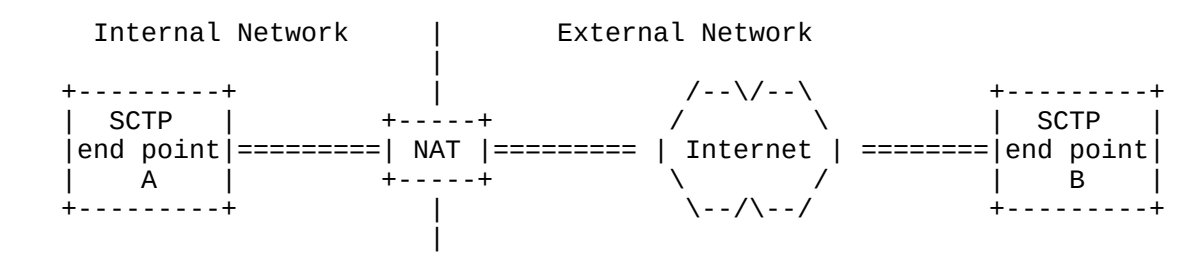
Figure 1: Architecture

4. SCTP NAT Traversal Scenarios

This section defines the notion of single and multi-point NAT traversal.

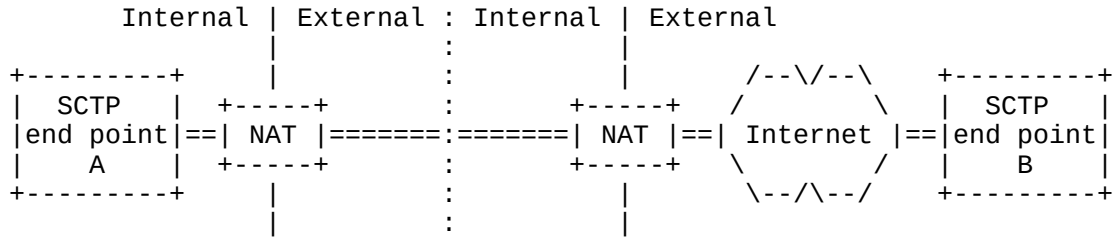
4.1. Single Point Traversal

In this case, all packets in the SCTP association go through a single NAT, as shown below:



Single NAT scenario

A variation of this case is shown below, i.e., multiple NATs in a single path:



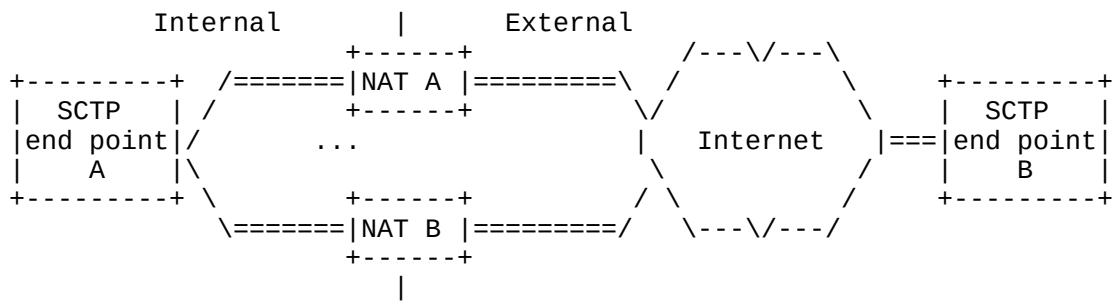
Serial NATs scenario

In this single point traversal scenario, we must acknowledge that while one of the main benefits of SCTP multi-homing is redundant paths, the NAT function represents a single point of failure in the path of the SCTP multi-home association. However, the rest of the path may still benefit from path diversity provided by SCTP multi-homing.

The two SCTP endpoints in this case can be either single-homed or multi-homed. However, the important thing is that the NAT (or NATs) in this case sees all the packets of the SCTP association.

4.2. Multi Point Traversal

This case involves multiple NATs and each NAT only sees some of the packets in the SCTP association. An example is shown below:



Parallel NATs scenario

This case does NOT apply to a single-homed SCTP association (i.e., BOTH endpoints in the association use only one IP address). The advantage here is that the existence of multiple NAT traversal points can preserve the path diversity of a multi-homed association for the

entire path. This in turn can improve the robustness of the communication.

5. Limitations of Classical NAPT for SCTP

Using classical NAPT may result in changing one of the SCTP port numbers during the processing which requires the recomputation of the transport layer checksum. Whereas for UDP and TCP this can be done very efficiently, for SCTP the checksum (CRC32c) over the entire packet needs to be recomputed. This would add considerable to the NAT computational burden, however hardware support may mitigate this in some implementations.

An SCTP endpoint may have multiple addresses but only has a single port number. To make multipoint traversal work, all the NATs involved must recognize the packets they see as belonging to the same SCTP association and perform port number translation in a consistent way. One possible way of doing this is to use pre-defined table of ports and addresses configured within each NAT. Other mechanisms could make use of NAT to NAT communication. Such mechanisms are considered by the authors not to be deployable on a wide scale base and thus not a recommended solution. Therefore the SCTP variant of NAT has been developed.

6. The SCTP Specific Variant of NAT

In this section we assume that we have multiple SCTP capable hosts behind a NAT which has one Public-Address. Furthermore we are focusing in this section on the single point traversal scenario.

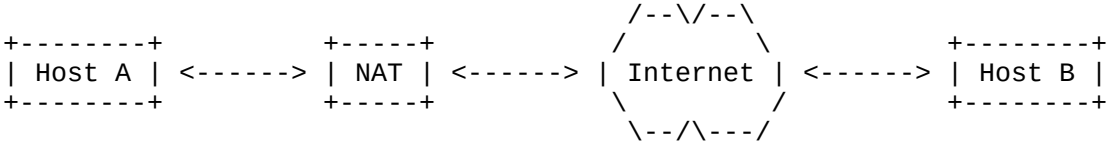
The modification of SCTP packets sent to the public Internet is easy. The source address of the packet has to be replaced with the Public-Address. It may also be necessary to establish some state in the NAT box to handle incoming packets, which is discussed later.

For SCTP packets coming from the public Internet the destination address of the packets has to be replaced with the Private-Address of the host the packet has to be delivered to. The lookup of the Private-Address is based on the External-VTag, External-Port, External-Address, Internal-VTag and the Internal-Port.

For the SCTP NAT processing the NAT box has to maintain a table of Internal-VTag, Internal-Port, Private-Address, External-VTag, External-Port and whether the restart procedure is disabled or not. An entry in that table is called a NAT state control block. The function Create() obtains the just mentioned parameters and returns a NAT-State control block.

The entries in this table fulfill some uniqueness conditions. There must not be more than one entry with the same pair of Internal-Port and External-Port. This rule can be relaxed, if all entries with the same Internal-Port and External-Port have the support for the restart procedure enabled. In this case there must be no more than one entry with the same Internal-Port, External-Port and Ext-VTag and no more than one entry with the same Internal-Port, External-Port and Int-VTag.

The processing of outgoing SCTP packets containing an INIT-chunk is described in the following figure. The scenario shown is valid for all message flows in this section.



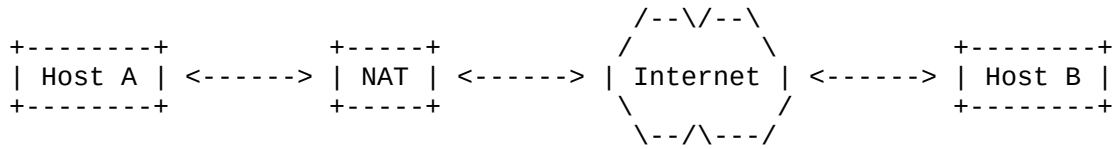

```
              INIT[Initiate-Tag]
Priv-Addr:Int-Port ------> Ext-Addr:Ext-Port
                  Ext-VTag=0

                   Create(Initiate-Tag, Int-Port, Priv-Addr, 0)
                   Returns(NAT-State control block)

            Translate To:

                        INIT[Initiate-Tag]
            Pub-Addr:Int-Port ------> Ext-Addr:Ext-Port
                            Ext-VTag=0
```

It should be noted that normally a NAT control block will be created. However, it is possible that there is already a NAT control block with the same External-Address, External-Port, Internal-Port, and Internal-VTag but different Private-Address. In this case the INIT SHOULD be dropped by the NAT and an ABORT SHOULD be sent back to the SCTP host with the M-Bit set and an appropriate error cause (see [I-D.ietf-tsvwg-natsupp] for the format). The source address of the packet containing the ABORT chunk MUST be the destination address of the packet containing the INIT chunk.

It is also possible that a connection to External-Address and External-Port exists without an Internal-VTag conflict but the

External-Address does not support the DISABLE_RESTART feature (noted in the NAT control block when the prior connection was established). In such a case the INIT SHOULD be dropped by the NAT and an ABORT SHOULD be sent back to the SCTP host with the M-Bit set and an appropriate error cause (see [I-D.ietf-tsvwg-natsupp] for the format).

The processing of outgoing SCTP packets containing no INIT-chunk is described in the following figure.

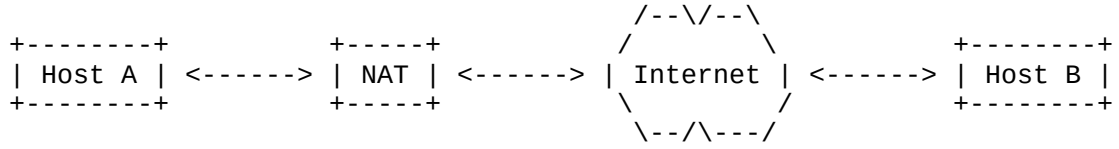


Priv-Addr:Int-Port ------> Ext-Addr:Ext-Port
Ext-VTag

Translate To:

Pub-Addr:Int-Port ------> Ext-Addr:Ext-Port
Ext-VTag

The processing of incoming SCTP packets containing INIT-ACK chunks is described in the following figure. The Lookup() function getting as input the Internal-VTag, Internal-Port, External-VTag (=0), External-Port, and External-Address, returns the corresponding entry of the NAT table and updates the External-VTag by substituting it with the value of the Initiate-Tag of the INIT-ACK chunk. The wildcard character signifies that the parameter's value is not considered in the Lookup() function or changed in the Update() function, respectively.

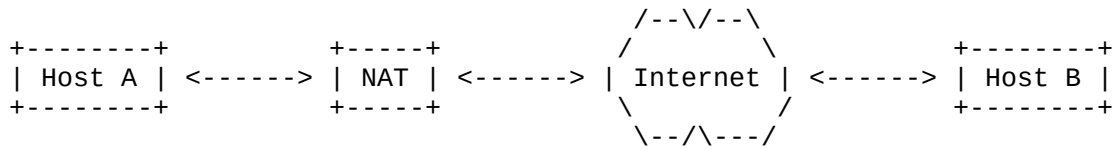


INIT-ACK[Initiate-Tag]
Pub-Addr:Int-Port <---- Ext-Addr:Ext-Port
Int-VTag

```
          Lookup(Int-VTag, Int-Port, *, 0, Ext-Port)
          Update(*, *, *, Initiate-Tag, *)

          Returns(NAT-State control block containing Private-Address)

                INIT-ACK[Initiate-Tag]
   Priv-Addr:Int-Port <------ Ext-Addr:Ext-Port
                      Int-VTag
```

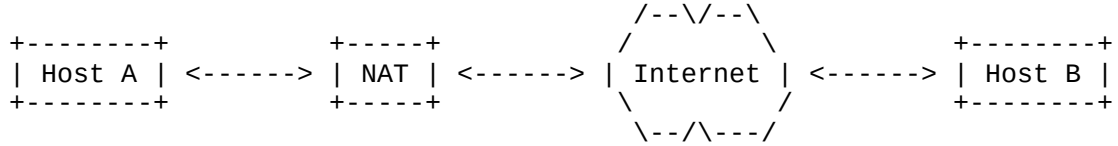
In the case Lookup fails, the SCTP packet is dropped. The Update routine inserts the External-VTag (the Initiate-Tag of the INIT-ACK chunk) in the NAT state control block.

The processing of incoming SCTP packets containing an ABORT or SHUTDOWN-COMPLETE chunk with the T-Bit set is described in the following figure.




```
                              Pub-Addr:Int-Port <------ Ext-Addr:Ext-Port
                                                Ext-VTag

          Lookup(0, Int-Port, *, Ext-VTag, Ext-Port)

          Returns(NAT-State control block containing Private-Address)

   Priv-Addr:Int-Port <------ Ext-Addr:Ext-Port
                      Ext-VTag
```

The processing of other incoming SCTP packets is described in the following figure.



```
                          Pub-Addr:Int-Port <------ Ext-Addr:Ext-Port
                                                    Int-VTag

          Lookup(Int-VTag, Int-Port, *, *, Ext-Port)

          Returns(NAT-State control block containing Local-Address)

   Priv-Addr:Int-Port <------ Ext-Addr:Ext-Port
                      Int-VTag
```

For an incoming packet containing an INIT-chunk a table lookup is made only based on the addresses and port numbers. If an entry with an External-VTag of zero is found, it is considered a match and the External-VTag is updated.

This allows the handling of INIT-collision through NAT.

7. NAT to SCTP

This document at various places discusses the sending of specialized SCTP chunks (e.g. an ABORT with M-Bit set). These chunks and procedures are not defined in this document, but instead are defined in [I-D.ietf-tsvwg-natsupp]. The NAT implementer should refer to [I-D.ietf-tsvwg-natsupp] for detailed descriptions of packet formats and procedures.

8. Handling of Fragmented SCTP Packets

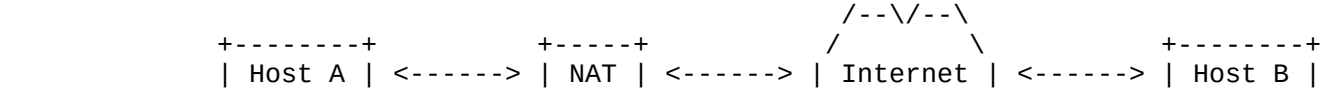
A NAT box MUST support IP reassembly of received fragmented SCTP packets. The fragments may arrive in any order.

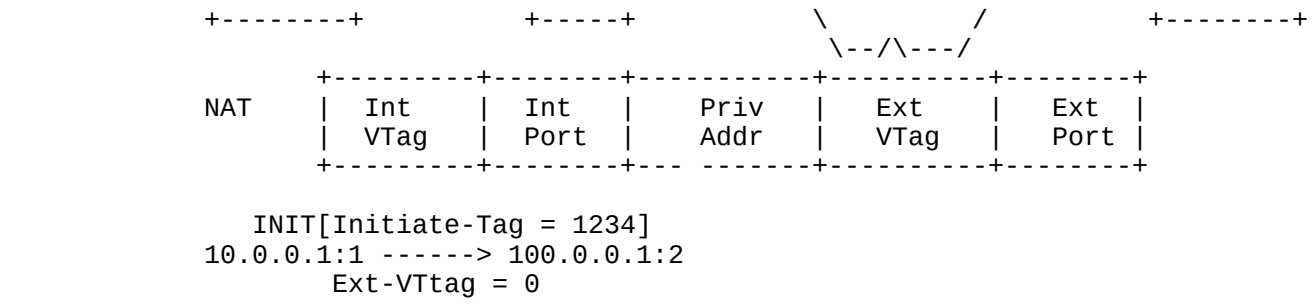
When an SCTP packet has to be fragmented by the NAT box and the IP header forbids fragmentation a corresponding ICMP packet SHOULD be sent.

9. Various Examples of NAT Traversals

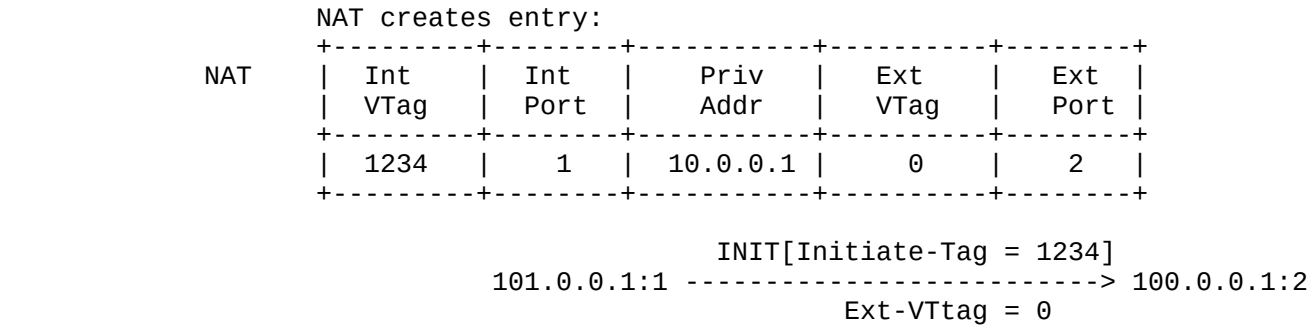
9.1. Single-homed Client to Single-homed Server

The internal client starts the association with the external server via a four-way-handshake. Host A starts by sending an INIT chunk.

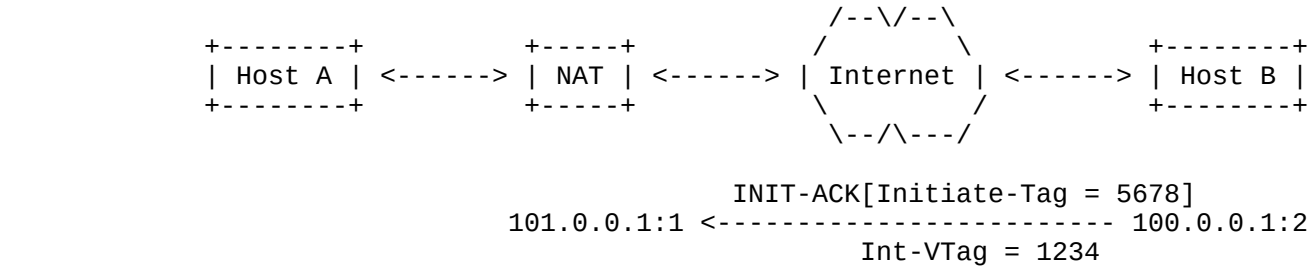




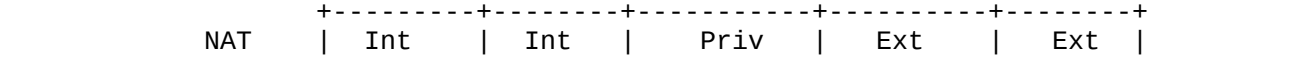
A NAT entry is created, the source address is substituted and the packet is sent on:



Host B receives the INIT and sends an INIT-ACK with the NAT's external address as destination address.



NAT updates entry:



```
|  VTag   |  Port  |    Addr   |   VTag   |   Port |
+---------+--------+-----------+----------+--------+
|  1234   |    1   |  10.0.0.1 |    5678  |    2   |
+---------+--------+-----------+----------+--------+
```

```
INIT-ACK[Initiate-Tag = 5678]
10.0.0.1:1 <------ 100.0.0.1:2
          Int-VTag = 1234
```

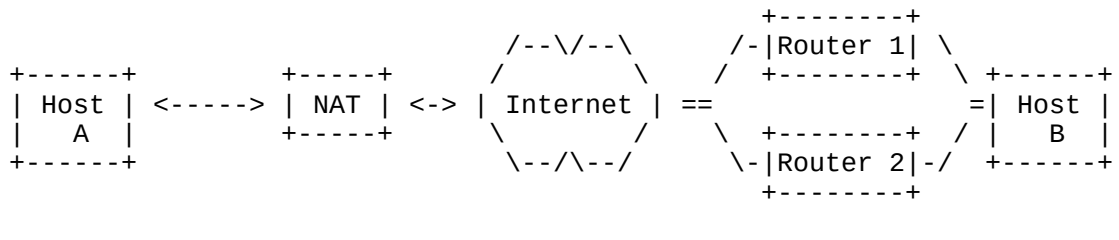
The handshake finishes with a COOKIE-ECHO acknowledged by a COOKIE-ACK.

```
                                      /--\/--\
+--------+          +-----+          /        \           +--------+
| Host A | <------> | NAT | <------> | Internet | <------> | Host B |
+--------+          +-----+          \        /           +--------+
                                      \--/\---/
```

```
         COOKIE-ECHO
10.0.0.1:1 ------> 100.0.0.1:2
       Ext-VTag = 5678

                                    COOKIE-ECHO
                  101.0.0.1:1 -------------------------> 100.0.0.1:2
                                   Ext-VTag = 5678


                                    COOKIE-ACK
                  101.0.0.1:1 <------------------------- 100.0.0.1:2
                                    Int-VTag = 1234

            COOKIE-ACK
10.0.0.1:1 <------ 100.0.0.1:2
           Int-VTag = 1234
```

9.2. Single-homed Client to Multi-homed Server

The internal client is single-homed whereas the external server is multi-homed. The client (Host A) sends an INIT like in the single-homed case.



NAT	Int VTag	Int Port	Priv Addr	Ext VTag	Ext Port

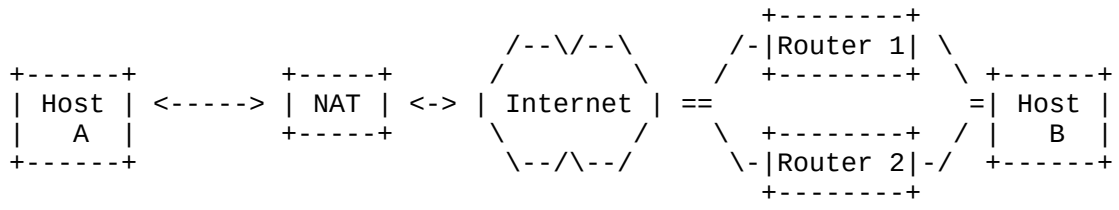
INIT[Initiate-Tag = 1234]
10.0.0.1:1 ---> 100.0.0.1:2
Ext-VTag = 0

NAT creates entry:

NAT	Int VTag	Int Port	Priv Addr	Ext VTag	Ext Port
	1234	1	10.0.0.1	0	2

INIT[Initiate-Tag = 1234]
101.0.0.1:1 ----------> 100.0.0.1:2
Ext-VTag = 0

The server (Host B) includes its two addresses in the INIT-ACK chunk, which results in two NAT entries.



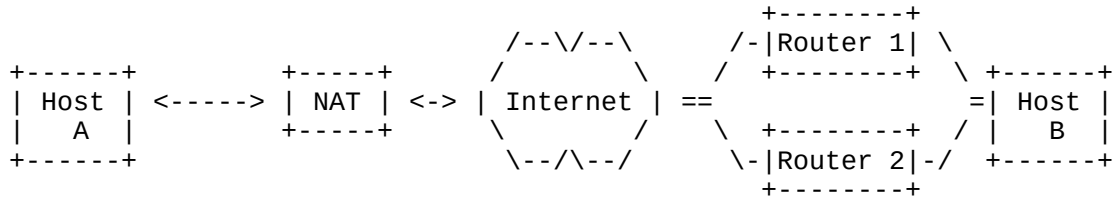
```
                  INIT-ACK[Initiate-tag = 5678, IP-Addr = 100.1.0.1]
               101.0.0.1:1 <---------------------------- 100.0.0.1:2
                                    Int-VTag = 1234
```

NAT does need to change the table for second address:

```
          +---------+--------+-----------+----------+--------+
     NAT  |  Int    |  Int   |    Priv   |   Ext    |   Ext  |
          |  VTag   |  Port  |    Addr   |   VTag   |   Port |
          +---------+--------+-----------+----------+--------+
          |  1234   |    1   |  10.0.0.1 |    5678  |    2   |
          +---------+--------+-----------+----------+--------+
```

```
     INIT-ACK[Initiate-Tag = 5678]
     10.0.0.1:1 <--- 100.0.0.1:2
              Int-VTag = 1234
```

The handshake finishes with a COOKIE-ECHO acknowledged by a COOKIE-ACK.



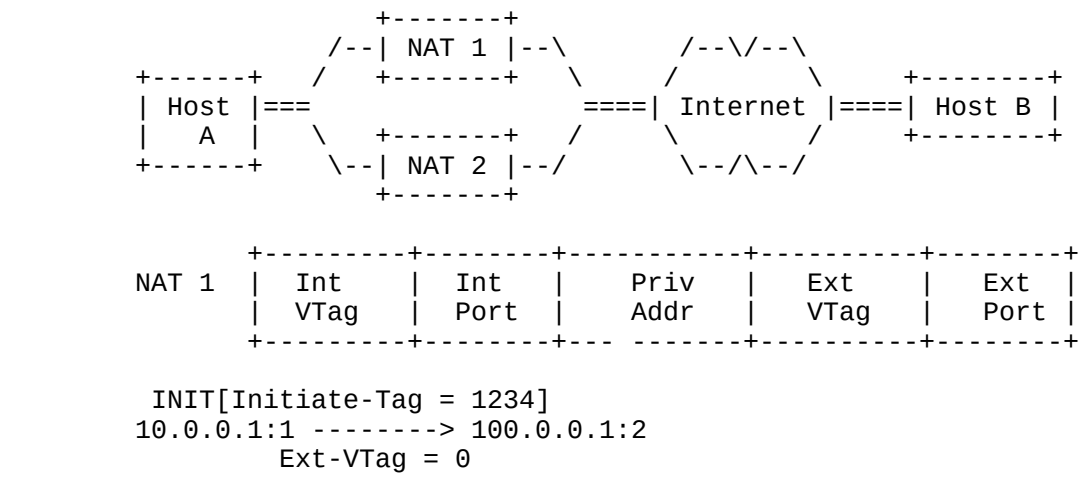

```
       COOKIE-ECHO
10.0.0.1:1 ---> 100.0.0.1:2
       ExtVTag = 5678
```

```
                                    COOKIE-ECHO
                  101.0.0.1:1 ----------------------------> 100.0.0.1:2
                                    Ext-VTag = 5678

                                    COOKIE-ACK
                  101.0.0.1:1 <---------------------------- 100.0.0.1:2
                                    Int-VTag = 1234
```

```
          COOKIE-ACK
  10.0.0.1:1 <--- 100.0.0.1:2
          Int-VTag = 1234
```

9.3. Multihomed Client and Server

The client (Host A) sends an INIT to the server (Host B), but does not include the second address.

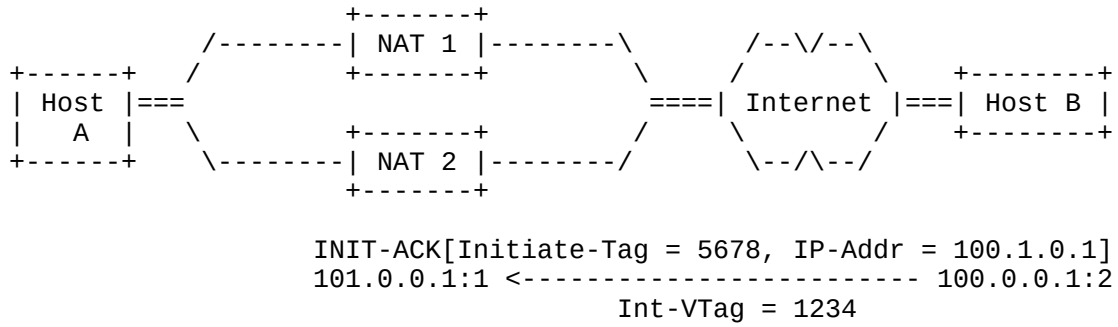


NAT 1 creates entry:

NAT 1	Int VTag	Int Port	Priv Addr	Ext VTag	Ext Port
	1234	1	10.0.0.1	0	2

```
                              INIT[Initiate-Tag = 1234]
                  101.0.0.1:1 ----------------------> 100.0.0.1:2
                                      ExtVTag = 0
```

Host B includes its second address in the INIT-ACK, which results in two NAT entries in NAT 1.

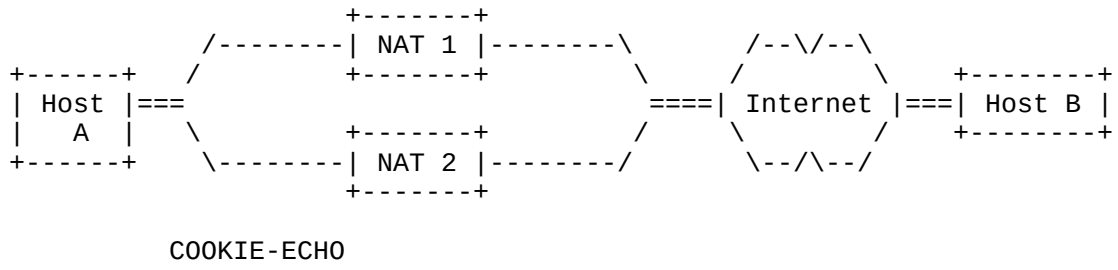


NAT 1 does not need to update the table for second address:

NAT 1	Int VTag	Int Port	Priv Addr	Ext VTag	Ext Port
	1234	1	10.0.0.1	5678	2

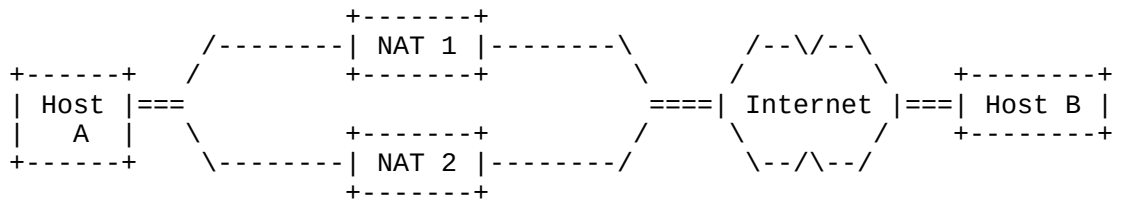
INIT-ACK[Initiate-Tag = 5678]
10.0.0.1:1 <---------100.0.0.1:2
Int-VTag = 1234

The handshake finishes with a COOKIE-ECHO acknowledged by a COOKIE-ACK.



```
10.0.0.1:1 --------> 100.0.0.1:2
          Ext-VTag = 5678

                                    COOKIE-ECHO
                        101.0.0.1:1 --------------------> 100.0.0.1:2
                                    Ext-VTag = 5678


                                     COOKIE-ACK
                        101.0.0.1:1 <-------------------- 100.0.0.1:2
                                    Int-VTag = 1234

             COOKIE-ACK
10.0.0.1:1 <------- 100.0.0.1:2
           Int-VTag = 1234
```

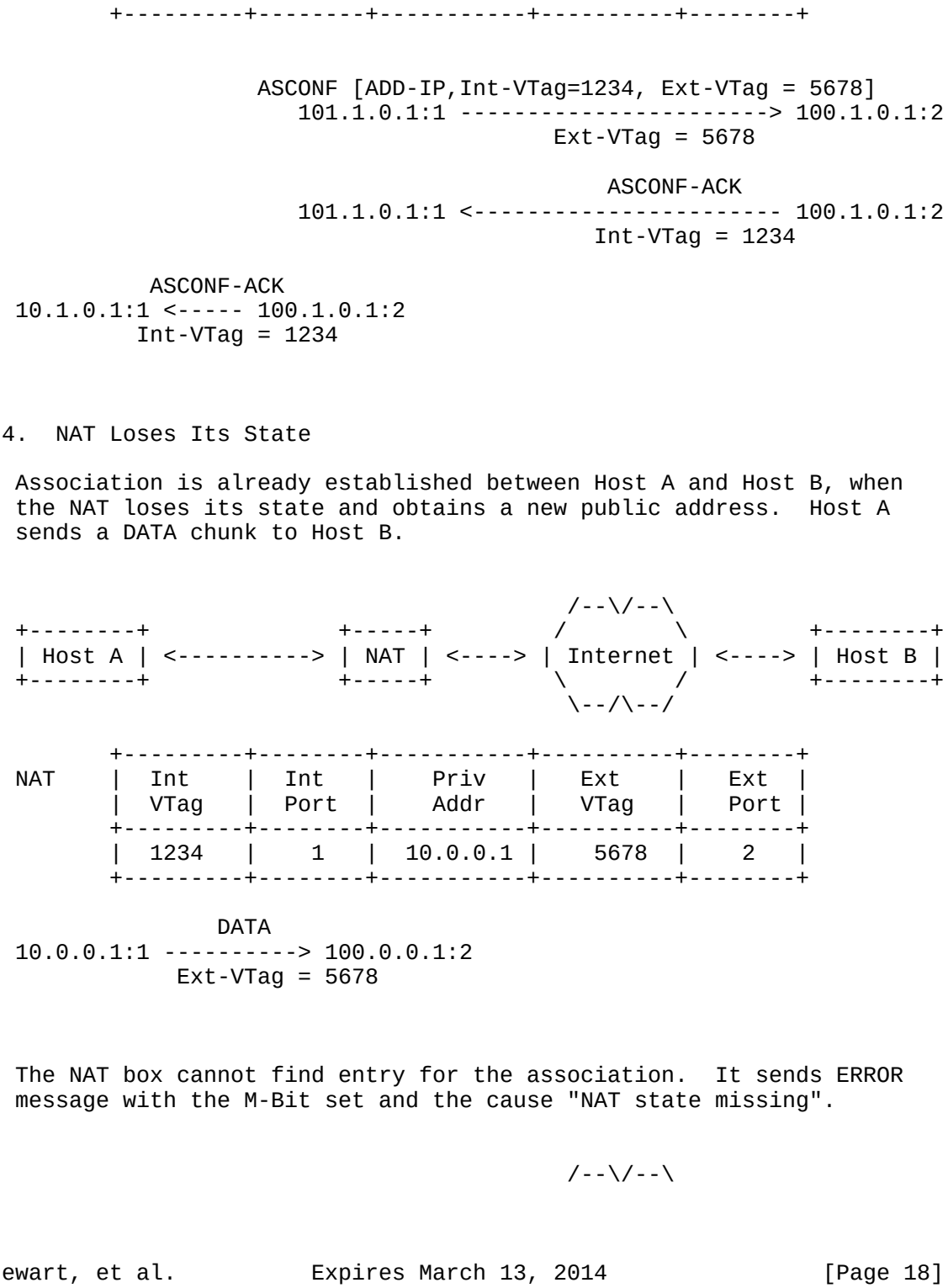
Host A announces its second address in an ASCONF chunk. The address parameter contains an undefined address (0) to indicate that the source address should be added. The lookup address parameter within the ASCONF chunk will also contain the pair of VTags (external and internal) so that the NAT may populate its table completely with this single packet.




```
ASCONF [ADD-IP=0.0.0.0, INT-VTag=1234, Ext-VTag = 5678]
10.1.0.1:1 --------> 100.1.0.1:2
          Ext-VTag = 5678
```

NAT 2 creates complete entry:

NAT 2	Int VTag	Int Port	Priv Addr	Ext VTag	Ext Port
	1234	1	10.1.0.1	5678	2

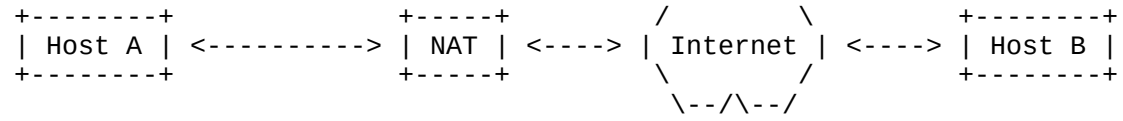


9.4. NAT Loses Its State

Association is already established between Host A and Host B, when the NAT loses its state and obtains a new public address. Host A sends a DATA chunk to Host B.

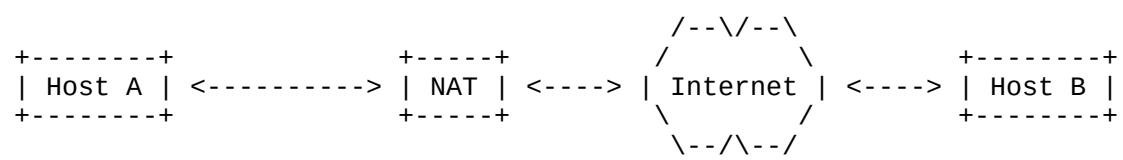
NAT	Int VTag	Int Port	Priv Addr	Ext VTag	Ext Port
	1234	1	10.0.0.1	5678	2

The NAT box cannot find entry for the association. It sends ERROR message with the M-Bit set and the cause "NAT state missing".




```
  ERROR [M-Bit, NAT state missing]
10.0.0.1:1 <---------- 100.0.0.1:2
          Ext-VTag = 5678
```

On reception of the ERROR message, Host A sends an ASCONF chunk indicating that the former information has to be deleted and the source address of the actual packet added.

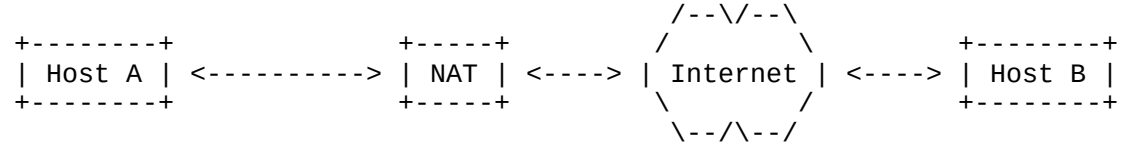


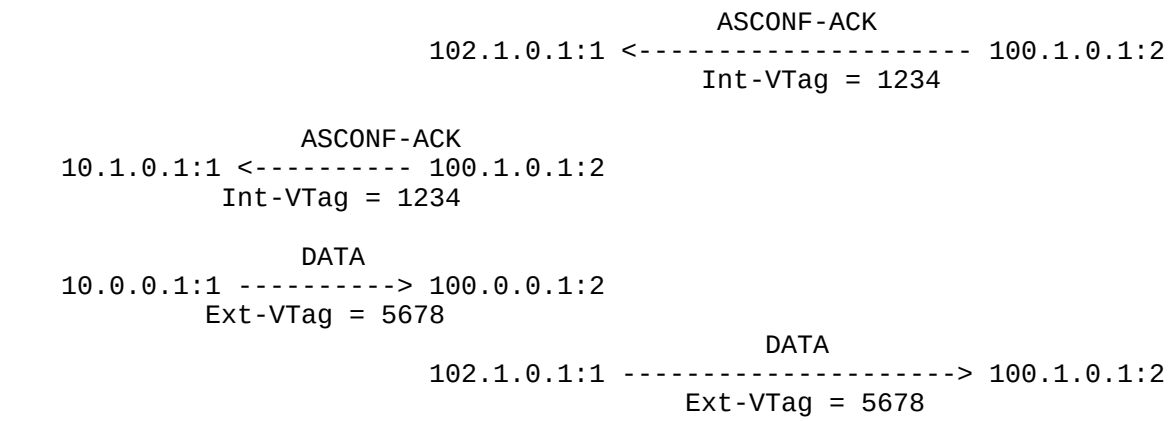

```
ASCONF [ADD-IP,DELETE-IP,Int-VTag=1234, Ext-VTag = 5678]
10.0.0.1:1 ----------> 100.1.0.1:2
          Ext-VTag = 5678
```

NAT	Int VTag	Int Port	Priv Addr	Ext VTag	Ext Port
	1234	1	10.0.0.1	5678	2

```
        ASCONF [ADD-IP,DELETE-IP,Int-VTag=1234, Ext-VTag = 5678]
                102.1.0.1:1 ---------------------> 100.1.0.1:2
                                    Ext-VTag = 5678
```

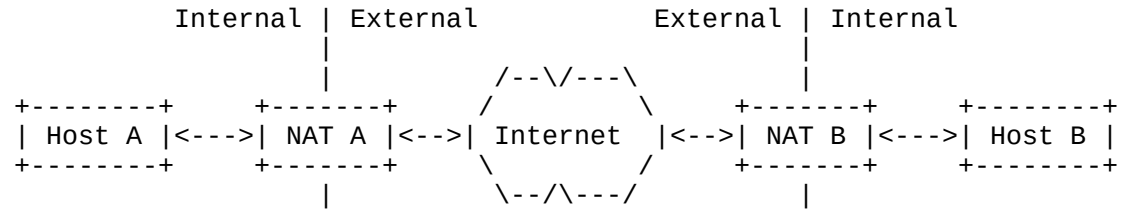
Host B adds the new source address and deletes all former entries.





9.5. Peer-to-Peer Communication

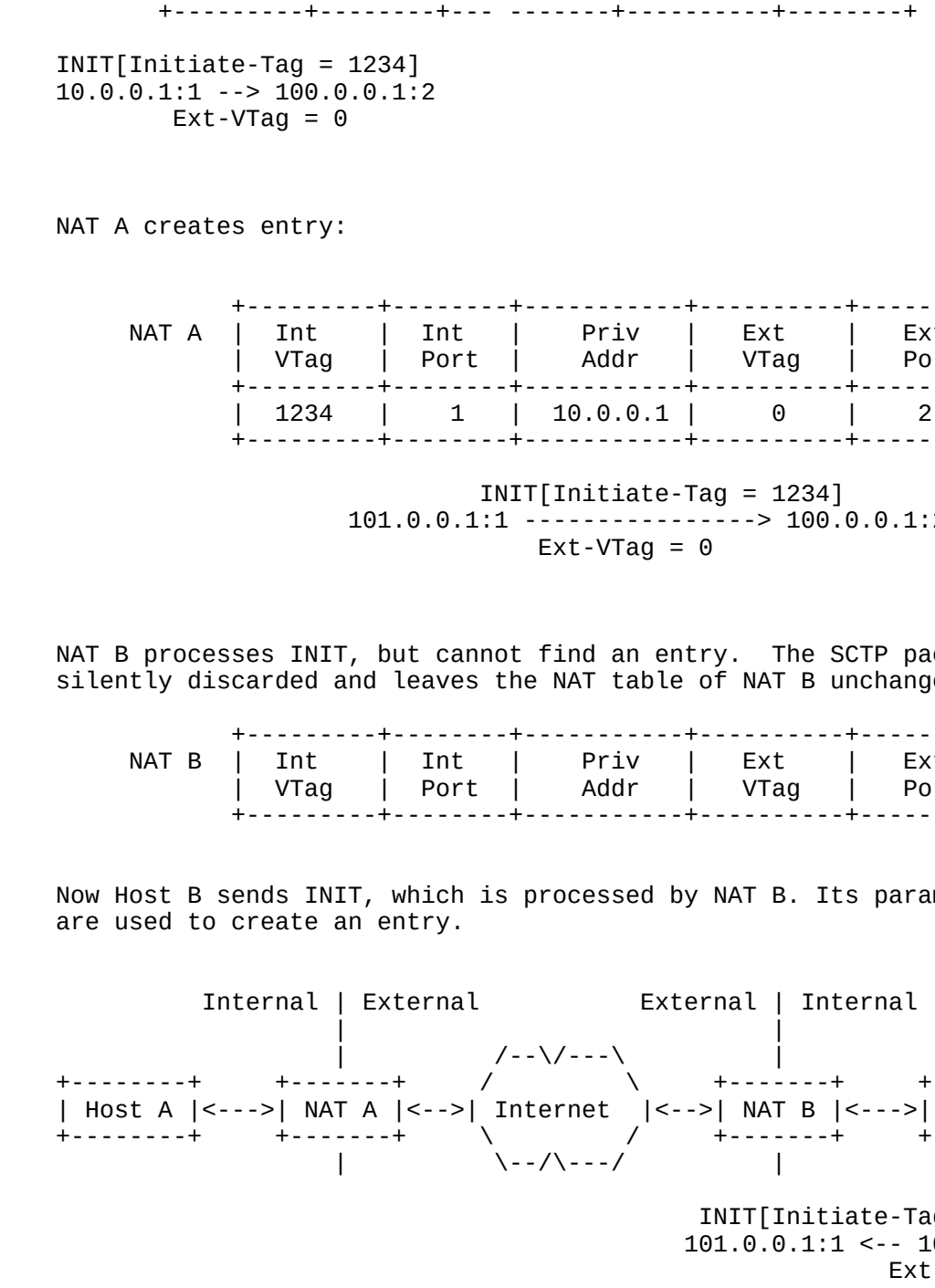
If two hosts are behind NATs, they have to get knowledge of the peer's public address. This can be achieved with a so-called rendezvous server. Afterwards the destination addresses are public, and the association is set up with the help of the INIT collision. The NAT boxes create their entries according to their internal peer's point of view. Therefore, NAT A's Internal-VTag and Internal-Port are NAT B's External-VTag and External-Port, respectively. The naming of the verification tag in the packet flow is done from the sending peer's point of view.



NAT-Tables

	Int VTag	Int Port	Priv Addr	Ext VTag	Ext Port
NAT A					
NAT B					

	Int v-tag	Int port	Priv addr	Ext v-tag	Ext port
NAT A					
NAT B					


INIT[Initiate-Tag = 1234]
10.0.0.1:1 --> 100.0.0.1:2
Ext-VTag = 0
NAT A creates entry:
NAT A
Int VTag
Int Port
Priv Addr
Ext VTag
Ext Port
1234
1
10.0.0.1
0
2
INIT[Initiate-Tag = 1234]
101.0.0.1:1 ----------> 100.0.0.1:2
Ext-VTag = 0
NAT B processes INIT, but cannot find an entry. The SCTP packet is silently discarded and leaves the NAT table of NAT B unchanged.
NAT B
Int VTag
Int Port
Priv Addr
Ext VTag
Ext Port
Now Host B sends INIT, which is processed by NAT B. Its parameters are used to create an entry.
Internal | External
External | Internal
Host A
NAT A
Internet
NAT B
Host B
INIT[Initiate-Tag = 5678]
101.0.0.1:1 <-- 10.1.0.1:2
Ext-VTag = 0

```
        +---------+--------+-----------+----------+--------+
NAT B   |  Int    |  Int   |    Priv   |   Ext    |   Ext  |
        |  VTag   |  Port  |    Addr   |   VTag   |   Port |
        +---------+--------+-----------+----------+--------+
        |  5678   |    2   |  10.1.0.1 |     0    |    1   |
        +---------+--------+-----------+----------+--------+
```

```
                      INIT[Initiate-Tag = 5678]
              101.0.0.1:1  <---------------- 100.0.0.1:2
                                  Ext-VTag = 0
```

NAT A processes INIT. As the outgoing INIT of Host A has already created an entry, the entry is found and updated:

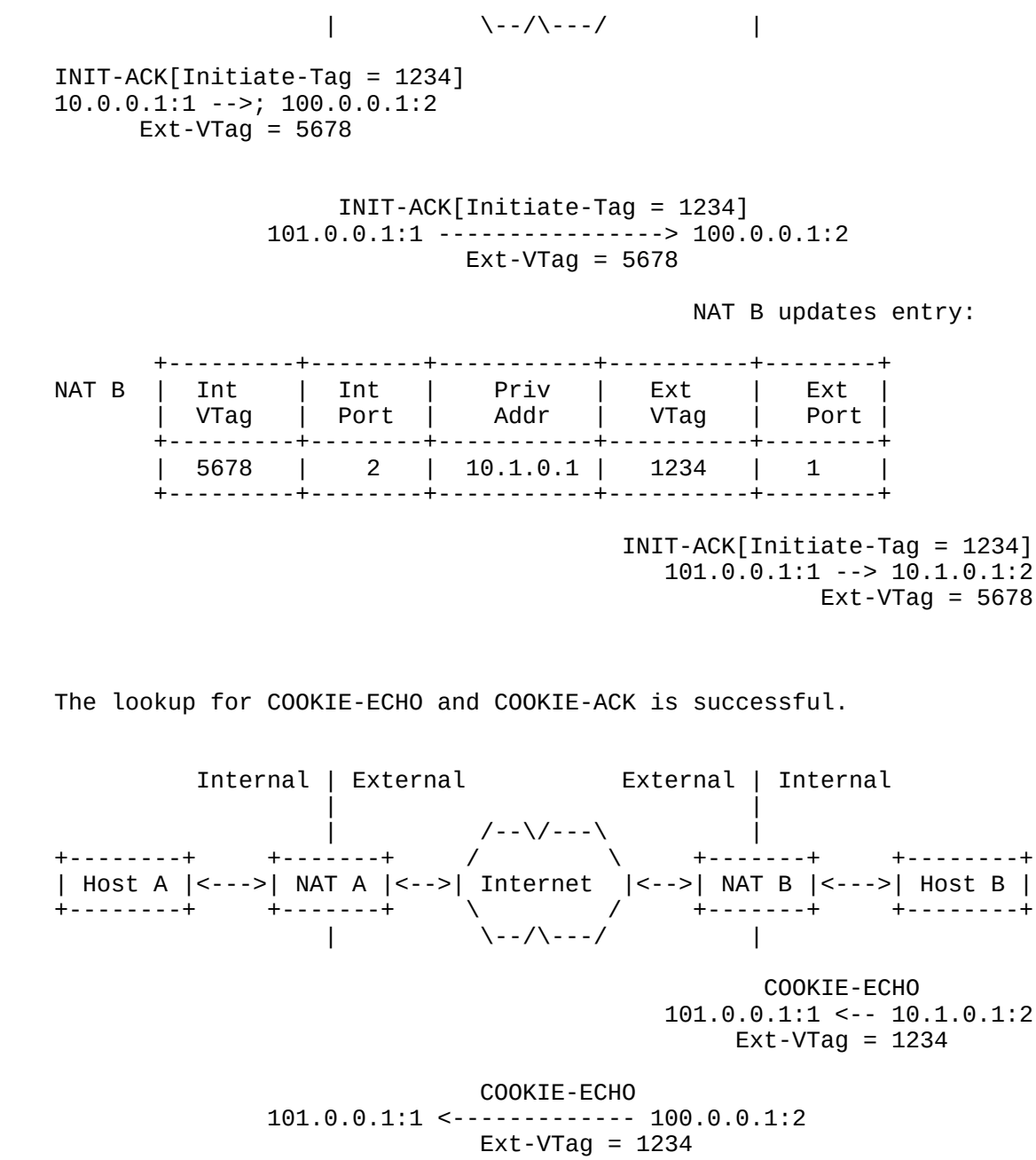
```
          Internal | External           External | Internal
                   |                             |
                   |          /--\/---\          |
+--------+     +-------+     /          \     +-------+     +--------+
| Host A |<--->| NAT A |<-->| Internet   |<-->| NAT B |<--->| Host B |
+--------+     +-------+     \          /     +-------+     +--------+
                   |          \--/\---/          |
```

```
              VTag != Int-VTag, but Ext-VTag == 0, find entry.
        +---------+--------+-----------+----------+--------+
NAT A   |  Int    |  Int   |    Priv   |   Ext    |   Ext  |
        |  VTag   |  Port  |    Addr   |   VTag   |   Port |
        +---------+--------+-----------+----------+--------+
        |   1234  |   1    |  10.0.0.1 |   5678   |    2   |
        +---------+--------+-----------+----------+--------+
```

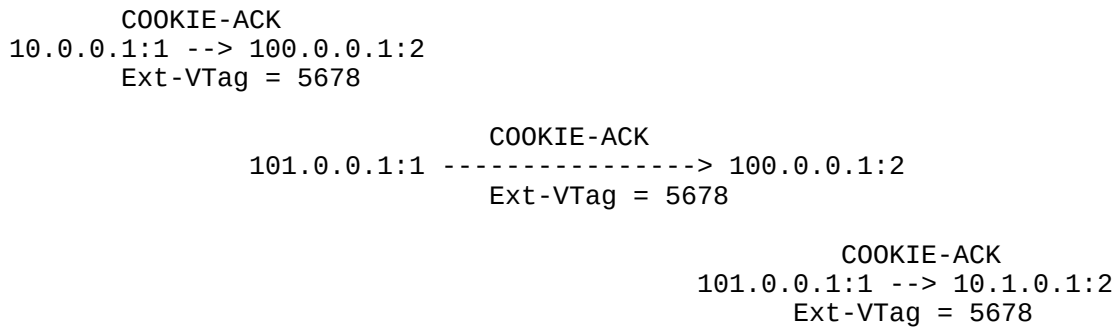
```
 INIT[Initiate-tag = 5678]
10.0.0.1:1 <-- 100.0.0.1:2
          Ext-VTag = 0
```

Host A send INIT-ACK, which can pass through NAT B:

```
          Internal | External           External | Internal
                   |                             |
                   |          /--\/---\          |
+--------+     +-------+     /          \     +-------+     +--------+
| Host A |<--->| NAT A |<-->| Internet   |<-->| NAT B |<--->| Host B |
+--------+     +-------+     \          /     +-------+     +--------+
```



The lookup for COOKIE-ECHO and COOKIE-ACK is successful.



10. IANA Considerations

This document requires no actions from IANA.

11. Security Considerations

State maintenance within a NAT is always a subject of possible Denial Of Service attacks. This document recommends that at a minimum a NAT runs a timer on any SCTP state so that old association state can be cleaned up.

12. Acknowledgments

The authors wish to thank Jason But Bryan Ford, David Hayes, Alfred Hines, Henning Peters, Timo Voelker, Dan Wing, and Qiaobing Xie for their invaluable comments.

13. References

13.1. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[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.

[I-D.ietf-tsvwg-natsupp]

Stewart, R., Tuexen, M., and I. Ruengeler, "Stream Control Transmission Protocol (SCTP) Network Address Translation Support", draft-ietf-tsvwg-natsupp-05 (work in progress), February 2013.

13.2. Informative References

[RFC5735] Cotton, M. and L. Vegoda, "Special Use IPv4 Addresses", RFC 5735, January 2010.

Authors' Addresses

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

Irene Ruengeler
Muenster University of Applied Sciences
Stegerwaldstrasse 39
48565 Steinfurt
DE

Email: i.ruengeler@fh-muenster.de

DCCP Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 27, 2012

T. Phelan
Sonus
G. Fairhurst
University of Aberdeen
C. Perkins
University of Glasgow
June 25, 2012

Datagram Congestion Control Protocol (DCCP) Encapsulation for NAT Traversal (DCCP-UDP)
draft-ietf-dccp-udpencap-11

Abstract

This document specifies an alternative encapsulation of the Datagram Congestion Control Protocol (DCCP), referred to as DCCP-UDP. This encapsulation allows DCCP to be carried through the current generation of Network Address Translation (NAT) middleboxes without modification of those middleboxes. This document also updates the SDP information for DCCP defined in RFC 5762.

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 December 27, 2012.

Copyright Notice

Table of Contents

1. Introduction

The Datagram Congestion Control Protocol (DCCP) [RFC4340] is a transport-layer protocol that provides upper layers with the ability to use non-reliable congestion-controlled flows. The current specification for DCCP specifies a direct native encapsulation in IPv4 or IPv6 packets.

DCCP support has been specified for devices that use Network Address Translation (NAT) or Network Address and Port Translation (NAPT) [RFC5597]. However, there is a significant installed base of NAT/NAPT devices that do not support RFC 5597. It is therefore useful to have an encapsulation for DCCP that is compatible with this installed base of NAT/NAPT devices that support [RFC4787], but do not support RFC 5597. This document specifies that encapsulation, which is referred to as DCCP-UDP. For convenience, the standard encapsulation for DCCP [RFC4340] (including [RFC5596] as required) is referred to as DCCP-STD.

The encapsulation described in this document may also be used as a transition mechanism to enable support for DCCP in devices that support UDP, but do not yet natively support DCCP. This also allows the DCCP transport to be implemented within an application using DCCP-UDP.

The document also updates the SDP specification for DCCP to convey the encapsulation type. In this respect only, it updates the method in [RFC5762].

The DCCP-UDP encapsulation specified in this document supports all of the features contained in DCCP-STD, but with limited functionality for partial checksums.

Network optimisations for DCCP-STP and UDP may need to be updated to allow these optimisations to take advantage of DCCP-UDP. Encapsulation with an additional UDP protocol header can complicate or prevent inspection of DCCP header fields by equipment along the network path in the case where multiple DCCP connections share the same UDP 4-tuple. For example, routers that wish to identify DCCP ports to perform Equal-Cost Multi-Path routing, ECMP, network devices that wish to inspect DCCP ports to inform algorithms for sharing the network load across multiple links; firewalls that wish to inspect DCCP ports and service codes to inform algorithms that implement access rules; media gateways that inspect SDP information to derive characteristics of the transport and session, etc.

2. Terminology

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. DCCP-UDP

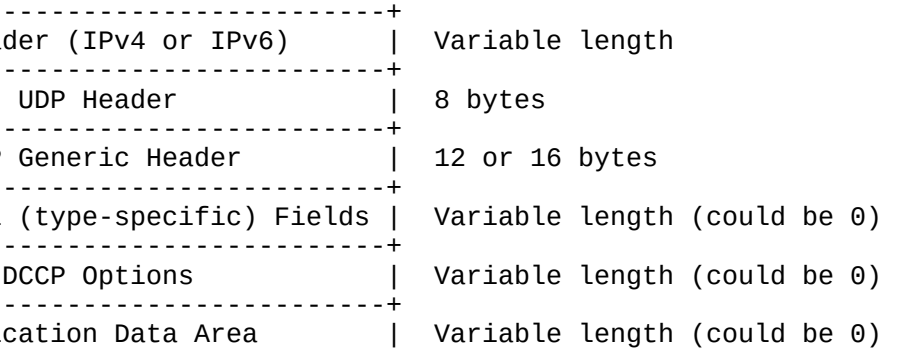
The basic approach is to insert a UDP [RFC0768] header between the IP header and the DCCP packet. Note that this is not a tunneling approach. The IP addresses of the communicating end systems are carried in the IP header. The method does not embed additional IP addresses.

The method is designed to support use when these addresses are modified by a device that implements NAT/NAPT. A NAT translates the IP addresses, which impacts the transport-layer checksum. A NAPT device may also translate the port values (usually the source port). In both cases, the outer transport header that includes these values would need to be updated by the NAT/NAPT.

A device offering or using DCCP services via DCCP-UDP encapsulation listens on a UDP port (default port, XXX IANA PORT XXX), or may bind to a specified port utilising out-of-band signalling, such as the Session Description Protocol (SDP). The DCCP-UDP server accepts incoming packets over the UDP transport and passes the received packets to the DCCP protocol module, after removing the UDP encapsulation.

A DCCP implementation endpoint may simultaneously provide services over any or all combinations of DCCP-STD and/or DCCP-UDP encapsulations with IPv4 and/or IPv6.

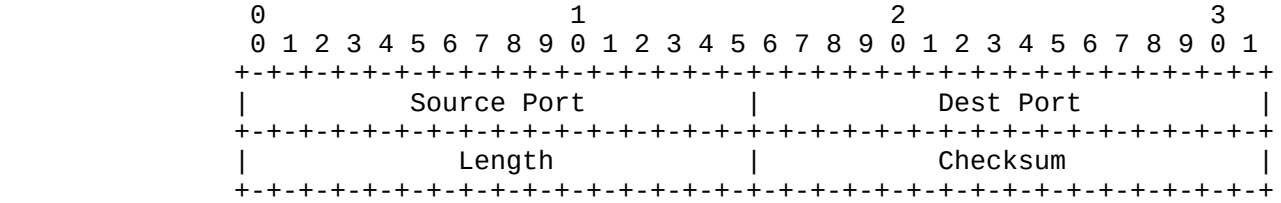
The basic format of a DCCP-UDP packet is:



Section 3.8 describes usage of UDP ports. This includes implementation of a DCCP-UDP encapsulation service as a daemon that listens on a well-known port, allowing multiplexing of different DCCP applications over the same port.

3.1. The UDP Header

The format of the UDP header is specified in [RFC0768]:



For DCCP-UDP, the fields are interpreted as follows:

Source and Dest(ination) Ports: 16 bits each

These fields identify the UDP ports on which the source and destination (respectively) of the packet are listening for incoming DCCP-UDP packets. The UDP port values do not identify the DCCP source and destination ports.

Length: 16 bits

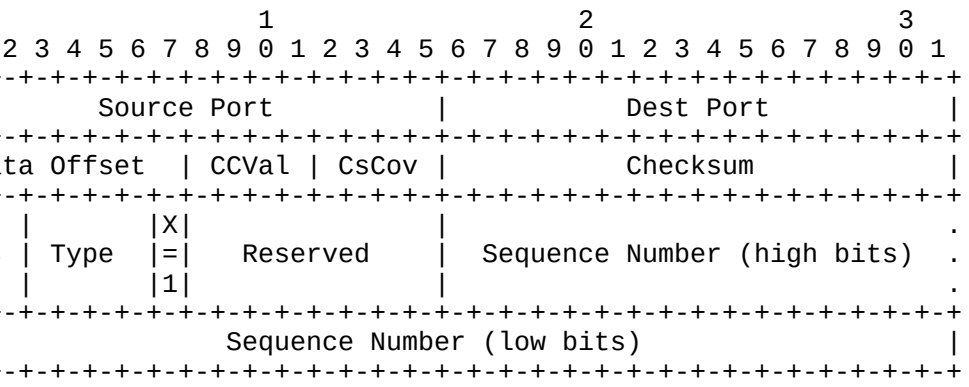
This field is the length of the UDP datagram, including the UDP header and the payload (for DCCP-UDP, the payload is a DCCP-UDP datagram).

Checksum: 16 bits

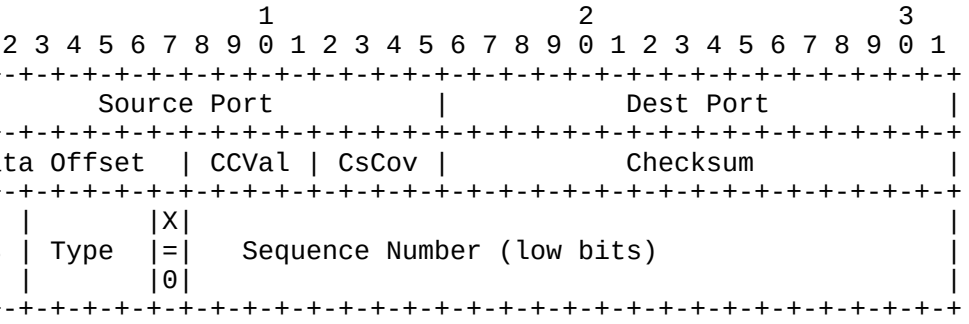
This field is the Internet checksum of a network-layer pseudoheader and Length bytes of the UDP packet [RFC0768]. The UDP checksum MUST NOT be zero for a UDP packet that carries DCCP-UDP.

3.2. The DCCP Generic Header

The DCCP Generic Header [RFC4340] takes two forms, one with long sequence numbers (48 bits) and the other with short sequence numbers (24 bits).



The Generic DCCP Header with long sequence numbers [RFC4340]



The Generic DCCP Header with short sequence numbers [RFC4340]

All generic header fields, except for the Checksum field, have the meaning specified in [RFC4340] updated by [RFC5596].

Section 3.8 describes how a DCCP-UDP implementation treats UDP and DCCP ports.

3.3. DCCP-UDP Checksum Procedures

DCCP-UDP employs a checksum at the UDP level and eliminates the use of the DCCP checksum. This approach was chosen to enable use of current NAT/NATP traversal methods developed for UDP. Such methods will generally be unaware whether DCCP is being encapsulated and hence do not update the inner checksum in the DCCP header. Standard DCCP requires protection of the DCCP header fields, this justifies any processing overhead incurred from calculating the UDP checksum.

In addition, UDP NAT traversal does not support partial checksums. Although this is still permitted end-to-end in the encapsulated DCCP

datagram, links along the path will treat these as UDP packets and can not enable special partial checksum processing.

DCCP-UDP does not update or modify the operation of UDP. The UDP transport protocol is used in the following way:

For DCCP-UDP, the function of the DCCP Checksum field is performed by the UDP checksum field. On transmit, the DCCP Checksum field SHOULD be set to zero. On receive, the DCCP Checksum field MUST be ignored.

The UDP checksum MUST NOT be zero for a UDP packet that is sent using DCCP-UDP. If the received UDP Checksum field is zero, the packet MUST be dropped [RFC5405].

If the UDP Length field is less than 20 (the UDP Header length and minimum DCCP-UDP header length), the packet MUST be dropped [RFC5405]..

If the UDP Checksum field, computed using standard UDP methods, is invalid, the packet MUST be dropped [RFC5405].

If the UDP Length field in a received packet is less than the length of the UDP header plus the entire DCCP-UDP header (including the generic header and type-specific fields and options, if present), or the UDP Length field is greater than the length of the packet from the beginning of the UDP header to the end of the packet, the packet MUST be dropped.

3.3.1. Partial Checksums and the Minimum Checksum Coverage Feature

This document describes an encapsulation for DCCP that uses the UDP transport. It requires the UDP checksum to be enabled. This checksum provides coverage of the entire encapsulated DCCP datagram.

DCCP-UDP supports the syntax of partial checksums. It also supports negotiation of the Minimum Checksum Coverage feature and settings of the CsCov field. However, the UDP checksum field in DCCP-UDP always covers the entire DCCP datagram and the DCCP checksum is ignored on receipt. An application that enables the partial checksums feature in the DCCP Module will therefore experience a service that is functionally identical to using full DCCP checksum coverage. This is also the service that the application would have received if it had used a network path that did not provide optimised processing for DCCP partial checksums.

3.4. Network Layer Options

A DCCP-UDP implementation MAY transfer network-layer options intended for DCCP to the network-layer header of the encapsulating UDP packet.

A DCCP-UDP endpoint that receives IP-options for the encapsulating UDP packet MAY forward these to the DCCP protocol module. If the endpoint forwards a specific network layer option to the DCCP module, it MUST also forward all subsequent packets with this option. Consistent forwarding is essential for correct operation of many end-to-end options.

3.5. Explicit Congestion Notification

A DCCP-UDP endpoint SHOULD follow the procedures of DCCP-STD section 12 by setting the ECN fields in the IP Headers of outgoing packets and examining the values received in the ECN fields of incoming IP packets, relaying any packet markings to the DCCP module.

Implementations that do not support ECN MUST follow the procedures in DCCP-STD section 12.1 with regard to implementations that are not ECN capable.

3.6. ICMP handling for messages relating to DCCP-UDP

To allow ICMP messages to be demultiplexed by the receiving endpoint, part of the original packet that resulted in the message is included in the payload of the ICMP error message. The receiving endpoint can therefore use this information to associate the ICMP error with the transport protocol instance that resulted in the ICMP message. When DCCP-UDP is used, the error message and the payload of the ICMP error message relate to the UDP transport.

DCCP-UDP endpoints SHOULD forward ICMP messages relating to a UDP packet that carries a DCCP-UDP to the DCCP module. This may imply translation of the payload of the ICMP message into a form that is recognised by the DCCP stack. [RFC5927] describes precautions that are desirable before TCP acts on the receipt of an ICMP message. Similar precautions are desirable prior to forwarding by DCCP-UDP to the DCCP module.

The minimal length ICMP error message generated in response to processing a UDP Datagram only identifies the Source UDP Port and Destination UDP Port. This ICMP message does not carry sufficient information to discover the encapsulated DCCP Port values. A DCCP-UDP endpoint that supports multiple DCCP connections over the same pair of UDP ports (see section Section 3.8) may not therefore be able to associate an ICMP message with a unique DCCP-UDP connection.

3.7. Path Maximum Transmission Unit Discovery

DCCP-UDP implementations MUST follow DCCP-STD [RFC4340], section 14 with regard to determining the maximum packet size and the use of Path Maximum Transmission Unit Discovery (PMTUD). This requires the processing of ICMP Destination Unreachable messages with a Code that indicates that an unfragmentable packet was too large to be forwarded (a "Datagram Too Big" message), as defined in RFC 4340.

An effect of encapsulation is to incur additional datagram overhead. This will reduce the Maximum Packet Size (MPS) at the DCCP level.

3.8. Usage of the UDP port by DCCP-UDP

A DCCP-UDP server (that is, an initially passive endpoint that wishes to receive DCCP-Request packets [RFC4340] over DCCP-UDP) listens for connections on one or more UDP ports. UDP port number XXX IANA PORT XXX has been reserved as the default listening UDP port for a DCCP-UDP server. Some NAT/NAPT topologies may require using a non-default listening port.

The purpose of this IANA-assigned port is for the operating system or a framework to receive and process DCCP-UDP datagrams for delivery to the DCCP module (e.g. to support a system-wide DCCP-UDP daemon serving multiple DCCP applications or a DCCP-UDP server placed behind a firewall).

An application-specific implementation SHOULD use an ephemeral port and advertise this port using outside means, e.g. SDP. This method of implementation SHOULD NOT use the IANA-assigned port to listen for incoming DCCP-UDP packets.

A DCCP-UDP client provides UDP source and destination ports as well as DCCP source and destination ports at connection initiation time. A client SHOULD ensure that each DCCP connection maps to a single DCCP-UDP connection by setting the UDP source port. Choosing a distinct source UDP port for each distinct DCCP connection ensures that UDP-based flow identifiers differ whenever DCCP-based flow identifiers differ. Specifically, two connections with different <source IP address, source DCCP port, destination IP address, destination DCCP port> DCCP 4-tuples will have different <source IP address, source UDP port, destination IP address, destination UDP port> UDP 4-tuples.

A DCCP-UDP server SHOULD accept datagrams from any UDP source port. There is a risk that the same DCCP source port number could be used by two endpoints each behind a NAPT. A DCCP-UDP server MUST therefore demultiplex a DCCP-UDP flow using both the UDP source and

destination port numbers and the encapsulated DCCP ports. This ensures than an active DCCP connection is uniquely identified by the 6-tuple <source IP address, source UDP port, source DCCP port, destination IP address, destination UDP port, destination DCCP port>. (The active state of a DCCP connection is defined in Section 3.8: A DCCP connection becomes active following transmission of a DCCP-Request, and become inactive after sending a DCCP-Close.)

This demultiplexing at a DCCP-UDP endpoint occurs in two stages:

1) In the first stage, DCCP-UDP packets are demultiplexed using the UDP 4-tuple: <source IP address, source UDP port, destination IP address, destination UDP port>.

2) In the second stage, a receiving endpoint MUST ensure that two independent DCCP connections that were multiplexed to the same UDP 4-tuple are not associated with the same connection in the DCCP module. The endpoint therefore needs to keep state for the set of active DCCP-UDP endpoints using each combination of a UDP 4-tuple: <source IP address, source UDP port, destination IP address, destination UDP port>. Two DCCP endpoint methods are specified. A DCCP-UDP implementation MUST implement exactly one of these:

- The DCCP server may accept only one active 6-tuple at any one time for a given UDP 4-tuple. In this method, DCCP-UDP packets that do not match an active 6-tuple MUST NOT be passed to the DCCP module and the DCCP Server SHOULD send a DCCP-Reset with with Reset Code XXX IANA Port Reuse XXX, "Encapsulated Port Reuse". An endpoint that receives a DCCP-Reset with this reset code will clear its connection state, but MAY immediately try again using a different 4-tuple. This provides protection should the same UDP 4-tuple be re-used by multiple DCCP connections, ensuring that only one DCCP connection is established at one time.

- The DCCP server may support multiple DCCP connections over the same UDP 4-tuple. In this method, the endpoint MUST then associate each 6-tuple with a single DCCP connection. If an endpoint is unable to demultiplex the 6-tuple (e.g. due to internal resource limits), it MUST discard DCCP-UDP packets that do not match an active 6-tuple instead of forwarding them to the DCCP module. The DCCP endpoint MAY send a DCCP-Reset with Reset Code XXX IANA Port Reuse XXX, "Encapsulated Port Reuse", indicating the connection has been closed, but may be retried using a different UDP 4-tuple.

3.9. Service Codes and the DCCP Port Registry

This section clarifies the usage of DCCP Service Codes and the registration of server ports by DCCP-UDP. The section is not intended to update the procedures for allocating Service Codes or server ports.

There is one Service Code registry and one DCCP port registration that apply to all combinations of encapsulation and IP version. A DCCP Service Code specifies an application using DCCP regardless of the combination of DCCP encapsulation and IP version. An application may choose not to support some combinations of encapsulation and IP version, but its Service Code will remain registered for those combinations and the Service Code must not be used by other applications. An application should not register different Service Codes for different combinations of encapsulation and IP version. [RFC5595] provides additional information about DCCP Service Codes.

Similarly, a DCCP port registration is applicable to all combinations of encapsulation and IP version. Again, an application may choose not to support some combinations of encapsulation and IP version on its registered DCCP port, although the port will remain registered for those combinations. Applications should not register different DCCP ports just for the purpose of using different combinations of encapsulation.

4. DCCP-UDP and Higher-Layer Protocols

The encapsulation of a higher-layer protocol within DCCP MUST be the same for both DCCP-STD and DCCP-UDP. Encapsulation of Datagram Transport Layer Security (DTLS) over DCCP is defined in [RFC5238] and RTP over DCCP is defined in [RFC5762]. This document therefore does not update these encapsulations when using DCCP-UDP.

5. Signaling the Use of DCCP-UDP

Applications often signal transport connection parameters through outside means, such as SDP. Applications that define such methods for DCCP MUST define how the DCCP encapsulation is chosen, and MUST allow either encapsulation to be signaled. Where DCCP-STD and DCCP-UDP are both supported, DCCP-STD SHOULD be preferred.

The Session Description Protocol (SDP) [RFC4566] and the offer/answer model [RFC3264] can be used to negotiate DCCP sessions, and [RFC5762] defines SDP extensions for signalling the use of an RTP session running over DCCP connections. However, since [RFC5762] predates

this document, it does not define a mechanism for signalling that the DCCP-UDP encapsulation is to be used. This section updates [RFC5762] to describe how SDP can be used to signal RTP sessions running over the DCCP-UDP encapsulation.

The new SDP support specified in this section is expected to be useful when the offering party is on the public Internet, or in the same private addressing realm as the answering party. In this case, the DCCP-UDP server has a public address. The client may either have a public address or be behind a NAT/NAPT. This scenario has the potential to be an important use-case. Some other NAT/NAPT topologies may result in the advertised port being unreachable via the NAT/NAPT.

5.1. Protocol Identification

SDP uses a media ("m=") line to convey details of the media format and transport protocol used. The ABNF syntax [RFC5124] of a media line for DCCP is as follows (from [RFC4566]):

```
media-field = %x6d "=" media SP port ["/" integer] SP proto

1*(SP fmt) CRLF
```

The proto field denotes the transport protocol used for the media, while the port indicates the transport port to which the media is sent, following [RFC5762]. This document defines the following five values of the proto field to indicate media transported using DCCP-UDP encapsulation:

UDP/DCCP

UDP/DCCP/RTP/AVP

UDP/DCCP/RTP/SAVP

UDP/DCCP/RTP/AVPF

UDP/DCCP/RTP/SAVPF

The "UDP/DCCP" protocol identifier is similar to the "DCCP" protocol identifier defined in [RFC5762] and denotes the DCCP transport protocol encapsulated in UDP, but not its upper-layer protocol.

The "UDP/DCCP/RTP/AVP" protocol identifier refers to RTP using the RTP Profile for Audio and Video Conferences with Minimal Control [RFC3551] running over the DCCP-UDP encapsulation.

The "UDP/DCCP/RTP/SAVP" protocol identifier refers to RTP using the Secure Real-time Transport Protocol [RFC3711] running over the DCCP-UDP encapsulation.

The "UDP/DCCP/RTP/AVPF" protocol identifier refers to RTP using the Extended RTP Profile for RTCP-based Feedback [RFC4585] running over the DCCP-UDP encapsulation.

The "UDP/DCCP/RTP/SAVPF" protocol identifier refers to RTP using the Extended Secure RTP Profile for RTCP-based Feedback [RFC5124] running over the DCCP-UDP encapsulation.

The fmt value in the "m=" line is used as described in [RFC5762].

The port number specified in the "m=" line indicates the UDP port that is used for the DCCP-UDP encapsulation service. The DCCP port number MUST be sent using an associated "a=dccp-port:" attribute, as described in Section 5.2.

The use of ports with DCCP-UDP encapsulation is described further in Section 3.8.

5.2. Signalling Encapsulated DCCP Ports

When using DCCP-UDP, the UDP port used for the encapsulation is signalled using the SDP "m=" line. The DCCP ports MUST NOT be included in the "m=" line, but are instead signalled using a new SDP attribute ("dccp-port") defined according to the following ABNF:

```
dccp-port-attr = %x61 "=dccp-port:" dccp-port

dccp-port = 1*DIGIT
```

where DIGIT is as defined in [RFC5234]. This is a media level attribute, that is not subject to the charset attribute. The "a=dccp-port:" attribute MUST be included when the protocol identifiers described in Section 5.1 are used.

The use of ports with DCCP-UDP encapsulation is described further in Section 3.8.

- If the "a=rtcp:" attribute [RFC3605] is used, then the signalled port is the DCCP port used for RTCP.
- If the "a=rtcp-mux" attribute [RFC5761] is negotiated, then RTP and RTCP are multiplexed onto a single DCCP port, otherwise separate DCCP ports are used for RTP and RTCP [RFC5762].

In each case, only a single UDP port is used for the DCCP-UDP encapsulation.

- If the "a=rtcp-mux" attribute is not present, then the second of the two demultiplexing methods described in Section 3.8 MUST be implemented, otherwise the second DCCP connection for the RTCP flow will be rejected. For this reason, using "a=rtcp-mux" is RECOMMENDED when using RTP over DCCP-UDP.

5.3. Connection Management

The "a=setup:" attribute is used in a manner compatible with [RFC5762] Section 5.3 to indicate which of the DCCP-UDP endpoints should initiate the DCCP-UDP connection establishment.

5.4. Negotiating the DCCP-UDP encapsulation versus native DCCP

An endpoint that supports both native DCCP and the DCCP-UDP encapsulation may wish to signal support for both options in an SDP offer, allowing the answering party the option of using native DCCP where possible, while falling back to the DCCP-UDP encapsulation otherwise.

An approach to doing this might be to include candidates for the DCCP-UDP encapsulation and native DCCP into an Interactive Connectivity Establishment (ICE) [RFC5245] exchange. Since DCCP is connection-oriented, these candidates would need to be encoded into ICE in a manner analogous to TCP candidates defined in [RFC6544]. Both active and passive candidates could be supported for native DCCP and DCCP-UDP encapsulation, as may DCCP simultaneous open [RFC5596]. In choosing local preference values, it may make sense to to prefer DCCP-UDP over native DCCP in cases where low connection setup time is important, and to prioritise native DCCP in cases where low overhead is preferred (on the assumption that DCCP-UDP is more likely to work through legacy NAT, but has higher overhead). The details of this encoding into ICE are left for future study.

While ICE is appropriate for selecting basic use of DCCP-UDP versus DCCP-STD, it may not be appropriate for negotiating different RTP profiles with each transport encapsulation. The SDP Capability Negotiation framework [RFC5939] may be be more suitable. Section 3.7 of RFC 5939 specifies how to provide attributes and transport protocols as capabilities and negotiate them using the framework .The details of the use of SDP Capability Negotiation with DCCP are left for future study.

5.5. Example of SDP use

The example below shows an SDP offer, where an application signals support for DCCP-UDP:

```
v=0
o=alice 1129377363 1 IN IP4 192.0.2.47
s=-
c=IN IP4 192.0.2.47
t=0 0
m=video 50234 UDP/DCCP/RTP/AVP 99
a=rtpmap:99 h261/90000
a=dccp-service-code:SC=x52545056
a=dccp-port:5004
a=rtcp:5005
a=setup:passive
a=connection:new
```

The answering party at 192.0.2.128 receives this offer and responds with the following answer:

```
v=0
o=bob 1129377364 1 IN IP4 192.0.2.128
s=-
c=IN IP4 192.0.2.128
t=0 0
m=video 40123 UDP/DCCP/RTP/AVP 99
a=rtpmap:99 h261/90000
a=dccp-service-code:SC:RTPV
a=dccp-port:9
a=setup:active
a=connection:new
```

Note that the "m=" line in the answer includes the UDP port number of the encapsulation service. The DCCP service code is set to "RTPV", signalled using the "a=dccp-service-code" attribute [RFC5762]. The "a=dccp-port:" attribute in the answer is set to 9 (the discard port) in the usual manner for an active connection-oriented endpoint.

The answering party will then attempt to establish a DCCP-UDP connection to the offering party. The connection request will use an ephemeral DCCP source port and DCCP destination port 5004. The UDP packet encapsulating that request will have UDP source port 40123 and UDP destination port 50234.

6. Security Considerations

DCCP-UDP provides all of the security risk-mitigation measures present in DCCP-STD, and also all of the security risks. It does not maintain additional state at the encapsulation layer.

The tunnel encapsulation recommends processing of ICMP messages received for packets sent using DCCP-UDP and translation to allow use by DCCP. [RFC5927] describes precautions that are desirable before TCP acts on receipt of ICMP messages. Similar precautions are desirable for endpoints processing ICMP for DCCP-UDP.The purpose of DCCP-UDP is to allow DCCP to pass through NAT/NAPT devices, and therefore it exposes DCCP to the risks associated with passing through NAT devices. It does not create any new risks with regard to NAT/NAPT devices.

DCCP-UDP may also allow DCCP applications to pass through existing firewall devices using rules for UDP, if the administrators of the devices so choose. A simple use may either allow all DCCP applications or allow none.

A firewall that interprets this specification could inspect the encapsulated DCCP header to filter based on the inner DCCP header information. Full control of DCCP connections by applications will require enhancements to firewalls, as discussed in [RFC4340] and related RFCs (e.g. [RFC5595]).

Datagram Transport Layer Security (DTLS) TLS provides mechanisms that can be used to provide security protection for the encapsulated DCCP packets. DTLS may be used in two ways:

- Individual DCCP connections may be protected in the same way that DTLS is used with native DCCP [RFC5595]. This does not encrypt the UDP transport header added by DCCP-UDP.

- This specification also permits the use of DTLS with the UDP transport that encapsulates DCCP packets. When DTLS is used at the encapsulation layer this protects the DCCP headers. This prevents the headers from being inspected or updated by network middleboxes (such as firewalls and NAPT). It also eliminates the need for a spearate DTLS handshake for each DCCP connection.

7. IANA Considerations

This document requests IANA to make the allocations described in the following sections.

7.1. UDP Port Allocation

IANA is requested to allocate a UDP port for the DCCP-UDP service. This port is allocated for use by a transport service, rather than an application. In this case, the name of the transport should explicitly appear in the registry. Use of this port is defined in section Section 3.8

XXX Note: IANA is requested to replace all occurrences of "XXX IANA PORT XXX" by the allocated port value prior to publication. XXX

7.2. DCCP Reset

IANA is requested to assign a new DCCP Reset Code in the DCCP Reset Codes Registry, with the short description "Encapsulated Port Reuse". This code applies to all DCCP congestion control IDs and should be allocated a value less than 120 decimal. Use of this reset code is defined in section Section 3.8. Section 5.6 of RFC4340 defines three "Data" bytes that are carried by a DCCP Reset. For this Reset Code these are defined as below:

- Data byte 1: The DCCP Packet Type of the DCCP datagram that resulted in the error message.

- Data byte 2 & 3: The encapsulated Source UDP Port from the DCCP-UDP datagram that triggered the ICMP message, in network order.

XXX Note: IANA is requested to replace all occurrences of "XXX IANA Port Reuse XXX" by the allocated DCCP reset code value prior to publication. XXX

7.3. SDP Attribute Allocation

IANA is requested to allocate the following new SDP attribute ("att-field"):

Contact name: DCCP Working Group

Attribute name: dccp-port

Long-form attribute name in English: Encapsulated DCCP Port

Type of attribute: Media level

Subject to charset attribute? No

Purpose of the attribute: See this document, section Section 5.1

Allowed attribute values: See this document, section Section 5.1

8. Acknowledgments

This document was produced by the DCCP WG. The following contributed during the working group last call:

Andrew Lentvorski, Lloyd Wood, Pasi Sarolahti, Gerrit Renker, Eddie Kohler, and Dan Wing.

9. References

9.1. Normative References

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

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

[RFC3605] Huitema, C., "Real Time Control Protocol (RTCP) attribute in Session Description Protocol (SDP)", RFC 3605, October 2003.

[RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, March 2006.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5762] Perkins, C., "RTP and the Datagram Congestion Control Protocol (DCCP)", RFC 5762, April 2010.

9.2. Informative References

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[RFC3551] Schulzrinne, H. and S. Casner, "RTP Profile for Audio and Video Conferences with Minimal Control", STD 65, RFC 3551, July 2003.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

[RFC4585] Ott, J., Wenger, S., Sato, N., Burmeister, C., and J. Rey, "Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)", RFC 4585, July 2006.

[RFC4787] Audet, F. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, January 2007.

[RFC5124] Ott, J. and E. Carrara, "Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)", RFC 5124, February 2008.

[RFC5238] Phelan, T., "Datagram Transport Layer Security (DTLS) over the Datagram Congestion Control Protocol (DCCP)", RFC 5238, May 2008.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.

[RFC5595] Fairhurst, G., "The Datagram Congestion Control Protocol (DCCP) Service Codes", RFC 5595, September 2009.

[RFC5596] Fairhurst, G., "Datagram Congestion Control Protocol (DCCP) Simultaneous-Open Technique to Facilitate NAT/ Middlebox Traversal", RFC 5596, September 2009.

[RFC5597] Denis-Courmont, R., "Network Address Translation (NAT) Behavioral Requirements for the Datagram Congestion Control Protocol", BCP 150, RFC 5597, September 2009.

[RFC5761] Perkins, C. and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port", RFC 5761, April 2010.

[RFC5927] Gont, F., "ICMP Attacks against TCP", RFC 5927, July 2010.

[RFC5939] Andreasen, F., "Session Description Protocol (SDP) Capability Negotiation", RFC 5939, September 2010.

[RFC6544] Rosenberg, J., Keranen, A., Lowekamp, B., and A. Roach, "TCP Candidates with Interactive Connectivity Establishment (ICE)", RFC 6544, March 2012.

Authors' Addresses

Tom Phelan
Sonus Networks
7 Technology Dr.
Westford, MA 01886
US

Phone: +1 978 614 8456
Email: tphelan@sonusnet.com

Godred Fairhurst
University of Aberdeen
School of Engineering
Fraser Noble Building
Aberdeen, Scotland AB24 3UE
UK

Email: gorry@erg.abdn.ac.uk
URI: http://www.erg.abdn.ac.uk

Colin Perkins
University of Glasgow
School of Computing Science
Glasgow, Scotland G12 8QQ
UK

Email: csp@csperkins.org
URI: http:http://csperkins.org/

Transport Area Working Group
Internet-Draft
Updates: 2309 (if approved)
Intended status: BCP
Expires: May 11, 2014

B. Briscoe
BT
J. Manner
Aalto University

November 07, 2013

Byte and Packet Congestion Notification
draft-ietf-tsvwg-byte-pkt-congest-12

Abstract

This document provides recommendations of best current practice for dropping or marking packets using any active queue management (AQM) algorithm, including random early detection (RED), BLUE, pre-congestion notification (PCN) and newer schemes such as CoDel (Controlled Delay) and PIE (Proportional Integral controller Enhanced). We give three strong recommendations: (1) packet size should be taken into account when transports detect and respond to congestion indications, (2) packet size should not be taken into account when network equipment creates congestion signals (marking, dropping), and therefore (3) in the specific case of RED, the byte-mode packet drop variant that drops fewer small packets should not be used. This memo updates RFC 2309 to deprecate deliberate preferential treatment of small packets in AQM algorithms.

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 May 11, 2014.

Copyright Notice

Table of Contents

1. Introduction

This document provides recommendations of best current practice for how we should correctly scale congestion control functions with respect to packet size for the long term. It also recognises that expediency may be necessary to deal with existing widely deployed protocols that don't live up to the long term goal.

When signalling congestion, the problem of how (and whether) to take packet sizes into account has exercised the minds of researchers and practitioners for as long as active queue management (AQM) has been discussed. Indeed, one reason AQM was originally introduced was to reduce the lock-out effects that small packets can have on large packets in drop-tail queues. This memo aims to state the principles we should be using and to outline how these principles will affect future protocol design, taking into account the existing deployments we have already.

The question of whether to take into account packet size arises at three stages in the congestion notification process:

Measuring congestion: When a congested resource measures locally how congested it is, should it measure its queue length in time, bytes or packets?

Encoding congestion notification into the wire protocol: When a congested network resource signals its level of congestion, should it drop / mark each packet dependent on the size of the particular packet in question?

Decoding congestion notification from the wire protocol: When a transport interprets the notification in order to decide how much to respond to congestion, should it take into account the size of each missing or marked packet?

Consensus has emerged over the years concerning the first stage, which Section 2.1 records in the RFC Series. In summary: If possible it is best to measure congestion by time in the queue, but otherwise the choice between bytes and packets solely depends on whether the resource is congested by bytes or packets.

The controversy is mainly around the last two stages: whether to allow for the size of the specific packet notifying congestion i) when the network encodes or ii) when the transport decodes the congestion notification.

Currently, the RFC series is silent on this matter other than a paper trail of advice referenced from [RFC2309], which conditionally

recommends byte-mode (packet-size dependent) drop [pktByteEmail]. Reducing drop of small packets certainly has some tempting advantages: i) it drops less control packets, which tend to be small and ii) it makes TCP's bit-rate less dependent on packet size. However, there are ways of addressing these issues at the transport layer, rather than reverse engineering network forwarding to fix the problems.

This memo updates [RFC2309] to deprecate deliberate preferential treatment of packets in AQM algorithms solely because of their size. It recommends that (1) packet size should be taken into account when transports detect and respond to congestion indications, (2) not when network equipment creates them. This memo also adds to the congestion control principles enumerated in BCP 41 [RFC2914].

In the particular case of Random early Detection (RED), this means that the byte-mode packet drop variant should not be used to drop fewer small packets, because that creates a perverse incentive for transports to use tiny segments, consequently also opening up a DoS vulnerability. Fortunately all the RED implementers who responded to our admittedly limited survey (Section 4.2.4) have not followed the earlier advice to use byte-mode drop, so the position this memo argues for seems to already exist in implementations.

However, at the transport layer, TCP congestion control is a widely deployed protocol that doesn't scale with packet size (i.e. its reduction in rate does not take into account the size of a lost packet). To date this hasn't been a significant problem because most TCP implementations have been used with similar packet sizes. But, as we design new congestion control mechanisms, this memo recommends that we should build in scaling with packet size rather than assuming we should follow TCP's example.

This memo continues as follows. First it discusses terminology and scoping. Section 2 gives the concrete formal recommendations, followed by motivating arguments in Section 3. We then critically survey the advice given previously in the RFC series and the research literature (Section 4), referring to an assessment of whether or not this advice has been followed in production networks (Appendix A). To wrap up, outstanding issues are discussed that will need resolution both to inform future protocol designs and to handle legacy (Section 5). Then security issues are collected together in Section 6 before conclusions are drawn in Section 8. The interested reader can find discussion of more detailed issues on the theme of byte vs. packet in the appendices.

This memo intentionally includes a non-negligible amount of material on the subject. For the busy reader Section 2 summarises the

recommendations for the Internet community.

1.1. Terminology and Scoping

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].

This memo applies to the design of all AQM algorithms, for example, Random Early Detection (RED) [RFC2309], BLUE [BLUE02], Pre-Congestion Notification (PCN) [RFC5670], Controlled Delay (CoDel) [I-D.nichols-tsvwg-codel] and the Proportional Integral controller Enhanced (PIE) [I-D.pan-tsvwg-pie]. Throughout, RED is used as a concrete example because it is a widely known and deployed AQM algorithm. There is no intention to imply that the advice is any less applicable to the other algorithms, nor that RED is preferred.

Congestion Notification: Congestion notification is a changing signal that aims to communicate the probability that the network resource(s) will not be able to forward the level of traffic load offered (or that there is an impending risk that they will not be able to).

The ‘impending risk’ qualifier is added, because AQM systems set a virtual limit smaller than the actual limit to the resource, then notify when this virtual limit is exceeded in order to avoid uncontrolled congestion of the actual capacity.

Congestion notification communicates a real number bounded by the range [0 , 1]. This ties in with the most well-understood measure of congestion notification: drop probability.

Explicit and Implicit Notification: The byte vs. packet dilemma concerns congestion notification irrespective of whether it is signalled implicitly by drop or using Explicit Congestion Notification (ECN [RFC3168] or PCN [RFC5670]). Throughout this document, unless clear from the context, the term marking will be used to mean notifying congestion explicitly, while congestion notification will be used to mean notifying congestion either implicitly by drop or explicitly by marking.

Bit-congestible vs. Packet-congestible: If the load on a resource depends on the rate at which packets arrive, it is called packet-congestible. If the load depends on the rate at which bits arrive it is called bit-congestible.

Examples of packet-congestible resources are route look-up engines and firewalls, because load depends on how many packet headers

they have to process. Examples of bit-congestible resources are transmission links, radio power and most buffer memory, because the load depends on how many bits they have to transmit or store. Some machine architectures use fixed size packet buffers, so buffer memory in these cases is packet-congestible (see Section 4.1.1).

The path through a machine will typically encounter both packet-congestible and bit-congestible resources. However, currently, a design goal of network processing equipment such as routers and firewalls is to size the packet-processing engine(s) relative to the lines in order to keep packet processing uncongested even under worst case packet rates with runs of minimum size packets. Therefore, packet-congestion is currently rare [RFC6077; S.3.3], but there is no guarantee that it will not become more common in future.

Note that information is generally processed or transmitted with a minimum granularity greater than a bit (e.g. octets). The appropriate granularity for the resource in question should be used, but for the sake of brevity we will talk in terms of bytes in this memo.

Coarser Granularity: Resources may be congestible at higher levels of granularity than bits or packets, for instance stateful firewalls are flow-congestible and call-servers are session-congestible. This memo focuses on congestion of connectionless resources, but the same principles may be applicable for congestion notification protocols controlling per-flow and per-session processing or state.

RED Terminology: In RED whether to use packets or bytes when measuring queues is called respectively "packet-mode queue measurement" or "byte-mode queue measurement". And whether the probability of dropping a particular packet is independent or dependent on its size is called respectively "packet-mode drop" or "byte-mode drop". The terms byte-mode and packet-mode should not be used without specifying whether they apply to queue measurement or to drop.

1.2. Example Comparing Packet-Mode Drop and Byte-Mode Drop

Taking RED as a well-known example algorithm, a central question addressed by this document is whether to recommend RED's packet-mode drop variant and to deprecate byte-mode drop. Table 1 compares how packet-mode and byte-mode drop affect two flows of different size packets. For each it gives the expected number of packets and of bits dropped in one second. Each example flow runs at the same bit-

rate of 48Mb/s, but one is broken up into small 60 byte packets and the other into large 1500 byte packets.

To keep up the same bit-rate, in one second there are about 25 times more small packets because they are 25 times smaller. As can be seen from the table, the packet rate is 100,000 small packets versus 4,000 large packets per second (pps).

Parameter	Formula	Small packets	Large packets
Packet size	s/8	60B	1,500B
Packet size	s	480b	12,000b
Bit-rate	x	48Mbps	48Mbps
Packet-rate	u = x/s	100kpps	4kpps
Packet-mode Drop			
Pkt loss probability	p	0.1%	0.1%
Pkt loss-rate	p*u	100pps	4pps
Bit loss-rate	p*u*s	48kbps	48kbps
Byte-mode Drop	MTU, M=12,000b		
Pkt loss probability	b = p*s/M	0.004%	0.1%
Pkt loss-rate	b*u	4pps	4pps
Bit loss-rate	b*u*s	1.92kbps	48kbps

Table 1: Example Comparing Packet-mode and Byte-mode Drop

For packet-mode drop, we illustrate the effect of a drop probability of 0.1%, which the algorithm applies to all packets irrespective of size. Because there are 25 times more small packets in one second, it naturally drops 25 times more small packets, that is 100 small packets but only 4 large packets. But if we count how many bits it drops, there are 48,000 bits in 100 small packets and 48,000 bits in 4 large packets--the same number of bits of small packets as large.

> The packet-mode drop algorithm drops any bit with the same probability whether the bit is in a small or a large packet.

For byte-mode drop, again we use an example drop probability of 0.1%, but only for maximum size packets (assuming the link maximum transmission unit (MTU) is 1,500B or 12,000b). The byte-mode algorithm reduces the drop probability of smaller packets proportional to their size, making the probability that it drops a small packet 25 times smaller at 0.004%. But there are 25 times more small packets, so dropping them with 25 times lower probability results in dropping the same number of packets: 4 drops in both cases. The 4 small dropped packets contain 25 times less bits than the 4 large dropped packets: 1,920 compared to 48,000.

The byte-mode drop algorithm drops any bit with a probability proportionate to the size of the packet it is in.

2. Recommendations

This section gives recommendations related to network equipment in Sections 2.1 and 2.2, and in Sections 2.3 and 2.4 we discuss the implications on the transport protocols.

2.1. Recommendation on Queue Measurement

Ideally, an AQM would measure the service time of the queue to measure congestion of a resource. However service time can only be measured as packets leave the queue, where it is not always expedient to implement a full AQM algorithm. To predict the service time as packets join the queue, an AQM algorithm needs to measure the length of the queue.

In this case, if the resource is bit-congestible, the AQM implementation SHOULD measure the length of the queue in bytes and, if the resource is packet-congestible, the implementation SHOULD measure the length of the queue in packets. Subject to the exceptions below, no other choice makes sense, because the number of packets waiting in the queue isn't relevant if the resource gets congested by bytes and vice versa. For example, the length of the queue into a transmission line would be measured in bytes, while the length of the queue into a firewall would be measured in packets.

To avoid the pathological effects of drop tail, the AQM can then transform this service time or queue length into the probability of dropping or marking a packet (e.g. RED's piecewise linear function between thresholds).

What this advice means for RED as a specific example:

1. A RED implementation SHOULD use byte mode queue measurement for measuring the congestion of bit-congestible resources and packet mode queue measurement for packet-congestible resources.

2. An implementation SHOULD NOT make it possible to configure the way a queue measures itself, because whether a queue is bit-congestible or packet-congestible is an inherent property of the queue.

Exceptions to these recommendations might be necessary, for instance where a packet-congestible resource has to be configured as a proxy bottleneck for a bit-congestible resource in an adjacent box that does not support AQM.

The recommended approach in less straightforward scenarios, such as fixed size packet buffers, resources without a queue and buffers comprising a mix of packet and bit-congestible resources, is discussed in Section 4.1. For instance, Section 4.1.1 explains that the queue into a line should be measured in bytes even if the queue consists of fixed-size packet-buffers, because the root-cause of any congestion is bytes arriving too fast for the line--packets filling buffers are merely a symptom of the underlying congestion of the line.

2.2. Recommendation on Encoding Congestion Notification

When encoding congestion notification (e.g. by drop, ECN or PCN), the probability that network equipment drops or marks a particular packet to notify congestion SHOULD NOT depend on the size of the packet in question. As the example in Section 1.2 illustrates, to drop any bit with probability 0.1% it is only necessary to drop every packet with probability 0.1% without regard to the size of each packet.

This approach ensures the network layer offers sufficient congestion information for all known and future transport protocols and also ensures no perverse incentives are created that would encourage transports to use inappropriately small packet sizes.

What this advice means for RED as a specific example:

1. The RED AQM algorithm SHOULD NOT use byte-mode drop, i.e. it ought to use packet-mode drop. Byte-mode drop is more complex, it creates the perverse incentive to fragment segments into tiny pieces and it is vulnerable to floods of small packets.

2. If a vendor has implemented byte-mode drop, and an operator has turned it on, it is RECOMMENDED to switch it to packet-mode drop, after establishing if there are any implications on the relative performance of applications using different packet sizes. The unlikely possibility of some application-specific legacy use of byte-mode drop is the only reason that all the above recommendations on encoding congestion notification are not phrased more strongly.

   RED as a whole SHOULD NOT be switched off. Without RED, a drop tail queue biases against large packets and is vulnerable to floods of small packets.

Note well that RED's byte-mode queue drop is completely orthogonal to byte-mode queue measurement and should not be confused with it. If a RED implementation has a byte-mode but does not specify what sort of byte-mode, it is most probably byte-mode queue measurement, which is

fine. However, if in doubt, the vendor should be consulted.

A survey (Appendix A) showed that there appears to be little, if any, installed base of the byte-mode drop variant of RED. This suggests that deprecating byte-mode drop will have little, if any, incremental deployment impact.

2.3. Recommendation on Responding to Congestion

When a transport detects that a packet has been lost or congestion marked, it SHOULD consider the strength of the congestion indication as proportionate to the size in octets (bytes) of the missing or marked packet.

In other words, when a packet indicates congestion (by being lost or marked) it can be considered conceptually as if there is a congestion indication on every octet of the packet, not just one indication per packet.

To be clear, the above recommendation solely describes how a transport should interpret the meaning of a congestion indication, as a long term goal. It makes no recommendation on whether a transport should act differently based on this interpretation. It merely aids interoperablity between transports, if they choose to make their actions depend on the strength of congestion indications.

This definition will be useful as the IETF transport area continues its programme of;

- updating host-based congestion control protocols to take account of packet size

- making transports less sensitive to losing control packets like SYNs and pure ACKs.

What this advice means for the case of TCP:

1. If two TCP flows with different packet sizes are required to run at equal bit rates under the same path conditions, this SHOULD be done by altering TCP (Section 4.2.2), not network equipment (the latter affects other transports besides TCP).

2. If it is desired to improve TCP performance by reducing the chance that a SYN or a pure ACK will be dropped, this SHOULD be done by modifying TCP (Section 4.2.3), not network equipment.

To be clear, we are not recommending at all that TCPs under equivalent conditions should aim for equal bit-rates. We are merely

saying that anyone trying to do such a thing should modify their TCP algorithm, not the network.

These recommendations are phrased as 'SHOULD' rather than 'MUST', because there may be cases where expediency dictates that compatibility with pre-existing versions of a transport protocol make the recommendations impractical.

2.4. Recommendation on Handling Congestion Indications when Splitting or Merging Packets

Packets carrying congestion indications may be split or merged in some circumstances (e.g. at a RTP/RTCP transcoder or during IP fragment reassembly). Splitting and merging only make sense in the context of ECN, not loss.

The general rule to follow is that the number of octets in packets with congestion indications SHOULD be equivalent before and after merging or splitting. This is based on the principle used above; that an indication of congestion on a packet can be considered as an indication of congestion on each octet of the packet.

The above rule is not phrased with the word "MUST" to allow the following exception. There are cases where pre-existing protocols were not designed to conserve congestion marked octets (e.g. IP fragment reassembly [RFC3168] or loss statistics in RTCP receiver reports [RFC3550] before ECN was added [RFC6679]). When any such protocol is updated, it SHOULD comply with the above rule to conserve marked octets. However, the rule may be relaxed if it would otherwise become too complex to interoperate with pre-existing implementations of the protocol.

One can think of a splitting or merging process as if all the incoming congestion-marked octets increment a counter and all the outgoing marked octets decrement the same counter. In order to ensure that congestion indications remain timely, even the smallest positive remainder in the conceptual counter should trigger the next outgoing packet to be marked (causing the counter to go negative).

3. Motivating Arguments

This section is informative. It justifies the recommendations given in the previous section.

3.1. Avoiding Perverse Incentives to (Ab)use Smaller Packets

Increasingly, it is being recognised that a protocol design must take care not to cause unintended consequences by giving the parties in

the protocol exchange perverse incentives [Evol_cc][RFC3426]. Given there are many good reasons why larger path maximum transmission units (PMTUs) would help solve a number of scaling issues, we do not want to create any bias against large packets that is greater than their true cost.

Imagine a scenario where the same bit rate of packets will contribute the same to bit-congestion of a link irrespective of whether it is sent as fewer larger packets or more smaller packets. A protocol design that caused larger packets to be more likely to be dropped than smaller ones would be dangerous in both the following cases:

Malicious transports: A queue that gives an advantage to small packets can be used to amplify the force of a flooding attack. By sending a flood of small packets, the attacker can get the queue to discard more traffic in large packets, allowing more attack traffic to get through to cause further damage. Such a queue allows attack traffic to have a disproportionately large effect on regular traffic without the attacker having to do much work.

Non-malicious transports: Even if an application designer is not actually malicious, if over time it is noticed that small packets tend to go faster, designers will act in their own interest and use smaller packets. Queues that give advantage to small packets create an evolutionary pressure for applications or transports to send at the same bit-rate but break their data stream down into tiny segments to reduce their drop rate. Encouraging a high volume of tiny packets might in turn unnecessarily overload a completely unrelated part of the system, perhaps more limited by header-processing than bandwidth.

Imagine two unresponsive flows arrive at a bit-congestible transmission link each with the same bit rate, say 1Mbps, but one consists of 1500B and the other 60B packets, which are 25x smaller. Consider a scenario where gentle RED [gentle_RED] is used, along with the variant of RED we advise against, i.e. where the RED algorithm is configured to adjust the drop probability of packets in proportion to each packet's size (byte mode packet drop). In this case, RED aims to drop 25x more of the larger packets than the smaller ones. Thus, for example if RED drops 25% of the larger packets, it will aim to drop 1% of the smaller packets (but in practice it may drop more as congestion increases [RFC4828; Appx B.4]). Even though both flows arrive with the same bit rate, the bit rate the RED queue aims to pass to the line will be 750kbps for the flow of larger packets but 990kbps for the smaller packets (because of rate variations it will actually be a little less than this target).

Note that, although the byte-mode drop variant of RED amplifies small

packet attacks, drop-tail queues amplify small packet attacks even more (see Security Considerations in Section 6). Wherever possible neither should be used.

3.2. Small != Control

Dropping fewer control packets considerably improves performance. It is tempting to drop small packets with lower probability in order to improve performance, because many control packets tend to be smaller (TCP SYNs & ACKs, DNS queries & responses, SIP messages, HTTP GETs, etc). However, we must not give control packets preference purely by virtue of their smallness, otherwise it is too easy for any data source to get the same preferential treatment simply by sending data in smaller packets. Again we should not create perverse incentives to favour small packets rather than to favour control packets, which is what we intend.

Just because many control packets are small does not mean all small packets are control packets.

So, rather than fix these problems in the network, we argue that the transport should be made more robust against losses of control packets (see 'Making Transports Robust against Control Packet Losses' in Section 4.2.3).

3.3. Transport-Independent Network

TCP congestion control ensures that flows competing for the same resource each maintain the same number of segments in flight, irrespective of segment size. So under similar conditions, flows with different segment sizes will get different bit-rates.

To counter this effect it seems tempting not to follow our recommendation, and instead for the network to bias congestion notification by packet size in order to equalise the bit-rates of flows with different packet sizes. However, in order to do this, the queuing algorithm has to make assumptions about the transport, which become embedded in the network. Specifically:

- The queuing algorithm has to assume how aggressively the transport will respond to congestion (see Section 4.2.4). If the network assumes the transport responds as aggressively as TCP NewReno, it will be wrong for Compound TCP and differently wrong for Cubic TCP, etc. To achieve equal bit-rates, each transport then has to guess what assumption the network made, and work out how to replace this assumed aggressiveness with its own aggressiveness.

- Also, if the network biases congestion notification by packet size it has to assume a baseline packet size--all proposed algorithms use the local MTU (for example see the byte-mode loss probability formula in Table 1). Then if the non-Reno transports mentioned above are trying to reverse engineer what the network assumed, they also have to guess the MTU of the congested link.

Even though reducing the drop probability of small packets (e.g. RED's byte-mode drop) helps ensure TCP flows with different packet sizes will achieve similar bit rates, we argue this correction should be made to any future transport protocols based on TCP, not to the network in order to fix one transport, no matter how predominant it is. Effectively, favouring small packets is reverse engineering of network equipment around one particular transport protocol (TCP), contrary to the excellent advice in [RFC3426], which asks designers to question "Why are you proposing a solution at this layer of the protocol stack, rather than at another layer?"

In contrast, if the network never takes account of packet size, the transport can be certain it will never need to guess any assumptions the network has made. And the network passes two pieces of information to the transport that are sufficient in all cases: i) congestion notification on the packet and ii) the size of the packet. Both are available for the transport to combine (by taking account of packet size when responding to congestion) or not. Appendix B checks that these two pieces of information are sufficient for all relevant scenarios.

When the network does not take account of packet size, it allows transport protocols to choose whether to take account of packet size or not. However, if the network were to bias congestion notification by packet size, transport protocols would have no choice; those that did not take account of packet size themselves would unwittingly become dependent on packet size, and those that already took account of packet size would end up taking account of it twice.

3.4. Partial Deployment of AQM

In overview, the argument in this section runs as follows:

- Because the network does not and cannot always drop packets in proportion to their size, it shouldn't be given the task of making drop signals depend on packet size at all.

- Transports on the other hand don't always want to make their rate response proportional to the size of dropped packets, but if they want to, they always can.

The argument is similar to the end-to-end argument that says "Don't do X in the network if end-systems can do X by themselves, and they want to be able to choose whether to do X anyway." Actually the following argument is stronger; in addition it says "Don't give the network task X that could be done by the end-systems, if X is not deployed on all network nodes, and end-systems won't be able to tell whether their network is doing X, or whether they need to do X themselves." In this case, the X in question is "making the response to congestion depend on packet size".

We will now re-run this argument taking each step in more depth. The argument applies solely to drop, not to ECN marking.

A queue drops packets for either of two reasons: a) to signal to host congestion controls that they should reduce the load and b) because there is no buffer left to store the packets. Active queue management tries to use drops as a signal for hosts to slow down (case a) so that drop due to buffer exhaustion (case b) should not be necessary.

AQM is not universally deployed in every queue in the Internet; many cheap Ethernet bridges, software firewalls, NATs on consumer devices, etc implement simple tail-drop buffers. Even if AQM were universal, it has to be able to cope with buffer exhaustion (by switching to a behaviour like tail-drop), in order to cope with unresponsive or excessive transports. For these reasons networks will sometimes be dropping packets as a last resort (case b) rather than under AQM control (case a).

When buffers are exhausted (case b), they don't naturally drop packets in proportion to their size. The network can only reduce the probability of dropping smaller packets if it has enough space to store them somewhere while it waits for a larger packet that it can drop. If the buffer is exhausted, it does not have this choice. Admittedly tail-drop does naturally drop somewhat fewer small packets, but exactly how few depends more on the mix of sizes than the size of the packet in question. Nonetheless, in general, if we wanted networks to do size-dependent drop, we would need universal deployment of (packet-size dependent) AQM code, which is currently unrealistic.

A host transport cannot know whether any particular drop was a deliberate signal from an AQM or a sign of a queue shedding packets due to buffer exhaustion. Therefore, because the network cannot universally do size-dependent drop, it should not do it all.

Whereas universality is desirable in the network, diversity is desirable between different transport layer protocols - some, like

NewReno TCP [RFC5681], may not choose to make their rate response proportionate to the size of each dropped packet, while others will (e.g. TFRC-SP [RFC4828]).

3.5. Implementation Efficiency

Biasing against large packets typically requires an extra multiply and divide in the network (see the example byte-mode drop formula in Table 1). Allowing for packet size at the transport rather than in the network ensures that neither the network nor the transport needs to do a multiply operation--multiplication by packet size is effectively achieved as a repeated add when the transport adds to its count of marked bytes as each congestion event is fed to it. Also the work to do the biasing is spread over many hosts, rather than concentrated in just the congested network element. These aren't principled reasons in themselves, but they are a happy consequence of the other principled reasons.

4. A Survey and Critique of Past Advice

This section is informative, not normative.

The original 1993 paper on RED [RED93] proposed two options for the RED active queue management algorithm: packet mode and byte mode. Packet mode measured the queue length in packets and dropped (or marked) individual packets with a probability independent of their size. Byte mode measured the queue length in bytes and marked an individual packet with probability in proportion to its size (relative to the maximum packet size). In the paper's outline of further work, it was stated that no recommendation had been made on whether the queue size should be measured in bytes or packets, but noted that the difference could be significant.

When RED was recommended for general deployment in 1998 [RFC2309], the two modes were mentioned implying the choice between them was a question of performance, referring to a 1997 email [pktByteEmail] for advice on tuning. A later addendum to this email introduced the insight that there are in fact two orthogonal choices:

- whether to measure queue length in bytes or packets (Section 4.1)
- whether the drop probability of an individual packet should depend on its own size (Section 4.2).

The rest of this section is structured accordingly.

4.1. Congestion Measurement Advice

The choice of which metric to use to measure queue length was left open in RFC2309. It is now well understood that queues for bit-congestible resources should be measured in bytes, and queues for packet-congestible resources should be measured in packets [pktByteEmail].

Congestion in some legacy bit-congestible buffers is only measured in packets not bytes. In such cases, the operator has to set the thresholds mindful of a typical mix of packets sizes. Any AQM algorithm on such a buffer will be oversensitive to high proportions of small packets, e.g. a DoS attack, and under-sensitive to high proportions of large packets. However, there is no need to make allowances for the possibility of such legacy in future protocol design. This is safe because any under-sensitivity during unusual traffic mixes cannot lead to congestion collapse given the buffer will eventually revert to tail drop, discarding proportionately more large packets.

4.1.1. Fixed Size Packet Buffers

The question of whether to measure queues in bytes or packets seems to be well understood. However, measuring congestion is confusing when the resource is bit congestible but the queue into the resource is packet congestible. This section outlines the approach to take.

Some, mostly older, queuing hardware allocates fixed sized buffers in which to store each packet in the queue. This hardware forwards to the line in one of two ways:

- With some hardware, any fixed sized buffers not completely filled by a packet are padded when transmitted to the wire. This case, should clearly be treated as packet-congestible, because both queuing and transmission are in fixed MTU-sized units. Therefore the queue length in packets is a good model of congestion of the link.

- More commonly, hardware with fixed size packet buffers transmits packets to line without padding. This implies a hybrid forwarding system with transmission congestion dependent on the size of packets but queue congestion dependent on the number of packets, irrespective of their size.

  Nonetheless, there would be no queue at all unless the line had become congested--the root-cause of any congestion is too many bytes arriving for the line. Therefore, the AQM should measure the queue length as the sum of all the packet sizes in bytes that

are queued up waiting to be serviced by the line, irrespective of whether each packet is held in a fixed size buffer.

In the (unlikely) first case where use of padding means the queue should be measured in packets, further confusion is likely because the fixed buffers are rarely all one size. Typically pools of different sized buffers are provided (Cisco uses the term 'buffer carving' for the process of dividing up memory into these pools [IOSArch]). Usually, if the pool of small buffers is exhausted, arriving small packets can borrow space in the pool of large buffers, but not vice versa. However, there is no need to consider all this complexity, because the root-cause of any congestion is still line overload--buffer consumption is only the symptom. Therefore, the length of the queue should be measured as the sum of the bytes in the queue that will be transmitted to line, including any padding. In the (unusual) case of transmission with padding this means the sum of the sizes of the small buffers queued plus the sum of the sizes of the large buffers queued.

We will return to borrowing of fixed sized buffers when we discuss biasing the drop/marking probability of a specific packet because of its size in Section 4.2.1. But here we can repeat the simple rule for how to measure the length of queues of fixed buffers: no matter how complicated the buffering scheme is, ultimately a transmission line is nearly always bit-congestible so the number of bytes queued up waiting for the line measures how congested the line is, and it is rarely important to measure how congested the buffering system is.

4.1.2. Congestion Measurement without a Queue

AQM algorithms are nearly always described assuming there is a queue for a congested resource and the algorithm can use the queue length to determine the probability that it will drop or mark each packet. But not all congested resources lead to queues. For instance, power limited resources are usually bit-congestible if energy is primarily required for transmission rather than header processing, but it is rare for a link protocol to build a queue as it approaches maximum power.

Nonetheless, AQM algorithms do not require a queue in order to work. For instance spectrum congestion can be modelled by signal quality using target bit-energy-to-noise-density ratio. And, to model radio power exhaustion, transmission power levels can be measured and compared to the maximum power available. [ECNFixedWireless] proposes a practical and theoretically sound way to combine congestion notification for different bit-congestible resources at different layers along an end to end path, whether wireless or wired, and whether with or without queues.

In wireless protocols that use request to send / clear to send (RTS / CTS) control, such as some variants of IEEE802.11, it is reasonable to base an AQM on the time spent waiting for transmission opportunities (TXOPs) even though wireless spectrum is usually regarded as congested by bits (for a given coding scheme). This is because requests for TXOPs queue up as the spectrum gets congested by all the bits being transferred. So the time that TXOPs are queued directly reflects bit congestion of the spectrum.

4.2. Congestion Notification Advice

4.2.1. Network Bias when Encoding

4.2.1.1. Advice on Packet Size Bias in RED

The previously mentioned email [pktByteEmail] referred to by [RFC2309] advised that most scarce resources in the Internet were bit-congestible, which is still believed to be true (Section 1.1). But it went on to offer advice that is updated by this memo. It said that drop probability should depend on the size of the packet being considered for drop if the resource is bit-congestible, but not if it is packet-congestible. The argument continued that if packet drops were inflated by packet size (byte-mode dropping), "a flow's fraction of the packet drops is then a good indication of that flow's fraction of the link bandwidth in bits per second". This was consistent with a referenced policing mechanism being worked on at the time for detecting unusually high bandwidth flows, eventually published in 1999 [pBox]. However, the problem could and should have been solved by making the policing mechanism count the volume of bytes randomly dropped, not the number of packets.

A few months before RFC2309 was published, an addendum was added to the above archived email referenced from the RFC, in which the final paragraph seemed to partially retract what had previously been said. It clarified that the question of whether the probability of dropping/marking a packet should depend on its size was not related to whether the resource itself was bit congestible, but a completely orthogonal question. However the only example given had the queue measured in packets but packet drop depended on the size of the packet in question. No example was given the other way round.

In 2000, Cnodder et al [REDbyte] pointed out that there was an error in the part of the original 1993 RED algorithm that aimed to distribute drops uniformly, because it didn't correctly take into account the adjustment for packet size. They recommended an algorithm called RED_4 to fix this. But they also recommended a further change, RED_5, to adjust drop rate dependent on the square of relative packet size. This was indeed consistent with one implied

motivation behind RED's byte mode drop--that we should reverse engineer the network to improve the performance of dominant end-to-end congestion control mechanisms. This memo makes a different recommendations in Section 2.

By 2003, a further change had been made to the adjustment for packet size, this time in the RED algorithm of the ns2 simulator. Instead of taking each packet's size relative to a 'maximum packet size' it was taken relative to a 'mean packet size', intended to be a static value representative of the 'typical' packet size on the link. We have not been able to find a justification in the literature for this change, however Eddy and Allman conducted experiments [REDbias] that assessed how sensitive RED was to this parameter, amongst other things. However, this changed algorithm can often lead to drop probabilities of greater than 1 (which gives a hint that there is probably a mistake in the theory somewhere).

On 10-Nov-2004, this variant of byte-mode packet drop was made the default in the ns2 simulator. It seems unlikely that byte-mode drop has ever been implemented in production networks (Appendix A), therefore any conclusions based on ns2 simulations that use RED without disabling byte-mode drop are likely to behave very differently from RED in production networks.

4.2.1.2. Packet Size Bias Regardless of AQM

The byte-mode drop variant of RED (or a similar variant of other AQM algorithms) is not the only possible bias towards small packets in queueing systems. We have already mentioned that tail-drop queues naturally tend to lock-out large packets once they are full.

But also queues with fixed sized buffers reduce the probability that small packets will be dropped if (and only if) they allow small packets to borrow buffers from the pools for larger packets (see Section 4.1.1). Borrowing effectively makes the maximum queue size for small packets greater than that for large packets, because more buffers can be used by small packets while less will fit large packets. Incidentally, the bias towards small packets from buffer borrowing is nothing like as large as that of RED's byte-mode drop.

Nonetheless, fixed-buffer memory with tail drop is still prone to lock-out large packets, purely because of the tail-drop aspect. So, fixed size packet-buffers should be augmented with a good AQM algorithm and packet-mode drop. If an AQM is too complicated to implement with multiple fixed buffer pools, the minimum necessary to prevent large packet lock-out is to ensure smaller packets never use the last available buffer in any of the pools for larger packets.

4.2.2. Transport Bias when Decoding

The above proposals to alter the network equipment to bias towards smaller packets have largely carried on outside the IETF process. Whereas, within the IETF, there are many different proposals to alter transport protocols to achieve the same goals, i.e. either to make the flow bit-rate take account of packet size, or to protect control packets from loss. This memo argues that altering transport protocols is the more principled approach.

A recently approved experimental RFC adapts its transport layer protocol to take account of packet sizes relative to typical TCP packet sizes. This proposes a new small-packet variant of TCP-friendly rate control [RFC5348] called TFRC-SP [RFC4828]. Essentially, it proposes a rate equation that inflates the flow rate by the ratio of a typical TCP segment size (1500B including TCP header) over the actual segment size [PktSizeEquCC]. (There are also other important differences of detail relative to TFRC, such as using virtual packets [CCvarPktSize] to avoid responding to multiple losses per round trip and using a minimum inter-packet interval.)

Section 4.5.1 of this TFRC-SP spec discusses the implications of operating in an environment where queues have been configured to drop smaller packets with proportionately lower probability than larger ones. But it only discusses TCP operating in such an environment, only mentioning TFRC-SP briefly when discussing how to define fairness with TCP. And it only discusses the byte-mode dropping version of RED as it was before Cnodder et al pointed out it didn't sufficiently bias towards small packets to make TCP independent of packet size.

So the TFRC-SP spec doesn't address the issue of which of the network or the transport _should_ handle fairness between different packet sizes. In its Appendix B.4 it discusses the possibility of both TFRC-SP and some network buffers duplicating each other's attempts to deliberately bias towards small packets. But the discussion is not conclusive, instead reporting simulations of many of the possibilities in order to assess performance but not recommending any particular course of action.

The paper originally proposing TFRC with virtual packets (VP-TFRC) [CCvarPktSize] proposed that there should perhaps be two variants to cater for the different variants of RED. However, as the TFRC-SP authors point out, there is no way for a transport to know whether some queues on its path have deployed RED with byte-mode packet drop (except if an exhaustive survey found that no-one has deployed it!--see Appendix A). Incidentally, VP-TFRC also proposed that byte-mode RED dropping should really square the packet-size compensation-factor

(like that of Cnodder's RED_5, but apparently unaware of it).

Pre-congestion notification [RFC5670] is an IETF technology to use a virtual queue for AQM marking for packets within one Diffserv class in order to give early warning prior to any real queuing. The PCN marking algorithms have been designed not to take account of packet size when forwarding through queues. Instead the general principle has been to take account of the sizes of marked packets when monitoring the fraction of marking at the edge of the network, as recommended here.

4.2.3. Making Transports Robust against Control Packet Losses

Recently, two RFCs have defined changes to TCP that make it more robust against losing small control packets [RFC5562] [RFC5690]. In both cases they note that the case for these two TCP changes would be weaker if RED were biased against dropping small packets. We argue here that these two proposals are a safer and more principled way to achieve TCP performance improvements than reverse engineering RED to benefit TCP.

Although there are no known proposals, it would also be possible and perfectly valid to make control packets robust against drop by requesting a scheduling class with lower drop probability, by re-marking to a Diffserv code point [RFC2474] within the same behaviour aggregate.

Although not brought to the IETF, a simple proposal from Wischik [DupTCP] suggests that the first three packets of every TCP flow should be routinely duplicated after a short delay. It shows that this would greatly improve the chances of short flows completing quickly, but it would hardly increase traffic levels on the Internet, because Internet bytes have always been concentrated in the large flows. It further shows that the performance of many typical applications depends on completion of long serial chains of short messages. It argues that, given most of the value people get from the Internet is concentrated within short flows, this simple expedient would greatly increase the value of the best efforts Internet at minimal cost. A similar but more extensive approach has been evaluated on Google servers [GentleAggro].

The proposals discussed in this sub-section are experimental approaches that are not yet in wide operational use, but they are existence proofs that transports can make themselves robust against loss of control packets. The examples are all TCP-based, but applications over non-TCP transports could mitigate loss of control packets by making similar use of Diffserv, data duplication, FEC etc.

4.2.4. Congestion Notification: Summary of Conflicting Advice

transport cc	RED_1 (packet mode drop)	RED_4 (linear byte mode drop)	RED_5 (square byte mode drop)
TCP or TFRC	s/sqrt(p)	sqrt(s/p)	1/sqrt(p)
TFRC-SP	1/sqrt(p)	1/sqrt(sp)	1/(s.sqrt(p))

Table 2: Dependence of flow bit-rate per RTT on packet size, s, and drop probability, p, when network and/or transport bias towards small packets to varying degrees

Table 2 aims to summarise the potential effects of all the advice from different sources. Each column shows a different possible AQM behaviour in different queues in the network, using the terminology of Cnodder et al outlined earlier (RED_1 is basic RED with packet-mode drop). Each row shows a different transport behaviour: TCP [RFC5681] and TFRC [RFC5348] on the top row with TFRC-SP [RFC4828] below. Each cell shows how the bits per round trip of a flow depends on packet size, s, and drop probability, p. In order to declutter the formulae to focus on packet-size dependence they are all given per round trip, which removes any RTT term.

Let us assume that the goal is for the bit-rate of a flow to be independent of packet size. Suppressing all inessential details, the table shows that this should either be achievable by not altering the TCP transport in a RED_5 network, or using the small packet TFRC-SP transport (or similar) in a network without any byte-mode dropping RED (top right and bottom left). Top left is the ‘do nothing’ scenario, while bottom right is the ‘do-both’ scenario in which bit-rate would become far too biased towards small packets. Of course, if any form of byte-mode dropping RED has been deployed on a subset of queues that congest, each path through the network will present a different hybrid scenario to its transport.

Whatever, we can see that the linear byte-mode drop column in the middle would considerably complicate the Internet. It’s a half-way house that doesn’t bias enough towards small packets even if one believes the network should be doing the biasing. Section 2 recommends that _all_ bias in network equipment towards small packets should be turned off--if indeed any equipment vendors have implemented it--leaving packet-size bias solely as the preserve of the transport layer (solely the leftmost, packet-mode drop column).

In practice it seems that no deliberate bias towards small packets

has been implemented for production networks. Of the 19% of vendors who responded to a survey of 84 equipment vendors, none had implemented byte-mode drop in RED (see Appendix A for details).

5. Outstanding Issues and Next Steps

5.1. Bit-congestible Network

For a connectionless network with nearly all resources being bit-congestible the recommended position is clear--that the network should not make allowance for packet sizes and the transport should. This leaves two outstanding issues:

- How to handle any legacy of AQM with byte-mode drop already deployed;
- The need to start a programme to update transport congestion control protocol standards to take account of packet size.

A survey of equipment vendors (Section 4.2.4) found no evidence that byte-mode packet drop had been implemented, so deployment will be sparse at best. A migration strategy is not really needed to remove an algorithm that may not even be deployed.

A programme of experimental updates to take account of packet size in transport congestion control protocols has already started with TFRC-SP [RFC4828].

5.2. Bit- & Packet-congestible Network

The position is much less clear-cut if the Internet becomes populated by a more even mix of both packet-congestible and bit-congestible resources (see Appendix B.2). This problem is not pressing, because most Internet resources are designed to be bit-congestible before packet processing starts to congest (see Section 1.1).

The IRTF Internet congestion control research group (ICCRG) has set itself the task of reaching consensus on generic forwarding mechanisms that are necessary and sufficient to support the Internet's future congestion control requirements (the first challenge in [RFC6077]). The research question of whether packet congestion might become common and what to do if it does may in the future be explored in the IRTF (the "Challenge 3: Packet Size" in [RFC6077]).

Note that sometimes it seems that resources might be congested by neither bits nor packets, e.g. where the queue for access to a wireless medium is in units of transmission opportunities. However,

the root cause of congestion of the underlying spectrum is overload of bits (see Section 4.1.2).

6. Security Considerations

This memo recommends that queues do not bias drop probability due to packets size. For instance dropping small packets less often than large creates a perverse incentive for transports to break down their flows into tiny segments. One of the benefits of implementing AQM was meant to be to remove this perverse incentive that drop-tail queues gave to small packets.

In practice, transports cannot all be trusted to respond to congestion. So another reason for recommending that queues do not bias drop probability towards small packets is to avoid the vulnerability to small packet DDoS attacks that would otherwise result. One of the benefits of implementing AQM was meant to be to remove drop-tail's DoS vulnerability to small packets, so we shouldn't add it back again.

If most queues implemented AQM with byte-mode drop, the resulting network would amplify the potency of a small packet DDoS attack. At the first queue the stream of packets would push aside a greater proportion of large packets, so more of the small packets would survive to attack the next queue. Thus a flood of small packets would continue on towards the destination, pushing regular traffic with large packets out of the way in one queue after the next, but suffering much less drop itself.

Appendix C explains why the ability of networks to police the response of _any_ transport to congestion depends on bit-congestible network resources only doing packet-mode not byte-mode drop. In summary, it says that making drop probability depend on the size of the packets that bits happen to be divided into simply encourages the bits to be divided into smaller packets. Byte-mode drop would therefore irreversibly complicate any attempt to fix the Internet's incentive structures.

7. IANA Considerations

This document has no actions for IANA.

8. Conclusions

This memo identifies the three distinct stages of the congestion notification process where implementations need to decide whether to take packet size into account. The recommendations provided in Section 2 of this memo are different in each case:

- When network equipment measures the length of a queue, if it is not feasible to use time it is recommended to count in bytes if the network resource is congested by bytes, or to count in packets if is congested by packets.

- When network equipment decides whether to drop (or mark) a packet, it is recommended that the size of the particular packet should not be taken into account

- However, when a transport algorithm responds to a dropped or marked packet, the size of the rate reduction should be proportionate to the size of the packet.

In summary, the answers are 'it depends', 'no' and 'yes' respectively

For the specific case of RED, this means that byte-mode queue measurement will often be appropriate but the use of byte-mode drop is very strongly discouraged.

At the transport layer the IETF should continue updating congestion control protocols to take account of the size of each packet that indicates congestion. Also the IETF should continue to make protocols less sensitive to losing control packets like SYNs, pure ACKs and DNS exchanges. Although many control packets happen to be small, the alternative of network equipment favouring all small packets would be dangerous. That would create perverse incentives to split data transfers into smaller packets.

The memo develops these recommendations from principled arguments concerning scaling, layering, incentives, inherent efficiency, security and policeability. But it also addresses practical issues such as specific buffer architectures and incremental deployment. Indeed a limited survey of RED implementations is discussed, which shows there appears to be little, if any, installed base of RED's byte-mode drop. Therefore it can be deprecated with little, if any, incremental deployment complications.

The recommendations have been developed on the well-founded basis that most Internet resources are bit-congestible not packet-congestible. We need to know the likelihood that this assumption will prevail longer term and, if it might not, what protocol changes will be needed to cater for a mix of the two. The IRTF Internet Congestion Control Research Group (ICCRG) is currently working on these problems [RFC6077].

9. Acknowledgements

Thank you to Sally Floyd, who gave extensive and useful review comments. Also thanks for the reviews from Philip Eardley, David Black, Fred Baker, David Taht, Toby Moncaster, Arnaud Jacquet and Mirja Kuehlewind as well as helpful explanations of different hardware approaches from Larry Dunn and Fred Baker. We are grateful to Bruce Davie and his colleagues for providing a timely and efficient survey of RED implementation in Cisco's product range. Also grateful thanks to Toby Moncaster, Will Dormann, John Regnault, Simon Carter and Stefaan De Cnodder who further helped survey the current status of RED implementation and deployment and, finally, thanks to the anonymous individuals who responded.

Bob Briscoe and Jukka Manner were partly funded by Trilogy, a research project (ICT- 216372) supported by the European Community under its Seventh Framework Programme. The views expressed here are those of the authors only.

10. Comments Solicited

Comments and questions are encouraged and very welcome. They can be addressed to the IETF Transport Area working group mailing list <tsvwg@ietf.org>, and/or to the authors.

11. References

11.1. Normative References

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

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

11.2. Informative References

[BLUE02] Feng, W-c., Shin, K., Kandlur, D., and D. Saha, "The BLUE active queue management algorithms", IEEE/ACM Transactions on Networking 10(4) 513--528, August 2002, <http://dx.doi.org/10.1109/TNET.2002.801399>.

[CCvarPktSize] Widmer, J., Boutremans, C., and J-Y. Le

Boudec, "Congestion Control for Flows with Variable Packet Size", ACM CCR 34(2) 137--151, 2004, <http://doi.acm.org/10.1145/997150.997162>.

[CHOKe_Var_Pkt] Psounis, K., Pan, R., and B. Prabhaker, "Approximate Fair Dropping for Variable Length Packets", IEEE Micro 21(1):48--56, January-February 2001, <http://www.stanford.edu/~balaji/papers/01approximatefair.pdf}>.

[DRQ] Shin, M., Chong, S., and I. Rhee, "Dual-Resource TCP/AQM for Processing-Constrained Networks", IEEE/ACM Transactions on Networking Vol 16, issue 2, April 2008, <http://dx.doi.org/10.1109/TNET.2007.900415>.

[DupTCP] Wischik, D., "Short messages", Philosphical Transactions of the Royal Society A 366(1872):1941-1953, June 2008, <http://rsta.royalsocietypublishing.org/content/366/1872/1941.full.pdf+html>.

[ECNFixedWireless] Siris, V., "Resource Control for Elastic Traffic in CDMA Networks", Proc. ACM MOBICOM'02 , September 2002, <http://www.ics.forth.gr/netlab/publications/resource_control_elastic_cdma.html>.

[Evol_cc] Gibbens, R. and F. Kelly, "Resource pricing and the evolution of congestion control", Automatica 35(12)1969--1985, December 1999, <http://www.statslab.cam.ac.uk/~frank/evol.html>.

[GentleAggro] Flach, T., Dukkipati, N., Terzis, A., Raghavan, B., Cardwell, N., Cheng, Y., Jain, A., Hao, S., Katz-Bassett, E., and R. Govindan, "Reducing Web Latency: the Virtue of Gentle Aggression", ACM SIGCOMM CCR 43(4)159--170, August 2013, <http://doi.acm.org/10.1145/2486001.2486014>.

[I-D.nichols-tsvwg-codel] Nichols, K. and V. Jacobson, "Controlled Delay Active Queue Management",

draft-nichols-tsvwg-codel-01 (work in progress), February 2013.

[I-D.pan-tsvwg-pie] Pan, R., Natarajan, P., Piglione, C., and M. Prabhu, "PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem", draft-pan-tsvwg-pie-00 (work in progress), December 2012.

[IOSArch] Bollapragada, V., White, R., and C. Murphy, "Inside Cisco IOS Software Architecture", Cisco Press: CCIE Professional Development ISBN13: 978-1-57870-181-0, July 2000.

[PktSizeEquCC] Vasallo, P., "Variable Packet Size Equation-Based Congestion Control", ICSI Technical Report tr-00-008, 2000, <http://http.icsi.berkeley.edu/ftp/global/pub/techreports/2000/tr-00-008.pdf>.

[RED93] Floyd, S. and V. Jacobson, "Random Early Detection (RED) gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking 1(4) 397--413, August 1993, <http://www.icir.org/floyd/papers/red/red.html>.

[REDbias] Eddy, W. and M. Allman, "A Comparison of RED's Byte and Packet Modes", Computer Networks 42(3) 261--280, June 2003, <http://www.ir.bbn.com/documents/articles/redbias.ps>.

[REDbyte] De Cnodder, S., Elloumi, O., and K. Pauwels, "RED behavior with different packet sizes", Proc. 5th IEEE Symposium on Computers and Communications (ISCC) 793--799, July 2000, <http://www.icir.org/floyd/red/Elloumi99.pdf>.

[RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet",

RFC 2309, April 1998.

[RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, December 1998.

[RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, September 2000.

[RFC3426] Floyd, S., "General Architectural and Policy Considerations", RFC 3426, November 2002.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[RFC3714] Floyd, S. and J. Kempf, "IAB Concerns Regarding Congestion Control for Voice Traffic in the Internet", RFC 3714, March 2004.

[RFC4828] Floyd, S. and E. Kohler, "TCP Friendly Rate Control (TFRC): The Small-Packet (SP) Variant", RFC 4828, April 2007.

[RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, September 2008.

[RFC5562] Kuzmanovic, A., Mondal, A., Floyd, S., and K. Ramakrishnan, "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets", RFC 5562, June 2009.

[RFC5670] Eardley, P., "Metering and Marking Behaviour of PCN-Nodes", RFC 5670, November 2009.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.

[RFC5690] Floyd, S., Arcia, A., Ros, D., and J. Iyengar, "Adding Acknowledgement Congestion Control to TCP", RFC 5690, February 2010.

[RFC6077] Papadimitriou, D., Welzl, M., Scharf, M., and B. Briscoe, "Open Research Issues in Internet Congestion Control", RFC 6077, February 2011.

[RFC6679] Westerlund, M., Johansson, I., Perkins, C., O'Hanlon, P., and K. Carlberg, "Explicit Congestion Notification (ECN) for RTP over UDP", RFC 6679, August 2012.

[RFC6789] Briscoe, B., Woundy, R., and A. Cooper, "Congestion Exposure (ConEx) Concepts and Use Cases", RFC 6789, December 2012.

[Rate_fair_Dis] Briscoe, B., "Flow Rate Fairness: Dismantling a Religion", ACM CCR 37(2)63--74, April 2007, <http://portal.acm.org/citation.cfm?id=1232926>.

[gentle_RED] Floyd, S., "Recommendation on using the "gentle_" variant of RED", Web page , March 2000, <http://www.icir.org/floyd/red/gentle.html>.

[pBox] Floyd, S. and K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet", IEEE/ACM Transactions on Networking 7(4) 458--472, August 1999, <http://www.aciri.org/floyd/end2end-paper.html>.

[pktByteEmail] Floyd, S., "RED: Discussions of Byte and Packet Modes", email , March 1997, <http://www-nrg.ee.lbl.gov/floyd/REDaveraging.txt>.

Appendix A. Survey of RED Implementation Status

This Appendix is informative, not normative.

In May 2007 a survey was conducted of 84 vendors to assess how widely drop probability based on packet size has been implemented in RED Table 3. About 19% of those surveyed replied, giving a sample size

of 16. Although in most cases we do not have permission to identify the respondents, we can say that those that have responded include most of the larger equipment vendors, covering a large fraction of the market. The two who gave permission to be identified were Cisco and Alcatel-Lucent. The others range across the large network equipment vendors at L3 & L2, firewall vendors, wireless equipment vendors, as well as large software businesses with a small selection of networking products. All those who responded confirmed that they have not implemented the variant of RED with drop dependent on packet size (2 were fairly sure they had not but needed to check more thoroughly). At the time the survey was conducted, Linux did not implement RED with packet-size bias of drop, although we have not investigated a wider range of open source code.

Response	No. of vendors	%age of vendors
Not implemented	14	17%
Not implemented (probably)	2	2%
Implemented	0	0%
No response	68	81%
Total companies/orgs surveyed	84	100%

Table 3: Vendor Survey on byte-mode drop variant of RED (lower drop probability for small packets)

Where reasons have been given, the extra complexity of packet bias code has been most prevalent, though one vendor had a more principled reason for avoiding it--similar to the argument of this document.

Our survey was of vendor implementations, so we cannot be certain about operator deployment. But we believe many queues in the Internet are still tail-drop. The company of one of the co-authors (BT) has widely deployed RED, but many tail-drop queues are bound to still exist, particularly in access network equipment and on middleboxes like firewalls, where RED is not always available.

Routers using a memory architecture based on fixed size buffers with borrowing may also still be prevalent in the Internet. As explained in Section 4.2.1, these also provide a marginal (but legitimate) bias towards small packets. So even though RED byte-mode drop is not prevalent, it is likely there is still some bias towards small packets in the Internet due to tail drop and fixed buffer borrowing.

Appendix B. Sufficiency of Packet-Mode Drop

This Appendix is informative, not normative.

Here we check that packet-mode drop (or marking) in the network gives sufficiently generic information for the transport layer to use. We check against a 2x2 matrix of four scenarios that may occur now or in the future (Table 4). The horizontal and vertical dimensions have been chosen because each tests extremes of sensitivity to packet size in the transport and in the network respectively.

Note that this section does not consider byte-mode drop at all. Having deprecated byte-mode drop, the goal here is to check that packet-mode drop will be sufficient in all cases.

Network \ Transport	a) Independent of packet size of congestion notifications	b) Dependent on packet size of congestion notifications
1) Predominantly bit-congestible network	Scenario a1)	Scenario b1)
2) Mix of bit-congestible and pkt-congestible network	Scenario a2)	Scenario b2)

Table 4: Four Possible Congestion Scenarios

Appendix B.1 focuses on the horizontal dimension of Table 4 checking that packet-mode drop (or marking) gives sufficient information, whether or not the transport uses it--scenarios b) and a) respectively.

Appendix B.2 focuses on the vertical dimension of Table 4, checking that packet-mode drop gives sufficient information to the transport whether resources in the network are bit-congestible or packet-congestible (these terms are defined in Section 1.1).

Notation: To be concrete, we will compare two flows with different packet sizes, s_1 and s_2. As an example, we will take s_1 = 60B = 480b and s_2 = 1500B = 12,000b.

A flow's bit rate, x [bps], is related to its packet rate, u [pps], by

$$x(t) = s.u(t).$$

In the bit-congestible case, path congestion will be denoted by p_b, and in the packet-congestible case by p_p. When either case is implied, the letter p alone will denote path congestion.

B.1. Packet-Size (In)Dependence in Transports

In all cases we consider a packet-mode drop queue that indicates congestion by dropping (or marking) packets with probability p irrespective of packet size. We use an example value of loss (marking) probability, p=0.1%.

A transport like RFC5681 TCP treats a congestion notification on any packet whatever its size as one event. However, a network with just the packet-mode drop algorithm does give more information if the transport chooses to use it. We will use Table 5 to illustrate this.

We will set aside the last column until later. The columns labelled "Flow 1" and "Flow 2" compare two flows consisting of 60B and 1500B packets respectively. The body of the table considers two separate cases, one where the flows have equal bit-rate and the other with equal packet-rates. In both cases, the two flows fill a 96Mbps link. Therefore, in the equal bit-rate case they each have half the bit-rate (48Mbps). Whereas, with equal packet-rates, flow 1 uses 25 times smaller packets so it gets 25 times less bit-rate--it only gets 1/(1+25) of the link capacity (96Mbps/26 = 4Mbps after rounding). In contrast flow 2 gets 25 times more bit-rate (92Mbps) in the equal packet rate case because its packets are 25 times larger. The packet rate shown for each flow could easily be derived once the bit-rate was known by dividing bit-rate by packet size, as shown in the column labelled "Formula".

Parameter	Formula	Flow 1	Flow 2	Combined
Packet size	s/8	60B	1,500B	(Mix)
Packet size	s	480b	12,000b	(Mix)
Pkt loss probability	p	0.1%	0.1%	0.1%
EQUAL BIT-RATE CASE				
Bit-rate	x	48Mbps	48Mbps	96Mbps
Packet-rate	u = x/s	100kpps	4kpps	104kpps
Absolute pkt-loss-rate	p*u	100pps	4pps	104pps
Absolute bit-loss-rate	p*u*s	48kbps	48kbps	96kbps
Ratio of lost/sent pkts	p*u/u	0.1%	0.1%	0.1%
Ratio of lost/sent bits	p*u*s/(u*s)	0.1%	0.1%	0.1%
EQUAL PACKET-RATE CASE				
Bit-rate	x	4Mbps	92Mbps	96Mbps
Packet-rate	u = x/s	8kpps	8kpps	15kpps
Absolute pkt-loss-rate	p*u	8pps	8pps	15pps
Absolute bit-loss-rate	p*u*s	4kbps	92kbps	96kbps
Ratio of lost/sent pkts	p*u/u	0.1%	0.1%	0.1%
Ratio of lost/sent bits	p*u*s/(u*s)	0.1%	0.1%	0.1%

Table 5: Absolute Loss Rates and Loss Ratios for Flows of Small and Large Packets and Both Combined

So far we have merely set up the scenarios. We now consider congestion notification in the scenario. Two TCP flows with the same round trip time aim to equalise their packet-loss-rates over time. That is the number of packets lost in a second, which is the packets per second (u) multiplied by the probability that each one is dropped (p). Thus TCP converges on the "Equal packet-rate" case, where both flows aim for the same "Absolute packet-loss-rate" (both 8pps in the table).

Packet-mode drop actually gives flows sufficient information to measure their loss-rate in bits per second, if they choose, not just packets per second. Each flow can count the size of a lost or marked packet and scale its rate-response in proportion (as TFRC-SP does). The result is shown in the row entitled "Absolute bit-loss-rate", where the bits lost in a second is the packets per second (u) multiplied by the probability of losing a packet (p) multiplied by the packet size (s). Such an algorithm would try to remove any imbalance in bit-loss-rate such as the wide disparity in the "Equal packet-rate" case (4kbps vs. 92kbps). Instead, a packet-size-dependent algorithm would aim for equal bit-loss-rates, which would drive both flows towards the "Equal bit-rate" case, by driving them to equal bit-loss-rates (both 48kbps in this example).

The explanation so far has assumed that each flow consists of packets of only one constant size. Nonetheless, it extends naturally to flows with mixed packet sizes. In the right-most column of Table 5 a flow of mixed size packets is created simply by considering flow 1 and flow 2 as a single aggregated flow. There is no need for a flow to maintain an average packet size. It is only necessary for the transport to scale its response to each congestion indication by the size of each individual lost (or marked) packet. Taking for example the "Equal packet-rate" case, in one second about 8 small packets and 8 large packets are lost (making closer to 15 than 16 losses per second due to rounding). If the transport multiplies each loss by its size, in one second it responds to 8*480b and 8*12,000b lost bits, adding up to 96,000 lost bits in a second. This double checks correctly, being the same as 0.1% of the total bit-rate of 96Mbps. For completeness, the formula for absolute bit-loss-rate is p(u1*s1+ u2*s2).

Incidentally, a transport will always measure the loss probability the same irrespective of whether it measures in packets or in bytes. In other words, the ratio of lost to sent packets will be the same as the ratio of lost to sent bytes. (This is why TCP's bit rate is still proportional to packet size even when byte-counting is used, as recommended for TCP in [RFC5681], mainly for orthogonal security reasons.) This is intuitively obvious by comparing two example flows; one with 60B packets, the other with 1500B packets. If both flows pass through a queue with drop probability 0.1%, each flow will lose 1 in 1,000 packets. In the stream of 60B packets the ratio of bytes lost to sent will be 60B in every 60,000B; and in the stream of 1500B packets, the loss ratio will be 1,500B out of 1,500,000B. When the transport responds to the ratio of lost to sent packets, it will measure the same ratio whether it measures in packets or bytes: 0.1% in both cases. The fact that this ratio is the same whether measured in packets or bytes can be seen in Table 5, where the ratio of lost to sent packets and the ratio of lost to sent bytes is always 0.1% in all cases (recall that the scenario was set up with p=0.1%).

This discussion of how the ratio can be measured in packets or bytes is only raised here to highlight that it is irrelevant to this memo! Whether a transport depends on packet size or not depends on how this ratio is used within the congestion control algorithm.

So far we have shown that packet-mode drop passes sufficient information to the transport layer so that the transport can take account of bit-congestion, by using the sizes of the packets that indicate congestion. We have also shown that the transport can choose not to take packet size into account if it wishes. We will now consider whether the transport can know which to do.

B.2. Bit-Congestible and Packet-Congestible Indications

As a thought-experiment, imagine an idealised congestion notification protocol that supports both bit-congestible and packet-congestible resources. It would require at least two ECN flags, one for each of bit-congestible and packet-congestible resources.

1. A packet-congestible resource trying to code congestion level p_p into a packet stream should mark the idealised 'packet congestion' field in each packet with probability p_p irrespective of the packet's size. The transport should then take a packet with the packet congestion field marked to mean just one mark, irrespective of the packet size.

2. A bit-congestible resource trying to code time-varying byte-congestion level p_b into a packet stream should mark the 'byte congestion' field in each packet with probability p_b, again irrespective of the packet's size. Unlike before, the transport should take a packet with the byte congestion field marked to count as a mark on each byte in the packet.

This hides a fundamental problem--much more fundamental than whether we can magically create header space for yet another ECN flag, or whether it would work while being deployed incrementally. Distinguishing drop from delivery naturally provides just one implicit bit of congestion indication information--the packet is either dropped or not. It is hard to drop a packet in two ways that are distinguishable remotely. This is a similar problem to that of distinguishing wireless transmission losses from congestive losses.

This problem would not be solved even if ECN were universally deployed. A congestion notification protocol must survive a transition from low levels of congestion to high. Marking two states is feasible with explicit marking, but much harder if packets are dropped. Also, it will not always be cost-effective to implement AQM at every low level resource, so drop will often have to suffice.

We are not saying two ECN fields will be needed (and we are not saying that somehow a resource should be able to drop a packet in one of two different ways so that the transport can distinguish which sort of drop it was!). These two congestion notification channels are a conceptual device to illustrate a dilemma we could face in the future. Section 3 gives four good reasons why it would be a bad idea to allow for packet size by biasing drop probability in favour of small packets within the network. The impracticality of our thought experiment shows that it will be hard to give transports a practical way to know whether to take account of the size of congestion indication packets or not.

Fortunately, this dilemma is not pressing because by design most equipment becomes bit-congested before its packet-processing becomes congested (as already outlined in Section 1.1). Therefore transports can be designed on the relatively sound assumption that a congestion indication will usually imply bit-congestion.

Nonetheless, although the above idealised protocol isn't intended for implementation, we do want to emphasise that research is needed to predict whether there are good reasons to believe that packet congestion might become more common, and if so, to find a way to somehow distinguish between bit and packet congestion [RFC3714].

Recently, the dual resource queue (DRQ) proposal [DRQ] has been made on the premise that, as network processors become more cost effective, per packet operations will become more complex (irrespective of whether more function in the network is desirable). Consequently the premise is that CPU congestion will become more common. DRQ is a proposed modification to the RED algorithm that folds both bit congestion and packet congestion into one signal (either loss or ECN).

Finally, we note one further complication. Strictly, packet-congestible resources are often cycle-congestible. For instance, for routing look-ups load depends on the complexity of each look-up and whether the pattern of arrivals is amenable to caching or not. This also reminds us that any solution must not require a forwarding engine to use excessive processor cycles in order to decide how to say it has no spare processor cycles.

Appendix C. Byte-mode Drop Complicates Policing Congestion Response

This section is informative, not normative.

There are two main classes of approach to policing congestion response: i) policing at each bottleneck link or ii) policing at the edges of networks. Packet-mode drop in RED is compatible with either, while byte-mode drop precludes edge policing.

The simplicity of an edge policer relies on one dropped or marked packet being equivalent to another of the same size without having to know which link the drop or mark occurred at. However, the byte-mode drop algorithm has to depend on the local MTU of the line--it needs to use some concept of a 'normal' packet size. Therefore, one dropped or marked packet from a byte-mode drop algorithm is not necessarily equivalent to another from a different link. A policing function local to the link can know the local MTU where the congestion occurred. However, a policer at the edge of the network cannot, at least not without a lot of complexity.

The early research proposals for type (i) policing at a bottleneck link [pBox] used byte-mode drop, then detected flows that contributed disproportionately to the number of packets dropped. However, with no extra complexity, later proposals used packet mode drop and looked for flows that contributed a disproportionate amount of dropped bytes [CHOKe_Var_Pkt].

Work is progressing on the congestion exposure protocol (ConEx [RFC6789]), which enables a type (ii) edge policer located at a user's attachment point. The idea is to be able to take an integrated view of the effect of all a user's traffic on any link in the internetwork. However, byte-mode drop would effectively preclude such edge policing because of the MTU issue above.

Indeed, making drop probability depend on the size of the packets that bits happen to be divided into would simply encourage the bits to be divided into smaller packets in order to confuse policing. In contrast, as long as a dropped/marked packet is taken to mean that all the bytes in the packet are dropped/marked, a policer can remain robust against bits being re-divided into different size packets or across different size flows [Rate_fair_Dis].

Appendix D. Changes from Previous Versions

To be removed by the RFC Editor on publication.

Full incremental diffs between each version are available at <http://tools.ietf.org/wg/tsvwg/draft-ietf-tsvwg-byte-pkt-congest/> (courtesy of the rfcdiff tool):

From -11 to -12: Following the second pass through the IESG:

* Section 2.1 [Barry Leiba]:
  + s/No other choice makes sense,/Subject to the exceptions below, no other choice makes sense,/
  + s/Exceptions to these recommendations MAY be necessary /Exceptions to these recommendations may be necessary /
* Sections 3.2 and 4.2.3 [Joel Jaeggli]:
  + Added comment to section 4.2.3 that the examples given are not in widespread production use, but they give evidence that it is possible to follow the advice given.
  + Section 4.2.3:

- OLD: Although there are no known proposals, it would also be possible and perfectly valid to make control packets robust against drop by explicitly requesting a lower drop probability using their Diffserv code point [RFC2474] to request a scheduling class with lower drop.
  NEW: Although there are no known proposals, it would also be possible and perfectly valid to make control packets robust against drop by requesting a scheduling class with lower drop probability, by re-marking to a Diffserv code point [RFC2474] within the same behaviour aggregate.

- appended "Similarly applications, over non-TCP transports could make any packets that are effectively control packets more robust by using Diffserv, data duplication, FEC etc."

+ Updated Wischik ref and added "Reducing Web Latency: the Virtue of Gentle Aggression" ref.

* Expanded more abbreviations (CoDel, PIE, MTU).

* Section 1. Intro [Stephen Farrell]:

  + In the places where the doc desribes the dichotomy between ’long-term goal’ and ’expediency’ the words long term goal and expedient have been introduced, to more explicitly refer back to this introductory para (S.2.1 & S.2.3).

  + Added explanation of what scaling with packet size means.

* Conclusions [Benoit Claise]:

  + OLD: For the specific case of RED, this means that byte-mode queue measurement will often be appropriate although byte-mode drop is strongly deprecated.
    NEW: For the specific case of RED, this means that byte-mode queue measurement will often be appropriate but the use of byte-mode drop is very strongly discouraged.

From -10 to -11: Following a further WGLC:

* Abstract: clarified that advice applies to all AQMs including newer ones

* Abstract & Intro: changed ’read’ to ’detect’, because you don’t read losses, you detect them.

* S.1. Introduction: Disambiguated summary of advice on queue measurement.

* Clarified that the doc deprecates any preference based solely on packet size, it's not only against preferring smaller packets.

* S.4.1.2. Congestion Measurement without a Queue: Explained that a queue of TXOPs represents a queue into spectrum congested by too many bits.

* S.5.2: Bit- & Packet-congestible Network: Referred to explanation in S.4.1.2 to make the point that TXOPs are not a primary unit of workload like bits and packets are, even though you get queues of TXOPs.

* 6. Security: Disambiguated 'bias towards'.

* 8. Conclusions: Made consistent with recommendation to use time if possible for queue measurement.

From -09 to -10: Following IESG review:

* Updates 2309: Left header unchanged reflecting eventual IESG consensus [Sean Turner, Pete Resnick].

* S.1 Intro: This memo adds to the congestion control principles enumerated in BCP 41 [Pete Resnick]

* Abstract, S.1, S.1.1, s.1.2 Intro, Scoping and Example: Made applicability to all AQMs clearer listing some more example AQMs and explained that we always use RED for examples, but this doesn't mean it's not applicable to other AQMs. [A number of reviewers have described the draft as "about RED"]

* S.1 & S.2.1 Queue measurement: Explained that the choice between measuring the queue in packets or bytes is only relevant if measuring it in time units is infeasible [So as not to imply that we haven't noticed the advances made by PDPC & CoDel]

* S.1.1. Terminology: Better explained why hybrid systems congested by both packets and bytes are often designed to be treated as bit-congestible [Richard Barnes].

* S.2.1. Queue measurement advice: Added examples. Added a counter-example to justify SHOULDs rather than MUSTs. Pointed to S.4.1 for a list of more complicated scenarios. [Benson

Schliesser, OpsDir]

* S2.2. Recommendation on Encoding Congestion Notification: Removed SHOULD treat packets equally, leaving only SHOULD NOT drop dependent on packet size, to avoid it sounding like we're saying QoS is not allowed. Pointed to possible app-specific legacy use of byte-mode as a counter-example that prevents us saying MUST NOT. [Pete Resnick]

* S.2.3. Recommendation on Responding to Congestion: capitalised the two SHOULDs in recommendations for TCP, and gave possible counter-examples. [noticed while dealing with Pete Resnick's point]

* S2.4. Splitting & Merging: RTCP -> RTP/RTCP [Pete McCann, Gen-ART]

* S.3.2 Small != Control: many control packets are small -> ...tend to be small [Stephen Farrell]

* S.3.1 Perverse incentives: Changed transport designers to app developers [Stephen Farrell]

* S.4.1.1. Fixed Size Packet Buffers: Nearly completely re-written to simplify and to reverse the advice when the underlying resource is bit-congestible, irrespective of whether the buffer consists of fixed-size packet buffers. [Richard Barnes & Benson Schliesser]

* S.4.2.1.2. Packet Size Bias Regardless of AQM: Largely re-written to reflect the earlier change in advice about fixed-size packet buffers, and to primarily focus on getting rid of tail-drop, not various nuances of tail-drop. [Richard Barnes & Benson Schliesser]

* Editorial corrections [Tim Bray, AppsDir, Pete McCann, Gen-ART and others]

* Updated refs (two I-Ds have become RFCs). [Pete McCann]

From -08 to -09: Following WG last call:

* S.2.1: Made RED-related queue measurement recommendations clearer

* S.2.3: Added to "Recommendation on Responding to Congestion" to make it clear that we are definitely not saying transports have to equalise bit-rates, just how to do it and not do it, if you

want to.

* S.3: Clarified motivation sections S.3.3 "Transport-Independent Network" and S.3.5 "Implementation Efficiency"

* S.3.4: Completely changed motivating argument from "Scaling Congestion Control with Packet Size" to "Partial Deployment of AQM".

From -07 to -08:

* Altered abstract to say it provides best current practice and highlight that it updates RFC2309

* Added null IANA section

* Updated refs

From -06 to -07:

* A mix-up with the corollaries and their naming in 2.1 to 2.3 fixed.

From -05 to -06:

* Primarily editorial fixes.

From -04 to -05:

* Changed from Informational to BCP and highlighted non-normative sections and appendices

* Removed language about consensus

* Added "Example Comparing Packet-Mode Drop and Byte-Mode Drop"

* Arranged "Motivating Arguments" into a more logical order and completely rewrote "Transport-Independent Network" & "Scaling Congestion Control with Packet Size" arguments. Removed "Why Now?"

* Clarified applicability of certain recommendations

* Shifted vendor survey to an Appendix

* Cut down "Outstanding Issues and Next Steps"

* Re-drafted the start of the conclusions to highlight the three distinct areas of concern

* Completely re-wrote appendices

* Editorial corrections throughout.

From -03 to -04:

* Reordered Sections 2 and 3, and some clarifications here and there based on feedback from Colin Perkins and Mirja Kuehlewind.

From -02 to -03 (this version)

* Structural changes:

  + Split off text at end of "Scaling Congestion Control with Packet Size" into new section "Transport-Independent Network"

  + Shifted "Recommendations" straight after "Motivating Arguments" and added "Conclusions" at end to reinforce Recommendations

  + Added more internal structure to Recommendations, so that recommendations specific to RED or to TCP are just corollaries of a more general recommendation, rather than being listed as a separate recommendation.

  + Renamed "State of the Art" as "Critical Survey of Existing Advice" and retitled a number of subsections with more descriptive titles.

  + Split end of "Congestion Coding: Summary of Status" into a new subsection called "RED Implementation Status".

  + Removed text that had been in the Appendix "Congestion Notification Definition: Further Justification".

* Reordered the intro text a little.

* Made it clearer when advice being reported is deprecated and when it is not.

* Described AQM as in network equipment, rather than saying "at the network layer" (to side-step controversy over whether functions like AQM are in the transport layer but in network

equipment).

* Minor improvements to clarity throughout

From -01 to -02:

* Restructured the whole document for (hopefully) easier reading and clarity. The concrete recommendation, in RFC2119 language, is now in Section 8.

From -00 to -01:

* Minor clarifications throughout and updated references

From briscoe-byte-pkt-mark-02 to ietf-byte-pkt-congest-00:

* Added note on relationship to existing RFCs

* Posed the question of whether packet-congestion could become common and deferred it to the IRTF ICCRG. Added ref to the dual-resource queue (DRQ) proposal.

* Changed PCN references from the PCN charter & architecture to the PCN marking behaviour draft most likely to imminently become the standards track WG item.

From -01 to -02:

* Abstract reorganised to align with clearer separation of issue in the memo.

* Introduction reorganised with motivating arguments removed to new Section 3.

* Clarified avoiding lock-out of large packets is not the main or only motivation for RED.

* Mentioned choice of drop or marking explicitly throughout, rather than trying to coin a word to mean either.

* Generalised the discussion throughout to any packet forwarding function on any network equipment, not just routers.

* Clarified the last point about why this is a good time to sort out this issue: because it will be hard / impossible to design new transports unless we decide whether the network or the transport is allowing for packet size.

* Added statement explaining the horizon of the memo is long term, but with short term expediency in mind.

* Added material on scaling congestion control with packet size (Section 3.4).

* Separated out issue of normalising TCP's bit rate from issue of preference to control packets (Section 3.2).

* Divided up Congestion Measurement section for clarity, including new material on fixed size packet buffers and buffer carving (Section 4.1.1 & Section 4.2.1) and on congestion measurement in wireless link technologies without queues (Section 4.1.2).

* Added section on 'Making Transports Robust against Control Packet Losses' (Section 4.2.3) with existing & new material included.

* Added tabulated results of vendor survey on byte-mode drop variant of RED (Table 3).

From -00 to -01:

* Clarified applicability to drop as well as ECN.

* Highlighted DoS vulnerability.

* Emphasised that drop-tail suffers from similar problems to byte-mode drop, so only byte-mode drop should be turned off, not RED itself.

* Clarified the original apparent motivations for recommending byte-mode drop included protecting SYNs and pure ACKs more than equalising the bit rates of TCPs with different segment sizes. Removed some conjectured motivations.

* Added support for updates to TCP in progress (ackcc & ecn-syn-ack).

* Updated survey results with newly arrived data.

* Pulled all recommendations together into the conclusions.

* Moved some detailed points into two additional appendices and a note.

* Considerable clarifications throughout.

* Updated references

Authors' Addresses

Bob Briscoe
BT
B54/77, Adastral Park
Martlesham Heath
Ipswich IP5 3RE
UK

Phone: +44 1473 645196
EMail: bob.briscoe@bt.com
URI: http://bobbriscoe.net/

Jukka Manner
Aalto University
Department of Communications and Networking (Comnet)
P.O. Box 13000
FIN-00076 Aalto
Finland

Phone: +358 9 470 22481
EMail: jukka.manner@aalto.fi
URI: http://www.netlab.tkk.fi/~jmanner/

Transport Area Working Group
Internet-Draft
Updates: 2780, 2782, 3828, 4340,
4960, 5595 (if approved)
Intended status: BCP
Expires: August 16, 2011

M. Cotton
ICANN
L. Eggert
Nokia
J. Touch
USC/ISI
M. Westerlund
Ericsson
S. Cheshire
Apple

February 12, 2011

Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry
draft-ietf-tsvwg-iana-ports-10

Abstract

This document defines the procedures that the Internet Assigned Numbers Authority (IANA) uses when handling assignment and other requests related to the Service Name and Transport Protocol Port Number Registry. It also discusses the rationale and principles behind these procedures and how they facilitate the long-term sustainability of the registry.

This document updates IANA's procedures by obsoleting the previous UDP and TCP port assignment procedures defined in Sections 8 and 9.1 of the IANA allocation guidelines [RFC2780], and it updates the IANA Service Name and Port assignment procedures for UDP-Lite [RFC3828], DCCP [RFC4340] [RFC5595] and SCTP [RFC4960]. It also updates the DNS SRV specification [RFC2782] to clarify what a service name is and how it is registered.

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 August 16, 2011.

Table of Contents

1. Introduction

For many years, the assignment of new service names and port number values for use with the Transmission Control Protocol (TCP) [RFC0793] and the User Datagram Protocol (UDP) [RFC0768] have had less than clear guidelines. New transport protocols have been added - the Stream Control Transmission Protocol (SCTP) [RFC4960] and the Datagram Congestion Control Protocol (DCCP) [RFC4342] - and new mechanisms like DNS SRV records [RFC2782] have been developed, each with separate registries and separate guidelines. The community also recognized the need for additional procedures beyond just assignment; notably modification, revocation, and release.

A key element of the procedural streamlining specified in this document is to establish identical assignment procedures for all IETF transport protocols. This document brings the IANA procedures for TCP and UDP in line with those for SCTP and DCCP, resulting in a single process that requesters and IANA follow for all requests for all transport protocols, including future protocols not yet defined.

In addition to detailing the IANA procedures for the initial assignment of service names and port numbers, this document also specifies post-assignment procedures that until now have been handled in an ad hoc manner. These include procedures to de-assign a port number that is no longer in use, to take a port number assigned for one service that is no longer in use and reuse it for another service, and the procedure by which IANA can unilaterally revoke a prior port number assignment. Section 8 discusses the specifics of these procedures and processes that requesters and IANA follow for all requests for all current and future transport protocols.

IANA is the authority for assigning service names and port numbers. The registries that are created to store these assignments are maintained by IANA. For protocols developed by IETF working groups, IANA now also offers a method for the "early assignment" [RFC4020] of service names and port numbers, as described in Section 8.1.

This document updates IANA’s procedures for UDP and TCP port numbers by obsoleting Sections 8 and 9.1 of the IANA assignment guidelines [RFC2780]. (Note that other sections of the IANA assignment guidelines, relating to the protocol field values in IPv4 headers, were also updated in February 2008 [RFC5237].) This document also updates the IANA assignment procedures for DCCP [RFC4340] [RFC5595]and SCTP [RFC4960].

The Lightweight User Datagram Protocol (UDP-Lite) shares the port space with UDP. The UDP-Lite specification [RFC3828] says: "UDP-Lite uses the same set of port number values assigned by the IANA for use

by UDP". An update of the UDP procedures therefore also results in a corresponding update of the UDP-Lite procedures.

This document also clarifies what a service name is and how it is assigned. This will impact the DNS SRV specification [RFC2782], because that specification merely makes a brief mention that the symbolic names of services are defined in "Assigned Numbers" [RFC1700], without stating to which section it refers within that 230-page document. The DNS SRV specification may have been referring to the list of Port Assignments (known as /etc/services on Unix), or to the "Protocol And Service Names" section, or to both, or to some other section. Furthermore, "Assigned Numbers" [RFC1700] has been obsoleted [RFC3232] and has been replaced by on-line registries [PORTREG][PROTSERVREG].

The development of new transport protocols is a major effort that the IETF does not undertake very often. If a new transport protocol is standardized in the future, it is expected to follow these guidelines and practices around using service names and port numbers as much as possible, for consistency.

2. Motivation

Information about the assignment procedures for the port registry has existed in three locations: the forms for requesting port number assignments on the IANA web site [SYSFORM][USRFORM], an introductory text section in the file listing the port number assignments themselves (known as the port numbers registry) [PORTREG], and two brief sections of the IANA Allocation Guidelines [RFC2780].

Similarly, the procedures surrounding service names have been historically unclear. Service names were originally created as mnemonic identifiers for port numbers without a well-defined syntax, apart from the 14-character limit mentioned on the IANA website [SYSFORM][USRFORM]. Even that length limit has not been consistently applied, and some assigned service names are 15 characters long. When service identification via DNS SRV Resource Records (RRs) was introduced [RFC2782], it became useful to start assigning service names alone, and because IANA had no procedure for assigning a service name without an associated port number, this lead to the creation of an informal temporary service name registry outside of the control of IANA, which now contains roughly 500 service names [SRVREG].

This document aggregates all this scattered information into a single reference that aligns and clearly defines the management procedures for both service names and port numbers. It gives more detailed

guidance to prospective requesters of service names and ports than the existing documentation, and it streamlines the IANA procedures for the management of the registry, so that requests can be completed in a timely manner.

This document defines rules for assignment of service names without associated port numbers, for such usages as DNS SRV records [RFC2782], which was not possible under the previous IANA procedures. The document also merges service name assignments from the non-IANA ad hoc registry [SRVREG] and from the IANA "Protocol and Service Names" registry [PROTSERVREG] into the IANA "Service Name and Transport Protocol Port Number" registry [PORTREG], which from here on is the single authoritative registry for service names and port numbers.

An additional purpose of this document is to describe the principles that guide the IETF and IANA in their role as the long-term joint stewards of the service name and port number registry. TCP and UDP have had remarkable success over the last decades. Thousands of applications and application-level protocols have service names and port numbers assigned for their use, and there is every reason to believe that this trend will continue into the future. It is hence extremely important that management of the registry follow principles that ensure its long-term usefulness as a shared resource. Section 7 discusses these principles in detail.

3. Background

The Transmission Control Protocol (TCP) [RFC0793] and the User Datagram Protocol (UDP) [RFC0768] have enjoyed a remarkable success over the decades as the two most widely used transport protocols on the Internet. They have relied on the concept of "ports" as logical entities for Internet communication. Ports serve two purposes: first, they provide a demultiplexing identifier to differentiate transport sessions between the same pair of endpoints, and second, they may also identify the application protocol and associated service to which processes connect. Newer transport protocols, such as the Stream Control Transmission Protocol (SCTP) [RFC4960] and the Datagram Congestion Control Protocol (DCCP) [RFC4342] have also adopted the concept of ports for their communication sessions and use 16-bit port numbers in the same way as TCP and UDP (and UDP-Lite [RFC3828], a variant of UDP).

Port numbers are the original and most widely used means for application and service identification on the Internet. Ports are 16-bit numbers, and the combination of source and destination port numbers together with the IP addresses of the communicating end

systems uniquely identifies a session of a given transport protocol. Port numbers are also known by their associated service names such as "telnet" for port number 23 and "http" (as well as "www" and "www-http") for port number 80.

Hosts running services, hosts accessing services on other hosts, and intermediate devices (such as firewalls and NATs) that restrict services need to agree on which service corresponds to a particular destination port. Although this is ultimately a local decision with meaning only between the endpoints of a connection, it is common for many services to have a default port upon which those servers usually listen, when possible, and these ports are recorded by the Internet Assigned Numbers Authority (IANA) through the service name and port number registry [PORTREG].

Over time, the assumption that a particular port number necessarily implies a particular service may become less true. For example, multiple instances of the same service on the same host cannot generally listen on the same port, and multiple hosts behind the same NAT gateway cannot all have a mapping for the same port on the external side of the NAT gateway, whether using static port mappings configured by hand by the user, or dynamic port mappings configured automatically using a port mapping protocol like NAT Port Mapping Protocol (NAT-PMP) [I-D.cheshire-nat-pmp] or Internet Gateway Device (IGD) [IGD].

Applications may use port numbers directly, look up port numbers based on service names via system calls such as getservbyname() on UNIX, look up port numbers by performing queries for DNS SRV records [RFC2782][I-D.cheshire-dnsext-dns-sd], or determine port numbers in a variety of other ways like the TCP Port Service Multiplexer (TCPMUX) [RFC1078].

Designers of applications and application-level protocols may apply to IANA for an assigned service name and port number for a specific application, and may - after assignment - assume that no other application will use that service name or port number for its communication sessions. Application designers also have the option of requesting only an assigned service name without a corresponding fixed port number if their application does not require one, such as applications that use DNS SRV records to look up port numbers dynamically at runtime. Because the port number space is finite (and therefore conservation is an important goal) the alternative of using service names instead of port numbers is RECOMMENDED whenever possible.

4. Conventions Used in this Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in "Key words for use in RFCs to Indicate Requirement Levels" [RFC2119].

This document uses the term "assignment" to refer to the procedure by which IANA provides service names and/or port numbers to requesting parties; other RFCs refer to this as "allocation" or "registration". This document assumes that all these terms have the same meaning, and will use terms other than "assignment" when quoting from or referring to text in these other documents.

5. Service Names

Service names are the unique key in the Service Name and Transport Protocol Port Number Registry. This unique symbolic name for a service may also be used for other purposes, such as in DNS SRV records [RFC2782]. Within the registry, this unique key ensures that different services can be unambiguously distinguished, thus preventing name collisions and avoiding confusion about who is the Assignee for a particular entry.

There may be more than one service name associated with a particular transport protocol and port. There are three ways that such port number overloading can occur:

- Overloading occurs when one service is an extension of another service, and an in-band mechanism exists for determining if the extension is present or not. One example is port 3478, which has the service name aliases "stun" and "turn". TURN [RFC5766] is an extension to the STUN [RFC5389] service. TURN-enabled clients wishing to locate TURN servers could attempt to discover "stun" services and then check in-band if the server also supports TURN, but this would be inefficient. Enabling them to directly query for "turn" servers by name is a better approach. (Note that TURN servers in this case should also be locatable via a "stun" discovery, because every TURN server is also a STUN server.)

- By historical accident, the service name "http" has two synonyms "www" and "www-http". When used in SRV records [RFC2782] and similar service discovery mechanisms, only the service name "http" should be used, not these additional names. If a server were to advertise "www", it would not be discovered by clients browsing for "http". Advertising or browsing for the aliases as well as the primary service name is inefficient, and achieves nothing that

is not already achieved by using the service name "http" exclusively.

- As indicated in this document in Section 10.1, overloading has been used to create replacement names that are consistent with the syntax this document prescribes for legacy names that do not conform to this syntax already. For such cases, only the new name should be used in SRV records, to avoid the same issues as with historical cases of multiple names, and also because the legacy names are incompatible with SRV record use.

Assignment requests for new names for existing registered services will be rejected, as a result. Implementers are requested to inform IANA if they discover other cases where a single service has multiple names, so that one name may be recorded as the primary name for service discovery purposes.

Service names are assigned on a "first come, first served" basis, as described in Section 8.1. Names should be brief and informative, avoiding words or abbreviations that are redundant in the context of the registry (e.g., "port", "service", "protocol", etc.) Names referring to discovery services, e.g., using multicast or broadcast to identify endpoints capable of a given service, SHOULD use an easily identifiable suffix (e.g., "-disc").

5.1. Service Name Syntax

Valid service names are hereby normatively defined as follows:

- MUST be at least 1 character and no more than 15 characters long
- MUST contain only US-ASCII [ANSI.X3-4.1986] letters 'A' - 'Z' and 'a' - 'z', digits '0' - '9', and hyphens ('-', ASCII 0x2D or decimal 45)
- MUST contain at least one letter ('A' - 'Z' or 'a' - 'z')
- MUST NOT begin or end with a hyphen
- hyphens MUST NOT be adjacent to other hyphens

The reason for requiring at least one letter is to avoid service names like "23" (could be confused with a numeric port) or "6000-6063" (could be confused with a numeric port range). Although service names may contain both upper-case and lower-case letters, case is ignored for comparison purposes, so both "http" and "HTTP" denote the same service.

Service names are purely opaque identifiers, and no semantics are implied by any superficial structure that a given service name may appear to have. For example, a company called "Example" may choose to register service names "Example-Foo" and "Example-Bar" for its "Foo" and "Bar" products, but the "Example" company cannot claim to "own" all service names beginning with "Example-"; they cannot prevent someone else from registering "Example-Baz" for a different service, and they cannot prevent other developers from using the "Example-Foo" and "Example-Bar" service types in order to interoperate with the "Foo" and "Bar" products. Technically speaking, in service discovery protocols, service names are merely a series of byte values on the wire; for the mnemonic convenience of human developers it can be convenient to interpret those byte values as human-readable ASCII characters, but software should treat them as purely opaque identifiers and not attempt to parse them for any additional embedded meaning.

In approximately 98% of cases, the new "service name" is exactly the same as the old historic "short name" from the IANA web forms [SYSFORM] [USRFORM]. In approximately 2% of cases, the new "service name" is derived from the old historic "short name" as described below in Section 10.1.

The rules for valid service names, excepting the limit of 15 characters maximum, are also expressed below (as a non-normative convenience) using ABNF [RFC5234].

```
SRVNAME = *(1*DIGIT [HYPHEN]) ALPHA *([HYPHEN] ALNUM)
ALNUM   = ALPHA / DIGIT     ; A-Z, a-z, 0-9
HYPHEN  = %x2D              ; "-"
ALPHA   = %x41-5A / %x61-7A ; A-Z / a-z [RFC5234]
DIGIT   = %x30-39           ; 0-9       [RFC5234]
```

5.2. Service Name Usage in DNS SRV Records

The DNS SRV specification [RFC2782] states that the Service Label part of the owner name of a DNS SRV record includes a "Service" element, described as "the symbolic name of the desired service", but as discussed above, it is not clear precisely what this means.

This document clarifies that the Service Label MUST be a service name as defined herein with an underscore prepended. The service name SHOULD be registered with IANA and recorded in the Service Name and Transport Protocol Port Number Registry [PORTREG].

The details of using Service Names in SRV Service Labels are

specified in the DNS SRV specification [RFC2782].

6. Port Number Ranges

TCP, UDP, UDP-Lite, SCTP and DCCP use 16-bit namespaces for their port number registries. The port registries for all of these transport protocols are subdivided into three ranges of numbers [RFC1340], and Section 8.1.2 describes the IANA procedures for each range in detail:

- the System Ports, also known as the Well Known Ports, from 0-1023 (assigned by IANA)
- the User Ports, also known as the Registered Ports, from 1024-49151 (assigned by IANA)
- the Dynamic Ports, also known as the Private or Ephemeral Ports, from 49152-65535 (never assigned)

Of the assignable port ranges (System Ports and User Ports, i.e., port numbers 0-49151), individual port numbers are in one of three states at any given time:

- Assigned: Assigned port numbers are currently assigned to the service indicated in the registry.
- Unassigned: Unassigned port numbers are currently available for assignment upon request, as per the procedures outlined in this document.
- Reserved: Reserved port numbers are not available for regular assignment; they are "assigned to IANA" for special purposes. Reserved port numbers include values at the edges of each range, e.g., 0, 1023, 1024, etc., which may be used to extend these ranges or the overall port number space in the future.

In order to keep the size of the registry manageable, IANA typically only records the Assigned and Reserved service names and port numbers in the registry. Unassigned values are typically not explicitly listed. (There are very many Unassigned service names and enumerating them all would not be practical.)

As a data point, when this document was written, approximately 76% of the TCP and UDP System Ports were assigned, and approximately 9% of the User Ports were assigned. (As noted, Dynamic Ports are never assigned.)

6.1. Service names and Port Numbers for Experimentation

Of the System Ports, two TCP and UDP port numbers (1021 and 1022), together with their respective service names ("exp1" and "exp2"), have been assigned for experimentation with new applications and application-layer protocols that require a port number in the assigned ports range [RFC4727].

Please refer to Sections 1 and 1.1 of "Assigning Experimental and Testing Numbers Considered Useful" [RFC3692] for how these experimental port numbers are to be used.

This document assigns the same two service names and port numbers for experimentation with new application-layer protocols over SCTP and DCCP in Section 10.2.

Unfortunately, it can be difficult to limit access to these ports. Users SHOULD take measures to ensure that experimental ports are connecting to the intended process. For example, users of these experimental ports might include a 64-bit nonce, once on each segment of a message-oriented channel (e.g., UDP), or once at the beginning of a byte-stream (e.g., TCP), which is used to confirm that the port is being used as intended. Such confirmation of intended use is especially important when these ports are associated with privileged (e.g., system or administrator) processes.

7. Principles for Service Name and Transport Protocol Port Number Registry Management

Management procedures for the service name and transport protocol port number registry include assignment of service names and port numbers upon request, as well as management of information about existing assignments. The latter includes maintaining contact and description information about assignments, revoking abandoned assignments, and redefining assignments when needed. Of these procedures, careful port number assignment is most critical, in order to continue to conserve the remaining port numbers.

As noted earlier, only about 9% of the User Port space is currently assigned. The current rate of assignment is approximately 400 ports per year, and has remained steady for the past 8 years. At that rate, if similar conservation continues, this resource will sustain another 85 years of assignment - without the need to resort to reassignment of released values or revocation. The namespace available for service names is much larger, which allows for simpler management procedures.

7.1. Past Principles

The principles for service name and port number management are based on the recommendations of the IANA "Expert Review" team. Until recently, that team followed a set of informal guidelines developed based on the review experience from previous assignment requests. These original guidelines, although informal, had never been publicly documented. They are recorded here for historical purposes only; the current guidelines are described in Section 7.2. These guidelines previously were:

- TCP and UDP ports were simultaneously assigned when either was requested
- Port numbers were the primary assignment; service names were informative only, and did not have a well-defined syntax
- Port numbers were conserved informally, and sometimes inconsistently (e.g., some services were assigned ranges of many port numbers even where not strictly necessary)
- SCTP and DCCP service name and port number registries were managed separately from the TCP/UDP registries
- Service names could not be assigned in the old ports registry without assigning an associated port number at the same time

7.2. Updated Principles

This section summarizes the current principles by which IANA both handles the Service Name and Transport Protocol Port Number Registry and attempts to conserve the port number space. This description is intended to inform applicants requesting service names and port numbers. IANA has flexibility beyond these principles when handling assignment requests; other factors may come into play, and exceptions may be made to best serve the needs of the Internet. Applicants should be aware that IANA decisions are not required to be bound to these principles. These principles and general advice to users on port use are expected to change over time and are therefore documented separately, please see [I-D.touch-tsvwg-port-use].

IANA strives to assign service names that do not request an associated port number assignment under a simple "First Come, First Served" policy [RFC5226]. IANA MAY, at its discretion, refer service name requests to "Expert Review" in cases of mass assignment requests or other situations where IANA believes expert review is advisable [RFC5226]; use of the "Expert Review" helps advise IANA informally in cases where "IETF Review" or "IESG Review" is used, as with most IETF

protocols.

The basic principle of service name and port number registry management is to conserve use of the port space where possible. Extensions to support larger port number spaces would require changing many core protocols of the current Internet in a way that would not be backward compatible and interfere with both current and legacy applications.

Conservation of the port number space is required because this space is a limited resource, so applications are expected to participate in the traffic demultiplexing process where feasible. The port numbers are expected to encode as little information as possible that will still enable an application to perform further demultiplexing by itself. In particular, the principles form a goal that IANA strives to achieve for new applications (with exceptions as deemed appropriate, especially as for extensions to legacy services) as follows:

- IANA strives to assign only one assigned port number per service or application
- IANA strives to assign only one assigned port number for all variants of a service (e.g., for updated versions of a service)
- IANA strives to encourage the deployment of secure protocols
- IANA strives to assign only one assigned port number for all different types of device using or participating in the same service
- IANA strives to assign port numbers only for the transport protocol(s) explicitly named in an assignment request
- IANA may recover unused port numbers, via the new procedures of de-assignment, revocation, and transfer

Where possible, a given service is expected to demultiplex messages if necessary. For example, applications and protocols are expected to include in-band version information, so that future versions of the application or protocol can share the same assigned port. Applications and protocols are also expected to be able to efficiently use a single assigned port for multiple sessions, either by demultiplexing multiple streams within one port, or using the assigned port to coordinate using dynamic ports for subsequent exchanges (e.g., in the spirit of FTP [RFC0959].

These principles of port conservation are explained further in

[I-D.touch-tsvwg-port-use]. That document explains in further detail how ports are used in various ways, notably:

- as endpoint process identifiers
- as application protocol identifiers
- for firewall filtering purposes

Both the process identifier and the protocol identifier uses suggest that anything a single process can demultiplex, or that can be encoded into a single protocol, should be. The firewall filtering use suggests that some uses that could be multiplexed or encoded could instead be separated to allow for easier firewall management. Note that this latter use is much less sound, because port numbers have meaning only for the two endpoints involved in a connection, and drawing conclusions about the service that generated a given flow based on observed port numbers is not always reliable.

IANA will begin assigning port numbers for only those transport protocols explicitly included in an assignment request. This ends the long-standing practice of automatically assigning a port number to an application for both TCP and UDP, even if the request is for only one of these transport protocols. The new assignment procedure conserves resources by assigning a port number to an application for only those transport protocols (TCP, UDP, SCTP and/or DCCP) it actually uses. The port number will be marked as Reserved - instead of Assigned - in the port number registries of the other transport protocols. When applications start supporting the use of some of those additional transport protocols, the Assignee for the assignment MUST request IANA convert these reserved ports into assignments. An application MUST NOT assume that it can use a port number assigned to it for use with one transport protocol with another transport protocol without IANA converting the reservation into an assignment.

When the available pool of unassigned numbers has run out in a port range, it will be necessary for IANA to consider the Reserved ports for assignment. This is part of the motivation for not automatically assigning ports for transport protocols other than the requested one(s). This will allow more ports to be available for assignment at that point. To help conserve ports, application developers SHOULD request assignment of only those transport protocols that their application currently uses.

Conservation of port numbers is improved by procedures that allow previously allocated port numbers to become Unassigned, either through de-assignment or through revocation, and by a procedure that lets application designers transfer an assigned but unused port

number to a new application. Section 8 describes these procedures, which until now were undocumented. Port number conservation is also improved by recommending that applications that do not require an assigned port should register only a service name without an associated port number.

8. IANA Procedures for Managing the Service Name and Transport Protocol Port Number Registry

This section describes the process for handling requests associated with IANA's management of the Service Name and Transport Protocol Port Number Registry. Such requests include initial assignment, de-assignment, reuse, changes to the service name, and updates to the contact information or description associated with an assignment. Revocation is as additional process, initiated by IANA.

8.1. Service Name and Port Number Assignment

Assignment refers to the process of providing service names or port numbers to applicants. All such assignments are made from service names or port numbers that are Unassigned or Reserved at the time of the assignment.

- Unassigned names and numbers are allocated according to the rules described in Section 8.1.2 below.

- Reserved numbers and names are generally only assigned by a Standards Action or an IESG Approval, and MUST be accompanied by a statement explaining the reason a Reserved number or name is appropriate for this action. The only exception to this rule is that the current Assignee of a port number MAY request the assignment of the corresponding Reserved port number for other transport protocols when needed. IANA will initiate an "Expert Review" [RFC5226] for such requests.

When an assignment for one or more transport protocols is approved, the port number for any non-requested transport protocol(s) will be marked as Reserved. IANA SHOULD NOT assign that port number to any other application or service until no other port numbers remain Unassigned in the requested range.

8.1.1. General Assignment Procedure

A service name or port number assignment request contains the following information. The service name is the unique identifier of a given service:

Service Name (REQUIRED)
Transport Protocol(s) (REQUIRED)
Assignee (REQUIRED)
Contact (REQUIRED)
Description (REQUIRED)
Reference (REQUIRED)
Port Number (OPTIONAL)
Service Code (REQUIRED for DCCP only)
Known Unauthorized Uses (OPTIONAL)
Assignment Notes (OPTIONAL)

- Service Name: A desired unique service name for the service associated with the assignment request MUST be provided. This name may be used with various service selection and discovery mechanisms (including, but not limited to, DNS SRV records [RFC2782]). The name MUST be compliant with the syntax defined in Section 5.1. In order to be unique, they MUST NOT be identical to any currently assigned service name in the IANA registry [PORTREG]. Service names are case-insensitive; they may be provided and entered into the registry with mixed case for clarity, but for the comparison purposes the case is ignored.

- Transport Protocol(s): The transport protocol(s) for which an assignment is requested MUST be provided. This field is currently limited to one or more of TCP, UDP, SCTP, and DCCP. Requests without any port assignment and only a service name are still required to indicate which protocol the service uses.

- Assignee: Name and email address of the party to whom the assignment is made. This is REQUIRED. The Assignee is the organization, company or individual person responsible for the initial assignment. For assignments done through RFCs published via the "IETF Document Stream" [RFC4844], the Assignee will be the IESG <iesg@ietf.org>.

- Contact: Name and email address of the Contact person for the assignment. This is REQUIRED. The Contact person is the responsible person for the Internet community to send questions to. This person is also authorized to submit changes on behalf of the Assignee; in cases of conflict between the Assignee and the Contact, the Assignee decisions take precedence. Additional address information MAY be provided. For assignments done through RFCs published via the "IETF Document Stream" [RFC4844], the Contact will be the IETF Chair <chair@ietf.org>.

- Description: A short description of the service associated with the assignment request is REQUIRED. It should avoid all but the most well-known acronyms.

- Reference: A description of (or a reference to a document describing) the protocol or application using this port. The description must state whether the protocol uses IP-layer broadcast, multicast, or anycast communication.

  For assignments requesting only a Service Name, or a Service Name and User Port, a statement that the protocol is proprietary and not publicly documented is also acceptable, provided that the required information regarding the use of IP broadcast, multicast, or anycast is given.

  For any assignment request that includes a User Port, the assignment request MUST explain why a port number in the Dynamic Ports range is unsuitable for the given application.

  For any assignment request that includes a System Port, the assignment request MUST explain why a port number in the User Ports or Dynamic Ports ranges is unsuitable, and a reference to a stable protocol specification document MUST be provided.

  IANA MAY accept early assignment [RFC4020] requests (known as "early allocation" therein) from IETF Working Groups that reference a sufficiently stable Internet Draft instead of a published Standards-Track RFC.

- Port Number: If assignment of a port number is desired, either the port number the requester suggests for assignment or indication of port range (user or system) MUST be provided. If only a service name is to be assigned, this field is left empty. If a specific port number is requested, IANA is encouraged to assign the requested number. If a range is specified, IANA will choose a suitable number from the User or System Ports ranges. Note that the applicant MUST NOT use the requested port prior to the completion of the assignment.

- Service Code: If the assignment request includes DCCP as a transport protocol then the request MUST include a desired unique DCCP service code [RFC5595], and MUST NOT include a requested DCCP service code otherwise. Section 19.8 of the DCCP specification [RFC4340] defines requirements and rules for assignment, updated by this document. Note that, as per [RFC5595], some service codes are not assigned; zero (absence of a meaningful service code) or 4294967295 (invalid service code) are permanently reserved, and the Private service codes 1056964608-1073741823 (i.e., 32-bit

values with the high-order byte equal to a value of 63, corresponding to the ASCII character '?') are not centrally assigned.

- Known Unauthorized Uses: A list of uses by applications or organizations who are not the Assignee. This list may be augmented by IANA after assignment when unauthorized uses are reported.

- Assignment Notes: Indications of owner/name change, or any other assignment process issue. This list may be updated by IANA after assignment to help track changes to an assignment, e.g., de-assignment, owner/name changes, etc.

If the assignment request is for the addition of a new transport protocol to an already-assigned service name and the requester is not the Assignee or Contact for the already-assigned service name, IANA needs to confirm with the Assignee for the existing assignment whether this addition is appropriate.

If the assignment request is for a new service name sharing the same port as an already-assigned service name (see port number overloading in Section 5), IANA needs to confirm with the Assignee for the existing service name and other appropriate experts whether the overloading is appropriate.

When IANA receives an assignment request - containing the above information - that is requesting a port number, IANA SHALL initiate an "Expert Review" [RFC5226] in order to determine whether an assignment should be made. For requests that are not seeking a port number, IANA SHOULD assign the service name under a simple "First Come First Served" policy [RFC5226].

8.1.2. Variances for Specific Port Number Ranges

Section 6 describes the different port number ranges. It is important to note that IANA applies slightly different procedures when managing the different port ranges of the service name and port number registry:

- Ports in the Dynamic Ports range (49152-65535) have been specifically set aside for local and dynamic use and cannot be assigned through IANA. Application software may simply use any dynamic port that is available on the local host, without any sort of assignment. On the other hand, application software MUST NOT assume that a specific port number in the Dynamic Ports range will always be available for communication at all times, and a port number in that range hence MUST NOT be used as a service

identifier.

- Ports in the User Ports range (1024-49151) are available for assignment through IANA, and MAY be used as service identifiers upon successful assignment. Because assigning a port number for a specific application consumes a fraction of the shared resource that is the port number registry, IANA will require the requester to document the intended use of the port number. For most IETF protocols, ports in the User Ports range will be assigned under the "IETF Review" or "IESG Approval" procedures [RFC5226] and no further documentation is required. Where these procedures do not apply, then the requester must input the documentation to the "Expert Review" procedure [RFC5226], by which IANA will have a technical expert review the request to determine whether to grant the assignment. Regardless of the path ("IETF Review", "IESG Approval", or "Expert Review"), the submitted documentation is expected to be the same, as described in this section, and MUST explain why using a port number in the Dynamic Ports range is unsuitable for the given application. Further, IANA MAY utilize the Expert Review process informally to inform their position in participating in "IETF Review" and "IESG Review"

- Ports in the System Ports range (0-1023) are also available for assignment through IANA. Because the System Ports range is both the smallest and the most densely allocated, the requirements for new assignments are more strict than those for the User Ports range, and will only be granted under the "IETF Review" or "IESG Approval" procedures [RFC5226]. A request for a System Port number MUST document *both* why using a port number from the Dynamic Ports range is unsuitable *and* why using a port number from the User Ports range is unsuitable for that application.

8.2. Service Name and Port Number De-Assignment

The Assignee of a granted port number assignment can return the port number to IANA at any time if they no longer have a need for it. The port number will be de-assigned and will be marked as Reserved. IANA should not re-assign port numbers that have been de-assigned until all unassigned port numbers in the specific range have been assigned.

Before proceeding with a port number de-assignment, IANA needs to reasonably establish that the value is actually no longer in use.

Because there is much less danger of exhausting the service name space compared to the port number space, it is RECOMMENDED that a given service name remain assigned even after all associated port number assignments have become de-assigned. Under this policy, it will appear in the registry as if it had been created through a

service name assignment request that did not include any port numbers.

On rare occasions, it may still be useful to de-assign a service name. In such cases, IANA will mark the service name as Reserved. IANA will involve their IESG-appointed expert in such cases.

IANA will include a comment in the registry when de-assignment happens to indicate its historic usage.

8.3. Service Name and Port Number Reuse

If the Assignee of a granted port number assignment no longer has a need for the assigned number, but would like to reuse it for a different application, they can submit a request to IANA to do so.

Logically, port number reuse is to be thought of as a de-assignment (Section 8.2) followed by an immediate (re-)assignment (Section 8.1) of the same port number for a new application. Consequently, the information that needs to be provided about the proposed new use of the port number is identical to what would need to be provided for a new port number assignment for the specific ports range.

Because there is much less danger of exhausting the service name space compared to the port number space, it is RECOMMENDED that the original service name associated with the prior use of the port number remains assigned, and a new service name be created and associated with the port number. This is again consistent with viewing a reuse request as a de-assignment followed by an immediate (re-)assignment. Re-using an assigned service name for a different application is NOT RECOMMENDED.

IANA needs to carefully review such requests before approving them. In some instances, the Expert Reviewer will determine that the application the port number was assigned to has found usage beyond the original Assignee, or that there is a concern that it may have such users. This determination MUST be made quickly. A community call concerning revocation of a port number (see below) MAY be considered, if a broader use of the port number is suspected.

8.4. Service Name and Port Number Revocation

A port number revocation can be thought of as an IANA-initiated de-assignment (Section 8.2), and has exactly the same effect on the registry.

Sometimes, it will be clear that a specific port number is no longer in use and that IANA can revoke it and mark it as Reserved. At other

times, it may be unclear whether a given assigned port number is still in use somewhere in the Internet. In those cases, IANA must carefully consider the consequences of revoking the port number, and SHOULD only do so if there is an overwhelming need.

With the help of their IESG-appointed Expert Reviewer, IANA SHALL formulate a request to the IESG to issue a four-week community call concerning the pending port number revocation. The IESG and IANA, with the Expert Reviewer's support, SHALL determine promptly after the end of the community call whether revocation should proceed and then communicate their decision to the community. This procedure typically involves similar steps to de-assignment except that it is initiated by IANA.

Because there is much less danger of exhausting the service name space compared to the port number space, revoking service names is NOT RECOMMENDED.

8.5. Service Name and Port Number Transfers

The value of service names and port numbers is defined by their careful management as a shared Internet resource, whereas enabling transfer allows the potential for associated monetary exchanges. As a result, the IETF does not permit service name or port number assignments to be transferred between parties, even when they are mutually consenting.

The appropriate alternate procedure is a coordinated de-assignment and assignment: The new party requests the service name or port number via an assignment and the previous party releases its assignment via the de-assignment procedure outlined above.

With the help of their IESG-appointed Expert Reviewer, IANA SHALL carefully determine if there is a valid technical, operational or managerial reason to grant the requested new assignment.

8.6. Maintenance Issues

In addition to the formal procedures described above, updates to the Description and Contact information are coordinated by IANA in an informal manner, and may be initiated by either the Assignee or by IANA, e.g., by the latter requesting an update to current Contact information. (Note that the Assignee cannot be changed as a separate procedure; see instead Section 8.5 above.)

8.7. Disagreements

In the case of disagreements around any request there is the possibility of appeal following the normal appeals process for IANA assignments as defined by Section 7 of "Guidelines for Writing an IANA Considerations Section in RFCs" [RFC5226].

9. Security Considerations

The IANA guidelines described in this document do not change the security properties of UDP, TCP, SCTP, or DCCP.

Assignment of a service name or port number does not in any way imply an endorsement of an application or product, and the fact that network traffic is flowing to or from an assigned port number does not mean that it is "good" traffic, or even that it is used by the assigned service. Firewall and system administrators should choose how to configure their systems based on their knowledge of the traffic in question, not based on whether or not there is an assigned service name or port number.

Services are expected to include support for security, either as default or dynamically negotiated in-band. The use of separate service name or port number assignments for secure and insecure variants of the same service is to be avoided in order to discourage the deployment of insecure services.

10. IANA Considerations

This document obsoletes Sections 8 and 9.1 of the March 2000 IANA Allocation Guidelines [RFC2780].

Upon approval of this document, IANA is requested to contact Stuart Cheshire, maintainer of the independent service name registry [SRVREG], in order to merge the contents of that private registry into the official IANA registry. It is expected that the independent registry web page will be updated with pointers to the IANA registry and to this RFC.

IANA is instructed to create a new service name entry in the service name and port number registry [PORTREG] for any entry in the "Protocol and Service Names" registry [PROTSERVREG] that does not already have one assigned.

IANA is also instructed to indicate in the Assignment Notes for "www" and "www-http" that they are duplicate terms that refer to the "http"

service, and should not be used for discovery purposes. For this conceptual service (human-readable web pages served over HTTP) the correct service name to use for service discovery purposes is "http" (see Section 5).

10.1. Service Name Consistency

Section 8.1 defines which character strings are well-formed service names, which until now had not been clearly defined. The definition in Section 8.1 was chosen to allow maximum compatibility of service names with current and future service discovery mechanisms.

As of August 5, 2009 approximately 98% of the so-called "Short Names" from existing port number assignments [PORTREG] meet the rules for legal service names stated in Section 8.1, and hence for these services their service name will be exactly the same as their "Short Name".

The remaining approximately 2% of the exiting "Short Names" are not suitable to be used directly as well-formed service names because they contain illegal characters such as asterisks, dots, pluses, slashes, or underscores. All existing "Short Names" conform to the length requirement of 15 characters or fewer. For these unsuitable "Short Names", listed in the table below, the service name will be the Short Name with any illegal characters replaced by hyphens. IANA SHALL add an entry to the registry giving the new well-formed primary service name for the existing service, that otherwise duplicates the original assignment information. In the description field of this new entry giving the primary service name, IANA SHALL record that it assigns a well-formed service name for the previous service and reference the original assignment. In the Assignment Notes field of the original assignment, IANA SHALL add a note that this entry is an alias to the new well-formed service name, and that the old service name is historic, not usable for use with many common service discovery mechanisms.

Names containing illegal characters to be replaced by hyphens:

914c/g	acmaint_dbd	acmaint_transd
atex_elmd	avanti_cdp	badm_priv
badm_pub	bdir_priv	bdir_pub
bmc_ctd_ldap	bmc_patroldb	boks_clntd
boks_servc	boks_servm	broker_service
bues_service	canit_store	cedros_fds
cl/1	contamac_icm	corel_vncadmin
csc_proxy	cvc_hostd	dbcontrol_agent
dec_dlm	dl_agent	documentum_s
dsmeter_iatc	dsx_monitor	elpro_tunnel
elvin_client	elvin_server	encrypted_admin
erunbook_agent	erunbook_server	esri_sde
EtherNet/IP-1	EtherNet/IP-2	event_listener
flr_agent	gds_db	ibm_wrless_lan
iceedcp_rx	iceedcp_tx	iclcnet_svinfo
idig_mux	ife_icorp	instl_bootc
instl_boots	intel_rci	interhdl_elmd
lan900_remote	LiebDevMgmt_A	LiebDevMgmt_C
LiebDevMgmt_DM	mapper-ws_ethd	matrix_vnet
mdbs_daemon	menandmice_noh	msl_lmd
nburn_id	ncr_ccl	nds_sso
netmap_lm	nms_topo_serv	notify_srvr
novell-lu6.2	nuts_bootp	nuts_dem
ocs_amu	ocs_cmu	pipe_server
pra_elmd	printer_agent	redstorm_diag
redstorm_find	redstorm_info	redstorm_join
resource_mgr	rmonitor_secure	rsvp_tunnel
sai_sentlm	sge_execd	sge_qmaster
shiva_confsrvr	sql*net	srvc_registry
stm_pproc	subntbcst_tftp	udt_os
universe_suite	veritas_pbx	vision_elmd
vision_server	wrs_registry	z39.50

Following the example set by the "application/whoispp-query" MIME Content-Type [RFC2957], the service name for "whois++" will be "whoispp".

10.2. Port Numbers for SCTP and DCCP Experimentation

Two System UDP and TCP ports, 1021 and 1022, have been reserved for experimental use [RFC4727]. This document assigns the same port numbers for SCTP and DCCP, updates the TCP and UDP assignments, and also instructs IANA to automatically assign these two port numbers for any future transport protocol with a similar 16-bit port number

namespace.

Note that these port numbers are meant for temporary experimentation and development in controlled environments. Before using these port numbers, carefully consider the advice in Section 6.1 in this document, as well as in Sections 1 and 1.1 of "Assigning Experimental and Testing Numbers Considered Useful" [RFC3692]. Most importantly, application developers must request a permanent port number assignment from IANA as described in Section 8.1 before any kind of non-experimental deployment.

Service Name	exp1
Transport Protocol	DCCP, SCTP, TCP, UDP
Assignee	IESG <iesg@ietf.org>
Contact	IETF Chair <chair@ietf.org>
Description	RFC3692-style Experiment 1
Reference	[RFC4727] [RFCyyyy]
Port Number	1021

Service Name	exp2
Transport Protocol	DCCP, SCTP, TCP, UDP
Assignee	IESG <iesg@ietf.org>
Contact	IETF Chair <chair@ietf.org>
Description	RFC3692-style Experiment 2
Reference	[RFC4727] [RFCyyyy]
Port Number	1022

[RFC Editor Note: Please change "yyyy" to the RFC number allocated to this document before publication.]

10.3. Updates to DCCP Registries

This document updates the IANA assignment procedures for the DCCP Port Number and DCCP Service Codes Registries [RFC4340].

10.3.1. DCCP Service Code Registry

Service Codes are assigned first-come-first-served according to Section 19.8 of the DCCP specification [RFC4340]. This document updates that section by extending the guidelines given there in the following ways:

- IANA MAY assign new Service Codes without seeking Expert Review using their discretion, but SHOULD seek expert review if a request asks for more than five Service Codes.

- IANA should feel free to contact the DCCP Expert Reviewer with any questions related to requests for DCCP-related codepoints.

10.3.2. DCCP Port Numbers Registry

The DCCP ports registry is defined by Section 19.9 of the DCCP specification [RFC4340]. Assignments in this registry require prior assignment of a Service Code. Not all Service Codes require IANA-assigned ports. This document updates that section by extending the guidelines given there in the following way:

- IANA should normally assign a value in the range 1024-49151 to a DCCP server port. IANA requests to assign port numbers in the System Ports range (0 through 1023), require an "IETF Review" [RFC5226] prior to assignment by IANA [RFC4340].

- IANA MUST NOT assign more than one DCCP server port to a single service code value.

- The assignment of multiple service codes to the same DCCP port is allowed, but subject to expert review.

- The set of Service Code values associated with a DCCP server port should be recorded in the service name and port number registry.

- A request for additional Service Codes to be associated with an already-allocated Port Number requires Expert Review. These requests will normally be accepted when they originate from the contact associated with the port assignment. In other cases, these applications will be expected to use an unallocated port, when this is available.

The DCCP specification [RFC4340] notes that a short port name MUST be associated with each DCCP server port that has been assigned. This document clarifies that this short port name is the Service Name as defined here, and this name MUST be unique.

11. Contributors

Alfred Hoenes (ah@tr-sys.de) and Allison Mankin (mankin@psg.com) have contributed text and ideas to this document.

12. Acknowledgments

The text in Section 10.3 is based on a suggestion originally proposed as a part of the DCCP Service Codes document [RFC5595] by Gorry Fairhurst.

Lars Eggert is partly funded by the Trilogy Project [TRILOGY], a research project supported by the European Commission under its Seventh Framework Program.

13. References

13.1. Normative References

[ANSI.X3-4.1986] American National Standards Institute, "Coded Character Set - 7-bit American Standard Code for Information Interchange", ANSI X3.4, 1986.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

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

[RFC2780] Bradner, S. and V. Paxson, "IANA Allocation Guidelines For Values In the Internet Protocol and Related Headers", BCP 37, RFC 2780, March 2000.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

[RFC3828] Larzon, L-A., Degermark, M., Pink, S., Jonsson, L-E., and G. Fairhurst, "The Lightweight User Datagram Protocol (UDP-Lite)", RFC 3828, July 2004.

[RFC4020] Kompella, K. and A. Zinin, "Early IANA Allocation of Standards Track Code Points", BCP 100, RFC 4020, February 2005.

[RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, March 2006.

[RFC4727] Fenner, B., "Experimental Values In IPv4, IPv6, ICMPv4, ICMPv6, UDP, and TCP Headers", RFC 4727, November 2006.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5595] Fairhurst, G., "The Datagram Congestion Control Protocol (DCCP) Service Codes", RFC 5595, September 2009.

13.2. Informative References

[I-D.cheshire-dnsext-dns-sd]
Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", draft-cheshire-dnsext-dns-sd-08 (work in progress), January 2011.

[I-D.cheshire-nat-pmp]
Cheshire, S., "NAT Port Mapping Protocol (NAT-PMP)", draft-cheshire-nat-pmp-03 (work in progress), April 2008.

[I-D.touch-tsvwg-port-use]
Touch, J., "Recommendations for Transport Port Uses", draft-touch-tsvwg-port-use-00 (work in progress), December 2010.

[IGD] UPnP Forum, "Internet Gateway Device (IGD) V 1.0", November 2001.

[PORTREG] Internet Assigned Numbers Authority (IANA), "Service Name and Transport Protocol Port Number Registry", http://www.iana.org/assignments/port-numbers.

[PROTSERVREG]
Internet Assigned Numbers Authority (IANA), "Protocol and Service Names Registry", http://www.iana.org/assignments/service-names.

[RFC0959] Postel, J. and J. Reynolds, "File Transfer Protocol", STD 9, RFC 959, October 1985.

[RFC1078] Lottor, M., "TCP port service Multiplexer (TCPMUX)", RFC 1078, November 1988.

[RFC1340] Reynolds, J. and J. Postel, "Assigned Numbers", RFC 1340,

July 1992.

[RFC1700] Reynolds, J. and J. Postel, "Assigned Numbers", RFC 1700, October 1994.

[RFC2957] Daigle, L. and P. Faltstrom, "The application/ whoispp-query Content-Type", RFC 2957, October 2000.

[RFC3232] Reynolds, J., "Assigned Numbers: RFC 1700 is Replaced by an On-line Database", RFC 3232, January 2002.

[RFC3692] Narten, T., "Assigning Experimental and Testing Numbers Considered Useful", BCP 82, RFC 3692, January 2004.

[RFC4342] Floyd, S., Kohler, E., and J. Padhye, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC)", RFC 4342, March 2006.

[RFC4844] Daigle, L. and Internet Architecture Board, "The RFC Series and RFC Editor", RFC 4844, July 2007.

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

[RFC5237] Arkko, J. and S. Bradner, "IANA Allocation Guidelines for the Protocol Field", BCP 37, RFC 5237, February 2008.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, April 2010.

[SRVREG] "DNS SRV Service Types Registry", http://www.dns-sd.org/ServiceTypes.html.

[SYSFORM] Internet Assigned Numbers Authority (IANA), "Application for System (Well Known) Port Number", http://www.iana.org/.

[TRILOGY] "Trilogy Project", http://www.trilogy-project.org/.

[USRFORM] Internet Assigned Numbers Authority (IANA), "Application for User (Registered) Port Number", http://www.iana.org/.

Authors' Addresses

Michelle Cotton
Internet Corporation for Assigned Names and Numbers
4676 Admiralty Way, Suite 330
Marina del Rey, CA 90292
USA

Phone: +1 310 823 9358
Email: michelle.cotton@icann.org
URI: http://www.iana.org/

Lars Eggert
Nokia Research Center
P.O. Box 407
Nokia Group 00045
Finland

Phone: +358 50 48 24461
Email: lars.eggert@nokia.com
URI: http://research.nokia.com/people/lars_eggert/

Joe Touch
USC/ISI
4676 Admiralty Way
Marina del Rey, CA 90292
USA

Phone: +1 310 448 9151
Email: touch@isi.edu
URI: http://www.isi.edu/touch

Magnus Westerlund
Ericsson
Farogatan 6
Stockholm 164 80
Sweden

Phone: +46 8 719 0000
Email: magnus.westerlund@ericsson.com

Stuart Cheshire
Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
USA

Phone: +1 408 974 3207
Email: cheshire@apple.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 10, 2012

R. Stewart
Adara Networks
M. Tuexen
Muenster Univ. of Appl. Sciences
P. Lei
Cisco Systems, Inc.
December 8, 2011

Stream Control Transmission Protocol (SCTP) Stream Reconfiguration
draft-ietf-tsvwg-sctp-strrst-13.txt

Abstract

Many applications that use SCTP want the ability to "reset" a stream. The intention of resetting a stream is to set the numbering sequence of the stream back to 'zero' with a corresponding notification to the application layer that the reset has been performed. Applications requiring this feature want it so that they can "re-use" streams for different purposes but still utilize the stream sequence number so that the application can track the message flows. Thus, without this feature, a new use of an old stream would result in message numbers greater than expected unless there is a protocol mechanism to "reset the streams back to zero". This document also includes methods for resetting the transport sequence numbers, adding additional streams and resetting all stream sequence numbers.

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 June 10, 2012.

Copyright Notice

Table of Contents

1. Introduction

Many applications that use SCTP as defined in [RFC4960] want the ability to "reset" a stream. The intention of resetting a stream is to set the stream sequence numbers (SSNs) of the stream back to 'zero' with a corresponding notification to the application layer that the reset has been performed. Applications requiring this feature want to "re-use" streams for different purposes but still utilize the stream sequence number so that the application can track the message flows. Thus, without this feature, a new use of an old stream would result in message numbers greater than expected unless there is a protocol mechanism to "reset the streams back to zero". This document also includes methods for resetting the transport sequence numbers (TSNs), adding additional streams and resetting all stream sequence numbers.

The socket API for SCTP defined in [I-D.ietf-tsvwg-sctpsocket] exposes the sequence numbers used by SCTP for user message transfer. Therefore, resetting them can be used by application writers. Please note that the corresponding sequence number for TCP is not exposed via the socket API for TCP.

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. New Chunk Type

This section defines the new chunk type that will be used to re-configure streams. Table 1 illustrates the new chunk type.

Chunk Type	Chunk Name
0x82	Re-configuration Chunk (RE-CONFIG)

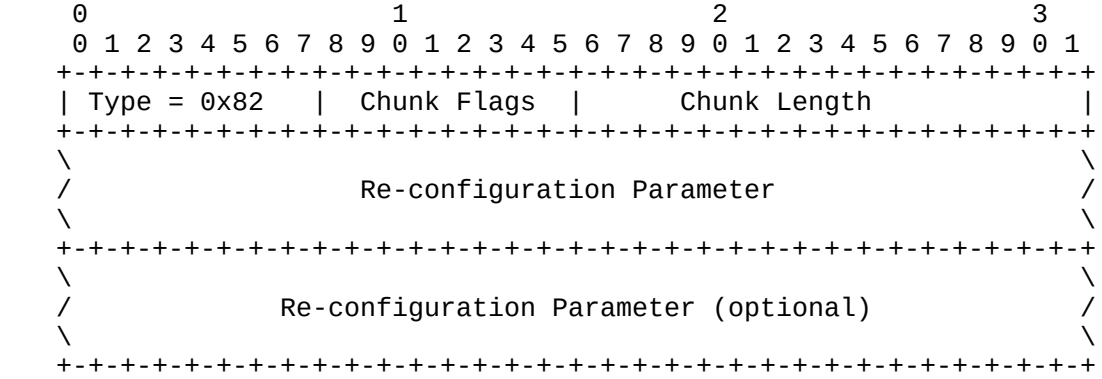
Table 1

It should be noted that the format of the RE-CONFIG chunk requires the receiver to ignore the chunk if it is not understood and continue processing all chunks that follow. This is accomplished by the use of the upper bits of the chunk type as described in section 3.2 of [RFC4960].

All transported integer numbers are in "network byte order" a.k.a., Big Endian.

3.1. RE-CONFIG Chunk

This document adds one new chunk type to SCTP. The chunk has the following format:



Chunk Type: 1 byte (unsigned integer)
This field holds the IANA defined chunk type for the RE-CONFIG chunk. The suggested value of this field for IANA is 0x82.

Chunk Flags: 1 byte (unsigned integer)
This field is set to 0 by the sender and ignored by the receiver.

Chunk Length: 2 bytes (unsigned integer)
This field holds the length of the chunk in bytes, including the Chunk Type, Chunk Flags and Chunk Length.

Re-configuration Parameter
This field holds a Re-configuration Request Parameter or a Re-configuration Response Parameter.

Note that each RE-CONFIG chunk holds at least one parameter and at most two parameters. Only the following combinations are allowed:

1. Outgoing SSN Reset Request Parameter.

2. Incoming SSN Reset Request Parameter.

3. Outgoing SSN Reset Request Parameter, Incoming SSN Reset Request Parameter.

4. SSN/TSN Reset Request Parameter.

5. Add Outgoing Streams Request Parameter.

6. Add Incoming Streams Request Parameter.

7. Add Outgoing Streams Request Parameter, Add Incoming Streams Request Parameter.

8. Re-configuration Response Parameter.

9. Re-configuration Response Parameter, Outgoing SSN Reset Request Parameter.

10. Re-configuration Response Parameter, Re-configuration Response Parameter.

If a sender transmits an unsupported combination, the receiver SHOULD send an ERROR chunk with a Protocol Violation cause as defined in section 3.3.10.13 of [RFC4960]).

4. New Parameter Types

This section defines the new parameter types that will be used in the RE-CONFIG chunk. Table 2 illustrates the new parameter types.

Parameter Type	Parameter Name
0x000d	Outgoing SSN Reset Request Parameter
0x000e	Incoming SSN Reset Request Parameter
0x000f	SSN/TSN Reset Request Parameter
0x0010	Re-configuration Response Parameter
0x0011	Add Outgoing Streams Request Parameter
0x0012	Add Incoming Streams Request Parameter

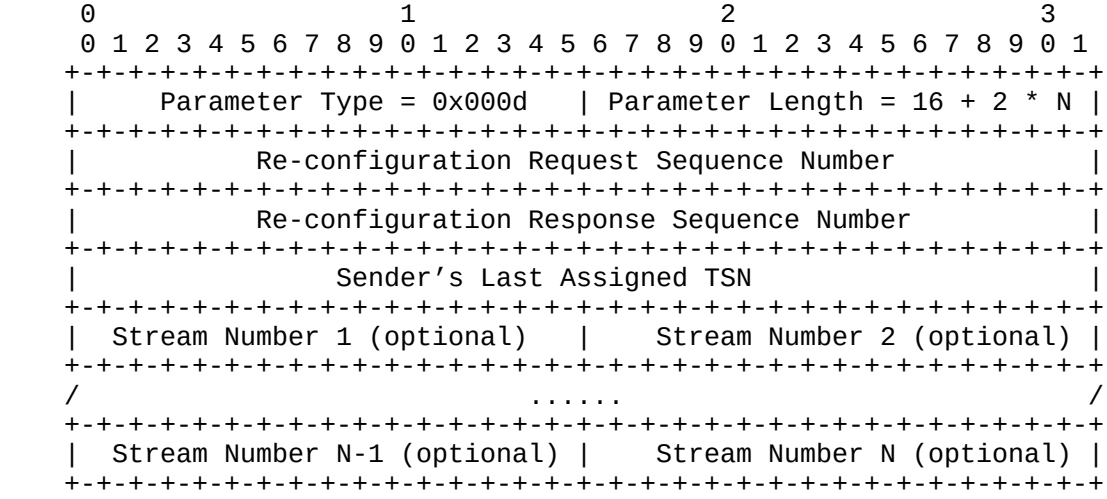
Table 2

It should be noted that the parameter format requires the receiver to stop processing the parameter and not to process any further parameters within the chunk if the parameter type is not recognized. This is accomplished by the use of the upper bits of the parameter type as described in section 3.2.1 of [RFC4960].

All transported integer numbers are in "network byte order" a.k.a., Big Endian.

4.1. Outgoing SSN Reset Request Parameter

This parameter is used by the sender to request the reset of some or all outgoing streams.



Parameter Type: 2 bytes (unsigned integer)
This field holds the IANA defined parameter type for the Outgoing SSN Reset Request Parameter. The suggested value of this field for IANA is 0x000d.

Parameter Length: 2 bytes (unsigned integer)
This field holds the length in bytes of the parameter; the value MUST be 16 + 2 * N, where N is the number of stream numbers listed.

Re-configuration Request Sequence Number: 4 bytes (unsigned integer)
This field is used to identify the request. It is a monotonically increasing number that is initialized to the same value as the Initial TSN number. It is increased by 1 whenever sending a new Re-configuration Request parameter.

Re-configuration Response Sequence Number: 4 bytes (unsigned integer)
When this Outgoing SSN Reset Request Parameter is sent in response to an Incoming SSN Reset Request Parameter this parameter is also an implicit response to the incoming request. Then this field holds the Re-configuration Request Sequence Number of the incoming request. In other cases it holds the next expected Re-configuration Request Sequence Number minus 1.

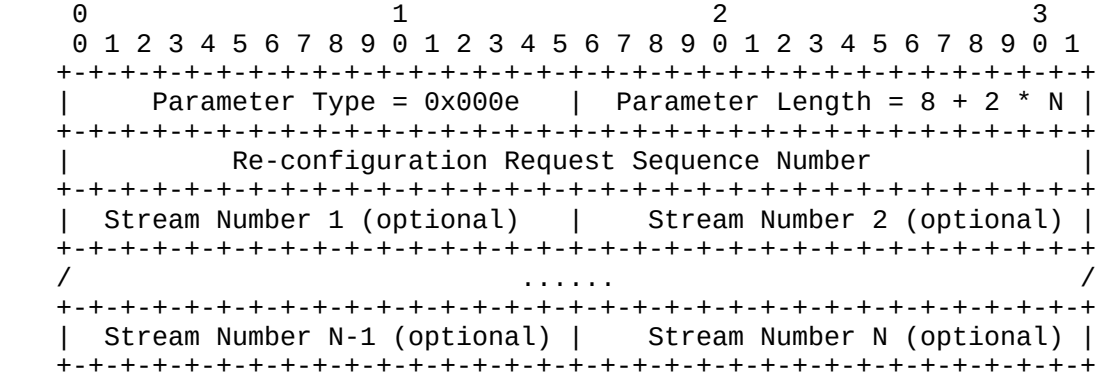
Sender's last assigned TSN: 4 bytes (unsigned integer)
This value holds the next TSN minus 1, in other words the last TSN that this sender assigned.

Stream Number 1..N: 2 bytes (unsigned integer)
This optional field, if included, is used to indicate specific streams that are to be reset. If no streams are listed, then all streams are to be reset.

This parameter can appear in a RE-CONFIG chunk. This parameter MUST NOT appear in any other chunk type.

4.2. Incoming SSN Reset Request Parameter

This parameter is used by the sender to request that the peer resets some or all of its outgoing streams.



Parameter Type: 2 bytes (unsigned integer)
This field holds the IANA defined parameter type for the Incoming SSN Reset Request Parameter. The suggested value of this field for IANA is 0x000e.

Parameter Length: 2 bytes (unsigned integer)
This field holds the length in bytes of the parameter; the value MUST be 8 + 2 * N.

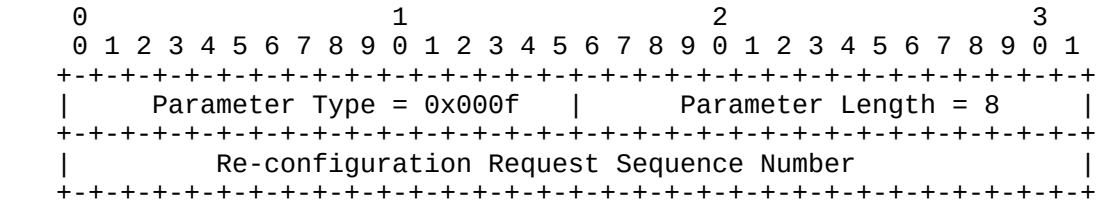
Re-configuration Request Sequence Number: 4 bytes (unsigned integer)
This field is used to identify the request. It is a monotonically increasing number that is initialized to the same value as the Initial TSN number. It is increased by 1 whenever sending a new Re-configuration Request parameter.

Stream Number 1..N: 2 bytes (unsigned integer)
This optional field, if included, is used to indicate specific streams that are to be reset. If no streams are listed, then all streams are to be reset.

This parameter can appear in a RE-CONFIG chunk. This parameter MUST NOT appear in any other chunk type.

4.3. SSN/TSN Reset Request Parameter

This parameter is used by the sender to request a reset of the TSN and SSN numbering of all incoming and outgoing streams.



Parameter Type: 2 bytes (unsigned integer)
This field holds the IANA defined parameter type for the SSN/TSN Reset Request Parameter. The suggested value of this field for IANA is 0x000f.

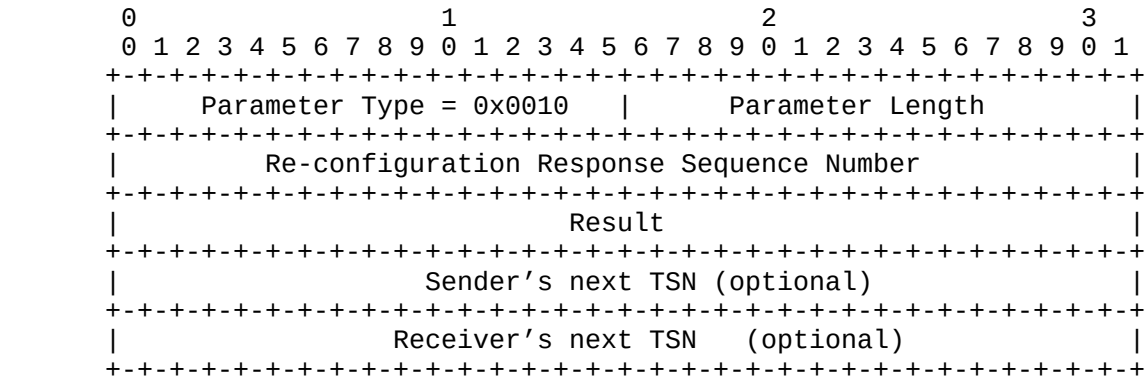
Parameter Length: 2 bytes (unsigned integer)
This field holds the length in bytes of the parameter; the value MUST be 8.

Re-configuration Request Sequence Number: 4 bytes (unsigned integer)
This field is used to identify the request. It is a monotonically increasing number that is initialized to the same value as the Initial TSN number. It is increased by 1 whenever sending a new Re-configuration Request parameter.

This parameter can appear in a RE-CONFIG chunk. This parameter MUST NOT appear in any other chunk type.

4.4. Re-configuration Response Parameter

This parameter is used by the receiver of a Re-configuration Request parameter to respond to the request.



Parameter Type: 2 bytes (unsigned integer)
This field holds the IANA defined parameter type for Re-configuration Response Parameter. The suggested value of this field for IANA is 0x0010.

Parameter Type Length: 2 bytes (unsigned integer)
This field holds the length in bytes of the parameter; the value MUST be 12 if the optional fields are not present and 20 otherwise.

Re-configuration Response Sequence Number: 4 bytes (unsigned integer)
This value is copied from the request parameter and is used by the receiver of the Re-configuration Response Parameter to tie the response to the request.

Result: 4 bytes (unsigned integer)
This value describes the result of the processing of the request. It is encoded as given by the following table

Result	Description
0	Success - Nothing to do
1	Success - Performed
2	Denied
3	Error - Wrong SSN
4	Error - Request already in progress
5	Error - Bad Sequence Number
6	In progress

Table 3

Sender's next TSN: 4 bytes (unsigned integer)
This field holds the TSN the sender of the response will use to send the next DATA chunk. The field is only applicable in responses to SSN/TSN reset requests.

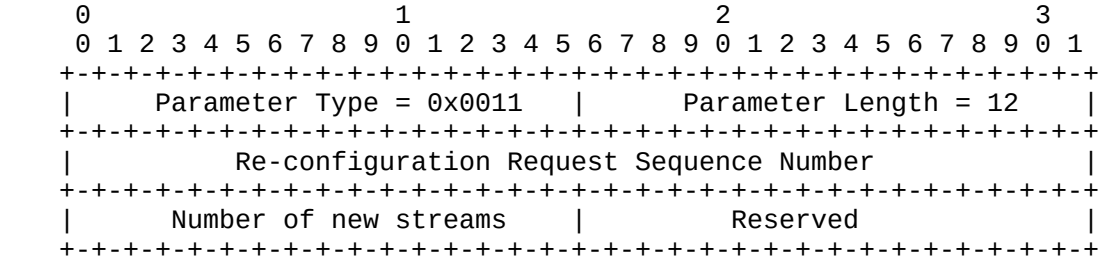
Receiver's next TSN: 4 bytes (unsigned integer)
This field holds the TSN the receiver of the response must use to send the next DATA chunk. The field is only applicable in responses to SSN/TSN reset requests.

Either both optional fields (Sender's next TSN and Receiver's next TSN) MUST be present or none.

This parameter can appear in a RE-CONFIG chunk. This parameter MUST NOT appear in any other chunk type.

4.5. Add Outgoing Streams Request Parameter

This parameter is used by the sender to request that an additional number of outgoing streams (i.e. the receiver's incoming streams) be added to the association.



Parameter Type: 2 bytes (unsigned integer)
This field holds the IANA defined parameter type for the the Add Outgoing Streams Request Parameter. The suggested value of this field for IANA is 0x0011.

Parameter Length: 2 bytes (unsigned integer)
This field holds the length in bytes of the parameter; the value MUST be 12.

Re-configuration Request Sequence Number: 4 bytes (unsigned integer)
This field is used to identify the request. It is a monotonically increasing number that is initialized to the same value as the Initial TSN number. It is increased by 1 whenever sending a new Re-configuration Request parameter.

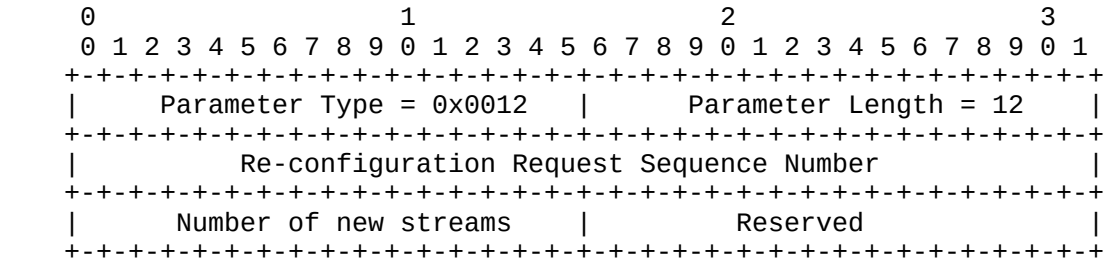
Number of new streams: 2 bytes (unsigned integer)
This value holds the number of additional outgoing streams the sender requests to be added to the association. Streams are added in order and are consecutive, e.g. if an association has four outgoing streams (0-3) and a requested is made to add 3 streams then the new streams will be 4, 5 and 6.

Reserved: 2 bytes (unsigned integer)
This field is reserved. It SHOULD be set to 0 by the sender and ignored by the receiver.

This parameter MAY appear in a RE-CONFIG chunk. This parameter MUST NOT appear in any other chunk type.

4.6. Add Incoming Streams Request Parameter

This parameter is used by the sender to request that the peer adds an additional number of outgoing streams (i.e. the sender's incoming streams) to the association.



Parameter Type: 2 bytes (unsigned integer)
This field holds the IANA defined parameter type for the the Add Incoming Streams Request Parameter. The suggested value of this field for IANA is 0x0012.

Parameter Length: 2 bytes (unsigned integer)
This field holds the length in bytes of the parameter; the value MUST be 12.

Re-configuration Request Sequence Number: 4 bytes (unsigned integer)
This field is used to identify the request. It is a monotonically increasing number that is initialized to the same value as the Initial TSN number. It is increased by 1 whenever sending a new Re-configuration Request parameter.

Number of new streams: 2 bytes (unsigned integer)
This value holds the number of additional incoming streams the sender requests to be added to the association. Streams are added in order and are consecutive, e.g. if an association has four outgoing streams (0-3) and a requested is made to add 3 streams then the new streams will be 4, 5 and 6.

Reserved: 2 bytes (unsigned integer)
This field is reserved. It SHOULD be set to 0 by the sender and ignored by the receiver.

This parameter MAY appear in a RE-CONFIG chunk. This parameter MUST NOT appear in any other chunk type.

5. Procedures

This section defines the procedures used by both the sender and receiver of a RE-CONFIG chunk. Various examples of re-configuration scenarios are given in Appendix A.

One important thing to remember about SCTP streams is that they are uni-directional. The endpoint for which a stream is an outgoing stream is called the outgoing side, the endpoint for which the stream is an incoming stream is called the incoming side. The procedures outlined in this section are designed so that the incoming side will always reset their stream sequence number first before the outgoing side which means the re-configuration request must always originate from the outgoing side. These two issues have important ramifications upon how an SCTP endpoint might request that its incoming streams be reset. In effect it must ask the peer to start an outgoing reset procedure and once that request is acknowledged let the peer actually control the reset operation.

5.1. Sender Side Procedures

This section describes the procedures related to the sending of RE-CONFIG chunks. A RE-CONFIG chunk is composed of one or two Type Length Value (TLV) parameters.

5.1.1. Sender Side Procedures for the RE-CONFIG Chunk

This SCTP extension uses the Supported Extensions Parameter defined in [RFC5061] for negotiating the support for it.

An SCTP endpoint supporting this extension MUST include the chunk type of the RE-CONFIG chunk in the Supported Extensions Parameter in either the INIT or INIT-ACK. Before sending a RE-CONFIG chunk the

sender MUST ensure that the peer advertised support for the re-configuration extension. If the chunk type of the RE-CONFIG chunk does not appear in the supported extensions list of chunks, then the sender MUST NOT send any re-configuration request to the peer, and any request by the application for such service SHOULD be responded to with an appropriate error indicating the peer SCTP stack does not support the re-configuration extension.

At any given time there MUST NOT be more than one request be in flight. So if the Re-configuration Timer is running and the the RE-CONFIG chunk contains at least one request parameter the chunk MUST be buffered.

After packaging the RE-CONFIG chunk and sending it to the peer the sender MUST start the Re-configuration Timer if the RE-CONFIG chunk contains at least one request parameter. If it contains no request parameters, the Re-configuration Timer MUST NOT be started. This timer MUST use the same value as SCTP's Data transmission timer (i.e. the RTO timer) and MUST use exponential backoff doubling the value at every expiration. If the timer expires, besides doubling the value, the sender MUST retransmit the RE-CONFIG chunk, increment the appropriate error counts (both for the association and the destination), and perform threshold management possibly destroying the association if SCTP retransmission thresholds are exceeded.

5.1.2. Sender Side Procedures for the Outgoing SSN Reset Request Parameter

When an SCTP sender wants to reset the SSNs of some or all outgoing streams it can send an Outgoing SSN Reset Request Parameter provided that the Re-configuration Timer is not running. The following steps must be followed:

A1: The sender MUST stop assigning new SSNs to new user data provided by the upper layer for the affected streams and queue it. This is because it is not known whether the receiver of the request will accept or deny it and moreover, a lost request might cause an out-of-sequence error in a stream that the receiver is not yet prepared to handle.

A2: The sender MUST assign the next re-configuration request sequence number and MUST put it into the Re-configuration Request Sequence Number field of the Outgoing SSN Reset Request Parameter. The next re-configuration request sequence number MUST then be incremented by 1.

A3: The Sender's Last Assigned TSN MUST be set to the next TSN the sender assigns minus 1.

A4: If this Outgoing SSN Reset Request Parameter is sent in response to an Incoming SSN Reset Request Parameter the Stream Numbers MUST be copied from the Incoming SSN Reset Request Parameter to the Outgoing SSN Reset Request Parameter. The Re-configuration Response Sequence Number of the Outgoing SSN Reset Request Parameter MUST be the Re-configuration Request Sequence Number of the Incoming SSN Reset Request Parameter. If this Outgoing SSN Reset Request Parameter is sent at the request of the upper layer and the sender requests all outgoing streams to be reset Stream Numbers SHOULD NOT be put into the Outgoing SSN Reset Request Parameter. If the sender requests only some outgoing streams to be reset these Stream Numbers MUST be placed in the Outgoing SSN Reset Request Parameter. Re-configuration Response Sequence Number is the next expected Re-configuration Request Sequence Number of the peer minus 1.

A5: The Outgoing SSN Reset Request Parameter MUST be put into a RE-CONFIG Chunk. The Outgoing SSN Reset Request Parameter MAY be put together with either an Incoming SSN Reset Request Parameter or an Re-configuration Response Parameter but not both. It MUST NOT be put together with any other parameter as described in Section 3.1.

A6: The RE-CONFIG chunk MUST be sent following the rules given in Section 5.1.1.

5.1.3. Sender Side Procedures for the Incoming SSN Reset Request Parameter

When an SCTP sender wants to reset the SSNs of some or all incoming streams it can send an Incoming SSN Reset Request Parameter provided that the Re-configuration Timer is not running. The following steps must be followed:

B1: The sender MUST assign the next re-configuration request sequence number and MUST put it into the Re-configuration Request Sequence Number field of the Incoming SSN Reset Request Parameter. After assigning it the next re-configuration request sequence number MUST be incremented by 1.

B2: If the sender wants all incoming streams to be reset Stream Numbers SHOULD NOT be put into the Incoming SSN Reset Request Parameter. If the sender wants only some incoming streams to be reset these Stream Numbers MUST be filled in the Incoming SSN Reset Request Parameter.

B3: The Incoming SSN Reset Request Parameter MUST be put into a RE-CONFIG Chunk. It MAY be put together with an Outgoing SSN Reset Request Parameter but MUST NOT be put together with any other parameter.

B4: The RE-CONFIG chunk MUST be sent following the rules given in Section 5.1.1.

When sending an Incoming SSN Reset Request there is a potential that the peer has just reset or is in the process of resetting the same streams via an Outgoing SSN Reset Request. This collision scenario is discussed in Section 5.2.3.

5.1.4. Sender Side Procedures for the SSN/TSN Reset Request Parameter

When an SCTP sender wants to reset the SSNs and TSNs it can send an SSN/TSN Reset Request Parameter provided that the Re-configuration Timer is not running. The following steps must be followed:

C1: The sender MUST assign the next re-configuration request sequence number and put it into the Re-configuration Request Sequence Number field of the SSN/TSN Reset Request Parameter. After assigning it the next re-configuration request sequence number MUST be incremented by 1.

C2: The sender has either no outstanding TSNs or considers all outstanding TSNs abandoned. The sender MUST queue any user data suspending any new transmissions and TSN assignment until the reset procedure is finished by the peer either acknowledging or denying the request.

C3: The SSN/TSN Reset Request Parameter MUST be put into a RE-CONFIG chunk. There MUST NOT be any other parameter in this chunk.

C4: The RE-CONFIG chunk MUST be sent following the rules given in Section 5.1.1.

Only one SSN/TSN Reset Request SHOULD be sent within 30 seconds, which is considered a maximum segment lifetime, the IP MSL.

5.1.5. Sender Side Procedures for the Re-configuration Response Parameter

When an implementation receives a reset request parameter it must respond with a Re-configuration Response Parameter in the following manner:

D1: The Re-configuration Request Sequence number of the incoming request MUST be copied to the Re-configuration Response Sequence Number field of the Re-configuration Response Parameter.

D2: The result of the processing of the incoming request according to Table 3 MUST be placed in the Result field of the Re-configuration Response Parameter.

D3: If the incoming request is an SSN/TSN reset request, the Sender's next TSN field MUST be filled with the next TSN the sender of this Re-configuration Response Parameter will assign. For other requests the Sender's next TSN field, which is optional, MUST NOT be used.

D4: If the incoming request is an SSN/TSN reset request, the Receiver's next TSN field MUST be filled with a TSN such that the sender of the Re-configuration Response Parameter can be sure it can discard received DATA chunks with smaller TSNs. The value SHOULD be the smallest TSN not acknowledged by the receiver of the request plus 2^31. For other requests the Receiver's next TSN field, which is optional, MUST NOT be used.

5.1.6. Sender Side Procedures for the Add Outgoing Streams Request Parameter

When an SCTP sender wants to increase the number of outbound streams to which it is able to send, it may add an Add Outgoing Streams Request parameter to the RE-CONFIG chunk. Upon sending the request the sender MUST await a positive acknowledgment (Success) before using any additional stream added by this request. Note that new streams are added adjacent to the previous streams with no gaps. This means that if a request is made to add 2 streams to an association that has already 5 (0-4) then the new streams, upon successful completion, are streams 5 and 6. A new stream MUST use the stream sequence number 0 for its first ordered message.

5.1.7. Sender Side Procedures for the Add Incoming Streams Request Parameter

When an SCTP sender wants to increase the number of inbound streams to which the peer is able to send, it may add an Add Incoming Streams Request parameter to the RE-CONFIG chunk. Note that new streams are added adjacent to the previous streams with no gaps. This means that if a request is made to add 2 streams to an association that has already 5 (0-4) then the new streams, upon successful completion, are streams 5 and 6. A new stream MUST use the stream sequence number 0 for its first ordered message.

5.2. Receiver Side Procedures

5.2.1. Receiver Side Procedures for the RE-CONFIG Chunk

Upon reception of a RE-CONFIG chunk each parameter within it SHOULD be processed. If multiple parameters have to be returned, they MUST be put into one RE_CONFIG chunk. If the received RE-CONFIG chunk contains at least one request parameter, a SACK chunk SHOULD be sent back and MAY be bundled with the RE-CONFIG chunk. If the received RE-CONFIG chunk contains at least one request and based on the analysis of the Re-configuration Request Sequence Numbers this is the last received RE-CONFIG chunk (i.e. a retransmission), the same RE-CONFIG chunk MUST to be sent back in response as was earlier.

The decision to deny a re-configuration request is an administrative decision and may be user configurable even after the association has formed. If for whatever reason the endpoint does not wish to process a received request parameter it MUST send a corresponding response parameter as described in Section 5.1.5 with an appropriate Result field.

Implementation Note: A SACK is recommended to be bundled with any re-configuration response so that any retransmission processing that needs to occur can be expedited. A SACK chunk is not required for this feature to work, but it will in effect help minimize the delay in completing a re-configuration operation in the face of any data loss.

5.2.2. Receiver Side Procedures for the Outgoing SSN Reset Request Parameter

In the case that the endpoint is willing to perform a stream reset the following steps must be followed:

E1: If the Re-configuration Timer is running for the Re-configuration Request Sequence Number indicated in the Re-configuration Response Sequence Number field, the Re-configuration Request Sequence Number MUST be marked as acknowledged. If all Re-configuration Request Sequence Numbers the Re-configuration Timer is running for are acknowledged, the Re-configuration Timer MUST be stopped.

E2: If the Sender’s Last Assigned TSN number is greater than the cumulative acknowledgment point, then the endpoint MUST enter "deferred reset processing". In this mode, any data arriving with a TSN number larger than the ’senders last assigned TSN’ for the affected stream(s) MUST be queued locally and held until the Cumulative Acknowledgment point reaches the ’senders last

assigned TSN number'. When the Cumulative Acknowledgment point reaches the last assigned TSN number then proceed to the next step. If the endpoint enters "deferred reset processing", it MUST put a Re-configuration Response Parameter into a RE-CONFIG chunk indicating 'In progress' and MUST send the RE-CONFIG chunk.

E3: If no Stream Numbers are listed in the parameter, then all incoming streams MUST be reset to 0 as the next expected stream sequence number. If specific Stream Numbers are listed, then only these specific streams MUST be reset to 0 and all other non-listed stream sequence numbers remain unchanged.

E4: Any queued TSN's (queued at step E2) MUST now be released and processed normally.

E5: A Re-configuration Response Parameter MUST be put into a RE-CONFIG chunk indicating successful processing.

E6: The RE-CONFIG chunk MUST be sent after the incoming RE-CONFIG chunk is processed completely.

5.2.3. Receiver Side Procedures for the Incoming SSN Reset Request Parameter

In the case that the endpoint is willing to perform a stream reset the following steps must be followed:

F1: An Outgoing SSN Reset Request Parameter MUST be put into an RE-CONFIG chunk according to Section 5.1.2.

F2: The RE-CONFIG chunk MUST be sent after the incoming RE-CONFIG chunk is processed completely.

When a peer endpoint requests an Incoming SSN Reset Request it is possible that the local endpoint has just sent an Outgoing SSN Reset Request on the same association and has not yet received a response. In such a case the local endpoint MUST do the following:

- If the just sent Outgoing SSN Reset Request Parameter completely overlaps the received Incoming SSN Reset Request Parameter respond to the peer with an acknowledgment indicating that there was 'Nothing to do'.

- Otherwise process the Incoming SSN Reset Request Parameter normally responding to the peer with an acknowledgment. Note that this case includes the situation where some of the streams requested overlap with the just sent Outgoing SSN Reset Request.

Even in such a situation the Incoming SSN Reset MUST be processed normally even though this means that (if the endpoint elects to do the stream reset) streams that are already at SSN 0, will be reset a subsequent time.

It is also possible that the Incoming request will arrive after the Outgoing SSN Reset Request just completed. In such a case all of the streams being requested will be already set to 0. If so, the local endpoint SHOULD send back a Re-configuration Response with the success code "Nothing to do".

Note that in either race condition the local endpoint could optionally also perform the reset. This would result in streams that are already at sequence 0 being reset again to 0 which would cause no harm to the application but will add an extra message to the network.

5.2.4. Receiver Side Procedures for the SSN/TSN Reset Request Parameter

In the case that the endpoint is willing to perform an SSN/TSN reset the following steps must be followed:

G1: Compute an appropriate value for the Receiver's next TSN, the TSN the peer should use to send the next DATA chunk. The value SHOULD be the smallest TSN not acknowledged by the receiver of the request plus 2^31.

G2: Compute an appropriate value for the local endpoint's next TSN, i.e. the receiver of the SSN/TSN reset chunk next TSN to be assigned. The value SHOULD be the highest TSN sent by the receiver of the request plus 1.

G3: The same processing as if a SACK chunk with no gap report and a cumulative TSN ACK of Sender's next TSN minus 1 was received MUST be performed.

G4: The same processing as if a FWD-TSN chunk as defined in [RFC3758] with all streams affected and a new cumulative TSN ACK of Receiver's next TSN minus 1 was received MUST be performed.

G5: The next expected and outgoing stream sequence numbers MUST be reset to 0 for all incoming and outgoing streams.

G6: A Re-configuration Response Parameter MUST be put into a RE-CONFIG chunk indicating successful processing.

G7: The RE-CONFIG chunk MUST be sent after the incoming RE-CONFIG chunk is processed completely.

5.2.5. Receiver Side Procedures for the Add Outgoing Streams Request Parameter

When an SCTP endpoint receives a re-configuration request adding additional streams, it MUST send a response parameter either acknowledging or denying the request. If the response is successful the receiver MUST add the requested number of inbound streams to the association, initializing the next expected stream sequence number to be 0. The SCTP endpoint SHOULD deny the request if the number of streams exceeds a limit which should be configurable by the application.

5.2.6. Receiver Side Procedures for the Add Incoming Streams Request Parameter

When an SCTP endpoint receives a re-configuration request adding additional incoming streams, it MUST either send a response parameter denying the request or sending a corresponding Add Outgoing Streams Request Parameter following the rules given in Section 5.1.6. The SCTP endpoint SHOULD deny the request if the number of streams exceeds a limit which should be configurable by the application.

5.2.7. Receiver Side Procedures for the Re-configuration Response Parameter

On receipt of a Re-configuration Response Parameter the following must be performed:

H1: If the Re-confguration Timer is running for the Re-configuration Request Sequence Number indicated in the Re-configuration Response Sequence Number field, the Re-configuration Request Sequence Number MUST be marked as acknowledged. If all Re-configuration Request Sequence Numbers the Re-configuration Timer is running for are acknowledged, the Re-configuration Timer MUST be stopped. If the timer was not running for the Re-configuration Request Sequence Number, the processing of the Re-configuration Response Parameter is complete.

H2: If the Result field indicates 'In progress', the timer for the Re-configuration Request Sequence Number is started again. If the timer runs off, the RE-CONFIG chunk MUST be retransmitted but the corresponding error counters MUST NOT be incremented.

H3: If the Result field does not indicate successful processing the processing of this response is complete.

H4: If the request was an Outgoing SSN Reset Request the affected streams MUST now be reset and all queued data should be processed now and assigning of stream sequence numbers is allowed again.

H5: If the request was an SSN/TSN Reset Request new data MUST be sent from Receiver's next TSN and beginning with stream sequence number 0 for all outgoing streams. All incoming streams MUST be reset to 0 as the next expected stream sequence number. The peer will send DATA chunks starting with Sender's next TSN.

H6: If the request was to add outgoing streams, the endpoint MUST add the additional streams to the association. Note that an implementation may allocate the memory at the time of the request, but it MUST NOT use the streams until the peer has responded with a positive acknowledgment.

6. Socket API Considerations

This section describes how the socket API defined in [I-D.ietf-tsvwg-sctpsocket] needs to be extended to make the features of SCTP re-configuration available to the application.

Please note that this section is informational only.

6.1. Events

When the SCTP_ASSOC_CHANGE notification is delivered and both peers support the extension described in this document, SCTP_ASSOC_SUPPORTS_RE_CONFIG should be listed in the sac_info field.

The union sctp_notification {} is extended to contain three new fields: sn_strreset_event, sn_assocreset_event, and sn_strchange_event:

```
union sctp_notification {
  struct {
    uint16_t sn_type;
    uint16_t sn_flags;
    uint32_t sn_length;
  } sn_header;
  ...
  struct sctp_stream_reset_event sn_strreset_event;
  struct sctp_assoc_reset_event sn_assocreset_event;
  struct sctp_stream_change_event sn_strchange_event;
  ...
}
```

The corresponding sn_type values are given in Table 4.

sn_type	valid field in union sctp_notification
SCTP_STREAM_RESET_EVENT	sn_strreset_event
SCTP_ASSOC_RESET_EVENT	sn_assocreset_event
SCTP_STREAM_CHANGE_EVENT	sn_strchange_event

Table 4

These events are delivered when an incoming request was processed successfully or the processing of an outgoing request has been finished.

6.1.1. Stream Reset Event

The event delivered has the following structure:

```
struct sctp_stream_reset_event {
  uint16_t strreset_type;
  uint16_t strreset_flags;
  uint32_t strreset_length;
  sctp_assoc_t strreset_assoc_id;
  uint16_t strreset_stream_list[];
};
```

strreset_type: It should be SCTP_STREAM_RESET_EVENT.

strreset_flags: This field is formed from the bitwise OR of one or more of the following currently defined flags:

SCTP_STREAM_RESET_INCOMING_SSN: The stream identifiers given in strreset_stream_list[] refer to incoming streams of the endpoint.

SCTP_STREAM_RESET_OUTGOING_SSN: The stream identifiers given in strreset_stream_list[] refer to outgoing streams of the endpoint.

SCTP_STREAM_RESET_DENIED: The corresponding request was denied by the peer.

SCTP_STREAM_RESET_FAILED: The corresponding request failed.

At least one of SCTP_STREAM_RESET_INCOMING_SSN and SCTP_STREAM_RESET_OUTGOING_SSN is set. SCTP_STREAM_RESET_DENIED and SCTP_STREAM_RESET_FAILED are mutually exclusive. If the request was successful, none of these are set.

strreset_length: This field is the total length in bytes of the delivered event, including the header.

strreset_assoc_id: The association id field, holds the identifier for the association. All notifications for a given association have the same association identifier. For one-to-one style sockets, this field is ignored.

strreset_stream_list: The list of stream identifiers this event refers to. An empty list identifies all streams as being reset. Depending on strreset_flags the identifiers refer to incoming or outgoing streams or both.

6.1.2. Association Reset Event

The event delivered has the following structure:

```
struct sctp_assoc_reset_event {
  uint16_t assocreset_type;
  uint16_t assocreset_flags;
  uint32_t assocreset_length;
  sctp_assoc_t assocreset_assoc_id;
  uint32_t assocreset_local_tsn;
  uint32_t assocreset_remote_tsn;
};
```

assocreset_type: It should be SCTP_ASSOC_RESET_EVENT.

assocreset_flags: This field is formed from the bitwise OR of one or more of the following currently defined flags:

SCTP_ASSOC_RESET_DENIED: The corresponding outgoing request was denied by the peer.

SCTP_ASSOC_RESET_FAILED: The corresponding outgoing request failed.

SCTP_ASSOC_RESET_DENIED and SCTP_ASSOC_RESET_FAILED are mutual exclusive. If the request was successful, none of these are set.

assocreset_length: This field is the total length in bytes of the delivered event, including the header.

assocreset_assoc_id: The association id field, holds the identifier for the association. All notifications for a given association have the same association identifier. For one-to-one style sockets, this field is ignored.

assocreset_local_tsn: The next TSN used by the endpoint.

assocreset_remote_tsn: The next TSN used by the peer.

6.1.3. Stream Change Event

The event delivered has the following structure:

```
struct sctp_stream_change_event {
  uint16_t strchange_type;
  uint16_t strchange_flags;
  uint32_t strchange_length;
  sctp_assoc_t strchange_assoc_id;
  uint16_t strchange_instrms;
  uint16_t strchange_outstrms;
};
```

strchange_type: It should be SCTP_STREAM_CHANGE_EVENT.

strchange_flags: This field is formed from the bitwise OR of one or more of the following currently defined flags:

SCTP_STREAM_CHANGE_DENIED: The corresponding request was denied by the peer.

SCTP_STREAM_CHANGE_FAILED: The corresponding request failed.

SCTP_STREAM_CHANGE_DENIED and SCTP_STREAM_CHANGE_FAILED are mutual exclusive. If the request was successful, none of these are set.

strchange_length: This field is the total length in bytes of the delivered event, including the header.

strchange_assoc_id: The association id field, holds the identifier for the association. All notifications for a given association have the same association identifier. For one-to-one style sockets, this field is ignored.

strchange_instrms: The number of streams that the peer is allowed to use outbound.

strchange_outstrms: The number of streams that the endpoint is allowed to use outbound.

6.2. Event Subscription

Subscribing to events as described in [I-D.ietf-tsvwg-sctpsocket] uses a setsockopt() call with the SCTP_EVENT socket option. This option takes the following structure that specifies the association, the event type (using the same value found in the event type field) and an on/off boolean.

```
struct sctp_event {
  sctp_assoc_t se_assoc_id;
  uint16_t     se_type;
  uint8_t      se_on;
};
```

The user fills in the se_type with the same value found in the strreset_type field i.e. SCTP_STREAM_RESET_EVENT. The user will also fill in the se_assoc_id field with either the association to set this event on (this field is ignored for one-to-one style sockets) or one of the reserved constant values defined in [I-D.ietf-tsvwg-sctpsocket]. Finally the se_on field is set with a 1 to enable the event or a 0 to disable the event.

6.3. Socket Options

The following table describes the new socket options which make the re-configuration features accessible to the user. They all use IPPROTO_SCTP as their level.

If a call to setsockopt() is used to issue a Re-configuration request

while the Re-configuration timer is running, setsockopt() will return -1 and error is set to EALREADY.

option name	data type	get	set
SCTP_ENABLE_STREAM_RESET	struct sctp_assoc_value	X	X
SCTP_RESET_STREAMS	struct sctp_reset_streams		X
SCTP_RESET_ASSOC	sctp_assoc_t		X
SCTP_ADD_STREAMS	struct sctp_add_streams		X

Table 5

6.3.1. Enable/Disable Stream Reset (SCTP_ENABLE_STREAM_RESET)

This option allows a user to control whether the SCTP implementation processes or denies incoming requests in STREAM_RESET chunks.

The default is to deny all incoming requests.

To set or get this option the user fills in the following structure:

```
struct sctp_assoc_value {
  sctp_assoc_t assoc_id;
  uint32_t assoc_value;
};
```

assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets this parameter indicates which association the user is performing an action upon.

assoc_value: It is formed from the bitwise OR of one or more of the following currently defined flags:

SCTP_ENABLE_RESET_STREAM_REQ: Process received Incoming/Outgoing SSN Reset Requests if this flag is set, deny them if not.

SCTP_ENABLE_RESET_ASSOC_REQ: Process received SSN/TSN Reset Requests if this flag is set, deny them if not.

SCTP_ENABLE_CHANGE_ASSOC_REQ: Process received Add Outgoing Streams Requests if this flag is set, deny them if not.

The default value is !(SCTP_ENABLE_RESET_STREAM_REQ| SCTP_ENABLE_RESET_ASSOC_REQ|SCTP_ENABLE_CHANGE_ASSOC_REQ).

Please note that using the option does not have any impact on

subscribing to any related events.

6.3.2. Reset Incoming and/or Outgoing Streams (SCTP_RESET_STREAMS)

This option allows the user to request the reset of incoming and/or outgoing streams.

To set or get this option the user fills in the following structure:

```
struct sctp_reset_streams {
  sctp_assoc_t srs_assoc_id;
  uint16_t srs_flags;
  uint16_t srs_number_streams;
  uint16_t srs_stream_list[];
};
```

srs_assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets this parameter indicates which association the user is performing an action upon.

srs_flags: This parameter describes which class of streams is reset. It is formed from the bitwise OR of one or more of the following currently defined flags:

* SCTP_STREAM_RESET_INCOMING

* SCTP_STREAM_RESET_OUTGOING

srs_number_streams: This parameter is the number of elements in the srs_stream_list. If it is zero, the operation is performed on all streams.

srs_stream_list: This parameter contains a list of stream identifiers the operation is performed upon. It contains srs_number_streams elements. If it is empty, the operation is performed on all streams. Depending on srs_flags the identifiers refer to incoming or outgoing streams or both.

6.3.3. Reset SSN/TSN (SCTP_RESET_ASSOC)

This option allows a user to request the reset of the SSN/TSN.

To set this option the user provides an option_value of type sctp_assoc_t.

On one-to-one style sockets the option_value is ignored. For one-to-many style sockets the option_value is the association identifier of the association the action is to be performed upon.

6.3.4. Add Incoming and/or Outgoing Streams (SCTP_ADD_STREAMS)

This option allows a user to request the addition of a number of incoming and/or outgoing streams.

To set this option the user fills in the following structure:

```
struct sctp_add_streams {
  sctp_assoc_t sas_assoc_id;
  uint16_t sas_instrms;
  uint16_t sas_outstrms;
};
```

sas_assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets this parameter indicates which association the user is performing an action upon.

sas_instrms: This parameter is the number of incoming streams to add.

sas_outstrms: This parameter is the number of outgoing streams to add.

An endpoint can limit the number of incoming and outgoing streams by using the sinit_max_instreams field in the struct sctp_initmsg{} when issuing an SCTP_INIT socket option, as defined in [I-D.ietf-tsvwg-sctpsocket]. An incoming request asking for more streams than allowed will be denied.

7. Security Considerations

The SCTP socket API as described in [I-D.ietf-tsvwg-sctpsocket] exposes the sequence numbers of received DATA chunks to the application. An application might expect them to be monotonically increasing. When using the re-configuration extension this might no longer be true. Therefore the applications must enable this extension explicitly before it is used. In addition, applications must subscribe explicitly to notifications related to the re-configuration extension before receiving them.

SCTP associations are protected against blind attackers by using the verification tags. This is still valid when using the re-configuration extension. Therefore this extension does not add any additional security risk to SCTP in relation to blind attackers.

When the both sequence numbers are reset, the maximum segment lifetime is used to avoid the wrap-around for the TSN.

8. 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.

8.1. A New Chunk Type

A chunk type has to be assigned by IANA. It is suggested to use the values given in Table 1. IANA should assign this value from the pool of chunks with the upper two bits set to '10'.

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

Chunk Types

ID Value	Chunk Type	Reference
130	Re-configuration Chunk (RE-CONFIG)	[RFCXXXX]

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

8.2. Six New Chunk Parameter Types

Six chunk parameter types have to be assigned by IANA. It is suggested to use the values given in Table 2. IANA should assign these values from the pool of parameters with the upper two bits set to '00'.

This requires six additional lines in the "Chunk Parameter Types" registry for SCTP:

Chunk Parameter Types

ID Value	Chunk Parameter Type	Reference
13	Outgoing SSN Reset Request Parameter	[RFCXXXX]
14	Incoming SSN Reset Request Parameter	[RFCXXXX]
15	SSN/TSN Reset Request Parameter	[RFCXXXX]
16	Re-configuration Response Parameter	[RFCXXXX]
17	Add Outgoing Streams Request Parameter	[RFCXXXX]
18	Add Incoming Streams Request Parameter	[RFCXXXX]

9. Acknowledgments

The authors wish to thank Paul Aitken, Gorry Fairhurst, Tom Petch, Kacheong Poon, Irene Ruengeler, Robin Seggelmann, Gavin Shearer, and Vlad Yasevich for there invaluable comments.

10. References

10.1. Normative References

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

[RFC3758] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", RFC 3758, May 2004.

[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.

[RFC6096] Tuexen, M. and R. Stewart, "Stream Control Transmission Protocol (SCTP) Chunk Flags Registration", RFC 6096, January 2011.

10.2. Informative References

[I-D.ietf-tsvwg-sctpsocket] Stewart, R., Tuexen, M., Poon, K., Lei, P., and V. Yasevich, "Sockets API Extensions for Stream Control Transmission Protocol (SCTP)",

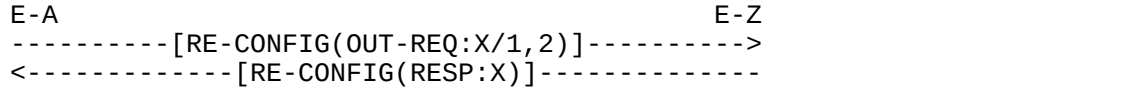
draft-ietf-tsvwg-sctpsocket-32 (work in progress), October 2011.

Appendix A. Examples of the Re-configuration procedures

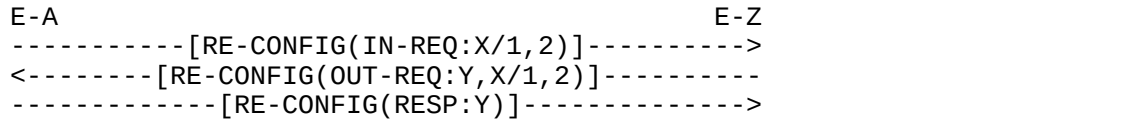
Please note that this appendix is informational only.

The following message flows between an Endpoint A and an Endpoint Z illustrate the described procedures. The time progresses in downward direction.

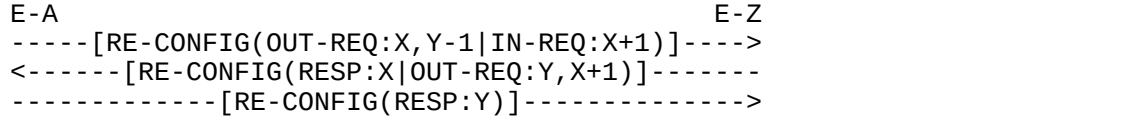
The following example illustrates an Endpoint A resetting stream 1 and 2 for just its outgoing streams.



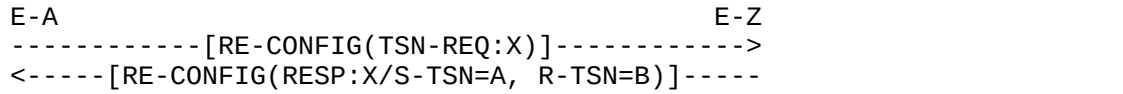
The following example illustrates an Endpoint A resetting stream 1 and 2 for just its incoming streams.



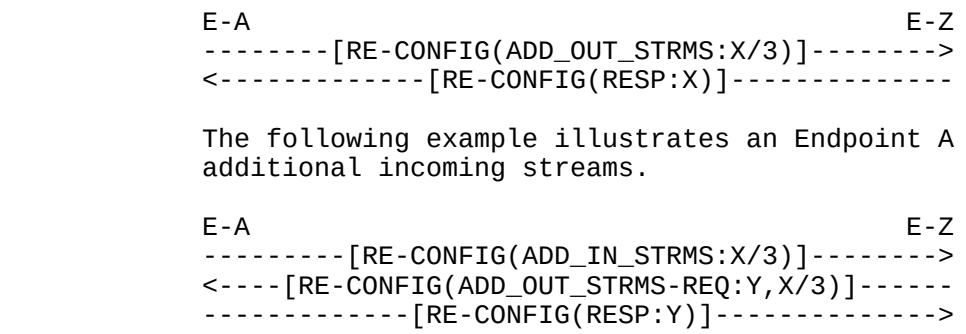
The following example illustrates an Endpoint A resetting all streams in both directions.



The following example illustrates an Endpoint A requesting the streams and TSNs be reset. At the completion E-A has the new sending TSN (selected by the peer) of B and E-Z has the new sending TSN of A (also selected by the peer).



The following example illustrates an Endpoint A requesting to add 3 additional outgoing streams.



The following example illustrates an Endpoint A requesting to add 3 additional incoming streams.

Authors' Addresses

Randall R. Stewart
Adara Networks
Chapin, SC 29036
USA

Email: randall@lakerest.net

Michael Tuexen
Muenster University of Applied Sciences
Stegerwaldstr. 39
48565 Steinfurt
DE

Email: tuexen@fh-muenster.de

Peter Lei
Cisco Systems, Inc.
8735 West Higgins Road
Suite 300
Chicago, IL 60631
USA

Email: peterlei@cisco.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: April 13, 2012

R. Stewart
Adara Networks
M. Tuexen
Muenster Univ. of Appl. Sciences
K. Poon
Oracle Corporation
P. Lei
Cisco Systems, Inc.
V. Yasevich
HP

October 11, 2011

Sockets API Extensions for Stream Control Transmission Protocol (SCTP)
draft-ietf-tsvwg-sctpsocket-32.txt

Abstract

This document describes a mapping of the Stream Control Transmission Protocol (SCTP) into a sockets API. The benefits of this mapping include compatibility for TCP applications, access to new SCTP features and a consolidated error and event notification scheme.

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 April 13, 2012.

Copyright Notice

Table of Contents

1. Introduction

The sockets API has provided a standard mapping of the Internet Protocol suite to many operating systems. Both TCP [RFC0793] and UDP [RFC0768] have benefited from this standard representation and access method across many diverse platforms. SCTP is a new protocol that provides many of the characteristics of TCP but also incorporates semantics more akin to UDP. This document defines a method to map the existing sockets API for use with SCTP, providing both a base for access to new features and compatibility so that most existing TCP applications can be migrated to SCTP with few (if any) changes.

There are three basic design objectives:

1. Maintain consistency with existing sockets APIs: We define a sockets mapping for SCTP that is consistent with other sockets API protocol mappings (for instance UDP, TCP, IPv4, and IPv6).

2. Support a one-to-many style interface: This set of semantics is similar to that defined for connection-less protocols, such as UDP. A one-to-many style SCTP socket should be able to control multiple SCTP associations. This is similar to a UDP socket, which can communicate with many peer endpoints. Each of these associations is assigned an association identifier so that an application can use the ID to differentiate them. Note that SCTP is connection-oriented in nature, and it does not support broadcast or multicast communications, as UDP does.

3. Support a one-to-one style interface: This interface supports a similar semantics as sockets for connection-oriented protocols, such as TCP. A one-to-one style SCTP socket should only control one SCTP association. One purpose of defining this interface is to allow existing applications built on other connection-oriented protocols to be ported to use SCTP with very little effort. Developers familiar with these semantics can easily adapt to SCTP. Another purpose is to make sure that existing mechanisms in most operating systems that support sockets, such as select(), should continue to work with this style of socket. Extensions are added to this mapping to provide mechanisms to exploit new features of SCTP.

Goals 2 and 3 are not compatible, so this document defines two modes of mapping, namely the one-to-many style mapping and the one-to-one style mapping. These two modes share some common data structures and operations, but will require the use of two different application programming styles. Note that all new SCTP features can be used with both styles of socket. The decision on which one to use depends mainly on the nature of applications.

A mechanism is defined to extract a one-to-many style SCTP association into a one-to-one style socket.

Some of the SCTP mechanisms cannot be adequately mapped to an existing socket interface. In some cases, it is more desirable to have a new interface instead of using existing socket calls. Section 9 of this document describes these new interfaces.

Please note that some elements of the SCTP socket API are declared as deprecated. During the evolution of this document, elements of the API were introduced, implemented and later on replaced by other elements. These replaced elements are declared as deprecated since they are still available in some implementations and the replacement functions are not. This applies especially to older versions of operating systems supporting SCTP. New SCTP socket implementations must implement at least the non deprecated elements. Implementations intending interoperability with older versions of the API should also include the deprecated functions.

2. Data Types

Whenever possible, POSIX data types defined in [IEEE-1003.1-2008] are used: uintN_t means an unsigned integer of exactly N bits (e.g. uint16_t). This document also assumes the argument data types from POSIX when possible (e.g. the final argument to setsockopt() is a socklen_t value). Whenever buffer sizes are specified, the POSIX size_t data type is used.

3. One-to-Many Style Interface

In the one-to-many style interface there is a 1 to many relationship between sockets and associations.

3.1. Basic Operation

A typical server in this style uses the following socket calls in sequence to prepare an endpoint for servicing requests:

- socket()
- bind()
- listen()
- recvmsg()

- sendmsg()
- close()

A typical client uses the following calls in sequence to setup an association with a server to request services:

- socket()
- sendmsg()
- recvmsg()
- close()

In this style, by default, all the associations connected to the endpoint are represented with a single socket. Each association is assigned an association identifier (type is sctp_assoc_t) so that an application can use it to differentiate among them. In some implementations, the peer endpoints' addresses can also be used for this purpose. But this is not required for performance reasons. If an implementation does not support using addresses to differentiate between different associations, the sendto() call can only be used to setup an association implicitly. It cannot be used to send data to an established association as the association identifier cannot be specified.

Once an association identifier is assigned to an SCTP association, that identifier will not be reused until the application explicitly terminates the use of the association. The resources belonging to that association will not be freed until that happens. This is similar to the close() operation on a normal socket. The only exception is when the SCTP_AUTOCLOSE option (Section 8.1.8) is set. In this case, after the association is terminated gracefully and automatically, the association identifier assigned to it can be reused. All applications using this option should be aware of this to avoid the possible problem of sending data to an incorrect peer endpoint.

If the server or client wishes to branch an existing association off to a separate socket, it is required to call sctp_peeloff() and to specify the association identifier. The sctp_peeloff() call will return a new one-to-one style socket which can then be used with recv() and send() functions for message passing. See Section 9.2 for more on branched-off associations.

Once an association is branched off to a separate socket, it becomes completely separated from the original socket. All subsequent

control and data operations to that association must be done through the new socket. For example, the close operation on the original socket will not terminate any associations that have been branched off to a different socket.

One-to-many style socket calls are discussed in more detail in the following subsections.

3.1.1. socket()

Applications use socket() to create a socket descriptor to represent an SCTP endpoint.

The function prototype is

```
int socket(int domain,
           int type,
           int protocol);
```

and one uses PF_INET or PF_INET6 as the domain, SOCK_SEQPACKET as the type and IPPROTO_SCTP as the protocol.

Here, SOCK_SEQPACKET indicates the creation of a one-to-many style socket.

The function returns a socket descriptor or -1 in case of an error.

Using the PF_INET domain indicates the creation of an endpoint which can use only IPv4 addresses, while PF_INET6 creates an endpoint which can use both IPv6 and IPv4 addresses.

3.1.2. bind()

Applications use bind() to specify which local address and port the SCTP endpoint should associate itself with.

An SCTP endpoint can be associated with multiple addresses. To do this, sctp_bindx() is introduced in Section 9.1 to help applications do the job of associating multiple addresses. But note that an endpoint can only be associated with one local port.

These addresses associated with a socket are the eligible transport addresses for the endpoint to send and receive data. The endpoint will also present these addresses to its peers during the association initialization process, see [RFC4960].

After calling bind(), if the endpoint wishes to accept new associations on the socket, it must call listen() (see

Section 3.1.3).

The function prototype of bind() is

```
int bind(int sd,
         struct sockaddr *addr,
         socklen_t addrlen);
```

and the arguments are

sd: The socket descriptor returned by socket().

addr: The address structure (struct sockaddr_in for an IPv4 address or struct sockaddr_in6 for an IPv6 address, see [RFC3493]).

addrlen: The size of the address structure.

It returns 0 on success and -1 in case of an error.

If sd is an IPv4 socket, the address passed must be an IPv4 address. If the sd is an IPv6 socket, the address passed can either be an IPv4 or an IPv6 address.

Applications cannot call bind() multiple times to associate multiple addresses to an endpoint. After the first call to bind(), all subsequent calls will return an error.

If the IP address part of addr is specified as a wildcard (INADDR_ANY for an IPv4 address, or as IN6ADDR_ANY_INIT or in6addr_any for an IPv6 address), the operating system will associate the endpoint with an optimal address set of the available interfaces. If the IPv4 sin_port or IPv6 sin6_port is set to 0, the operating system will choose an ephemeral port for the endpoint.

If a bind() is not called prior to a sendmsg() call that initiates a new association, the system picks an ephemeral port and will choose an address set equivalent to binding with a wildcard address. One of those addresses will be the primary address for the association. This automatically enables the multi-homing capability of SCTP.

The completion of this bind() process does not allow the SCTP endpoint to accept inbound SCTP association requests. Until a listen() system call, described below, is performed on the socket, the SCTP endpoint will promptly reject an inbound SCTP INIT request with an SCTP ABORT.

3.1.3. listen()

By default, a one-to-many style socket does not accept new association requests. An application uses listen() to mark a socket as being able to accept new associations.

The function prototype is

```
int listen(int sd,
           int backlog);
```

and the arguments are

sd: The socket descriptor of the endpoint.

backlog: If backlog is non-zero, enable listening, else disable listening.

It returns 0 on success and -1 in case of an error.

Note that one-to-many style socket consumers do not need to call accept to retrieve new associations. Calling accept() on a one-to-many style socket should return EOPNOTSUPP. Rather, new associations are accepted automatically, and notifications of the new associations are delivered via recvmsg() with the SCTP_ASSOC_CHANGE event (if these notifications are enabled). Clients will typically not call listen(), so that they can be assured that only actively initiated associations are possible on the socket. Server or peer-to-peer sockets, on the other hand, will always accept new associations, so a well-written application using server one-to-many style sockets must be prepared to handle new associations from unwanted peers.

Also note that the SCTP_ASSOC_CHANGE event provides the association identifier for a new association, so if applications wish to use the association identifier as a parameter to other socket calls, they should ensure that the SCTP_ASSOC_CHANGE event is enabled.

3.1.4. sendmsg() and recvmsg()

An application uses the sendmsg() and recvmsg() call to transmit data to and receive data from its peer.

The function prototypes are

```
ssize_t sendmsg(int sd,
                const struct msghdr *message,
                int flags);
```

and

```
ssize_t recvmsg(int sd,
                struct msghdr *message,
                int flags);
```

using the arguments:

sd: The socket descriptor of the endpoint.

message: Pointer to the msghdr structure which contains a single user message and possibly some ancillary data. See Section 5 for complete description of the data structures.

flags: No new flags are defined for SCTP at this level. See Section 5 for SCTP specific flags used in the msghdr structure.

sendmsg() returns the number of bytes accepted by the kernel or -1 in case of an error. recvmsg() returns the number of bytes received or -1 in case of an error.

As described in Section 5, different types of ancillary data can be sent and received along with user data. When sending, the ancillary data is used to specify the sent behavior, such as the SCTP stream number to use. When receiving, the ancillary data is used to describe the received data, such as the SCTP stream sequence number of the message.

When sending user data with sendmsg(), the msg_name field in the msghdr structure will be filled with one of the transport addresses of the intended receiver. If there is no existing association between the sender and the intended receiver, the sender's SCTP stack will set up a new association and then send the user data (see Section 7.5 for more on implicit association setup). If sendmsg() is called with no data and there is no existing association, a new one will be established. The SCTP_INIT type ancillary data can be used to change some of the parameters used to set up a new association. If sendmsg() is called with NULL data, and there is no existing association but the SCTP_ABORT or SCTP_EOF flags are set as described in Section 5.3.4, then -1 is returned and errno is set to EINVAL. Sending a message using sendmsg() is atomic unless explicit EOR marking is enabled on the socket specified by sd (see Section 8.1.26).

If a peer sends a SHUTDOWN, an SCTP_SHUTDOWN_EVENT notification will be delivered if that notification has been enabled, and no more data can be sent to that association. Any attempt to send more data will cause sendmsg() to return with an ESHUTDOWN error. Note that the

socket is still open for reading at this point so it is possible to retrieve notifications.

When receiving a user message with recvmsg(), the msg_name field in the msghdr structure will be populated with the source transport address of the user data. The caller of recvmsg() can use this address information to determine to which association the received user message belongs. Note that if SCTP_ASSOC_CHANGE events are disabled, applications must use the peer transport address provided in the msg_name field by recvmsg() to perform correlation to an association, since they will not have the association identifier.

If all data in a single message has been delivered, MSG_EOR will be set in the msg_flags field of the msghdr structure (see Section 5.1).

If the application does not provide enough buffer space to completely receive a data message, MSG_EOR will not be set in msg_flags. Successive reads will consume more of the same message until the entire message has been delivered, and MSG_EOR will be set.

If the SCTP stack is running low on buffers, it may partially deliver a message. In this case, MSG_EOR will not be set, and more calls to recvmsg() will be necessary to completely consume the message. Only one message at a time can be partially delivered in any stream. The socket option SCTP_FRAGMENT_INTERLEAVE controls various aspects of what interlacing of messages occurs for both the one-to-one and the one-to-many model sockets. Please consult Section 8.1.20 for further details on message delivery options.

3.1.5. close()

Applications use close() to perform graceful shutdown (as described in Section 10.1 of [RFC4960]) on all the associations currently represented by a one-to-many style socket.

The function prototype is

```
int close(int sd);
```

and the argument is

sd: The socket descriptor of the associations to be closed.

0 is returned on success and -1 in case of an error.

To gracefully shutdown a specific association represented by the one-to-many style socket, an application should use the sendmsg() call, and include the SCTP_EOF flag. A user may optionally terminate an

association non-gracefully by sending with the SCTP_ABORT flag set and possibly passing a user specified abort code in the data field. Both flags SCTP_EOF and SCTP_ABORT are passed with ancillary data (see Section 5.3.4) in the sendmsg() call.

If sd in the close() call is a branched-off socket representing only one association, the shutdown is performed on that association only.

3.1.6. connect()

An application may use the connect() call in the one-to-many style to initiate an association without sending data.

The function prototype is

```
int connect(int sd,
            const struct sockaddr *nam,
            socklen_t len);
```

and the arguments are

sd: The socket descriptor to have a new association added to.

nam: The address structure (struct sockaddr_in for an IPv4 address or struct sockaddr_in6 for an IPv6 address, see [RFC3493]).

len: The size of the address.

0 is returned on success and -1 in case of an error.

Multiple connect() calls can be made on the same socket to create multiple associations. This is different from the semantics of connect() on a UDP socket.

Note that SCTP allows data exchange, similar to T/TCP [RFC1644], during the association set up phase. If an application wants to do this, it cannot use the connect() call. Instead, it should use sendto() or sendmsg() to initiate an association. If it uses sendto() and it wants to change the initialization behavior, it needs to use the SCTP_INITMSG socket option before calling sendto(). Or it can use sendmsg() with SCTP_INIT type ancillary data to initiate an association without calling setsockopt(). Note that the implicit setup is supported for the one-to-many style sockets.

SCTP does not support half close semantics. This means that unlike T/TCP, MSG_EOF should not be set in the flags parameter when calling sendto() or sendmsg() when the call is used to initiate a connection. MSG_EOF is not an acceptable flag with an SCTP socket.

3.2. Non-blocking mode

Some SCTP application may wish to avoid being blocked when calling a socket interface function.

Once a bind() and/or subsequent sctp_bindx() calls are complete on a one-to-many style socket, an application may set the non-blocking option by a fcntl() (such as O_NONBLOCK). After setting the socket to non-blocking mode, the sendmsg() function returns immediately. The success or failure of sending the data message (with possible SCTP_INITMSG ancillary data) will be signaled by the SCTP_ASSOC_CHANGE event with SCTP_COMM_UP or SCTP_CANT_START_ASSOC. If user data could not be sent (due to a SCTP_CANT_START_ASSOC), the sender will also receive an SCTP_SEND_FAILED_EVENT event. Events can be received by the user calling recvmsg(). A server (having called listen()) is also notified of an association up event by the reception of an SCTP_ASSOC_CHANGE with SCTP_COMM_UP via the calling of recvmsg() and possibly the reception of the first data message.

To shutdown the association gracefully, the user must call sendmsg() with no data and with the SCTP_EOF flag set as described in Section 5.3.4. The function returns immediately, and completion of the graceful shutdown is indicated by an SCTP_ASSOC_CHANGE notification of type SHUTDOWN_COMPLETE (see Section 6.1.1). Note that this can also be done using the sctp_sendv() call described in Section 9.12.

An application is recommended to use caution when using select() (or poll()) for writing on a one-to-many style socket. The reason being that the interpretation of select on write is implementation specific. Generally a positive return on a select on write would only indicate that one of the associations represented by the one-to-many socket is writable. An application that writes after the select() returns may still block since the association that was writeable is not the destination association of the write call. Likewise select() (or poll()) for reading from a one-to-many socket will only return an indication that one of the associations represented by the socket has data to be read.

An application that wishes to know that a particular association is ready for reading or writing should either use the one-to-one style or use the sctp_peeloff() (see Section 9.2) function to separate the association of interest from the one-to-many socket.

Note some implementations may have an extended select call such as epoll or kqueue that may escape this limitation and allow a select on a specific association of a one-to-many socket, but this is an implementation specific detail that a portable application cannot

depend on.

3.3. Special considerations

The fact that a one-to-many style socket can provide access to many SCTP associations through a single socket descriptor, has important implications for both application programmers and system programmers implementing this API. A key issue is how buffer space inside the sockets layer is managed. Because this implementation detail directly affects how application programmers must write their code to ensure correct operation and portability, this section provides some guidance to both implementers and application programmers.

An important feature that SCTP shares with TCP is flow control. Specifically, a sender may not send data faster than the receiver can consume it.

For TCP, flow control is typically provided for in the sockets API as follows. If the reader stops reading, the sender queues messages in the socket layer until the send socket buffer is completely filled. This results in a "stalled connection". Further attempts to write to the socket will block or return the error EAGAIN or EWOULDBLOCK for a non-blocking socket. At some point, either the connection is closed, or the receiver begins to read again freeing space in the output queue.

For one-to-one style SCTP sockets (this includes sockets descriptors that were separated from a one-to-many style socket with sctp_peeloff()) the behavior is identical. For one-to-many style SCTP sockets there are multiple associations for a single socket, which makes the situation more complicated. If the implementation uses a single buffer space allocation shared by all associations, a single stalled association can prevent the further sending of data on all associations active on a particular one-to-many style socket.

For a blocking socket, it should be clear that a single stalled association can block the entire socket. For this reason, application programmers may want to use non-blocking one-to-many style sockets. The application should at least be able to send messages to the non-stalled associations.

But a non-blocking socket is not sufficient if the API implementer has chosen a single shared buffer allocation for the socket. A single stalled association would eventually cause the shared allocation to fill, and it would become impossible to send even to non-stalled associations.

The API implementer can solve this problem by providing each

association with its own allocation of outbound buffer space. Each association should conceptually have as much buffer space as it would have if it had its own socket. As a bonus, this simplifies the implementation of sctp_peeloff().

To ensure that a given stalled association will not prevent other non-stalled associations from being writable, application programmers should either:

- demand that the underlying implementation dedicates independent buffer space reservation to each association (as suggested above), or
- verify that their application layer protocol does not permit large amounts of unread data at the receiver (this is true of some request-response protocols, for example), or
- use one-to-one style sockets for association which may potentially stall (either from the beginning, or by using sctp_peeloff before sending large amounts of data that may cause a stalled condition).

4. One-to-One Style Interface

The goal of this style is to follow as closely as possible the current practice of using the sockets interface for a connection oriented protocol, such as TCP. This style enables existing applications using connection oriented protocols to be ported to SCTP with very little effort.

One-to-one style sockets can be connected (explicitly or implicitly) at most once, similar to TCP sockets.

Note that some new SCTP features and some new SCTP socket options can only be utilized through the use of sendmsg() and recvmsg() calls, see Section 4.1.8.

4.1. Basic Operation

A typical server in one-to-one style uses the following system call sequence to prepare an SCTP endpoint for servicing requests:

- socket()
- bind()
- listen()

- accept()

The accept() call blocks until a new association is set up. It returns with a new socket descriptor. The server then uses the new socket descriptor to communicate with the client, using recv() and send() calls to get requests and send back responses.

Then it calls

- close()

to terminate the association.

A typical client uses the following system call sequence to setup an association with a server to request services:

- socket()

- connect()

After returning from connect(), the client uses send()/sendmsg() and recv()/recvmsg() calls to send out requests and receive responses from the server.

The client calls

- close()

to terminate this association when done.

4.1.1. socket()

Applications call socket() to create a socket descriptor to represent an SCTP endpoint.

The function prototype is

```
int socket(int domain,
           int type,
           int protocol);
```

and one uses PF_INET or PF_INET6 as the domain, SOCK_STREAM as the type and IPPROTO_SCTP as the protocol.

Here, SOCK_STREAM indicates the creation of a one-to-one style socket.

Using the PF_INET domain indicates the creation of an endpoint which

can use only IPv4 addresses, while PF_INET6 creates an endpoint which can use both IPv6 and IPv4 addresses.

4.1.2. bind()

Applications use bind() to specify which local address and port the SCTP endpoint should associate itself with.

An SCTP endpoint can be associated with multiple addresses. To do this, sctp_bindx() is introduced in Section 9.1 to help applications do the job of associating multiple addresses. But note that an endpoint can only be associated with one local port.

These addresses associated with a socket are the eligible transport addresses for the endpoint to send and receive data. The endpoint will also present these addresses to its peers during the association initialization process, see [RFC4960].

The function prototype of bind() is

```
int bind(int sd,
         struct sockaddr *addr,
         socklen_t addrlen);
```

and the arguments are

sd: The socket descriptor returned by socket().

addr: The address structure (struct sockaddr_in for an IPv4 address or struct sockaddr_in6 for an IPv6 address, see [RFC3493]).

addrlen: The size of the address structure.

If sd is an IPv4 socket, the address passed must be an IPv4 address. If sd is an IPv6 socket, the address passed can either be an IPv4 or an IPv6 address.

Applications cannot call bind() multiple times to associate multiple addresses to the endpoint. After the first call to bind(), all subsequent calls will return an error.

If the IP address part of addr is specified as a wildcard (INADDR_ANY for an IPv4 address, or as IN6ADDR_ANY_INIT or in6addr_any for an IPv6 address), the operating system will associate the endpoint with an optimal address set of the available interfaces. If the IPv4 sin_port or IPv6 sin6_port is set to 0, the operating system will choose an ephemeral port for the endpoint.

If a bind() is not called prior to the connect() call, the system picks an ephemeral port and will choose an address set equivalent to binding with a wildcard address. One of these addresses will be the primary address for the association. This automatically enables the multi-homing capability of SCTP.

The completion of this bind() process does not allow the SCTP endpoint to accept inbound SCTP association requests. Until a listen() system call, described below, is performed on the socket, the SCTP endpoint will promptly reject an inbound SCTP INIT request with an SCTP ABORT.

4.1.3. listen()

Applications use listen() to allow the SCTP endpoint to accept inbound associations.

The function prototype is

```
int listen(int sd,
           int backlog);
```

and the arguments are

sd: the socket descriptor of the SCTP endpoint.

backlog: this specifies the max number of outstanding associations allowed in the socket's accept queue. These are the associations that have finished the four-way initiation handshake (see Section 5 of [RFC4960]) and are in the ESTABLISHED state. Note, a backlog of '0' indicates that the caller no longer wishes to receive new associations.

It returns 0 on success an -1 in case of an error.

4.1.4. accept()

Applications use the accept() call to remove an established SCTP association from the accept queue of the endpoint. A new socket descriptor will be returned from accept() to represent the newly formed association.

The function prototype is

```
int accept(int sd,
           struct sockaddr *addr,
           socklen_t *addrlen);
```

and the arguments are

sd: The listening socket descriptor.

addr: On return, addr (struct sockaddr_in for an IPv4 address or struct sockaddr_in6 for an IPv6 address, see [RFC3493]) will contain the primary address of the peer endpoint.

addrlen: On return, addrlen will contain the size of addr.

The function returns the socket descriptor for the newly formed association on success and -1 in case of an error.

4.1.5. connect()

Applications use connect() to initiate an association to a peer.

The function prototype is

```
int connect(int sd,
            const struct sockaddr *addr,
            socklen_t addrlen);
```

and the arguments are

sd: The socket descriptor of the endpoint.

addr: The peer's (struct sockaddr_in for an IPv4 address or struct sockaddr_in6 for an IPv6 address, see [RFC3493]) address.

addrlen: The size of the address.

It returns 0 on success and -1 on error.

This operation corresponds to the ASSOCIATE primitive described in Section 10.1 of [RFC4960].

The number of outbound streams the new association has is stack dependent. Applications can use the SCTP_INITMSG option described in Section 8.1.3 before connecting to change the number of outbound streams.

If a bind() is not called prior to the connect() call, the system picks an ephemeral port and will choose an address set equivalent to binding with INADDR_ANY and IN6ADDR_ANY_INIT for IPv4 and IPv6 socket respectively. One of the addresses will be the primary address for the association. This automatically enables the multi-homing capability of SCTP.

Note that SCTP allows data exchange, similar to T/TCP [RFC1644], during the association set up phase. If an application wants to do this, it cannot use the connect() call. Instead, it should use sendto() or sendmsg() to initiate an association. If it uses sendto() and it wants to change the initialization behavior, it needs to use the SCTP_INITMSG socket option before calling sendto(). Or it can use sendmsg() with SCTP_INIT type ancillary data to initiate an association without calling setsockopt(). Note that the implicit setup is supported for the one-to-one style sockets.

SCTP does not support half close semantics. This means that unlike T/TCP, MSG_EOF should not be set in the flags parameter when calling sendto() or sendmsg() when the call is used to initiate a connection. MSG_EOF is not an acceptable flag with an SCTP socket.

4.1.6. close()

Applications use close() to gracefully close down an association.

The function prototype is

```
int close(int sd);
```

and the argument is

sd: The socket descriptor of the association to be closed.

It returns 0 on success and -1 in case of an error.

After an application calls close() on a socket descriptor, no further socket operations will succeed on that descriptor.

4.1.7. shutdown()

SCTP differs from TCP in that it does not have half closed semantics. Hence the shutdown() call for SCTP is an approximation of the TCP shutdown() call, and solves some different problems. Full TCP-compatibility is not provided, so developers porting TCP applications to SCTP may need to recode sections that use shutdown(). (Note that it is possible to achieve the same results as half close in SCTP using SCTP streams.)

The function prototype is

```
int shutdown(int sd,
             int how);
```

and the arguments are

sd: The socket descriptor of the association to be closed.

how: Specifies the type of shutdown. The values are as follows:

SHUT_RD: Disables further receive operations. No SCTP protocol action is taken.

SHUT_WR: Disables further send operations, and initiates the SCTP shutdown sequence.

SHUT_RDWR: Disables further send and receive operations and initiates the SCTP shutdown sequence.

It returns 0 on success and -1 in case of an error.

The major difference between SCTP and TCP shutdown() is that SCTP SHUT_WR initiates immediate and full protocol shutdown, whereas TCP SHUT_WR causes TCP to go into the half closed state. SHUT_RD behaves the same for SCTP as TCP. The purpose of SCTP SHUT_WR is to close the SCTP association while still leaving the socket descriptor open. This allows the caller to receive back any data which SCTP is unable to deliver (see Section 6.1.4 for more information) and receive event notifications.

To perform the ABORT operation described in [RFC4960] Section 10.1, an application can use the socket option SO_LINGER. It is described in Section 8.1.4.

4.1.8. sendmsg() and recvmsg()

With a one-to-one style socket, the application can also use sendmsg() and recvmsg() to transmit data to and receive data from its peer. The semantics is similar to those used in the one-to-many style (see Section 3.1.4), with the following differences:

1. When sending, the msg_name field in the msghdr is not used to specify the intended receiver, rather it is used to indicate a preferred peer address if the sender wishes to discourage the stack from sending the message to the primary address of the receiver. If the socket is connected and the transport address given is not part of the current association, the data will not be sent and an SCTP_SEND_FAILED_EVENT event will be delivered to the application if send failure events are enabled.

2. Using sendmsg() on a non-connected one-to-one style socket for implicit connection setup may or may not work depending on the SCTP implementation.

4.1.9. getpeername()

Applications use getpeername() to retrieve the primary socket address of the peer. This call is for TCP compatibility, and is not multi-homed. It may not work with one-to-many style sockets depending on the implementation. See Section 9.3 for a multi-homed style version of the call.

The function prototype is

```
int getpeername(int sd,
                struct sockaddr *address,
                socklen_t *len);
```

and the arguments are:

sd: The socket descriptor to be queried.

address: On return, the peer primary address is stored in this buffer. If the socket is an IPv4 socket, the address will be IPv4. If the socket is an IPv6 socket, the address will be either an IPv6 or IPv4 address.

len: The caller should set the length of address here. On return, this is set to the length of the returned address.

It returns 0 on success and -1 in case of an error.

If the actual length of the address is greater than the length of the supplied sockaddr structure, the stored address will be truncated.

5. Data Structures

This section discusses important data structures which are specific to SCTP and are used with sendmsg() and recvmsg() calls to control SCTP endpoint operations and to access ancillary information and notifications.

5.1. The msghdr and cmsghdr Structures

The msghdr structure used in the sendmsg() and recvmsg() calls, as well as the ancillary data carried in the structure, is the key for the application to set and get various control information from the SCTP endpoint.

The msghdr and the related cmsghdr structures are defined and discussed in detail in [RFC3542]. They are defined as:

```
struct msghdr {
  void *msg_name;           /* ptr to socket address structure */
  socklen_t msg_namelen;    /* size of socket address structure */
  struct iovec *msg_iov;    /* scatter/gather array */
  int msg_iovlen;           /* # elements in msg_iov */
  void *msg_control;        /* ancillary data */
  socklen_t msg_controllen; /* ancillary data buffer length */
  int msg_flags;            /* flags on received message */
};

struct cmsghdr {
  socklen_t cmsg_len; /* #bytes, including this header */
  int cmsg_level;     /* originating protocol */
  int cmsg_type;      /* protocol-specific type */
                      /* followed by unsigned char cmsg_data[]; */
};
```

In the msghdr structure, the usage of msg_name has been discussed in previous sections (see Section 3.1.4 and Section 4.1.8).

The scatter/gather buffers, or I/O vectors (pointed to by the msg_iov field) are treated by SCTP as a single user message for both sendmsg() and recvmsg().

SCTP stack uses the ancillary data (msg_control field) to communicate the attributes, such as SCTP_RCVINFO, of the message stored in msg_iov to the socket end point. The different ancillary data types are described in Section 5.3.

The msg_flags are not used when sending a message with sendmsg().

If a notification has arrived, recvmsg() will return the notification in msg_iov field and set MSG_NOTIFICATION flag in msg_flags. If the MSG_NOTIFICATION flag is not set, recvmsg() will return data. See Section 6 for more information about notifications.

If all portions of a data frame or notification have been read, recvmsg() will return with MSG_EOR set in msg_flags.

5.2. Ancillary Data Considerations and Semantics

Programming with ancillary socket data (msg_control) contains some subtleties and pitfalls, which are discussed below.

5.2.1. Multiple Items and Ordering

Multiple ancillary data items may be included in any call to sendmsg() or recvmsg(); these may include multiple SCTP or non-SCTP,

such as IP level items, or both.

The ordering of ancillary data items (either by SCTP or another protocol) is not significant and is implementation-dependent, so applications must not depend on any ordering.

SCTP_SNDRCV/SCTP_SNDINFO/SCTP_RCVINFO type ancillary data always correspond to the data in the msghdr's msg_iov member. There can be only one single such type ancillary data for each sendmsg() or recvmsg() call.

5.2.2. Accessing and Manipulating Ancillary Data

Applications can infer the presence of data or ancillary data by examining the msg_iovlen and msg_controllen msghdr members, respectively.

Implementations may have different padding requirements for ancillary data, so portable applications should make use of the macros CMSG_FIRSTHDR, CMSG_NXTHDR, CMSG_DATA, CMSG_SPACE, and CMSG_LEN. See [RFC3542] and the SCTP implementation's documentation for more information. The following is an example, from [RFC3542], demonstrating the use of these macros to access ancillary data:

```
struct msghdr msg;
struct cmsghdr *cmsgptr;

/* fill in msg */

/* call recvmsg() */

for (cmsgptr = CMSG_FIRSTHDR(&msg); cmsgptr != NULL;
     cmsgptr = CMSG_NXTHDR(&msg, cmsgptr)) {
  if (cmsgptr->cmsg_level == ... && cmsgptr->cmsg_type == ... ) {
    u_char  *ptr;

    ptr = CMSG_DATA(cmsgptr);
    /* process data pointed to by ptr */
  }
}
```

5.2.3. Control Message Buffer Sizing

The information conveyed via SCTP_SNDRCV/SCTP_SNDINFO/SCTP_RCVINFO ancillary data will often be fundamental to the correct and sane operation of the sockets application. This is particularly true of the one-to-many semantics, but also of the one-to-one semantics. For example, if an application needs to send and receive data on

different SCTP streams, SCTP_SNDRCV/SCTP_SNDINFO/SCTP_RCVINFO ancillary data is indispensable.

Given that some ancillary data is critical, and that multiple ancillary data items may appear in any order, applications should be carefully written to always provide a large enough buffer to contain all possible ancillary data that can be presented by recvmsg(). If the buffer is too small, and crucial data is truncated, it may pose a fatal error condition.

Thus, it is essential that applications be able to deterministically calculate the maximum required buffer size to pass to recvmsg(). One constraint imposed on this specification that makes this possible is that all ancillary data definitions are of a fixed length. One way to calculate the maximum required buffer size might be to take the sum the sizes of all enabled ancillary data item structures, as calculated by CMSG_SPACE. For example, if we enabled SCTP_SNDRCV_INFO and IPV6_RECVPKTINFO [RFC3542], we would calculate and allocate the buffer size as follows:

```
size_t total;
void *buf;

total = CMSG_SPACE(sizeof(struct sctp_sndrcvinfo)) +
        CMSG_SPACE(sizeof(struct in6_pktinfo));

buf = malloc(total);
```

We could then use this buffer (buf) for msg_control on each call to recvmsg() and be assured that we would not lose any ancillary data to truncation.

5.3. SCTP msg_control Structures

A key element of all SCTP specific socket extensions is the use of ancillary data to specify and access SCTP specific data via the struct msghdr's msg_control member used in sendmsg() and recvmsg(). Fine-grained control over initialization and sending parameters are handled with ancillary data.

Each ancillary data item is proceeded by a struct cmsghdr (see Section 5.1), which defines the function and purpose of the data contained in the cmsg_data[] member.

By default on either style socket, SCTP will pass no ancillary data; Specific ancillary data items can be enabled with socket options defined for SCTP; see Section 6.2.

Note that all ancillary types are fixed length; see Section 5.2 for further discussion on this. These data structures use struct sockaddr_storage (defined in [RFC3493]) as a portable, fixed length address format.

Other protocols may also provide ancillary data to the socket layer consumer. These ancillary data items from other protocols may intermingle with SCTP data. For example, the IPv6 socket API definitions ([RFC3542] and [RFC3493]) define a number of ancillary data items. If a socket API consumer enables delivery of both SCTP and IPv6 ancillary data, they both may appear in the same msg_control buffer in any order. An application may thus need to handle other types of ancillary data besides those passed by SCTP.

The sockets application must provide a buffer large enough to accommodate all ancillary data provided via recvmsg(). If the buffer is not large enough, the ancillary data will be truncated and the msghdr's msg_flags will include MSG_CTRUNC.

5.3.1. SCTP Initiation Structure (SCTP_INIT)

This cmsghdr structure provides information for initializing new SCTP associations with sendmsg(). The SCTP_INITMSG socket option uses this same data structure. This structure is not used for recvmsg().

cmsg_level	cmsg_type	cmsg_data[]
IPPROTO_SCTP	SCTP_INIT	struct sctp_initmsg

The sctp_initmsg structure is defined below:

```
struct sctp_initmsg {
  uint16_t sinit_num_ostreams;
  uint16_t sinit_max_instreams;
  uint16_t sinit_max_attempts;
  uint16_t sinit_max_init_timeo;
};
```

sinit_num_ostreams: This is an integer number representing the number of streams that the application wishes to be able to send to. This number is confirmed in the SCTP_COMM_UP notification and must be verified since it is a negotiated number with the remote endpoint. The default value of 0 indicates to use the endpoint default value.

sinit_max_instreams: This value represents the maximum number of inbound streams the application is prepared to support. This value is bounded by the actual implementation. In other words the user may be able to support more streams than the Operating System. In such a case, the Operating System limit overrides the value requested by the user. The default value of 0 indicates to use the endpoints default value.

sinit_max_attempts: This integer specifies how many attempts the SCTP endpoint should make at resending the INIT. This value overrides the system SCTP 'Max.Init.Retransmits' value. The default value of 0 indicates to use the endpoints default value. This is normally set to the system's default 'Max.Init.Retransmit' value.

sinit_max_init_timeo: This value represents the largest Time-Out or RTO value (in milliseconds) to use in attempting an INIT. Normally the 'RTO.Max' is used to limit the doubling of the RTO upon timeout. For the INIT message this value may override 'RTO.Max'. This value must not influence 'RTO.Max' during data transmission and is only used to bound the initial setup time. A default value of 0 indicates to use the endpoints default value. This is normally set to the system's 'RTO.Max' value (60 seconds).

5.3.2. SCTP Header Information Structure (SCTP_SNDRCV) - DEPRECATED

This cmsghdr structure specifies SCTP options for sendmsg() and describes SCTP header information about a received message through recvmsg(). This structure mixes the send and receive path. SCTP_SNDINFO described in Section 5.3.4 and SCTP_RCVINFO described in Section 5.3.5 split this information. These structures should be used, when possible, since SCTP_SNDRCV is deprecated.

cmsg_level	cmsg_type	cmsg_data[]
IPPROTO_SCTP	SCTP_SNDRCV	struct sctp_sndrcvinfo

The sctp_sndrcvinfo structure is defined below:

```
struct sctp_sndrcvinfo {
  uint16_t sinfo_stream;
  uint16_t sinfo_ssn;
  uint16_t sinfo_flags;
  uint32_t sinfo_ppid;
  uint32_t sinfo_context;
  uint32_t sinfo_timetolive;
  uint32_t sinfo_tsn;
  uint32_t sinfo_cumtsn;
  sctp_assoc_t sinfo_assoc_id;
};
```

sinfo_stream: For recvmsg() the SCTP stack places the message's stream number in this value. For sendmsg() this value holds the stream number that the application wishes to send this message to. If a sender specifies an invalid stream number an error indication is returned and the call fails.

sinfo_ssn: For recvmsg() this value contains the stream sequence number that the remote endpoint placed in the DATA chunk. For fragmented messages this is the same number for all deliveries of the message (if more than one recvmsg() is needed to read the message). The sendmsg() call will ignore this parameter.

sinfo_flags: This field may contain any of the following flags and is composed of a bitwise OR of these values.

recvmsg() flags:

SCTP_UNORDERED: This flag is present when the message was sent un-ordered.

sendmsg() flags:

SCTP_UNORDERED: This flag requests the un-ordered delivery of the message. If this flag is clear the datagram is considered an ordered send.

SCTP_ADDR_OVER: This flag, in the one-to-many style, requests the SCTP stack to override the primary destination address with the address found with the sendto/sendmsg call.

SCTP_ABORT: Setting this flag causes the specified association to abort by sending an ABORT message to the peer. The ABORT chunk will contain an error cause 'User Initiated Abort' with cause code 12. The cause specific information of this error cause is provided in msg_iov.

SCTP_EOF: Setting this flag invokes the SCTP graceful shutdown procedure on the specified association. Graceful shutdown assures that all data queued by both endpoints is successfully transmitted before closing the association.

SCTP_SENDALL: This flag, if set, will cause a one-to-many model socket to send the message to all associations that are currently established on this socket. For the one-to-one socket, this flag has no effect.

sinfo_ppid: This value in sendmsg() is an unsigned integer that is passed to the remote end in each user message. In recvmsg() this value is the same information that was passed by the upper layer in the peer application. Please note that the SCTP stack performs no byte order modification of this field. For example, if the DATA chunk has to contain a given value in network byte order, the SCTP user has to perform the htonl() computation.

sinfo_context: This value is an opaque 32 bit context datum that is used in the sendmsg() function. This value is passed back to the upper layer if an error occurs on the send of a message and is retrieved with each undelivered message.

sinfo_timetolive: For the sending side, this field contains the message time to live in milliseconds. The sending side will expire the message within the specified time period if the message as not been sent to the peer within this time period. This value will override any default value set using any socket option. Also note that the value of 0 is special in that it indicates no timeout should occur on this message.

sinfo_tsn: For the receiving side, this field holds a TSN that was assigned to one of the SCTP Data Chunks. For the sending side it is ignored.

sinfo_cumtsn: This field will hold the current cumulative TSN as known by the underlying SCTP layer. Note this field is ignored when sending.

sinfo_assoc_id: The association handle field, sinfo_assoc_id, holds the identifier for the association announced in the SCTP_COMM_UP notification. All notifications for a given association have the same identifier. Ignored for one-to-one style sockets.

An sctp_sndrcvinfo item always corresponds to the data in msg_iov.

5.3.3. Extended SCTP Header Information Structure (SCTP_EXTRCV) - DEPRECATED

This cmsghdr structure specifies SCTP options for SCTP header information about a received message via recvmsg(). Note that this structure is an extended version of SCTP_SNDRCV (see Section 5.3.2) and will only be received if the user has set the socket option SCTP_USE_EXT_RCVINFO to true in addition to any event subscription needed to receive ancillary data. See Section 8.1.22 on this socket option. Note that next message data is not valid unless the current message is completely read, i.e. the MSG_EOR is set, in other words if the application has more data to read from the current message then no next message information will be available.

SCTP_NXTINFO described in Section 5.3.6 should be used when possible, since SCTP_EXTRCV is considered deprecated.

cmsg_level	cmsg_type	cmsg_data[]
IPPROTO_SCTP	SCTP_EXTRCV	struct sctp_extrcvinfo

The sctp_extrcvinfo structure is defined below:

```
struct sctp_extrcvinfo {
  uint16_t sinfo_stream;
  uint16_t sinfo_ssn;
  uint16_t sinfo_flags;
  uint32_t sinfo_ppid;
  uint32_t sinfo_context;
  uint32_t sinfo_pr_value;
  uint32_t sinfo_tsn;
  uint32_t sinfo_cumtsn;
  uint16_t serinfo_next_flags;
  uint16_t serinfo_next_stream;
  uint32_t serinfo_next_aid;
  uint32_t serinfo_next_length;
  uint32_t serinfo_next_ppid;
  sctp_assoc_t sinfo_assoc_id;
};
```

sinfo_*: Please see Section 5.3.2 for the details for these fields.

serinfo_next_flags: This bitmask will hold one or more of the following values:

SCTP_NEXT_MSG_AVAIL: This bit, when set to 1, indicates that next message information is available i.e.: next_stream, next_asocid, next_length and next_ppid fields all have valid values. If this bit is set to 0, then these fields are not valid and should be ignored.

SCTP_NEXT_MSG_ISCOMPLETE: This bit, when set, indicates that the next message is completely in the receive buffer. The next_length field thus contains the entire message size. If this flag is set to 0, then the next_length field only contains part of the message size since the message is still being received (it is being partially delivered).

SCTP_NEXT_MSG_IS_UNORDERED: This bit, when set, indicates that the next message to be received was sent by the peer as unordered. If this bit is not set (i.e. the bit is 0) the next message to be read is an ordered message in the stream specified.

SCTP_NEXT_MSG_IS_NOTIFICATION: This bit, when set, indicates that the next message to be received is not a message from the peer, but instead is a MSG_NOTIFICATION from the local SCTP stack.

serinfo_next_stream: This value, when valid (see serinfo_next_flags), contains the next stream number that will be received on a subsequent call to one of the receive message functions.

serinfo_next_aid: This value, when valid (see serinfo_next_flags), contains the next association identifier that will be received on a subsequent call to one of the receive message functions.

serinfo_next_length: This value, when valid (see serinfo_next_flags), contains the length of the next message that will be received on a subsequent call to one of the receive message functions. Note that this length may be a partial length depending on the settings of next_flags.

serinfo_next_ppid: This value, when valid (see serinfo_next_flags), contains the ppid of the next message that will be received on a subsequent call to one of the receive message functions.

5.3.4. SCTP Send Information Structure (SCTP_SNDINFO)

This cmsghdr structure specifies SCTP options for sendmsg().

cmsg_level	cmsg_type	cmsg_data[]
IPPROTO_SCTP	SCTP_SNDINFO	struct sctp_sndinfo

The sctp_sndinfo structure is defined below:

```
struct sctp_sndinfo {
  uint16_t snd_sid;
  uint16_t snd_flags;
  uint32_t snd_ppid;
  uint32_t snd_context;
  sctp_assoc_t snd_assoc_id;
};
```

snd_sid: This value holds the stream number that the application wishes to send this message to. If a sender specifies an invalid stream number an error indication is returned and the call fails.

snd_flags: This field may contain any of the following flags and is composed of a bitwise OR of these values.

SCTP_UNORDERED: This flag requests the un-ordered delivery of the message. If this flag is clear the datagram is considered an ordered send.

SCTP_ADDR_OVER: This flag, in the one-to-many style, requests the SCTP stack to override the primary destination address with the address found with the sendto()/sendmsg call.

SCTP_ABORT: Setting this flag causes the specified association to abort by sending an ABORT message to the peer. The ABORT chunk will contain an error cause 'User Initiated Abort' with cause code 12. The cause specific information of this error cause is provided in msg_iov.

SCTP_EOF: Setting this flag invokes the SCTP graceful shutdown procedures on the specified association. Graceful shutdown assures that all data queued by both endpoints is successfully transmitted before closing the association.

SCTP_SENDALL: This flag, if set, will cause a one-to-many model socket to send the message to all associations that are currently established on this socket. For the one-to-one socket, this flag has no effect.

snd_ppid: This value in sendmsg() is an unsigned integer that is passed to the remote end in each user message. Please note that the SCTP stack performs no byte order modification of this field. For example, if the DATA chunk has to contain a given value in network byte order, the SCTP user has to perform the htonl() computation.

snd_context: This value is an opaque 32 bit context datum that is used in the sendmsg() function. This value is passed back to the upper layer if an error occurs on the send of a message and is retrieved with each undelivered message.

snd_assoc_id: The association handle field, sinfo_assoc_id, holds the identifier for the association announced in the SCTP_COMM_UP notification. All notifications for a given association have the same identifier. Ignored for one-to-one style sockets.

An sctp_sndinfo item always corresponds to the data in msg_iov.

5.3.5. SCTP Receive Information Structure (SCTP_RCVINFO)

This cmsghdr structure describes SCTP receive information about a received message through recvmsg().

To enable the delivery of this information an application must use the SCTP_RECVRCVINFO socket option (see Section 8.1.29).

cmsg_level	cmsg_type	cmsg_data[]
IPPROTO_SCTP	SCTP_RCVINFO	struct sctp_rcvinfo

The sctp_rcvinfo structure is defined below:

```
struct sctp_rcvinfo {
  uint16_t rcv_sid;
  uint16_t rcv_ssn;
  uint16_t rcv_flags;
  uint32_t rcv_ppid;
  uint32_t rcv_tsn;
  uint32_t rcv_cumtsn;
  uint32_t rcv_context;
  sctp_assoc_t rcv_assoc_id;
};
```

rcv_sid: The SCTP stack places the message's stream number in this value.

rcv_ssn: This value contains the stream sequence number that the remote endpoint placed in the DATA chunk. For fragmented messages this is the same number for all deliveries of the message (if more than one recvmsg() is needed to read the message).

rcv_flags: This field may contain any of the following flags and is composed of a bitwise OR of these values.

SCTP_UNORDERED: This flag is present when the message was sent un-ordered.

rcv_ppid: This value is the same information that was passed by the upper layer in the peer application. Please note that the SCTP stack performs no byte order modification of this field. For example, if the DATA chunk has to contain a given value in network byte order, the SCTP user has to perform the ntohl() computation.

rcv_tsn: This field holds a TSN that was assigned to one of the SCTP Data Chunks.

rcv_cumtsn: This field will hold the current cumulative TSN as known by the underlying SCTP layer.

rcv_assoc_id: The association handle field, sinfo_assoc_id, holds the identifier for the association announced in the SCTP_COMM_UP notification. All notifications for a given association have the same identifier. Ignored for one-to-one style sockets.

rcv_context: This value is an opaque 32 bit context datum that was set by the user with the SCTP_CONTEXT socket option. This value is passed back to the upper layer if an error occurs on the send of a message and is retrieved with each undelivered message.

An sctp_rcvinfo item always corresponds to the data in msg_iov.

5.3.6. SCTP Next Receive Information Structure (SCTP_NXTINFO)

This cmsghdr structure describes SCTP receive information of the next message which will be delivered through recvmsg() if this information is already available when delivering the current message.

To enable the delivery of this information an application must use the SCTP_RECVNXTINFO socket option (see Section 8.1.30).

cmsg_level	cmsg_type	cmsg_data[]
IPPROTO_SCTP	SCTP_NXTINFO	struct sctp_nxtinfo

The sctp_nxtinfo structure is defined below:

```
struct sctp_nxtinfo {
  uint16_t nxt_sid;
  uint16_t nxt_flags;
  uint32_t nxt_ppid;
  uint32_t nxt_length;
  sctp_assoc_t nxt_assoc_id;
};
```

nxt_sid: The SCTP stack places the next message's stream number in this value.

nxt_flags: This field may contain any of the following flags and is composed of a bitwise OR of these values.

SCTP_UNORDERED: This flag is present when the next message was sent un-ordered.

SCTP_COMPLETE: This flag indicates that the entire message has been received and is in the socket buffer. Note that this has special implications with respect to the nxt_length field, see nxt_length description below.

SCTP_NOTIFICATION: This flag is present when the next message is not a user message but instead is a notification.

nxt_ppid: This value is the same information that was passed by the upper layer in the peer application for the next message. Please note that the SCTP stack performs no byte order modification of this field. For example, if the DATA chunk has to contain a given value in network byte order, the SCTP user has to perform the ntohl() computation.

nxt_length: This value is the length of the message currently within the socket buffer. This might NOT be the entire length of the message since a partial delivery may be in progress. Only if the flag SCTP_COMPLETE is set in the nxt_flags field does this field represent the entire next message size.

nxt_assoc_id: The association handle field of the next message, nxt_assoc_id, holds the identifier for the association announced in the SCTP_COMM_UP notification. All notifications for a given association have the same identifier. Ignored for one-to-one style sockets.

5.3.7. SCTP PR-SCTP Information Structure (SCTP_PRINFO)

This cmsghdr structure specifies SCTP options for sendmsg().

cmsg_level	cmsg_type	cmsg_data[]
IPPROTO_SCTP	SCTP_PRINFO	struct sctp_prinfo

The sctp_prinfo structure is defined below:

```
struct sctp_prinfo {
  uint16_t pr_policy;
  uint32_t pr_value;
};
```

pr_policy: This specifies which PR-SCTP policy is used. Using SCTP_PR_SCTP_NONE results in a reliable transmission. When SCTP_PR_SCTP_TTL is used, the PR-SCTP policy "timed reliability" defined in [RFC3758] is used. In this case, the lifetime is provided in pr_value.

pr_value: The meaning of this field depends on the PR-SCTP policy specified by the pr_policy field. It is ignored when SCTP_PR_SCTP_NONE is specified. In case of SCTP_PR_SCTP_TTL the lifetime in milliseconds is specified.

An sctp_prinfo item always corresponds to the data in msg_iov.

5.3.8. SCTP AUTH Information Structure (SCTP_AUTHINFO)

This cmsghdr structure specifies SCTP options for sendmsg().

cmsg_level	cmsg_type	cmsg_data[]
IPPROTO_SCTP	SCTP_AUTHINFO	struct sctp_authinfo

The sctp_authinfo structure is defined below:

```
struct sctp_authinfo {
  uint16_t auth_keynumber;
};
```

auth_keynumber: This specifies the shared key identifier used for sending the user message.

An sctp_authinfo item always corresponds to the data in msg_iov. Please note that the SCTP implementation must not bundle user messages that needs to be authenticated using different shared key identifiers.

5.3.9. SCTP Destination IPv4 Address Structure (SCTP_DSTADDRV4)

This cmsghdr structure specifies SCTP options for sendmsg().

cmsg_level	cmsg_type	cmsg_data[]
IPPROTO_SCTP	SCTP_DSTADDRV4	struct in_addr

This ancillary data can be used to provide more than one destination address to sendmsg(). It can be used to implement sctp_sendv() using sendmsg().

5.3.10. SCTP Destination IPv6 Address Structure (SCTP_DSTADDRV6)

This cmsghdr structure specifies SCTP options for sendmsg().

cmsg_level	cmsg_type	cmsg_data[]
IPPROTO_SCTP	SCTP_DSTADDRV6	struct in6_addr

This ancillary data can be used to provide more than one destination address to sendmsg(). It can be used to implement sctp_sendv() using sendmsg().

6. SCTP Events and Notifications

An SCTP application may need to understand and process events and errors that happen on the SCTP stack. These events include network status changes, association startups, remote operational errors and undeliverable messages. All of these can be essential for the application.

When an SCTP application layer does a recvmsg() the message read is normally a data message from a peer endpoint. If the application wishes to have the SCTP stack deliver notifications of non-data events, it sets the appropriate socket option for the notifications it wants. See Section 6.2 for these socket options. When a notification arrives, recvmsg() returns the notification in the application-supplied data buffer via msg_iov, and sets MSG_NOTIFICATION in msg_flags.

This section details the notification structures. Every notification structure carries some common fields which provide general information.

A recvmsg() call will return only one notification at a time. Just as when reading normal data, it may return part of a notification if the msg_iov buffer is not large enough. If a single read is not sufficient, msg_flags will have MSG_EOR clear. The user must finish reading the notification before subsequent data can arrive.

6.1. SCTP Notification Structure

The notification structure is defined as the union of all notification types.

```
union sctp_notification {
  struct sctp_tlv {
    uint16_t sn_type; /* Notification type. */
    uint16_t sn_flags;
    uint32_t sn_length;
  } sn_header;
  struct sctp_assoc_change sn_assoc_change;
  struct sctp_paddr_change sn_paddr_change;
  struct sctp_remote_error sn_remote_error;
  struct sctp_send_failed sn_send_failed;
  struct sctp_shutdown_event sn_shutdown_event;
  struct sctp_adaptation_event sn_adaptation_event;
  struct sctp_pdapi_event sn_pdapi_event;
  struct sctp_authkey_event sn_auth_event;
  struct sctp_sender_dry_event sn_sender_dry_event;
  struct sctp_send_failed_event sn_send_failed_event;
};
```

sn_type: The following list describes the SCTP notification and event types for the field sn_type.

SCTP_ASSOC_CHANGE: This tag indicates that an association has either been opened or closed. Refer to Section 6.1.1 for details.

SCTP_PEER_ADDR_CHANGE: This tag indicates that an address that is part of an existing association has experienced a change of state (e.g. a failure or return to service of the reachability of an endpoint via a specific transport address). Please see Section 6.1.2 for data structure details.

SCTP_REMOTE_ERROR: The attached error message is an Operational Error received from the remote peer. It includes the complete TLV sent by the remote endpoint. See Section 6.1.3 for the detailed format.

SCTP_SEND_FAILED_EVENT: The attached datagram could not be sent to the remote endpoint. This structure includes the original SCTP_SNDINFO that was used in sending this message i.e. this structure uses the sctp_sndinfo per Section 6.1.11.

SCTP_SHUTDOWN_EVENT: The peer has sent a SHUTDOWN. No further data should be sent on this socket.

SCTP_ADAPTATION_INDICATION: This notification holds the peer's indicated adaptation layer. Please see Section 6.1.6.

SCTP_PARTIAL_DELIVERY_EVENT: This notification is used to tell a receiver that the partial delivery has been aborted. This may indicate the association is about to be aborted. Please see Section 6.1.7.

SCTP_AUTHENTICATION_EVENT: This notification is used to tell a receiver that either an error occurred on authentication, or a new key was made active. See Section 6.1.8.

SCTP_SENDER_DRY_EVENT: This notification is used to inform the application that the sender has no more user data queued for transmission nor retransmission. See Section 6.1.9.

sn_flags: These are notification-specific flags.

sn_length: This is the length of the whole sctp_notification structure including the sn_type, sn_flags, and sn_length fields.

6.1.1. SCTP_ASSOC_CHANGE

Communication notifications inform the application that an SCTP association has either begun or ended. The identifier for a new association is provided by this notification. The notification information has the following format:

```
struct sctp_assoc_change {
  uint16_t sac_type;
  uint16_t sac_flags;
  uint32_t sac_length;
  uint16_t sac_state;
  uint16_t sac_error;
  uint16_t sac_outbound_streams;
  uint16_t sac_inbound_streams;
  sctp_assoc_t sac_assoc_id;
  uint8_t  sac_info[];
};
```

sac_type: It should be SCTP_ASSOC_CHANGE.

sac_flags: Currently unused.

sac_length: This field is the total length of the notification data, including the notification header.

sac_state: This field holds one of a number of values that communicate the event that happened to the association. They include:

SCTP_COMM_UP: A new association is now ready and data may be exchanged with this peer. When an association has been established successfully, this notification should be the first one.

SCTP_COMM_LOST: The association has failed. The association is now in the closed state. If SEND_FAILED notifications are turned on, an SCTP_COMM_LOST is accompanied by a series of SCTP_SEND_FAILED_EVENT events, one for each outstanding message.

SCTP_RESTART: SCTP has detected that the peer has restarted.

SCTP_SHUTDOWN_COMP: The association has gracefully closed.

SCTP_CANT_STR_ASSOC: The association failed to setup. If non blocking mode is set and data was sent (on a one-to-many style socket), an SCTP_CANT_STR_ASSOC is accompanied by a series of SCTP_SEND_FAILED_EVENT events, one for each outstanding message.

sac_error: If the state was reached due to an error condition (e.g. SCTP_COMM_LOST) any relevant error information is available in this field. This corresponds to the protocol error codes defined in [RFC4960].

sac_outbound_streams:

sac_inbound_streams: The maximum number of streams allowed in each direction are available in sac_outbound_streams and sac_inbound streams.

sac_assoc_id: The sac_assoc_id field holds the identifier for the association. All notifications for a given association have the same association identifier. For a one-to-one style socket, this field is ignored.

sac_info: If the sac_state is SCTP_COMM_LOST and an ABORT chunk was received for this association, sac_info[] contains the complete ABORT chunk as defined in the SCTP specification [RFC4960] Section 3.3.7. If the sac_state is SCTP_COMM_UP or SCTP_RESTART, sac_info may contain an array of uint8_t describing the features that the current association supports. Features may include

SCTP_ASSOC_SUPPORTS_PR: Both endpoints support the protocol extension described in [RFC3758].

SCTP_ASSOC_SUPPORTS_AUTH: Both endpoints support the protocol extension described in [RFC4895].

SCTP_ASSOC_SUPPORTS_ASCONF: Both endpoints support the protocol extension described in [RFC5061].

SCTP_ASSOC_SUPPORTS_MULTIBUF: For a one-to-many style socket, the local endpoints use separate send and/or receive buffers for each SCTP association.

6.1.2. SCTP_PEER_ADDR_CHANGE

When a destination address of a multi-homed peer encounters a state change a peer address change event is sent. The notification has the following format:

```
struct sctp_paddr_change {
  uint16_t spc_type;
  uint16_t spc_flags;
  uint32_t spc_length;
  struct sockaddr_storage spc_aaddr;
  uint32_t spc_state;
  uint32_t spc_error;
  sctp_assoc_t spc_assoc_id;
}
```

spc_type: It should be SCTP_PEER_ADDR_CHANGE.

spc_flags: Currently unused.

spc_length: This field is the total length of the notification data, including the notification header.

spc_aaddr: The affected address field holds the remote peer's address that is encountering the change of state.

spc_state: This field holds one of a number of values that communicate the event that happened to the address. They include:

SCTP_ADDR_AVAILABLE: This address is now reachable. This notification is provided whenever an address becomes reachable.

SCTP_ADDR_UNREACHABLE: The address specified can no longer be reached. Any data sent to this address is rerouted to an alternate until this address becomes reachable. This notification is provided whenever an address becomes unreachable.

SCTP_ADDR_REMOVED: The address is no longer part of the association.

SCTP_ADDR_ADDED: The address is now part of the association.

SCTP_ADDR_MADE_PRIM: This address has now been made to be the primary destination address. This notification is provided whenever an address is made primary.

spc_error: If the state was reached due to any error condition (e.g. SCTP_ADDR_UNREACHABLE) any relevant error information is available in this field.

spc_assoc_id: The spc_assoc_id field holds the identifier for the association. All notifications for a given association have the same association identifier. For a one-to-one style socket, this field is ignored.

6.1.3. SCTP_REMOTE_ERROR

A remote peer may send an Operational Error message to its peer. This message indicates a variety of error conditions on an association. The entire ERROR chunk as it appears on the wire is included in an SCTP_REMOTE_ERROR event. Please refer to the SCTP specification [RFC4960] and any extensions for a list of possible error formats. An SCTP error notification has the following format:

```
struct sctp_remote_error {
  uint16_t sre_type;
  uint16_t sre_flags;
  uint32_t sre_length;
  uint16_t sre_error;
  sctp_assoc_t sre_assoc_id;
  uint8_t sre_data[];
};
```

sre_type: It should be SCTP_REMOTE_ERROR.

sre_flags: Currently unused.

sre_length: This field is the total length of the notification data, including the notification header and the contents of sre_data.

sre_error: This value represents one of the Operational Error causes defined in the SCTP specification, in network byte order.

sre_assoc_id: The sre_assoc_id field holds the identifier for the association. All notifications for a given association have the same association identifier. For a one-to-one style socket, this field is ignored.

sre_data: This contains the ERROR chunk as defined in the SCTP specification [RFC4960] Section 3.3.10.

6.1.4. SCTP_SEND_FAILED - DEPRECATED

Please note that this notification is deprecated. Use SCTP_SEND_FAILED_EVENT instead.

If SCTP cannot deliver a message, it can return back the message as a notification if the SCTP_SEND_FAILED event is enabled. The

notification has the following format:

```
struct sctp_send_failed {
  uint16_t ssf_type;
  uint16_t ssf_flags;
  uint32_t ssf_length;
  uint32_t ssf_error;
  struct sctp_sndrcvinfo ssf_info;
  sctp_assoc_t ssf_assoc_id;
  uint8_t ssf_data[];
};
```

ssf_type: It should be SCTP_SEND_FAILED.

ssf_flags: The flag value will take one of the following values:

SCTP_DATA_UNSENT: Indicates that the data was never put on the wire.

SCTP_DATA_SENT: Indicates that the data was put on the wire. Note that this does not necessarily mean that the data was (or was not) successfully delivered.

ssf_length: This field is the total length of the notification data, including the notification header and the payload in ssf_data.

ssf_error: This value represents the reason why the send failed, and if set, will be an SCTP protocol error code as defined in [RFC4960] Section 3.3.10.

ssf_info: The ancillary data (struct sctp_sndrcvinfo) used to send the undelivered message. Regardless of if ancillary data is used or not, the ssf_info.sinfo_flags field indicates if the complete message or only part of the message is returned in ssf_data. If only part of the message is returned, it means that the part which is not present has been sent successfully to the peer.

If the complete message cannot be sent, the SCTP_DATA_NOT_FRAG flags is set in ssf_info.sinfo_flags. If the first part of the message is sent successfully, the SCTP_DATA_LAST_FRAG is set. This means that the tail end of the message is returned in ssf_data.

ssf_assoc_id: The ssf_assoc_id field, ssf_assoc_id, holds the identifier for the association. All notifications for a given association have the same association identifier. For a one-to-one style socket, this field is ignored.

ssf_data: The undelivered message or part of the undelivered message will be present in the ssf_data field. Note that the ssf_info.sinfo_flags field as noted above should be used to determine if a complete message is present or just a piece of the message. Note that only user data is present in this field, any chunk headers or SCTP common headers must be removed by the SCTP stack.

6.1.5. SCTP_SHUTDOWN_EVENT

When a peer sends a SHUTDOWN, SCTP delivers this notification to inform the application that it should cease sending data.

```
struct sctp_shutdown_event {
    uint16_t sse_type;
    uint16_t sse_flags;
    uint32_t sse_length;
    sctp_assoc_t sse_assoc_id;
};
```

sse_type: It should be SCTP_SHUTDOWN_EVENT.

sse_flags: Currently unused.

sse_length: This field is the total length of the notification data, including the notification header. It will generally be sizeof(struct sctp_shutdown_event).

sse_flags: Currently unused.

sse_assoc_id: The sse_assoc_id field holds the identifier for the association. All notifications for a given association have the same association identifier. For a one-to-one style socket, this field is ignored.

6.1.6. SCTP_ADAPTATION_INDICATION

When a peer sends an Adaptation Layer Indication parameter as described in [RFC5061], SCTP delivers this notification to inform the application about the peer's adaptation layer indication.

```
struct sctp_adaptation_event {
  uint16_t sai_type;
  uint16_t sai_flags;
  uint32_t sai_length;
  uint32_t sai_adaptation_ind;
  sctp_assoc_t sai_assoc_id;
};
```

sai_type: It should be SCTP_ADAPTATION_INDICATION.

sai_flags: Currently unused.

sai_length: This field is the total length of the notification data, including the notification header. It will generally be sizeof(struct sctp_adaptation_event).

sai_adaptation_ind: This field holds the bit array sent by the peer in the adaptation layer indication parameter.

sai_assoc_id: The sai_assoc_id field holds the identifier for the association. All notifications for a given association have the same association identifier. For a one-to-one style socket, this field is ignored.

6.1.7. SCTP_PARTIAL_DELIVERY_EVENT

When a receiver is engaged in a partial delivery of a message this notification will be used to indicate various events.

```
struct sctp_pdapi_event {
  uint16_t pdapi_type;
  uint16_t pdapi_flags;
  uint32_t pdapi_length;
  uint32_t pdapi_indication;
  uint32_t pdapi_stream;
  uint32_t pdapi_seq;
  sctp_assoc_t pdapi_assoc_id;
};
```

pdapi_type: It should be SCTP_PARTIAL_DELIVERY_EVENT.

pdapi_flags: Currently unused.

pdapi_length: This field is the total length of the notification data, including the notification header. It will generally be sizeof(struct sctp_pdapi_event).

pdapi_indication: This field holds the indication being sent to the application. Currently there is only one defined value:

SCTP_PARTIAL_DELIVERY_ABORTED: This indicates that the partial delivery of a user message has been aborted. This happens, for example, if an association is aborted while a partial delivery is going on or the user message gets abandoned using PR-SCTP while the partial delivery of this message is going on.

pdapi_stream: This field holds the stream on which the partial delivery event happened.

pdapi_seq: This field holds the stream sequence number which was being partially delivered.

pdapi_assoc_id: The pdapi_assoc_id field holds the identifier for the association. All notifications for a given association have the same association identifier. For a one-to-one style socket this field is ignored.

6.1.8. SCTP_AUTHENTICATION_EVENT

[RFC4895] defines an extension to authenticate SCTP messages. The following notification is used to report different events relating to the use of this extension.

```
struct sctp_authkey_event {
  uint16_t auth_type;
  uint16_t auth_flags;
  uint32_t auth_length;
  uint16_t auth_keynumber;
  uint32_t auth_indication;
  sctp_assoc_t auth_assoc_id;
};
```

auth_type: It should be SCTP_AUTHENTICATION_EVENT.

auth_flags: Currently unused.

auth_length: This field is the total length of the notification data, including the notification header. It will generally be sizeof(struct sctp_authkey_event).

auth_keynumber: This field holds the keynumber for the affected key indicated in the event (depends on auth_indication).

auth_indication: This field holds the error or indication being reported. The following values are currently defined:

SCTP_AUTH_NEW_KEY: This report indicates that a new key has been made active (used for the first time by the peer) and is now the active key. The auth_keynumber field holds the user specified key number.

SCTP_AUTH_NO_AUTH: This report indicates that the peer does not support SCTP AUTH as defined in [RFC4895].

SCTP_AUTH_FREE_KEY: This report indicates that the SCTP implementation will no longer use the key identifier specified in auth_keynumber.

auth_assoc_id: The auth_assoc_id field holds the identifier for the association. All notifications for a given association have the same association identifier. For a one-to-one style socket this field is ignored.

6.1.9. SCTP_SENDER_DRY_EVENT

When the SCTP stack has no more user data to send or retransmit, this notification is given to the user. Also, at the time when a user app subscribes to this event, if there is no data to be sent or retransmit, the stack will immediately send up this notification.

```
struct sctp_sender_dry_event {
  uint16_t sender_dry_type;
  uint16_t sender_dry_flags;
  uint32_t sender_dry_length;
  sctp_assoc_t sender_dry_assoc_id;
};
```

sender_dry_type: It should be SCTP_SENDER_DRY_EVENT.

sender_dry_flags: Currently unused.

sender_dry_length: This field is the total length of the notification data, including the notification header. It will generally be sizeof(struct sctp_sender_dry_event).

sender_dry_assoc_id: The sender_dry_assoc_id field holds the identifier for the association. All notifications for a given association have the same association identifier. For a one-to-one style socket this field is ignored.

6.1.10. SCTP_NOTIFICATIONS_STOPPED_EVENT

SCTP notifications, when subscribed to, are reliable. They are always delivered as long as there is space in the socket receive buffer. However, if an implementation experiences a notification storm, it may run out of socket buffer space. When this occurs it may wish to disable notifications. If the implementation chooses to do this, it will append a final notification SCTP_NOTIFICATIONS_STOPPED_EVENT. This notification is a union

sctp_notification, where only the struct sctp_tlv (see the union above) is used. It only contains this type in the sn_type field, the sn_length field set to the size of an sctp_tlv structure and the sn_flags set to 0. If an application receives this notification, it will need to re-subscribe to any notifications of interest to it, except for the sctp_data_io_event (note that SCTP_EVENTS is deprecated).

An endpoint is automatically subscribed to this event as soon as it is subscribed to any event other than data io events.

6.1.11. SCTP_SEND_FAILED_EVENT

If SCTP cannot deliver a message, it can return back the message as a notification if the SCTP_SEND_FAILED_EVENT event is enabled. The notification has the following format:

```
struct sctp_send_failed_event {
  uint16_t ssfe_type;
  uint16_t ssfe_flags;
  uint32_t ssfe_length;
  uint32_t ssfe_error;
  struct sctp_sndinfo ssfe_info;
  sctp_assoc_t ssfe_assoc_id;
  uint8_t ssfe_data[];
};
```

ssfe_type: It should be SCTP_SEND_FAILED_EVENT.

ssfe_flags: The flag value will take one of the following values:

- SCTP_DATA_UNSENT: Indicates that the data was never put on the wire.
- SCTP_DATA_SENT: Indicates that the data was put on the wire. Note that this does not necessarily mean that the data was (or was not) successfully delivered.

ssfe_length: This field is the total length of the notification data, including the notification header and the payload in ssf_data.

ssfe_error: This value represents the reason why the send failed, and if set, will be an SCTP protocol error code as defined in [RFC4960] Section 3.3.10.

ssfe_info: The ancillary data (struct sctp_sndinfo) used to send the undelivered message. Regardless of if ancillary data is used or not, the ssfe_info.sinfo_flags field indicates if the complete message or only part of the message is returned in ssf_data. If only part of the message is returned, it means that the part which is not present has been sent successfully to the peer.

If the complete message cannot be sent, the SCTP_DATA_NOT_FRAG flags is set in ssfe_info.sinfo_flags. If the first part of the message is sent successfully, the SCTP_DATA_LAST_FRAG is set. This means that the tail end of the message is returned in ssf_data.

ssfe_assoc_id: The ssfe_assoc_id field, ssf_assoc_id, holds the identifier for the association. All notifications for a given association have the same association identifier. For a one-to-one style socket, this field is ignored.

ssfe_data: The undelivered message or part of the undelivered message will be present in the ssf_data field. Note that the ssf_info.sinfo_flags field as noted above should be used to determine if a complete message is present or just a piece of the message. Note that only user data is present in this field, any chunk headers or SCTP common headers must be removed by the SCTP stack.

6.2. Notification Interest Options

6.2.1. SCTP_EVENTS option - DEPRECATED

Please note that this option is deprecated. Use the SCTP_EVENT option described in Section 6.2.2 instead.

To receive SCTP event notifications, an application registers its interest by setting the SCTP_EVENTS socket option. The application then uses recvmsg() to retrieve notifications. A notification is stored in the data part (msg_iov) of the struct msghdr. The socket option uses the following structure:

```
struct sctp_event_subscribe {
  uint8_t sctp_data_io_event;
  uint8_t sctp_association_event;
  uint8_t sctp_address_event;
  uint8_t sctp_send_failure_event;
  uint8_t sctp_peer_error_event;
  uint8_t sctp_shutdown_event;
  uint8_t sctp_partial_delivery_event;
  uint8_t sctp_adaptation_layer_event;
  uint8_t sctp_authentication_event;
  uint8_t sctp_sender_dry_event;
};
```

sctp_data_io_event: Setting this flag to 1 will cause the reception of SCTP_SNDRCV information on a per message basis. The application will need to use the recvmsg() interface so that it can receive the event information contained in the msg_control field. Setting the flag to 0 will disable the reception of the message control information. Note that this is not really a notification and this is stored in the ancillary data (msg_control), not in the data part (msg_iov).

sctp_association_event: Setting this flag to 1 will enable the reception of association event notifications. Setting the flag to 0 will disable association event notifications.

sctp_address_event: Setting this flag to 1 will enable the reception of address event notifications. Setting the flag to 0 will disable address event notifications.

sctp_send_failure_event: Setting this flag to 1 will enable the reception of send failure event notifications. Setting the flag to 0 will disable send failure event notifications.

sctp_peer_error_event: Setting this flag to 1 will enable the reception of peer error event notifications. Setting the flag to 0 will disable peer error event notifications.

sctp_shutdown_event: Setting this flag to 1 will enable the reception of shutdown event notifications. Setting the flag to 0 will disable shutdown event notifications.

sctp_partial_delivery_event: Setting this flag to 1 will enable the reception of partial delivery notifications. Setting the flag to 0 will disable partial delivery event notifications.

sctp_adaptation_layer_event: Setting this flag to 1 will enable the reception of adaptation layer notifications. Setting the flag to 0 will disable adaptation layer event notifications.

sctp_authentication_event: Setting this flag to 1 will enable the reception of authentication layer notifications. Setting the flag to 0 will disable authentication layer event notifications.

sctp_sender_dry_event: Setting this flag to 1 will enable the reception of sender dry notifications. Setting the flag to 0 will disable sender dry event notifications.

An example where an application would like to receive data_io_events and association_events but no others would be as follows:

```
{
  struct sctp_event_subscribe events;

  memset(&events, 0, sizeof(events));

  events.sctp_data_io_event = 1;
  events.sctp_association_event = 1;

  setsockopt(sd, IPPROTO_SCTP, SCTP_EVENTS, &events, sizeof(events));
}
```

Note that for one-to-many style SCTP sockets, the caller of recvmsg() receives ancillary data and notifications for all associations bound to the file descriptor. For one-to-one style SCTP sockets, the caller receives ancillary data and notifications only for the single association bound to the file descriptor.

By default both the one-to-one style and the one-to-many style socket do not subscribe to any notification.

6.2.2. SCTP_EVENT option

The SCTP_EVENTS socket option has one issue for future compatibility. As new features are added the structure (sctp_event_subscribe) must be expanded. This can cause an application binary interface (ABI) issue unless an implementation has added padding at the end of the structure. To avoid this problem, SCTP_EVENTS has been deprecated and a new socket option SCTP_EVENT has taken its place. The option is used with the following structure:

```
struct sctp_event {
        sctp_assoc_t se_assoc_id;
        uint16_t     se_type;
        uint8_t      se_on;
};
```

se_assoc_id: The se_assoc_id field is ignored for one-to-one style sockets. For one-to-many style sockets this field can be a particular association identifier or SCTP_{FUTURE|CURRENT|ALL}_ASSOC.

se_type: The se_type field can be filled with any value that would show up in the respective sn_type field (in the sctp_tlv structure of the notification).

se_on: The se_on field is set to 1 to turn on an event and set to 0 to turn off an event.

To use this option the user fills in this structure and then calls the setsockopt() to turn on or off an individual event. The following is an example use of this option:

```
{
  struct sctp_event event;

  memset(&event, 0, sizeof(event));

  event.se_assoc_id = SCTP_FUTURE_ASSOC;
  event.se_type = SCTP_SENDER_DRY_EVENT;
  event.se_on = 1;
  setsockopt(sd, IPPROTO_SCTP, SCTP_EVENT, &event, sizeof(event));
}
```

By default both the one-to-one style and the one-to-many style socket do not subscribe to any notification.

7. Common Operations for Both Styles

7.1. send(), recv(), sendto(), and recvfrom()

Applications can use send() and sendto() to transmit data to the peer of an SCTP endpoint. recv() and recvfrom() can be used to receive data from the peer.

The function prototypes are

```
ssize_t send(int sd,
             const void *msg,
             size_t len,
             int flags);


ssize_t sendto(int sd,
               const void *msg,
               size_t len,
               int flags,
               const struct sockaddr *to,
               socklen_t tolen);


ssize_t recv(int sd,
             void *buf,
             size_t len,
             int flags);


ssize_t recvfrom(int sd,
                 void *buf,
                 size_t len,
                 int flags,
                 struct sockaddr *from,
                 socklen_t *fromlen);
```

and the arguments are

sd: The socket descriptor of an SCTP endpoint.

msg: The message to be sent.

len: The size of the message or the size of the buffer.

to: One of the peer addresses of the association to be used to send the message.

tolen: The size of the address.

buf: The buffer to store a received message.

from: The buffer to store the peer address used to send the received message.

fromlen: The size of the from address.

flags: (described below).

These calls give access to only basic SCTP protocol features. If either peer in the association uses multiple streams, or sends unordered data, these calls will usually be inadequate, and may deliver the data in unpredictable ways.

SCTP has the concept of multiple streams in one association. The above calls do not allow the caller to specify on which stream a message should be sent. The system uses stream 0 as the default stream for send() and sendto(). recv() and recvfrom() return data from any stream, but the caller can not distinguish the different streams. This may result in data seeming to arrive out of order. Similarly, if a data chunk is sent unordered, recv() and recvfrom() provide no indication.

SCTP is message based. The msg buffer above in send() and sendto() is considered to be a single message. This means that if the caller wants to send a message that is composed by several buffers, the caller needs to combine them before calling send() or sendto(). Alternately, the caller can use sendmsg() to do that without combining them. Sending a message using send() or sendto() is atomic unless explicit EOR marking is enabled on the socket specified by sd. Using sendto() on a non-connected one-to-one style socket for implicit connection setup may or may not work depending on the SCTP implementation. recv() and recvfrom() cannot distinguish message boundaries (i.e. there is no way to observe the MSG_EOR flag to detect partial delivery).

In receiving, if the buffer supplied is not large enough to hold a complete message, the receive call acts like a stream socket and returns as much data as will fit in the buffer.

Note, the send() and recv() calls may not be used for a one-to-many style socket.

Note, if an application calls a send() or sendto() function with no user data the SCTP implementation should reject the request with an appropriate error message. An implementation is not allowed to send a DATA chunk with no user data [RFC4960].

7.2. setsockopt() and getsockopt()

Applications use setsockopt() and getsockopt() to set or retrieve socket options. Socket options are used to change the default behavior of socket calls. They are described in Section 8.

The function prototypes are

```
int getsockopt(int sd,
               int level,
               int optname,
               void *optval,
               socklen_t *optlen);
```

and

```
int setsockopt(int sd,
               int level,
               int optname,
               const void *optval,
               socklen_t optlen);
```

and the arguments are

sd: The socket descriptor.

level: Set to IPPROTO_SCTP for all SCTP options.

optname: The option name.

optval: The buffer to store the value of the option.

optlen: The size of the buffer (or the length of the option returned).

They return 0 on success and -1 in case of an error.

All socket options set on a one-to-one style listening socket also apply to all future accepted sockets. For one-to-many style sockets often a socket option will pass a structure that includes an assoc_id field. This field can be filled with the association identifier of a particular association and unless otherwise specified can be filled with one of the following constants:

SCTP_FUTURE_ASSOC: Specifies that only future associations created after this socket option will be affected by this call.

SCTP_CURRENT_ASSOC: Specifies that only currently existing associations will be affected by this call, future associations will still receive the previous default value.

SCTP_ALL_ASSOC: Specifies that all current and future associations will be affected by this call.

7.3. read() and write()

Applications can use read() and write() to send and receive data to and from a peer. They have the same semantics as send() and recv() except that the flags parameter cannot be used.

7.4. getsockname()

Applications use getsockname() to retrieve the locally-bound socket address of the specified socket. This is especially useful if the caller let SCTP choose a local port. This call is for single homed endpoints. It does not work well with multi-homed endpoints. See Section 9.5 for a multi-homed version of the call.

The function prototype is

```
int getsockname(int sd,
                struct sockaddr *address,
                socklen_t *len);
```

and the arguments are

sd: The socket descriptor to be queried.

address: On return, one locally bound address (chosen by the SCTP stack) is stored in this buffer. If the socket is an IPv4 socket, the address will be IPv4. If the socket is an IPv6 socket, the address will be either an IPv6 or IPv4 address.

len: The caller should set the length of the address here. On return, this is set to the length of the returned address.

It returns 0 on success and -1 in case of an error.

If the actual length of the address is greater than the length of the supplied sockaddr structure, the stored address will be truncated.

If the socket has not been bound to a local name, the value stored in the object pointed to by address is unspecified.

7.5. Implicit Association Setup

The application can begin sending and receiving data using the sendmsg()/recvmsg() or sendto()/recvfrom() calls, without going through any explicit association setup procedures (i.e., no connect()

calls required).

Whenever sendmsg() or sendto() is called and the SCTP stack at the sender finds that no association exists between the sender and the intended receiver (identified by the address passed either in the msg_name field of msghdr structure in the sendmsg() call or the dest_addr field in the sendto() call), the SCTP stack will automatically setup an association to the intended receiver.

Upon the successful association setup an SCTP_COMM_UP notification will be dispatched to the socket at both the sender and receiver side. This notification can be read by the recvmsg() system call (see Section 3.1.4).

Note, if the SCTP stack at the sender side supports bundling, the first user message may be bundled with the COOKIE ECHO message [RFC4960].

When the SCTP stack sets up a new association implicitly, the SCTP_INIT type ancillary data may also be passed along (see Section 5.3.1 for details of the data structures) to change some parameters used in setting up a new association.

If this information is not present in the sendmsg() call, or if the implicit association setup is triggered by a sendto() call, the default association initialization parameters will be used. These default association parameters may be set with respective setsockopt() calls or be left to the system defaults.

Implicit association setup cannot be initiated by send() calls.

8. Socket Options

The following sub-section describes various SCTP level socket options that are common to both styles. SCTP associations can be multi-homed. Therefore, certain option parameters include a sockaddr_storage structure to select which peer address the option should be applied to.

For the one-to-many style sockets, an sctp_assoc_t (association identifier) parameter is used to identify the association instance that the operation affects. So it must be set when using this style.

For the one-to-one style sockets and branched off one-to-many style sockets (see Section 9.2) this association ID parameter is ignored.

Note that socket or IP level options are set or retrieved per socket.

This means that for one-to-many style sockets, the options will be applied to all associations (similar to using SCTP_ALL_ASSOC as the association identifier) belonging to the socket. For one-to-one style, these options will be applied to all peer addresses of the association controlled by the socket. Applications should be careful in setting those options.

For some IP stacks getsockopt() is read-only; so a new interface will be needed when information must be passed both into and out of the SCTP stack. The syntax for sctp_opt_info() is

```
int sctp_opt_info(int sd,
                  sctp_assoc_t id,
                  int opt,
                  void *arg,
                  socklen_t *size);
```

The sctp_opt_info() call is a replacement for getsockopt() only and will not set any options associated with the specified socket. A setsockopt() must be used to set any writeable option.

For one-to-many style sockets, id specifies the association to query. For one-to-one style sockets, id is ignored. For one-to-many sockets, any association identifier in the structure provided as arg is ignored and id takes precedence.

Note that SCTP_CURRENT_ASSOC and SCTP_ALL_ASSOC cannot be used with sctp_opt_info() or in getsockopt() calls. Using them will result in an error (returning -1 and errno set to EINVAL). SCTP_FUTURE_ASSOC can be used to query information for future associations.

The field opt specifies which SCTP socket option to get. It can get any socket option currently supported that requests information (either read/write options or read only) such as:

SCTP_RTOINFO

SCTP_ASSOCINFO

SCTP_PRIMARY_ADDR

SCTP_PEER_ADDR_PARAMS

SCTP_DEFAULT_SEND_PARAM

SCTP_MAX_SEG

SCTP_AUTH_ACTIVE_KEY

SCTP_DELAYED_SACK

SCTP_MAX_BURST

SCTP_CONTEXT

SCTP_EVENT

SCTP_DEFAULT_SNDINFO

SCTP_DEFAULT_PRINFO

SCTP_STATUS

SCTP_GET_PEER_ADDR_INFO

SCTP_PEER_AUTH_CHUNKS

SCTP_LOCAL_AUTH_CHUNKS

The arg field is an option-specific structure buffer provided by the caller. See the rest of this sections subsections for more information on these options and option-specific structures.

sctp_opt_info() returns 0 on success, or on failure returns -1 and sets errno to the appropriate error code.

8.1. Read / Write Options

8.1.1. Retransmission Timeout Parameters (SCTP_RTOINFO)

The protocol parameters used to initialize and limit the retransmission timeout (RTO) are tunable. See [RFC4960] for more information on how these parameters are used in RTO calculation.

The following structure is used to access and modify these parameters:

```
struct sctp_rtoinfo {
  sctp_assoc_t srto_assoc_id;
  uint32_t srto_initial;
  uint32_t srto_max;
  uint32_t srto_min;
};
```

srto_initial: This contains the initial RTO value.

srto_max and srto_min: These contain the maximum and minimum bounds for all RTOs.

srto_assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets the application may fill in an association identifier or SCTP_FUTURE_ASSOC. It is an error to use SCTP_{CURRENT|ALL}_ASSOC in srto_assoc_id.

All times are given in milliseconds. A value of 0, when modifying the parameters, indicates that the current value should not be changed.

To access or modify these parameters, the application should call getsockopt() or setsockopt() respectively with the option name SCTP_RTOINFO.

8.1.2. Association Parameters (SCTP_ASSOCINFO)

This option is used to both examine and set various association and endpoint parameters. See [RFC4960] for more information on how this parameter is used.

The following structure is used to access and modify these parameters:

```
struct sctp_assocparams {
  sctp_assoc_t sasoc_assoc_id;
  uint16_t sasoc_asocmaxrxt;
  uint16_t sasoc_number_peer_destinations;
  uint32_t sasoc_peer_rwnd;
  uint32_t sasoc_local_rwnd;
  uint32_t sasoc_cookie_life;
};
```

sasoc_assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets the application may fill in an association identifier or SCTP_FUTURE_ASSOC. It is an error to use SCTP_{CURRENT|ALL}_ASSOC in sasoc_assoc_id.

sasoc_asocmaxrxt: This contains the maximum retransmission attempts to make for the association.

sasoc_number_peer_destinations: This is the number of destination addresses that the peer has.

sasoc_peer_rwnd: This holds the current value of the peers rwnd (reported in the last SACK) minus any outstanding data (i.e. data in flight).

sasoc_local_rwnd: This holds the last reported rwnd that was sent to the peer.

sasoc_cookie_life: This is the association's cookie life value used when issuing cookies.

The values of the sasoc_peer_rwnd is meaningless when examining endpoint information (i.e. it is only valid when examining information on a specific association).

All time values are given in milliseconds. A value of 0, when modifying the parameters, indicates that the current value should not be changed.

The values of the sasoc_asocmaxrxt and sasoc_cookie_life may be set on either an endpoint or association basis. The rwnd and destination counts (sasoc_number_peer_destinations, sasoc_peer_rwnd, sasoc_local_rwnd) are not settable and any value placed in these is ignored.

To access or modify these parameters, the application should call getsockopt() or setsockopt() respectively with the option name SCTP_ASSOCINFO.

The maximum number of retransmissions before an address is considered unreachable is also tunable, but is address-specific, so it is covered in a separate option. If an application attempts to set the value of the association maximum retransmission parameter to more than the sum of all maximum retransmission parameters, setsockopt() may return an error. The reason for this, from [RFC4960] Section 8.2:

Note: When configuring the SCTP endpoint, the user should avoid having the value of 'Association.Max.Retrans' (sasoc_maxrxt in this option) larger than the summation of the 'Path.Max.Retrans' (see Section 8.1.12 on spp_pathmaxrxt) of all the destination addresses for the remote endpoint. Otherwise, all the destination addresses may become inactive while the endpoint still considers the peer endpoint reachable.

8.1.3. Initialization Parameters (SCTP_INITMSG)

Applications can specify protocol parameters for the default association initialization. The structure used to access and modify

these parameters is defined in Section 5.3.1. The option name argument to setsockopt() and getsockopt() is SCTP_INITMSG.

Setting initialization parameters is effective only on an unconnected socket (for one-to-many style sockets only future associations are affected by the change).

8.1.4. SO_LINGER

An application can use this option to perform the SCTP ABORT primitive. This option affects all associations related to the socket.

The linger option structure is:

```
struct linger {
  int l_onoff;  /* option on/off */
  int l_linger; /* linger time   */
};
```

To enable the option, set l_onoff to 1. If the l_linger value is set to 0, calling close() is the same as the ABORT primitive. If the value is set to a negative value, the setsockopt() call will return an error. If the value is set to a positive value linger_time, the close() can be blocked for at most linger_time. Please note that the time unit is seconds according to POSIX, but might be different on specific platforms. If the graceful shutdown phase does not finish during this period, close() will return but the graceful shutdown phase will continue in the system.

Note, this is a socket level option, not an SCTP level option. When using this option, an application must specify a level of SOL_SOCKET in the call.

8.1.5. SCTP_NODELAY

Turn on/off any Nagle-like algorithm. This means that packets are generally sent as soon as possible and no unnecessary delays are introduced, at the cost of more packets in the network. In particular, not using any Nagle-like algorithm might reduce the bundling of small user messages in cases where this would require an additional delay.

Turning this option on disables any Nagle-like algorithm.

This option expects an integer boolean flag, where a non-zero value turns on the option, and a zero value turns off the option.

8.1.6. SO_RCVBUF

Sets the receive buffer size in octets. For SCTP one-to-one style sockets, this controls the receiver window size. For one-to-many style sockets the meaning is implementation dependent. It might control the receive buffer for each association bound to the socket descriptor or it might control the receive buffer for the whole socket. This option expects an integer.

Note, this is a socket level option, not an SCTP level option. When using this option, an application must specify a level of SOL_SOCKET in the call.

8.1.7. SO_SNDBUF

Sets the send buffer size. For SCTP one-to-one style sockets, this controls the amount of data SCTP may have waiting in internal buffers to be sent. This option therefore bounds the maximum size of data that can be sent in a single send call. For one-to-many style sockets, the effect is the same, except that it applies to one or all associations (see Section 3.3) bound to the socket descriptor used in the setsockopt() or getsockopt() call. The option applies to each association's window size separately. This option expects an integer.

Note, this is a socket level option, not an SCTP level option. When using this option, an application must specify a level of SOL_SOCKET in the call.

8.1.8. Automatic Close of Associations (SCTP_AUTOCLOSE)

This socket option is applicable to the one-to-many style socket only. When set it will cause associations that are idle for more than the specified number of seconds to automatically close using the graceful shutdown procedure. An association being idle is defined as an association that has not sent or received user data. The special value of '0' indicates that no automatic close of any association should be performed, this is the default value. This option expects an integer defining the number of seconds of idle time before an association is closed.

An application using this option should enable receiving the association change notification. This is the only mechanism an application is informed about the closing of an association. After an association is closed, the association identifier assigned to it can be reused. An application should be aware of this to avoid the possible problem of sending data to an incorrect peer endpoint.

8.1.9. Set Primary Address (SCTP_PRIMARY_ADDR)

Requests that the local SCTP stack uses the enclosed peer address as the association's primary. The enclosed address must be one of the association peer's addresses.

The following structure is used to make a set peer primary request:

```
struct sctp_setprim {
  sctp_assoc_t ssp_assoc_id;
  struct sockaddr_storage ssp_addr;
};
```

ssp_addr: The address to set as primary. No wildcard address is allowed.

ssp_assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets it identifies the association for this request. Note that the special sctp_assoc_t SCTP_{FUTURE|ALL|CURRENT}_ASSOC are not allowed.

8.1.10. Set Adaptation Layer Indicator (SCTP_ADAPTATION_LAYER)

Requests that the local endpoint set the specified Adaptation Layer Indication parameter for all future INIT and INIT-ACK exchanges.

The following structure is used to access and modify this parameter:

```
struct sctp_setadaptation {
  uint32_t   ssb_adaptation_ind;
};
```

ssb_adaptation_ind: The adaptation layer indicator that will be included in any outgoing Adaptation Layer Indication parameter.

8.1.11. Enable/Disable Message Fragmentation (SCTP_DISABLE_FRAGMENTS)

This option is a on/off flag and is passed as an integer where a non-zero is on and a zero is off. If enabled no SCTP message fragmentation will be performed. The effect of enabling this option are that if a message being sent exceeds the current PMTU size, the message will not be sent and instead an error will be indicated to the user. If this option is disabled (the default) then a message exceeding the size of the PMTU will be fragmented and reassembled by the peer.

8.1.12. Peer Address Parameters (SCTP_PEER_ADDR_PARAMS)

Applications can enable or disable heartbeats for any peer address of an association, modify an address's heartbeat interval, force a heartbeat to be sent immediately, and adjust the address's maximum number of retransmissions sent before an address is considered unreachable.

The following structure is used to access and modify an address's parameters:

```
struct sctp_paddrparams {
  sctp_assoc_t spp_assoc_id;
  struct sockaddr_storage spp_address;
  uint32_t spp_hbinterval;
  uint16_t spp_pathmaxrxt;
  uint32_t spp_pathmtu;
  uint32_t spp_flags;
  uint32_t spp_ipv6_flowlabel;
  uint8_t spp_dscp;
};
```

spp_assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets the application may fill in an association identifier or SCTP_FUTURE_ASSOC for this query. It is an error to use SCTP_{CURRENT|ALL}_ASSOC in spp_assoc_id.

spp_address: This specifies which address is of interest. If a wildcard address is provided it applies to all current and future paths.

spp_hbinterval: This contains the value of the heartbeat interval, in milliseconds (HB.Interval in [RFC4960]). Note that unless the spp_flag is set to SPP_HB_ENABLE the value of this field is ignored. Note also that a value of zero indicates the current setting should be left unchanged. To set an actual value of zero the use of the flag SPP_HB_TIME_IS_ZERO should be used. Even when it is set to 0, it does not mean that SCTP will continuously send out heartbeat since the actual interval also includes the current RTO and jitter (see Section 8.3 in [RFC4960]).

spp_pathmaxrxt: This contains the maximum number of retransmissions before this address shall be considered unreachable. Note that a value of zero indicates the current setting should be left unchanged.

spp_pathmtu: The current path MTU of the peer address. It is the number of bytes available in an SCTP packet for chunks. Providing a value of 0 does not change the current setting. If a positive value is provided and SPP_PMTUD_DISABLE is set in the spp_flags, the given value is used as the path MTU. If SPP_PMTUD_ENABLE is set in the spp_flags, the spp_pathmtu field is ignored.

spp_ipv6_flowlabel: This field is used in conjunction with the SPP_IPV6_FLOWLABEL flag and contains the IPv6 flowlabel. The 20 least significant bits are used for the flowlabel. This setting has precedence over any IPv6 layer setting.

spp_dscp: This field is used in conjunction with the SPP_DSCP flag and contains the Differentiated Services Code Point (DSCP). The 6 most significant bits are used for the DSCP. This setting has precedence over any IPv4 or IPv6 layer setting.

spp_flags: These flags are used to control various features on an association. The flag field is a bit mask which may contain zero or more of the following options:

SPP_HB_ENABLE: Enable heartbeats on the specified address.

SPP_HB_DISABLE: Disable heartbeats on the specified address. Note that SPP_HB_ENABLE and SPP_HB_DISABLE are mutually exclusive, only one of these two should be specified. Enabling both fields will have undetermined results.

SPP_HB_DEMAND: Request a user initiated heartbeat to be made immediately. This must not be used in conjunction with a wildcard address.

SPP_HB_TIME_IS_ZERO: Specifies that the time for heartbeat delay is to be set to the value of 0 milliseconds.

SPP_PMTUD_ENABLE: This field will enable PMTU discovery upon the specified address.

SPP_PMTUD_DISABLE: This field will disable PMTU discovery upon the specified address. Note that if the address field is empty then all addresses on the association are affected. Note also that SPP_PMTUD_ENABLE and SPP_PMTUD_DISABLE are mutually exclusive. Enabling both will have undetermined results.

SPP_IPV6_FLOWLABEL: Setting this flag enables the setting of the IPV6 flowlabel value. The value is contained in the spp_ipv6_flowlabel field.

Upon retrieval, this flag will be set to indicate that the spp_ipv6_flowlabel field has a valid value returned. If a specific destination address is set (in the spp_address field), then the value returned is that of the address. If just an association is specified (and no address), then the association's default flowlabel is returned. If neither an association nor a destination is specified, then the socket's default flowlabel is returned. For non IPv6 sockets, this flag will be left cleared.

SPP_DSCP: Setting this flag enables the setting of the DSCP value associated with either the association or a specific address. The value is obtained in the spp_dscp field.

Upon retrieval, this flag will be set to indicate that the spp_dscp field has a valid value returned. If a specific destination address is set when called (in the spp_address field) then that specific destination address' DSCP value is returned. If just an association is specified then the association default DSCP is returned. If neither an association nor a destination is specified, then the sockets default DSCP is returned.

Please note that changing the flowlabel or DSCP value will affect all packets sent by the SCTP stack after setting these parameters. The flowlabel might also be set via the sin6_flowinfo field of the sockaddr_in6 structure.

8.1.13. Set Default Send Parameters (SCTP_DEFAULT_SEND_PARAM) - DEPRECATED

Please note that this options is deprecated. Section 8.1.31 should be used instead.

Applications that wish to use the sendto() system call may wish to specify a default set of parameters that would normally be supplied through the inclusion of ancillary data. This socket option allows such an application to set the default sctp_sndrcvinfo structure. The application that wishes to use this socket option simply passes the sctp_sndrcvinfo structure defined in Section 5.3.2 to this call. The input parameters accepted by this call include sinfo_stream, sinfo_flags, sinfo_ppid, sinfo_context, and sinfo_timetolive. The sinfo_flags is composed of a bitwise OR of SCTP_UNORDERED, SCTP_EOF, and SCTP_SENDALL. The sinfo_assoc_id field specifies the association to apply the parameters to. For a one-to-many style socket any of the predefined constants are also allowed in this field. The field is ignored on the one-to-one style.

8.1.14. Set Notification and Ancillary Events (SCTP_EVENTS) - DEPRECATED

This socket option is used to specify various notifications and ancillary data the user wishes to receive. Please see Section 6.2.1 for a full description of this option and its usage. Note that this option is considered deprecated and present for backward compatibility. New applications should use the SCTP_EVENT option. See Section 6.2.2 for a full description of that option as well.

8.1.15. Set/Clear IPv4 Mapped Addresses (SCTP_I_WANT_MAPPED_V4_ADDR)

This socket option is a boolean flag which turns on or off the mapping of IPv4 addresses. If this option is turned on, then IPv4 addresses will be mapped to V6 representation. If this option is turned off, then no mapping will be done of V4 addresses and a user will receive both PF_INET6 and PF_INET type addresses on the socket. See [RFC3542] for more details on mapped V6 addresses.

If this socket option is used on a socket of type PF_INET an error is returned.

By default this option is turned off and expects an integer to be passed where a non-zero value turns on the option and a zero value turns off the option.

8.1.16. Get or Set the Maximum Fragmentation Size (SCTP_MAXSEG)

This option will get or set the maximum size to put in any outgoing SCTP DATA chunk. If a message is larger than this size it will be fragmented by SCTP into the specified size. Note that the underlying SCTP implementation may fragment into smaller sized chunks when the PMTU of the underlying association is smaller than the value set by the user. The default value for this option is '0' which indicates the user is not limiting fragmentation and only the PMTU will affect SCTP's choice of DATA chunk size. Note also that values set larger than the maximum size of an IP datagram will effectively let SCTP control fragmentation (i.e. the same as setting this option to 0).

The following structure is used to access and modify this parameter:

```
struct sctp_assoc_value {
  sctp_assoc_t assoc_id;
  uint32_t assoc_value;
};
```

assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets this parameter indicates which association the user is performing an action upon. It is an error to use SCTP_{CURRENT|ALL}_ASSOC in assoc_id.

assoc_value: This parameter specifies the maximum size in bytes.

8.1.17. Get or Set the List of Supported HMAC Identifiers (SCTP_HMAC_IDENT)

This option gets or sets the list of HMAC algorithms that the local endpoint requires the peer to use.

The following structure is used to get or set these identifiers:

```
struct sctp_hmacalgo {
  uint32_t shmac_number_of_idents;
  uint16_t shmac_idents[];
};
```

shmac_number_of_idents: This field gives the number of elements present in the array shmac_idents.

shmac_idents: This parameter contains an array of HMAC identifiers that the local endpoint is requesting the peer to use, in priority order. The following identifiers are valid:

* SCTP_AUTH_HMAC_ID_SHA1

* SCTP_AUTH_HMAC_ID_SHA256

Note that the list supplied must include SCTP_AUTH_HMAC_ID_SHA1 and may include any of the other values in its preferred order (lowest list position has the highest preference in algorithm selection). Note also that the lack of SCTP_AUTH_HMAC_ID_SHA1, or the inclusion of an unknown HMAC identifier (including optional identifiers unknown to the implementation) will cause the set option to fail and return an error.

8.1.18. Get or Set the Active Shared Key (SCTP_AUTH_ACTIVE_KEY)

This option will get or set the active shared key to be used to build the association shared key.

The following structure is used to access and modify these parameters:

```
struct sctp_authkeyid {
  sctp_assoc_t scact_assoc_id;
  uint16_t scact_keynumber;
};
```

scact_assoc_id: This parameter sets the active key of the specified association. The special SCTP_{FUTURE|CURRENT|ALL}_ASSOC can be used. For one-to-one sockets, this parameter is ignored. Note, however, that this option will set the active key on the association if the socket is connected, otherwise this will set the default active key for the endpoint.

scact_keynumber: This parameter is the shared key identifier which the application is requesting to become the active shared key to be used for sending authenticated chunks. The key identifier must correspond to an existing shared key. Note that shared key identifier '0' defaults to a null key.

When used with setsockopt() the SCTP implementation must use the indicated shared key identifier for all messages being given to an SCTP implementation via a send call after the setsockopt() call until changed again. Therefore, the SCTP implementation must not bundle user messages which should be authenticated using different shared key identifiers.

Initially the key with key identifier 0 is the active key.

8.1.19. Get or Set Delayed SACK Timer (SCTP_DELAYED_SACK)

This option will affect the way delayed sacks are performed. This option allows the application to get or set the delayed sack time, in milliseconds. It also allows changing the delayed sack frequency. Changing the frequency to 1 disables the delayed sack algorithm. Note that if sack_delay or sack_freq are 0 when setting this option, the current values will remain unchanged.

The following structure is used to access and modify these parameters:

```
struct sctp_sack_info {
  sctp_assoc_t sack_assoc_id;
  uint32_t sack_delay;
  uint32_t sack_freq;
};
```

sack_assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets this parameter indicates which association the user is performing an action upon. The special SCTP_{FUTURE|CURRENT|ALL}_ASSOC can also be used.

sack_delay: This parameter contains the number of milliseconds that the user is requesting the delayed SACK timer to be set to. Note that this value is defined in the standard to be between 200 and 500 milliseconds.

sack_freq: This parameter contains the number of packets that must be received before a sack is sent without waiting for the delay timer to expire. The default value is 2, setting this value to 1 will disable the delayed sack algorithm.

8.1.20. Get or Set Fragmented Interleave (SCTP_FRAGMENT_INTERLEAVE)

Fragmented interleave controls how the presentation of messages occurs for the message receiver. There are three levels of fragment interleave defined. Two of the levels affect the one-to-one model, while the one-to-many model is affected by all three levels.

This option takes an integer value. It can be set to a value of 0, 1 or 2. Attempting to set this level to other values will return an error.

Setting the three levels provides the following receiver interactions:

level 0: Prevents the interleaving of any messages. This means that when a partial delivery begins, no other messages will be received except the message being partially delivered. If another message arrives on a different stream (or association) that could be delivered, it will be blocked waiting for the user to read all of the partially delivered message.

level 1: Allows interleaving of messages that are from different associations. For the one-to-one model, level 0 and level 1 thus have the same meaning since a one-to-one socket always receives messages from the same association. Note that setting the one-to-many model to this level may cause multiple partial deliveries from different associations but for any given association, only one message will be delivered until all parts of a message have been delivered. This means that one large message, being read with an association identifier of "X", will block other messages from association "X" from being delivered.

level 2: Allows complete interleaving of messages. This level requires that the sender carefully observes not only the peer association identifier (or address) but must also pay careful attention to the stream number. With this option enabled a partially delivered message may begin being delivered for association "X" stream "Y" and the next subsequent receive may return a message from association "X" stream "Z". Note that no other messages would be delivered for association "X" stream "Y" until all of stream "Y"'s partially delivered message was read. Note that this option also affects the one-to-one model. Also note that for the one-to-many model not only another stream's message from the same association may be delivered upon the next receive, some other association's message may be delivered upon the next receive.

An implementation should default the one-to-many model to level 1. The reason for this is that otherwise it is possible that a peer could begin sending a partial message and thus block all other peers from sending data. However a setting of level 2 requires the application to not only be aware of the association (via the association identifier or peer's address) but also the stream number. The stream number is not present unless the user has subscribed to the sctp_data_io_event (see Section 6.2), which is deprecated, or has enabled the SCTP_RECVRCVINFO socket option (see Section 8.1.29). This is also why we recommend that the one-to-one model be defaulted to level 0 (level 1 for the one-to-one model has no effect). Note that an implementation should return an error if an application attempts to set the level to 2 and has not subscribed to the sctp_data_io_event event, which is deprecated, or has enabled the SCTP_RECVRCVINFO socket option.

For applications that have subscribed to events, those events appear in the normal socket buffer data stream. This means that unless the user has set the fragmentation interleave level to 0, notifications may also be interleaved with partially delivered messages.

8.1.21. Set or Get the SCTP Partial Delivery Point (SCTP_PARTIAL_DELIVERY_POINT)

This option will set or get the SCTP partial delivery point. This point is the size of a message where the partial delivery API will be invoked to help free up rwnd space for the peer. Setting this to a lower value will cause partial deliveries to happen more often. This option expects an integer that sets or gets the partial delivery point in bytes. Note also that the call will fail if the user attempts to set this value larger than the socket receive buffer size.

Note that any single message having a length smaller than or equal to the SCTP partial delivery point will be delivered in one single read call as long as the user provided buffer is large enough to hold the message.

8.1.22. Set or Get the Use of Extended Receive Info (SCTP_USE_EXT_RCVINFO) - DEPRECATED

This option will enable or disable the use of the extended version of the sctp_sndrcvinfo structure. If this option is disabled, then the normal sctp_sndrcvinfo structure is returned in all receive message calls. If this option is enabled then the sctp_extrcvinfo structure is returned in all receive message calls. The default is off.

Note that the sctp_extrcvinfo structure is never used in any send call.

This option is present for compatibility with older applications and is deprecated. Future applications should use SCTP_NXTINFO to retrieve this same information via ancillary data.

8.1.23. Set or Get the Auto ASCONF Flag (SCTP_AUTO_ASCONF)

This option will enable or disable the use of the automatic generation of ASCONF chunks to add and delete addresses to an existing association. Note that this option has two caveats namely: a) it only affects sockets that are bound to all addresses available to the SCTP stack, and b) the system administrator may have an overriding control that turns the ASCONF feature off no matter what setting the socket option may have.

This option expects an integer boolean flag, where a non-zero value turns on the option, and a zero value turns off the option.

8.1.24. Set or Get the Maximum Burst (SCTP_MAX_BURST)

This option will allow a user to change the maximum burst of packets that can be emitted by this association. Note that the default value is 4, and some implementations may restrict this setting so that it can only be lowered to positive values.

To set or get this option the user fills in the following structure:

```
struct sctp_assoc_value {
  sctp_assoc_t assoc_id;
  uint32_t assoc_value;
};
```

assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets this parameter indicates which association the user is performing an action upon. The special SCTP_{FUTURE|CURRENT|ALL}_ASSOC can also be used.

assoc_value: This parameter contains the maximum burst. Setting the value to 0 disables burst mitigation.

8.1.25. Set or Get the Default Context (SCTP_CONTEXT)

The context field in the sctp_sndrcvinfo structure is normally only used when a failed message is retrieved holding the value that was sent down on the actual send call. This option allows the setting of a default context on an association basis that will be received on reading messages from the peer. This is especially helpful in the one-to-many model for an application to keep some reference to an internal state machine that is processing messages on the association. Note that the setting of this value only affects received messages from the peer and does not affect the value that is saved with outbound messages.

To set or get this option the user fills in the following structure:

```
struct sctp_assoc_value {
  sctp_assoc_t assoc_id;
  uint32_t assoc_value;
};
```

assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets this parameter indicates which association the user is performing an action upon. The special SCTP_{FUTURE|CURRENT|ALL}_ASSOC can also be used.

assoc_value: This parameter contains the context.

8.1.26. Enable or Disable Explicit EOR Marking (SCTP_EXPLICIT_EOR)

This boolean flag is used to enable or disable explicit end of record (EOR) marking. When this option is enabled, a user may make multiple send system calls to send a record and must indicate that they are finished sending a particular record by including the SCTP_EOR flag. If this boolean flag is disabled then each individual send system call is considered to have an SCTP_EOR indicator set on it implicitly without the user having to explicitly add this flag. The default is off.

This option expects an integer boolean flag, where a non-zero value turns on the option, and a zero value turns off the option.

8.1.27. Enable SCTP Port Reusage (SCTP_REUSE_PORT)

This option only supports one-to-one style SCTP sockets. If used on a one-to-many style SCTP socket an error is indicated.

This option expects an integer boolean flag, where a non-zero value turns on the option, and a zero value turns off the option.

This socket option must not be used after calling bind() or sctp_bindx() for a one-to-one style SCTP socket. If using bind() or sctp_bindx() on a socket with the SCTP_REUSE_PORT option, all other SCTP sockets bound to the same port must have set the SCTP_REUSE_PORT. Calling bind() or sctp_bindx() for a socket without having set the SCTP_REUSE_PORT option will fail if there are other sockets bound to the same port. At most one socket being bound to the same port may be listening.

It should be noted that the behavior of the socket level socket option to reuse ports and/or addresses for SCTP sockets is unspecified.

8.1.28. Set Notification Event (SCTP_EVENT)

This socket option is used to set a specific notification option. Please see Section 6.2.2 for a full description of this option and its usage.

8.1.29. Enable or Disable the Delivery of SCTP_RCVINFO as Ancillary Data (SCTP_RECVRCVINFO)

Setting this option specifies that SCTP_RCVINFO defined in Section 5.3.5 is returned as ancillary data by recvmsg().

This option expects an integer boolean flag, where a non-zero value turns on the option, and a zero value turns off the option.

8.1.30. Enable or Disable the Delivery of SCTP_NXTINFO as Ancillary Data (SCTP_RECVNXTINFO)

Setting this option specifies that SCTP_NXTINFO defined in Section 5.3.6 is returned as ancillary data by recvmsg().

This option expects an integer boolean flag, where a non-zero value turns on the option, and a zero value turns off the option.

8.1.31. Set Default Send Parameters (SCTP_DEFAULT_SNDINFO)

Applications that wish to use the sendto() system call may wish to specify a default set of parameters that would normally be supplied through the inclusion of ancillary data. This socket option allows such an application to set the default sctp_sndinfo structure. The application that wishes to use this socket option simply passes the sctp_sndinfo structure defined in Section 5.3.4 to this call. The input parameters accepted by this call include snd_sid, snd_flags, snd_ppid, snd_context. The snd_flags is composed of a bitwise OR of SCTP_UNORDERED, SCTP_EOF, and SCTP_SENDALL. The snd_assoc_id field specifies the association to apply the parameters to. For a one-to-many style socket any of the predefined constants are also allowed in this field. The field is ignored on the one-to-one style.

8.1.32. Set Default PR-SCTP Parameters (SCTP_DEFAULT_PRINFO)

This option sets and gets the default parameters for PR-SCTP. They can be overwritten by specific information provided in send calls.

The following structure is used to access and modify these parameters:

```
struct sctp_default_prinfo {
  uint16_t pr_policy;
  uint32_t pr_value;
  sctp_assoc_t pr_assoc_id;
};
```

pr_policy: Same as described in Section 5.3.7.

pr_value: Same as described in Section 5.3.7.

pr_assoc_id: This field is ignored for one-to-one style sockets. For one-to-many style sockets pr_assoc_id can be a particular association identifier or SCTP_{FUTURE|CURRENT|ALL}_ASSOC.

8.2. Read-Only Options

The options defined in this subsection are read-only. Using this option in a setsockopt() call will result in an error indicating EOPNOTSUPP.

8.2.1. Association Status (SCTP_STATUS)

Applications can retrieve current status information about an association, including association state, peer receiver window size, number of unacknowledged data chunks, and number of data chunks

pending receipt. This information is read-only.

The following structure is used to access this information:

```
struct sctp_status {
  sctp_assoc_t sstat_assoc_id;
  int32_t sstat_state;
  uint32_t sstat_rwnd;
  uint16_t sstat_unackdata;
  uint16_t sstat_penddata;
  uint16_t sstat_instrms;
  uint16_t sstat_outstrms;
  uint32_t sstat_fragmentation_point;
  struct sctp_paddrinfo sstat_primary;
};
```

sstat_assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets it holds the identifier for the association. All notifications for a given association have the same association identifier. The special SCTP_{FUTURE|CURRENT|ALL}_ASSOC cannot be used.

sstat_state: This contains the association's current state, i.e. one of the following values:

* SCTP_CLOSED
* SCTP_BOUND
* SCTP_LISTEN
* SCTP_COOKIE_WAIT
* SCTP_COOKIE_ECHOED
* SCTP_ESTABLISHED
* SCTP_SHUTDOWN_PENDING
* SCTP_SHUTDOWN_SENT
* SCTP_SHUTDOWN_RECEIVED
* SCTP_SHUTDOWN_ACK_SENT

sstat_rwnd: This contains the association peer's current receiver window size.

sstat_unackdata: This is the number of unacknowledged data chunks.

sstat_penddata: This is the number of data chunks pending receipt.

sstat_instrms: The number of streams that the peer will be using outbound.

sstat_outstrms: The number of streams that the endpoint is allowed to use outbound.

sstat_fragmentation_point: The size at which SCTP fragmentation will occur.

sstat_primary: This is information on the current primary peer address.

To access these status values, the application calls getsockopt() with the option name SCTP_STATUS.

8.2.2. Peer Address Information (SCTP_GET_PEER_ADDR_INFO)

Applications can retrieve information about a specific peer address of an association, including its reachability state, congestion window, and retransmission timer values. This information is read-only.

The following structure is used to access this information:

```
struct sctp_paddrinfo {
  sctp_assoc_t spinfo_assoc_id;
  struct sockaddr_storage spinfo_address;
  int32_t spinfo_state;
  uint32_t spinfo_cwnd;
  uint32_t spinfo_srtt;
  uint32_t spinfo_rto;
  uint32_t spinfo_mtu;
};
```

spinfo_assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets the following applies: This field may be filled by the application, if so, this field will have priority in looking up the association instead of using the address specified in spinfo_address. Note that if the address does not belong to the association specified then this call will fail. If the application does not fill in the spinfo_assoc_id,

then the address will be used to lookup the association and on return this field will have the valid association identifier. In other words, this call can be used to translate an address into an association identifier. Note that the predefined constants are not allowed on this option.

spinfo_address: This is filled by the application, and contains the peer address of interest.

spinfo_state: This contains the peer address' state:

SCTP_UNCONFIRMED: The initial state of a peer address.

SCTP_ACTIVE: The state is entered the first time after path verification. It can also be entered if the state is SCTP_INACTIVE and the path supervision detects that the peer address is reachable again.

SCTP_INACTIVE: This state is entered whenever a path failure is detected.

spinfo_cwnd: This contains the peer address' current congestion window.

spinfo_srtt: This contains the peer address' current smoothed round-trip time calculation in milliseconds.

spinfo_rto: This contains the peer address' current retransmission timeout value in milliseconds.

spinfo_mtu: The current path MTU of the peer address. It is the number of bytes available in an SCTP packet for chunks.

8.2.3. Get the List of Chunks the Peer Requires to be Authenticated (SCTP_PEER_AUTH_CHUNKS)

This option gets a list of chunk types (see [RFC4960]) for a specified association that the peer requires to be received authenticated only.

The following structure is used to access these parameters:

```
struct sctp_authchunks {
  sctp_assoc_t gauth_assoc_id;
  uint32_t gauth_number_of_chunks
  uint8_t gauth_chunks[];
};
```

gauth_assoc_id: This parameter indicates for which association the user is requesting the list of peer authenticated chunks. For one-to-one sockets, this parameter is ignored. Note that the predefined constants are not allowed with this option.

gauth_number_of_chunks: This parameter gives the number of elements in the array gauth_chunks.

gauth_chunks: This parameter contains an array of chunk types that the peer is requesting to be authenticated. If the passed in buffer size is not large enough to hold the list of chunk types, ENOBUFS is returned.

8.2.4. Get the List of Chunks the Local Endpoint Requires to be Authenticated (SCTP_LOCAL_AUTH_CHUNKS)

This option gets a list of chunk types (see [RFC4960]) for a specified association that the local endpoint requires to be received authenticated only.

The following structure is used to access these parameters:

```
struct sctp_authchunks {
  sctp_assoc_t gauth_assoc_id;
  uint32_t gauth_number_of_chunks;
  uint8_t gauth_chunks[];
};
```

gauth_assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets the application may fill in an association identifier or SCTP_FUTURE_ASSOC. It is an error to use SCTP_{CURRENT|ALL}_ASSOC in gauth_assoc_id.

gauth_number_of_chunks: This parameter gives the number of elements in the array gauth_chunks.

gauth_chunks: This parameter contains an array of chunk types that the local endpoint is requesting to be authenticated. If the passed in buffer is not large enough to hold the list of chunk types, ENOBUFS is returned.

8.2.5. Get the Current Number of Associations (SCTP_GET_ASSOC_NUMBER)

This option gets the current number of associations that are attached to a one-to-many style socket. The option value is an uint32_t. Note that this number is only a snap shot. This means that the number of associations may have changed when the caller gets back the option result.

For a one-to-one style socket, this socket option results in an error.

8.2.6. Get the Current Identifiers of Associations (SCTP_GET_ASSOC_ID_LIST)

This option gets the current list of SCTP association identifiers of the SCTP associations handled by a one-to-many style socket.

The option value has the structure

```
struct sctp_assoc_ids {
  uint32_t gaids_number_of_ids;
  sctp_assoc_t gaids_assoc_id[];
};
```

The caller must provide a large enough buffer to hold all association identifiers. If the buffer is too small, an error must be returned. The user can use the SCTP_GET_ASSOC_NUMBER socket option to get an idea how large the buffer has to be. gaids_number_of_ids gives the number of elements in the array gaids_assoc_id. Note also that some or all of sctp_assoc_t returned in the array may become invalid by the time the caller gets back the result.

For a one-to-one style socket, this socket option results in an error.

8.3. Write-Only Options

The options defined in this subsection are write-only. Using this option in a getsockopt() or sctp_opt_info() call will result in an error indicating EOPNOTSUPP.

8.3.1. Set Peer Primary Address (SCTP_SET_PEER_PRIMARY_ADDR)

Requests that the peer marks the enclosed address as the association primary (see [RFC5061]). The enclosed address must be one of the association’s locally bound addresses.

The following structure is used to make a set peer primary request:

```
struct sctp_setpeerprim {
  sctp_assoc_t sspp_assoc_id;
  struct sockaddr_storage sspp_addr;
};
```

sspp_assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets it identifies the association for this request. Note that the predefined constants are not allowed on this option.

sspp_addr: The address to set as primary.

8.3.2. Add a Chunk that must be Authenticated (SCTP_AUTH_CHUNK)

This set option adds a chunk type that the user is requesting to be received only in an authenticated way. Changes to the list of chunks will only affect future associations on the socket.

The following structure is used to add a chunk:

```
struct sctp_authchunk {
  uint8_t sauth_chunk;
};
```

sauth_chunk: This parameter contains a chunk type that the user is requesting to be authenticated.

The chunk types for INIT, INIT-ACK, SHUTDOWN-COMPLETE, and AUTH chunks must not be used. If they are used, an error must be returned. The usage of this option enables SCTP AUTH in cases where it is not required by other means (for example the use of dynamic address reconfiguration).

8.3.3. Set a Shared Key (SCTP_AUTH_KEY)

This option will set a shared secret key that is used to build an association shared key.

The following structure is used to access and modify these parameters:

```
struct sctp_authkey {
  sctp_assoc_t sca_assoc_id;
  uint16_t sca_keynumber;
  uint16_t sca_keylength;
  uint8_t sca_key[];
};
```

sca_assoc_id: This parameter indicates what association the shared key is being set upon. The special SCTP_{FUTURE|CURRENT|ALL}_ASSOC can be used. For one-to-one sockets, this parameter is ignored. Note, however on one to one sockets, that this option will set a key on the association if the socket is connected,

otherwise this will set a key on the endpoint.

sca_keynumber: This parameter is the shared key identifier by which the application will refer to this shared key. If a key of the specified index already exists, then this new key will replace the old existing key. Note that shared key identifier '0' defaults to a null key.

sca_keylength: This parameter is the length of the array sca_key.

sca_key: This parameter contains an array of bytes that is to be used by the endpoint (or association) as the shared secret key. Note, if the length of this field is zero, a null key is set.

8.3.4. Deactivate a Shared Key (SCTP_AUTH_DEACTIVATE_KEY)

This set option indicates that the application will no longer send user messages using the indicated key identifier.

```
struct sctp_authkeyid {
  sctp_assoc_t scact_assoc_id;
  uint16_t scact_keynumber;
};
```

scact_assoc_id: This parameter indicates which association the shared key identifier is being deleted from. The special SCTP_{FUTURE|CURRENT|ALL}_ASSOC can be used. For one-to-one sockets, this parameter is ignored. Note, however, that this option will deactivate the key from the association if the socket is connected, otherwise this will deactivate the key from the endpoint.

scact_keynumber: This parameter is the shared key identifier which the application is requesting to be deactivated. The key identifier must correspond to an existing shared key. Note if this parameter is zero, use of the null key identifier '0' is deactivated on the endpoint and/or association.

The currently active key cannot be deactivated.

8.3.5. Delete a Shared Key (SCTP_AUTH_DELETE_KEY)

This set option will delete a shared secret key which has been deactivated of an SCTP association.

```
struct sctp_authkeyid {
  sctp_assoc_t scact_assoc_id;
  uint16_t scact_keynumber;
```

};

scact_assoc_id: This parameter indicates which association the shared key identifier is being deleted from. The special SCTP_{FUTURE|CURRENT|ALL}_ASSOC can be used. For one-to-one sockets, this parameter is ignored. Note, however, that this option will delete the key from the association if the socket is connected, otherwise this will delete the key from the endpoint.

scact_keynumber: This parameter is the shared key identifier which the application is requesting to be deleted. The key identifier must correspond to an existing shared key and must not be in use for any packet being sent by the SCTP implementation. This means in particular, that it must be deactivated first. Note if this parameter is zero, use of the null key identifier '0' is deleted from the endpoint and/or association.

Only deactivated keys that are no longer used by an association can be deleted.

9. New Functions

Depending on the system, the following interface can be implemented as a system call or library function.

9.1. sctp_bindx()

This function allows the user to bind a specific subset of addresses or, if the SCTP extension described in [RFC5061] is supported, add or delete specific addresses.

The function prototype is

```
int sctp_bindx(int sd,
               struct sockaddr *addrs,
               int addrcnt,
               int flags);
```

If sd is an IPv4 socket, the addresses passed must be IPv4 addresses. If the sd is an IPv6 socket, the addresses passed can either be IPv4 or IPv6 addresses.

A single address may be specified as INADDR_ANY or IN6ADDR_ANY, see Section 3.1.2 for this usage.

addrs is a pointer to an array of one or more socket addresses. Each address is contained in its appropriate structure. For an IPv6

socket, an array of sockaddr_in6 is used. For a IPv4 socket, an array of sockaddr_in is used. The caller specifies the number of addresses in the array with addrcnt. Note that the wildcard addresses cannot be used in combination with non wildcard addresses on a socket with this function, doing so will result in an error.

On success, sctp_bindx() returns 0. On failure, sctp_bindx() returns -1 and sets errno to the appropriate error code.

For SCTP, the port given in each socket address must be the same, or sctp_bindx() will fail, setting errno to EINVAL.

The flags parameter is formed from the bitwise OR of zero or more of the following currently defined flags:

- SCTP_BINDX_ADD_ADDR

- SCTP_BINDX_REM_ADDR

SCTP_BINDX_ADD_ADDR directs SCTP to add the given addresses to the socket (i.e. endpoint), and SCTP_BINDX_REM_ADDR directs SCTP to remove the given addresses from the socket. The two flags are mutually exclusive; if both are given, sctp_bindx() will fail with EINVAL. A caller may not remove all addresses from a socket; sctp_bindx() will reject such an attempt with EINVAL.

An application can use sctp_bindx(SCTP_BINDX_ADD_ADDR) to associate additional addresses with an endpoint after calling bind(). Or use sctp_bindx(SCTP_BINDX_REM_ADDR) to remove some addresses a listening socket is associated with, so that no new association accepted will be associated with these addresses. If the endpoint supports dynamic address reconfiguration, an SCTP_BINDX_REM_ADDR or SCTP_BINDX_ADD_ADDR may cause an endpoint to send the appropriate message to its peers to change the peers' address lists.

Adding and removing addresses from established associations is an optional functionality. Implementations that do not support this functionality should return -1 and set errno to EOPNOTSUPP.

sctp_bindx() can be called on an already bound socket or on an unbound socket. If the socket is unbound and the first port number in the addrs is zero, the kernel will choose a port number. All port numbers after the first one being 0 must also be zero. If the first port number is not zero, the following port numbers must be zero or have the same value as the first one. For an already bound socket, all port numbers provided must be the bound one or 0.

sctp_bindx() is an atomic operation. Therefore, the binding will be

either successful on all addresses or fail on all addresses. If multiple addresses are provided and the sctp_bindx() call fails there is no indication which address is responsible for the failure. The only way to identify the specific error indication is to call sctp_bindx() sequentially with only one address per call.

9.2. sctp_peeloff()

After an association is established on a one-to-many style socket, the application may wish to branch off the association into a separate socket/file descriptor.

This is particularly desirable when, for instance, the application wishes to have a number of sporadic message senders/receivers remain under the original one-to-many style socket, but branch off these associations carrying high volume data traffic into their own separate socket descriptors.

The application uses the sctp_peeloff() call to branch off an association into a separate socket (Note the semantics are somewhat changed from the traditional one-to-one style accept() call). Note that the new socket is a one-to-one style socket. Thus it will be confined to operations allowed for a one-to-one style socket.

The function prototype is

```
int sctp_peeloff(int sd,
                 sctp_assoc_t assoc_id);
```

and the arguments are

sd: The original one-to-many style socket descriptor returned from the socket() system call (see Section 3.1.1).

assoc_id: the specified identifier of the association that is to be branched off to a separate file descriptor (Note, in a traditional one-to-one style accept() call, this would be an out parameter, but for the one-to-many style call, this is an in parameter).

The function returns a non-negative file descriptor representing the branched-off association, or -1 if an error occurred. The variable errno is then set appropriately.

9.3. sctp_getpaddrs()

sctp_getpaddrs() returns all peer addresses in an association.

The function prototype is:

```
int sctp_getpaddrs(int sd,
                   sctp_assoc_t id,
                   struct sockaddr **addrs);
```

On return, addrs will point to a dynamically allocated array of sockaddr structures of the appropriate type for the socket type. The caller should use sctp_freepaddrs() to free the memory. Note that the in/out parameter addrs must not be NULL.

If sd is an IPv4 socket, the addresses returned will be all IPv4 addresses. If sd is an IPv6 socket, the addresses returned can be a mix of IPv4 or IPv6 addresses, with IPv4 addresses returned according to the SCTP_I_WANT_MAPPED_V4_ADDR option setting.

For one-to-many style sockets, id specifies the association to query. For one-to-one style sockets, id is ignored.

On success, sctp_getpaddrs() returns the number of peer addresses in the association. If there is no association on this socket, sctp_getpaddrs() returns 0, and the value of *addrs is undefined. If an error occurs, sctp_getpaddrs() returns -1, and the value of *addrs is undefined.

9.4. sctp_freepaddrs()

sctp_freepaddrs() frees all resources allocated by sctp_getpaddrs().

The function prototype is

```
void sctp_freepaddrs(struct sockaddr *addrs);
```

and addrs is the array of peer addresses returned by sctp_getpaddrs().

9.5. sctp_getladdrs()

sctp_getladdrs() returns all locally bound address(es) on a socket.

The function prototype is

```
int sctp_getladdrs(int sd,
                   sctp_assoc_t id,
                   struct sockaddr **addrs);
```

On return, addrs will point to a dynamically allocated array of sockaddr structures of the appropriate type for the socket type. The caller should use sctp_freeladdrs() to free the memory. Note that the in/out parameter addrs must not be NULL.

If sd is an IPv4 socket, the addresses returned will be all IPv4 addresses. If sd is an IPv6 socket, the addresses returned can be a mix of IPv4 or IPv6 addresses, with IPv4 addresses returned according to the SCTP_I_WANT_MAPPED_V4_ADDR option setting.

For one-to-many style sockets, id specifies the association to query. For one-to-one style sockets, id is ignored.

If the id field is set to the value '0' then the locally bound addresses are returned without regard to any particular association.

On success, sctp_getladdrs() returns the number of local addresses bound to the socket. If the socket is unbound, sctp_getladdrs() returns 0, and the value of *addrs is undefined. If an error occurs, sctp_getladdrs() returns -1, and the value of *addrs is undefined.

9.6. sctp_freeladdrs()

sctp_freeladdrs() frees all resources allocated by sctp_getladdrs().

The function prototype is

```
void sctp_freeladdrs(struct sockaddr *addrs);
```

and addrs is the array of local addresses returned by sctp_getladdrs().

9.7. sctp_sendmsg() - DEPRECATED

This function is deprecated, sctp_sendv() (see Section 9.12) should be used instead.

An implementation may provide a library function (or possibly system call) to assist the user with the advanced features of SCTP.

The function prototype is

```
ssize_t sctp_sendmsg(int sd,
                     const void *msg,
                     size_t len,
                     const struct sockaddr *to,
                     socklen_t tolen,
                     uint32_t ppid,
                     uint32_t flags,
                     uint16_t stream_no,
                     uint32_t timetolive,
                     uint32_t context);
```

and the arguments are:

sd: The socket descriptor.

msg: The message to be sent.

len: The length of the message.

to: The destination address of the message.

tolen: The length of the destination address.

ppid: The same as sinfo_ppid (see Section 5.3.2).

flags: The same as sinfo_flags (see Section 5.3.2).

stream_no: The same as sinfo_stream (see Section 5.3.2).

timetolive: The same as sinfo_timetolive (see Section 5.3.2).

context: The same as sinfo_context (see Section 5.3.2).

The call returns the number of characters sent, or -1 if an error occurred. The variable errno is then set appropriately.

Sending a message using sctp_sendmsg() is atomic (unless explicit EOR marking is enabled on the socket specified by sd).

Using sctp_sendmsg() on a non-connected one-to-one style socket for implicit connection setup may or may not work depending on the SCTP implementation.

9.8. sctp_recvmsg() - DEPRECATED

This function is deprecated, sctp_recvv() (see Section 9.13) should be used instead.

An implementation may provide a library function (or possibly system call) to assist the user with the advanced features of SCTP. Note that in order for the sctp_sndrcvinfo structure to be filled in by sctp_recvmsg() the caller must enable the sctp_data_io_event with the SCTP_EVENTS option. Note that the setting of the SCTP_USE_EXT_RCVINFO will affect this function as well, causing the sctp_sndrcvinfo information to be extended.

The function prototype is

```
ssize_t sctp_recvmsg(int sd,
                     void *msg,
                     size_t len,
                     struct sockaddr *from,
                     socklen_t *fromlen
                     struct sctp_sndrcvinfo *sinfo
                     int *msg_flags);
```

and the arguments are

sd: The socket descriptor.

msg: The message buffer to be filled.

len: The length of the message buffer.

from: A pointer to an address to be filled with the sender of this messages address.

fromlen: An in/out parameter describing the from length.

sinfo: A pointer to an sctp_sndrcvinfo structure to be filled upon receipt of the message.

msg_flags: A pointer to an integer to be filled with any message flags (e.g. MSG_NOTIFICATION). Note that this field is an in-out field. Options for the receive may also be passed into the value (e.g. MSG_PEEK). On return from the call, the msg_flags value will be different than what was sent in to the call. If implemented via a recvmsg() call, the msg_flags should only contain the value of the flags from the recvmsg() call.

The call returns the number of bytes received, or -1 if an error occurred. The variable errno is then set appropriately.

9.9. sctp_connectx()

An implementation may provide a library function (or possibly system call) to assist the user with associating to an endpoint that is multi-homed. Much like sctp_bindx() this call allows a caller to specify multiple addresses at which a peer can be reached. The way the SCTP stack uses the list of addresses to set up the association is implementation dependent. This function only specifies that the stack will try to make use of all the addresses in the list when needed.

Note that the list of addresses passed in is only used for setting up the association. It does not necessarily equal the set of addresses

the peer uses for the resulting association. If the caller wants to find out the set of peer addresses, it must use sctp_getpaddrs() to retrieve them after the association has been set up.

The function prototype is

```
int sctp_connectx(int sd,
                  struct sockaddr *addrs,
                  int addrcnt,
                  sctp_assoc_t *id);
```

and the arguments are:

sd: The socket descriptor.

addrs: An array of addresses.

addrcnt: The number of addresses in the array.

id: An output parameter that if passed in as a non-NULL will return the association identifier for the newly created association (if successful).

The call returns 0 on success or -1 if an error occurred. The variable errno is then set appropriately.

9.10. sctp_send() - DEPRECATED

This function is deprecated, sctp_sendv() should be used instead.

An implementation may provide another alternative function or system call to assist an application with the sending of data without the use of the CMSG header structures.

The function prototype is

```
ssize_t sctp_send(int sd,
                  const void *msg,
                  size_t len,
                  const struct sctp_sndrcvinfo *sinfo,
                  int flags);
```

and the arguments are

sd: The socket descriptor.

msg: The message to be sent.

len: The length of the message.

sinfo: A pointer to an sctp_sndrcvinfo structure used as described in Section 5.3.2 for a sendmsg() call.

flags: The same flags as used by the sendmsg() call flags (e.g. MSG_DONTROUTE).

The call returns the number of bytes sent, or -1 if an error occurred. The variable errno is then set appropriately.

This function call may also be used to terminate an association using an association identifier by setting the sinfo.sinfo_flags to SCTP_EOF and the sinfo.sinfo_assoc_id to the association that needs to be terminated. In such a case the len of the message can be zero.

Using sctp_send() on a non-connected one-to-one style socket for implicit connection setup may or may not work depending on the SCTP implementation.

Sending a message using sctp_send() is atomic unless explicit EOR marking is enabled on the socket specified by sd.

9.11. sctp_sendx() - DEPRECATED

This function is deprecated, sctp_sendv() should be used instead.

An implementation may provide another alternative function or system call to assist an application with the sending of data without the use of the CMSG header structures that also gives a list of addresses. The list of addresses is provided for implicit association setup. In such a case the list of addresses serves the same purpose as the addresses given in sctp_connectx() (see Section 9.9).

The function prototype is

```
ssize_t sctp_sendx(int sd,
                   const void *msg,
                   size_t len,
                   struct sockaddr *addrs,
                   int addrcnt,
                   struct sctp_sndrcvinfo *sinfo,
                   int flags);
```

and the arguments are:

sd: The socket descriptor.

msg: The message to be sent.

len: The length of the message.

addrs: Is an array of addresses.

addrcnt: The number of addresses in the array.

sinfo: A pointer to an sctp_sndrcvinfo structure used as described in Section 5.3.2 for a sendmsg() call.

flags: The same flags as used by the sendmsg() call flags (e.g. MSG_DONTROUTE).

The call returns the number of bytes sent, or -1 if an error occurred. The variable errno is then set appropriately.

Note that in case of implicit connection setup, on return from this call the sinfo_assoc_id field of the sinfo structure will contain the new association identifier.

This function call may also be used to terminate an association using an association identifier by setting the sinfo.sinfo_flags to SCTP_EOF and the sinfo.sinfo_assoc_id to the association that needs to be terminated. In such a case the len of the message would be zero.

Sending a message using sctp_sendx() is atomic unless explicit EOR marking is enabled on the socket specified by sd.

Using sctp_sendx() on a non-connected one-to-one style socket for implicit connection setup may or may not work depending on the SCTP implementation.

9.12. sctp_sendv()

The function prototype is

```
ssize_t sctp_sendv(int sd,
                   const struct iovec *iov,
                   int iovcnt,
                   struct sockaddr *addrs,
                   int addrcnt,
                   void *info,
                   socklen_t infolen,
                   unsigned int infotype,
```

```
                    int flags);
```

The function sctp_sendv() provides an extensible way for an application to communicate different send attributes to the SCTP stack when sending a message. An implementation may provide sctp_sendv() as a library function or a system call.

This document defines three types of attributes which can be used to describe a message to be sent. They are struct sctp_sndinfo (Section 5.3.4), struct sctp_prinfo (Section 5.3.7), and struct sctp_authinfo (Section 5.3.8). The following structure sctp_sendv_spa is defined to be used when more than one of the above attributes are needed to describe a message to be sent.

```
struct sctp_sendv_spa {
  uint32_t sendv_flags;
  struct sctp_sndinfo sendv_sndinfo;
  struct sctp_prinfo sendv_prinfo;
  struct sctp_authinfo sendv_authinfo;
};
```

The sendv_flags field holds a bit wise OR of SCTP_SEND_SNDINFO_VALID, SCTP_SEND_PRINFO_VALID and SCTP_SEND_AUTHINFO_VALID indicating if the sendv_sndinfo/sendv_prinfo/sendv_authinfo fields contain valid information.

In future, when new send attributes are needed, new structures can be defined. But those new structures do not need to be based on any of the above defined structures.

The function takes the following arguments:

sd: The socket descriptor.

iov: The gather buffer. The data in the buffer is treated as one single user message.

iovcnt: The number of elements in iov.

addrs: An array of addresses to be used to set up an association or one single address to be used to send the message. Pass in NULL if the caller does not want to set up an association nor want to send the message to a specific address.

addrcnt: The number of addresses in the addrs array.

info: A pointer to the buffer containing the attribute associated with the message to be sent. The type is indicated by the info_type parameter.

infolen: The length in bytes of info.

infotype: Identifies the type of the information provided in info. The current defined values are:

SCTP_SENDV_NOINFO: No information is provided. The parameter info is a NULL pointer and infolen is 0.

SCTP_SENDV_SNDINFO: The parameter info is pointing to a struct sctp_sndinfo.

SCTP_SENDV_PRINFO: The parameter info is pointing to a struct sctp_prinfo.

SCTP_SENDV_AUTHINFO: The parameter info is pointing to a struct sctp_authinfo.

SCTP_SENDV_SPA: The parameter info is pointing to a struct sctp_sendv_spa.

flags: The same flags as used by the sendmsg() call flags (e.g. MSG_DONTROUTE).

The call returns the number of bytes sent, or -1 if an error occurred. The variable errno is then set appropriately.

A note on one-to-many style socket. The struct sctp_sndinfo attribute must always be used in order to specify the association the message is to be sent on. The only case where it is not needed is when this call is used to set up a new association.

The caller provides a list of addresses in the addrs parameter to set up an association. This function will behave like calling sctp_connectx() (see Section 9.9) first using the list of addresses and then calling sendmsg() with the given message and attributes. For an one-to-many style socket, if struct sctp_sndinfo attribute is provided, the snd_assoc_id field must be 0. When this function returns, the snd_assoc_id field will contain the association identifier of the newly established association. Note that struct sctp_sndinfo attribute is not required to set up an association for one-to-many style socket. If this attribute is not provided, the caller can enable the SCTP_ASSOC_CHANGE notification and use the SCTP_COMM_UP message to find out the association identifier.

If the caller wants to send the message to a specific peer address (hence overriding the primary address), it can provide the specific address in the addrs parameter and provide a struct sctp_sndinfo attribute with the field snd_flags set to SCTP_ADDR_OVER.

This function call may also be used to terminate an association. The caller provides an sctp_sndinfo attribute with the snd_flags set to SCTP_EOF. In this case the len of the message would be zero.

Sending a message using sctp_sendv() is atomic unless explicit EOR marking is enabled on the socket specified by sd.

9.13. sctp_recvv()

The function prototype is

```
ssize_t sctp_recvv(int sd,
                   const struct iovec *iov,
                   int iovlen,
                   struct sockaddr *from,
                   socklen_t *fromlen,
                   void *info,
                   socklen_t *infolen,
                   unsigned int *infotype,
                   int *flags);
```

The function sctp_recvv() provides an extensible way for the SCTP stack to pass up different SCTP attributes associated with a received message to an application. An implementation may provide sctp_recvv() as a library function or as a system call.

This document defines two types of attributes which can be returned by this call, the attribute of the received message and the attribute of the next message in receive buffer. The caller enables the SCTP_RECVRCVINFO and SCTP_RECVNXTINFO socket option to receive these attributes respectively. Attributes of the received message are returned in struct sctp_rcvinfo (Section 5.3.5) and attributes of the next message are returned in struct sctp_nxtinfo (Section 5.3.6). If both options are enabled, both attributes are returned using the following structure.

```
struct sctp_recvv_rn {
  struct sctp_rcvinfo recvv_rcvinfo;
  struct sctp_nxtinfo recvv_nxtinfo;
};
```

In future, new structures can be defined to hold new types of attributes. The new structures do not need to be based on struct

sctp_recvv_rn or struct sctp_rcvinfo.

This function takes the following arguments:

sd: The socket descriptor.

iov: The scatter buffer. Only one user message is returned in this buffer.

iovlen: The number of elements in iov.

from: A pointer to an address to be filled with the sender of the received message's address.

fromlen: An in/out parameter describing the from length.

info: A pointer to the buffer to hold the attributes of the received message. The structure type of info is determined by the info_type parameter.

infolen: An in/out parameter describing the size of the info buffer.

infotype: In return, *info_type is set to the type of the info buffer. The current defined values are:

SCTP_RECVV_NOINFO: If both SCTP_RECVRCVINFO and SCTP_RECVNXTINFO options are not enabled, no attribute will be returned. If only the SCTP_RECVNXTINFO option is enabled but there is no next message in the buffer, there will also no attribute be returned. In these cases *info_type will be set to SCTP_RECVV_NOINFO.

SCTP_RECVV_RCVINFO: The type of info is struct sctp_rcvinfo and the attribute is about the received message.

SCTP_RECVV_NXTINFO: The type of info is struct sctp_nxtinfo and the attribute is about the next message in receive buffer. This is the case when only the SCTP_RECVNXTINFO option is enabled and there is a next message in buffer.

SCTP_RECVV_RN: The type of info is struct sctp_recvv_rn. The recvv_rcvinfo field is the attribute of the received message and the recvv_nxtinfo field is the attribute of the next message in buffer. This is the case when both SCTP_RECVRCVINFO and SCTP_RECVNXTINFO options are enabled and there is a next message in the receive buffer.

flags: A pointer to an integer to be filled with any message flags (e.g. MSG_NOTIFICATION). Note that this field is an in/out parameter. Options for the receive may also be passed into the value (e.g. MSG_PEEK). On return from the call, the flags value will be different than what was sent in to the call. If implemented via a recvmsg() call, the flags should only contain the value of the flags from the recvmsg() call when calling sctp_recvv() and on return it has the value from msg_flags.

The call returns the number of bytes received, or -1 if an error occurred. The variable errno is then set appropriately.

10. IANA Considerations

This document requires no actions from IANA.

11. Security Considerations

Many TCP and UDP implementations reserve port numbers below 1024 for privileged users. If the target platform supports privileged users, the SCTP implementation should restrict the ability to call bind() or sctp_bindx() on these port numbers to privileged users.

Similarly unprivileged users should not be able to set protocol parameters that could result in the congestion control algorithm being more aggressive than permitted on the public Internet. These parameters are:

o struct sctp_rtoinfo

If an unprivileged user inherits a one-to-many style socket with open associations on a privileged port, it may be permitted to accept new associations, but it should not be permitted to open new associations. This could be relevant for the r* family of protocols.

Applications using the one-to-many style sockets and using the interleave level if 0 are subject to denial of service attacks as described in Section 8.1.20.

Applications needing transport layer security can use DTLS/SCTP as specified in [RFC6083]. This can be implemented using the socket API described in this document.

12. Acknowledgments

Special acknowledgment is given to Ken Fujita, Jonathan Woods, Qiaobing Xie, and La Monte Yarroll, who helped extensively in the early formation of this document.

The authors also wish to thank Kavitha Baratakke, Mike Bartlett, Martin Becke, Jon Berger, Mark Butler, Thomas Dreibholz, Andreas Fink, Scott Kimble, Jonathan Leighton, Renee Revis, Irene Ruengeler, Dan Wing, and many others on the TSVWG mailing list for contributing valuable comments.

A special thanks to Phillip Conrad, for his suggested text, quick and constructive insights, and most of all his persistent fighting to keep the interface to SCTP usable for the application programmer.

13. References

13.1. Normative References

[IEEE-1003.1-2008] Institute of Electrical and Electronics Engineers, "Information Technology - Portable Operating System Interface (POSIX)", IEEE Standard 1003.1, 2008.

[RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, February 2003.

[RFC3542] Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei, "Advanced Sockets Application Program Interface (API) for IPv6", RFC 3542, May 2003.

[RFC3758] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", RFC 3758, May 2004.

[RFC4895] Tuexen, M., Stewart, R., Lei, P., and E. Rescorla, "Authenticated Chunks for the Stream Control Transmission Protocol (SCTP)", RFC 4895, August 2007.

[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.

13.2. Informative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC1644] Braden, B., "T/TCP -- TCP Extensions for Transactions Functional Specification", RFC 1644, July 1994.

[RFC6083] Tuexen, M., Seggelmann, R., and E. Rescorla, "Datagram Transport Layer Security (DTLS) for Stream Control Transmission Protocol (SCTP)", RFC 6083, January 2011.

Appendix A. One-to-One Style Code Example

The following code is an implementation of a simple client which sends a number of messages marked for unordered delivery to an echo server making use of all outgoing streams. The example shows how to use some features of one-to-one style IPv4 SCTP sockets, including:

- Creating and connecting an SCTP socket.
- Requesting to negotiate a number of outgoing streams.
- Determining the negotiated number of outgoing streams.
- Setting an adaptation layer indication.
- Sending messages with a given payload protocol identifier on a particular stream using sctp_sendv().

```
/*

   Copyright (c) 2011 IETF Trust and the persons identified
   as authors of the code. All rights reserved.

   Redistribution and use in source and binary forms, with
   or without modification, is permitted pursuant to, and subject
   to the license terms contained in, the Simplified BSD License
   set forth in Section 4.c of the IETF Trust's Legal Provisions
   Relating to IETF Documents (http://trustee.ietf.org/license-info).
```

```
   */

   #include <sys/types.h>
   #include <sys/socket.h>
   #include <netinet/in.h>
   #include <netinet/sctp.h>
   #include <arpa/inet.h>
   #include <string.h>
   #include <stdio.h>
   #include <unistd.h>
   #include <stdlib.h>

   #define PORT 9
   #define ADDR "127.0.0.1"
   #define SIZE_OF_MESSAGE 1000
   #define NUMBER_OF_MESSAGES 10
   #define PPID 1234

   int
   main(void) {
     unsigned int i;
     int sd;
     struct sockaddr_in addr;
     char buffer[SIZE_OF_MESSAGE];
     struct iovec iov;
     struct sctp_status status;
     struct sctp_initmsg init;
     struct sctp_sndinfo info;
     struct sctp_setadaptation ind;
     socklen_t opt_len;

     /* Create a one-to-one style SCTP socket. */
     if ((sd = socket(AF_INET, SOCK_STREAM, IPPROTO_SCTP)) < 0) {
       perror("socket");
       exit(1);
     }

     /* Prepare for requesting 2048 outgoing streams. */
     memset(&init, 0, sizeof(init));
     init.sinit_num_ostreams = 2048;
     if (setsockopt(sd, IPPROTO_SCTP, SCTP_INITMSG,
                    &init, (socklen_t)sizeof(init)) < 0) {
       perror("setsockopt");
       exit(1);
     }

     ind.ssb_adaptation_ind  = 0x01020304;
     if (setsockopt(sd, IPPROTO_SCTP, SCTP_ADAPTATION_LAYER,
```

```
                  &ind, (socklen_t)sizeof(ind)) < 0) {
      perror("setsockopt");
      exit(1);
    }

    /* Connect to the discard server. */
    memset(&addr, 0, sizeof(addr));
  #ifdef HAVE_SIN_LEN
    addr.sin_len         = sizeof(struct sockaddr_in);
  #endif
    addr.sin_family      = AF_INET;
    addr.sin_port        = htons(PORT);
    addr.sin_addr.s_addr = inet_addr(ADDR);
    if (connect(sd,
                (const struct sockaddr *)&addr,
                sizeof(struct sockaddr_in)) < 0) {
      perror("connect");
      exit(1);
    }

    /* Get the actual number of outgoing streams. */
    memset(&status, 0, sizeof(status));
    opt_len = (socklen_t)sizeof(status);
    if (getsockopt(sd, IPPROTO_SCTP, SCTP_STATUS,
                   &status, &opt_len) < 0) {
      perror("getsockopt");
      exit(1);
    }

    memset(&info, 0, sizeof(info));
    info.snd_ppid = htonl(PPID);
    info.snd_flags = SCTP_UNORDERED;
    memset(buffer, 'A', SIZE_OF_MESSAGE);
    iov.iov_base = buffer;
    iov.iov_len = SIZE_OF_MESSAGE;
    for (i = 0; i <  NUMBER_OF_MESSAGES; i++) {
      info.snd_sid = i % status.sstat_outstrms;
      if (sctp_sendv(sd,
                     (const struct iovec *)&iov, 1,
                     NULL, 0,
                     &info, sizeof(info), SCTP_SENDV_SNDINFO,
                     0) < 0) {
        perror("sctp_sendv");
        exit(1);
      }
    }

    if (close(sd) < 0) {
```

```
      perror("close");
      exit(1);
    }
    return(0);
  }
```

Appendix B. One-to-Many Style Code Example

The following code is a simple implementation of a discard server over SCTP. The example shows how to use some features of one-to-many style IPv6 SCTP sockets, including:

- Opening and binding of a socket.
- Enabling notifications.
- Handling notifications.
- Configuring the auto close timer.
- Using sctp_recvv() to receive messages.

Please note that this server can be used in combination with the client described in Appendix A.

```
/*

   Copyright (c) 2011 IETF Trust and the persons identified
   as authors of the code. All rights reserved.

   Redistribution and use in source and binary forms, with
   or without modification, is permitted pursuant to, and subject
   to the license terms contained in, the Simplified BSD License
   set forth in Section 4.c of the IETF Trust's Legal Provisions
   Relating to IETF Documents (http://trustee.ietf.org/license-info).

*/

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/sctp.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

```
   #define BUFFER_SIZE (1<<16)
   #define PORT 9
   #define ADDR "0.0.0.0"
   #define TIMEOUT 5

   static void
   print_notification(void *buf)
   {
     struct sctp_assoc_change *sac;
     struct sctp_paddr_change *spc;
     struct sctp_adaptation_event *sad;
     union sctp_notification *snp;
     char addrbuf[INET6_ADDRSTRLEN];
     const char *ap;
     struct sockaddr_in *sin;
     struct sockaddr_in6 *sin6;

     snp = buf;

     switch (snp->sn_header.sn_type) {
     case SCTP_ASSOC_CHANGE:
       sac = &snp->sn_assoc_change;
       printf("^^^ Association change: ");
       switch (sac->sac_state) {
       case SCTP_COMM_UP:
         printf("Communication up (streams (in/out)=(%u/%u)).\n",
                sac->sac_inbound_streams, sac->sac_outbound_streams);
         break;
       case SCTP_COMM_LOST:
         printf("Communication lost (error=%d).\n", sac->sac_error);
         break;
       case SCTP_RESTART:
         printf("Communication restarted (streams (in/out)=(%u/%u).\n",
                sac->sac_inbound_streams, sac->sac_outbound_streams);
         break;
       case SCTP_SHUTDOWN_COMP:
         printf("Communication completed.\n");
         break;
       case SCTP_CANT_STR_ASSOC:
         printf("Communication couldn't be started.\n");
         break;
       default:
         printf("Unknown state: %d.\n", sac->sac_state);
         break;
       }
       break;
     case SCTP_PEER_ADDR_CHANGE:
       spc = &snp->sn_paddr_change;
```

```
      if (spc->spc_aaddr.ss_family == AF_INET) {
        sin = (struct sockaddr_in *)&spc->spc_aaddr;
        ap = inet_ntop(AF_INET, &sin->sin_addr,
                       addrbuf, INET6_ADDRSTRLEN);
      } else {
        sin6 = (struct sockaddr_in6 *)&spc->spc_aaddr;
        ap = inet_ntop(AF_INET6, &sin6->sin6_addr,
                       addrbuf, INET6_ADDRSTRLEN);
      }
      printf("^^^ Peer Address change: %s ", ap);
      switch (spc->spc_state) {
      case SCTP_ADDR_AVAILABLE:
        printf("is available.\n");
        break;
      case SCTP_ADDR_UNREACHABLE:
        printf("is not available (error=%d).\n", spc->spc_error);
        break;
      case SCTP_ADDR_REMOVED:
        printf("was removed.\n");
        break;
      case SCTP_ADDR_ADDED:
        printf("was added.\n");
        break;
      case SCTP_ADDR_MADE_PRIM:
        printf("is primary.\n");
        break;
      default:
        printf("unknown state (%d).\n", spc->spc_state);
        break;
      }
      break;
    case SCTP_SHUTDOWN_EVENT:
      printf("^^^ Shutdown received.\n");
      break;
    case SCTP_ADAPTATION_INDICATION:
      sad = &snp->sn_adaptation_event;
      printf("^^^ Adaptation indication 0x%08x received.\n",
             sad->sai_adaptation_ind);
      break;
    default:
      printf("^^^ Unknown event of type: %u.\n",
             snp->sn_header.sn_type);
      break;
    };
  }

  int
  main(void) {
```

```
int sd, flags, timeout, on;
ssize_t n;
unsigned int i;
union {
  struct sockaddr sa;
  struct sockaddr_in sin;
  struct sockaddr_in6 sin6;
} addr;
socklen_t fromlen, infolen;
struct sctp_rcvinfo info;
unsigned int infotype;
struct iovec iov;
char buffer[BUFFER_SIZE];
struct sctp_event event;
uint16_t event_types[] = {SCTP_ASSOC_CHANGE,
                          SCTP_PEER_ADDR_CHANGE,
                          SCTP_SHUTDOWN_EVENT,
                          SCTP_ADAPTATION_INDICATION};

/* Create a 1-to-many style SCTP socket. */
if ((sd = socket(AF_INET6, SOCK_SEQPACKET, IPPROTO_SCTP)) < 0) {
  perror("socket");
  exit(1);
}

/* Enable the events of interest. */
memset(&event, 0, sizeof(event));
event.se_assoc_id = SCTP_FUTURE_ASSOC;
event.se_on = 1;
for (i = 0; i < sizeof(event_types)/sizeof(uint16_t); i++) {
  event.se_type = event_types[i];
  if (setsockopt(sd, IPPROTO_SCTP, SCTP_EVENT,
                 &event, sizeof(event)) < 0) {
    perror("setsockopt");
    exit(1);
  }
}

/* Configure auto-close timer. */
timeout = TIMEOUT;
if (setsockopt(sd, IPPROTO_SCTP, SCTP_AUTOCLOSE,
               &timeout, sizeof(timeout)) < 0) {
  perror("setsockopt SCTP_AUTOCLOSE");
  exit(1);
}

/* Enable delivery of SCTP_RCVINFO. */
on = 1;
```

```
    if (setsockopt(sd, IPPROTO_SCTP, SCTP_RECVRCVINFO,
                   &on, sizeof(on)) < 0) {
      perror("setsockopt SCTP_RECVRCVINFO");
      exit(1);
    }

    /* Bind the socket to all local addresses. */
    memset(&addr, 0, sizeof(addr));
  #ifdef HAVE_SIN6_LEN
    addr.sin6.sin6_len         = sizeof(addr.sin6);
  #endif
    addr.sin6.sin6_family      = AF_INET6;
    addr.sin6.sin6_port        = htons(PORT);
    addr.sin6.sin6_addr        = in6addr_any;
    if (bind(sd, &addr.sa, sizeof(addr.sin6)) < 0) {
      perror("bind");
      exit(1);
    }
    /* Enable accepting associations. */
    if (listen(sd, 1) < 0) {
      perror("listen");
      exit(1);
    }

    for (;;) {
      flags = 0;
      memset(&addr, 0, sizeof(addr));
      fromlen = (socklen_t)sizeof(addr);
      memset(&info, 0, sizeof(info));
      infolen = (socklen_t)sizeof(info);
      infotype = 0;
      iov.iov_base = buffer;
      iov.iov_len = BUFFER_SIZE;

      n = sctp_recvv(sd, &iov, 1,
                     &addr.sa, &fromlen,
                     &info, &infolen, &infotype,
                     &flags);

      if (flags & MSG_NOTIFICATION) {
        print_notification(iov.iov_base);
      } else {
        char addrbuf[INET6_ADDRSTRLEN];
        const char *ap;
        in_port_t port;

        if (addr.sa.sa_family == AF_INET) {
               ap = inet_ntop(AF_INET, &addr.sin.sin_addr,
```

```
                              addrbuf, INET6_ADDRSTRLEN);
                port = ntohs(addr.sin.sin_port);
            } else {
                ap = inet_ntop(AF_INET6, &addr.sin6.sin6_addr,
                              addrbuf, INET6_ADDRSTRLEN);
                port = ntohs(addr.sin6.sin6_port);
            }
            printf("Message received from %s:%u: len=%d",
                ap, port, (int)n);
            switch (infotype) {
            case SCTP_RECVV_RCVINFO:
              printf(", sid=%u", info.rcv_sid);
              if (info.rcv_flags & SCTP_UNORDERED) {
                printf(", unordered");
              } else {
                printf(", ssn=%u", info.rcv_ssn);
              }
              printf(", tsn=%u", info.rcv_tsn);
              printf(", ppid=%u.\n", ntohl(info.rcv_ppid));
              break;
            case SCTP_RECVV_NOINFO:
            case SCTP_RECVV_NXTINFO:
            case SCTP_RECVV_RN:
              printf(".\n");
              break;
            default:
              printf(" unknown infotype.\n");
            }
          }
        }

        if (close(sd) < 0) {
          perror("close");
          exit(1);
        }

        return (0);
      }
```

Authors' Addresses

Randall R. Stewart
Adara Networks
Chapin, SC 29036
USA

Email: randall@lakerest.net

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

Email: tuexen@fh-muenster.de

Kacheong Poon
Oracle Corporation

Email: ka-cheong.poon@oracle.com

Peter Lei
Cisco Systems, Inc.
9501 Technology Blvd
West Office Center
Rosemont, IL 60018
USA

Email: peterlei@cisco.com

Vladislav Yasevich
HP
110 Spitrook Rd
Nashua, NH 03062
USA

Email: vladislav.yasevich@hp.com

TSVWG WG
Internet-Draft
Expires: September 14, 2011
Intended Status: Standards Track (PS)
Updates: RFC 2205, 2210, & 4495 (if published as an RFC)

James Polk
Subha Dhesikan
Cisco Systems
March 14, 2011

Integrated Services (IntServ) Extension to Allow Signaling of Multiple Traffic Specifications and Multiple Flow Specifications in RSVPv1
draft-polk-tsvwg-intserv-multiple-tspec-06

Abstract

This document defines extensions to Integrated Services (IntServ) allowing multiple traffic specifications and multiple flow specifications to be conveyed in the same Resource Reservation Protocol (RSVPv1) reservation message exchange. This ability helps optimize an agreeable bandwidth through a network between endpoints in a single round trip.

Table of Contents

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

1. Introduction

This document defines how Integrated Services (IntServ) [RFC2210] includes multiple traffic specifications and multiple flow specifications in the same Resource Reservation Protocol (RSVPv1) [RFC2205] message. This ability helps optimize an agreeable bandwidth through a network between endpoints in a single round trip.

There is a separation of function between RSVP and IntServ, in which RSVP does not define the internal objects to establish controlled load or guarantee services. These are generally left to be opaque in RSVP. At the same time, IntServ does not require that RSVP be the only reservation protocol for transporting both the controlled load or guaranteed service objects - but RSVP does often carry the objects anyway. This makes the two independent - yet related in usage, but are also frequently talked about as if they are one and the same. They are not.

The 'traffic specification' contains the traffic characteristics of a sender's data flow and is a required object in a PATH message. The TSPEC object is defined in RFC 2210 to convey the traffic specification from the sender and is opaque to RSVP. The ADSPEC object - for 'advertising specification' - is used to gather information along the downstream data path to aid the receiver in the computation of QoS properties of this data path. The ADSPEC is also opaque to RSVP and is defined in RFC 2210. Both of these IntServ objects are part of the Sender Descriptor [RFC2205].

Once the Sender Descriptor is received at its destination node, after having traveled through the network of routers, the SENDER_TSPEC information is matched with the information gathered in the ADSPEC, if present, about the data path. Together, these two objects help the receiver build its flow specification (encoded in the FLOWSPEC object) for the RESV message. The RESV message establishes the reservation through the network of routers on the data path established by the PATH message. If the ADSPEC is not present in the Sender_Descriptor, it cannot aid the receiver in building the flow specification.

The SENDER_TSPEC is not changed in transit between endpoints (i.e., there are no bandwidth request adjustments along the way). However, the ADSPEC is changed, based on the conditions experienced through the network (i.e., bandwidth availability within each router) as the RSVP message travels hop-by-hop.

Today, real-time applications have evolved such that they are able to dynamically adapt to available bandwidth, not only by dropping and adding layers, but also by reducing frame rates and resolution. It is therefore limiting to have a single bandwidth request in Integrated Services, and by extension, RSVP.

With only one traffic specification in a PATH message and only one flow specification in a RESV message (with some styles of reservations a RESV message may actually contain multiple flow specifications, but then there is only one per sender), applications will either have to give up altogether on session establishment in case of failure of the reservation establishment for the highest "bandwidth or will have to resort to multiple successive RSVP signaling attempts in a trial-and-error manner until they finally establish the reservation a lower "bandwidth". These multiple signaling round-trip would affect the session establishment time and in turn would negatively impact the end user experience.

The objective of this document is to avoid such roundtrips as well as allow applications to successfully receive some level of bandwidth allotment that it can use for its sessions.

While the ADSPEC provides an indication of the bandwidth available along the path and can be used by the receiver in creating the FLOWSPEC, it does not prevent failures or multiple round-trips as described above. The intermediary routers provide a best attempt estimate of available bandwidth in the ADSPEC object. However, it does not take into account external policy considerations (RFC 2215). In addition, the available bandwidth at the time of creating the ADSPEC may not be available at the time of an actual request in an RESV message. These reasons may cause the RESV message to be rejected. Therefore, the ADSPEC object cannot, by itself, satisfy the requirements of the current generations of real-time applications.

It needs to be noted that the ADSPEC is unchanged by this new mechanism. If ADSPEC is included in the PATH message, it is suggested that the receiver use this object in determining the flow specification.

This document creates a means for conveying more than one "bandwidth" within the same RSVP reservation set-up (both PATH and RESV) messages to optimize the determination of an agreed upon bandwidth for this reservation. Allowing multiple traffic specifications within the same PATH message allows the sender to communicate to the receiver multiple "bandwidths" that match the different sending rates that the sender is capable of transmitting at. This allows the receiver to convey this multiple "bandwidths" in the RESV so those can be considered when RSVP makes the actual reservation admission into the network. This allows the applications to dynamically adapt their data stream to available network resources.

The concept of RSVP signaling is shown in a single direction below, in Figure 1. Although the TSPEC is opaque to RSVP, it is shown along with the RSVP messages for completeness. The RSVP messages themselves need not be the focus of the reader. Instead, the number of round trips it takes to establish a reservation is the

focus here.

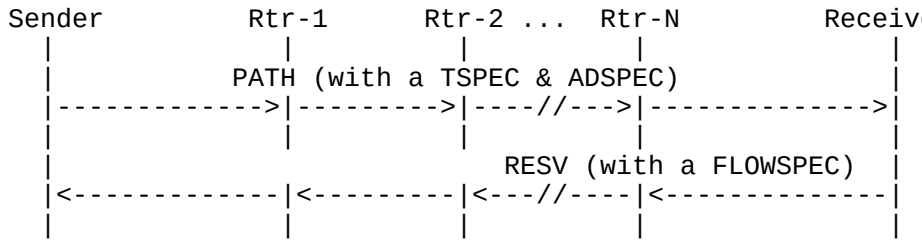


Figure 1. Concept of RSVP in a Single Direction

Figure 1 shows a successful one-way reservation using RSVP and IntServ.

Figure 2 shows a scenario where the RESV message, containing a FLOWSPEC, which is generated by the Receiver, after considering both the Sender TSPEC and the ADSPEC, is rejected by an intermediary router.

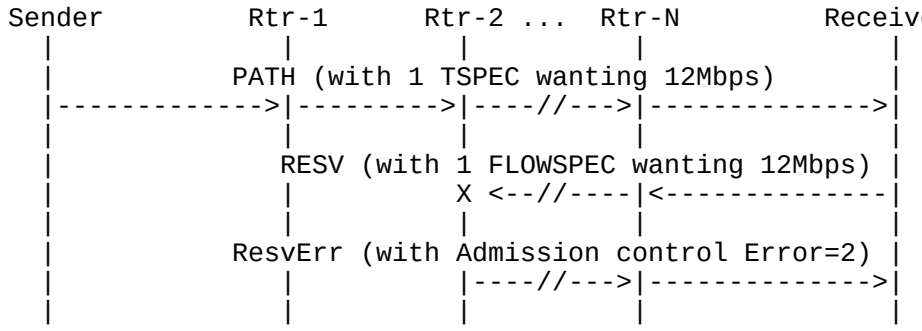


Figure 2. Concept of RSVP Rejection due to Limited Bandwidth

The scenario above is where multiple TSPEC and multiple FLOWSPEC optimization helps. The Sender may support multiple bandwidths for a given application (i.e., more than one codec for voice or video) and therefore might want to establish a reservation with the highest (or best) bandwidth that the network can provide for a particular codec.

For example, bandwidths of:

12Mbps,
4Mbps, and
1.5Mbps

for the three video codecs the Sender supports.

This document will discuss the overview of the proposal to include multiple TSPECs and FLOWSPECs RSVP in section 2. In section 3, the overview of the entire solution is provided. This section also contains the new parameters which are defined in this document. The multiple TSPECs in a PATH message and the multiple FLOWSPEC in a RESV message, both for controlled load and guaranteed service are described in this section. Section 4 will cover the rules of usage of this IntServ extension. This section contains how this document needs to extend the scenario of when a router in the middle of a reservation cannot accept a preferred bandwidth (i.e., FLOWSPEC), meaning previous routers that accepted that greater bandwidth now have too much bandwidth reserved. This requires an extension to RFC 4495 (RSVP Bandwidth Reduction) to cover reservations being established, as well as existing reservations. Section 4 also includes the merging rules.

2. Overview of Proposal for Including Multiple TSPECs and FLOWSPECS

Presently, this is the format of a PATH message [RFC2205]:

```
        <PATH Message> ::= <Common Header> [ <INTEGRITY> ]

                                    <SESSION> <RSVP_HOP>

                                    <TIME_VALUES>

                                   [ <POLICY_DATA> ... ]

                                   [ <sender descriptor> ]

        <sender descriptor> ::= <SENDER_TEMPLATE> <SENDER_TSPEC>
                                                   ^^^^^^^^^^^^
                                   [ <ADSPEC> ]
```

where the SENDER_TSPEC object contains a single traffic specification.

For the PATH message, the focus of this document is to modify the <sender_descriptor> in such a way to include more than one traffic specification. This solution does this by retaining the existing SENDER_TSPEC object above, highlighted by the '^^^^' characters, and complementing it with a new optional MULTI_TSPEC object to convey additional traffic specifications in this PATH message. No other object within the PATH message is affected by this IntServ extension.

This extension modifies the sender descriptor by specifically augmenting it to allow an optional <MULTI_TSPEC> object after the optional <ADSPEC>, as shown below.

```
          <sender descriptor> ::= <SENDER_TEMPLATE> <SENDER_TSPEC>

                                  [ <ADSPEC> ]  [ <MULTI_TSPEC> ]
                                                  ^^^^^^^^^^^^
```

As can be seen above, the MULTI_TSPEC is in addition to the SENDER_TSPEC - and is only to be used, per this extension, when more than one TSPEC is to be included in the PATH message.

Here is another way of looking at the proposal choices:

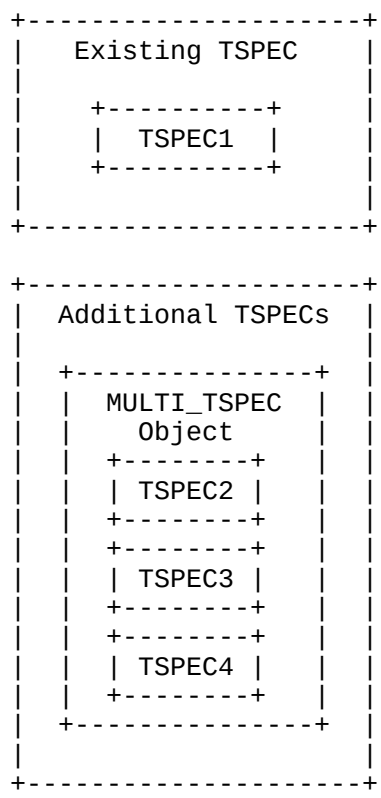


Figure 3. Encoding of Multiple Traffic Specifications in the TSPEC and MULTI_TSPEC objects

This solution is backwards compatible with existing implementations of [RFC2205] and [RFC2210], as the multiple TSPECs and FLOWSPECs are inserted as optional objects and such objects do not need to be processed, especially if they are not understood.

This solution defines a similar approach for encoding multiple flow specifications in the RESV message. Flow specifications beyond the first one can be encoded in a new "MULTI_FLOWSPEC" object contained

in the RESV message.

In this proposal, the original SENDER_TSPEC and the FLOWSPEC are left untouched, allowing routers not supporting this extension to process the PATH and the RESV message without issue. Two new additional objects are defined in this document. They are the MULTI_TSPEC and the MULTI_FLOWSPEC for the PATH and the RESV message, respectively. The additional TSPECs (in the new MULTI_TSPEC Object) are included in the PATH and the additional FLOWSPECS (in the new MULTI_FLOWSPEC Object) are included in the RESV message as new (optional) objects. These additional objects will have a class number of 11bbbbbb, allowing older routers to ignore the object(s) and forward each unexamined and unchanged, as defined in section 3.10 of [RFC 2205].

NOTE: it is important to emphasize here that including more than one FLOWSPEC in the RESV message does not cause more than one FLOWSPEC to be granted. This document requires that the receiver arrange these multiple FLOWSPECs in the order of preference according to the order remaining from the MULTI_TSPECs in the PATH message. The benefit of this arrangement is that RSVP does not have to process the rest of the FLOWSPEC if it can admit the first one.

3. Multi_TSPEC and MULTI_FLOWSPEC Solution

For the Sender Descriptor within the PATH message, the original TSPEC remains where it is, and is untouched by this IntServ extension. What is new is the use of a new <MULTI_TSPEC> object inside the sender descriptor as shown here:

```
        <sender descriptor> ::= <SENDER_TEMPLATE> <SENDER_TSPEC>

                                [ <ADSPEC> ]  [ <MULTI_TSPEC> ]
                                                ^^^^^^^^^^^
```

The preferred order of TSPECs sent by the sender is this:

- preferred TSPEC is in the original SENDER_TSPEC

- the next in line preferred TSPEC is the first TSPEC in the MULTI_TSPEC object

- the next in line preferred TSPEC is the second TSPEC in the MULTI_TSPEC object

- and so on...

The composition of the flow descriptor list in a Resv message depends upon the reservation style. Therefore, the following shows

the inclusion of the MULTI_FLOWSPEC object with each of the styles:

WF Style:

```
<flow descriptor list> ::=  <WF flow descriptor>

<WF flow descriptor> ::= <FLOWSPEC> [MULTI_FLOWSPEC]
```

FF style:

```
<flow descriptor list> ::=

          <FLOWSPEC>  <FILTER_SPEC>  [MULTI_FLOWSPEC] |

          <flow descriptor list> <FF flow descriptor>

<FF flow descriptor> ::=

          [ <FLOWSPEC> ] <FILTER_SPEC> [MULTI_FLOWSPEC]
```

SE style:

```
<flow descriptor list> ::= <SE flow descriptor>

<SE flow descriptor> ::=

         <FLOWSPEC> <filter spec list> [MULTI_FLOWSPEC]

<filter spec list> ::=  <FILTER_SPEC>

                  |  <filter spec list> <FILTER_SPEC>
```

3.1 New MULTI_TSPEC and MULTI-RSPEC Parameters

This extension to Integrated Services defines two new parameters They are:

1. <parameter name> Multiple_Token_Bucket_Tspec, with a parameter number of 125.

2. <parameter name> Multiple_Guaranteed_Service_RSpec with a parameter number of 124

These are IANA registered in this document.

The original SENDER_TSPEC and FLOWSPEC for Controlled Service maintain the <parameter name> of Token_Bucket_Tspec with a parameter number of 127. The original FLOWSPEC for Guaranteed Service maintains the <parameter name> of Guaranteed_Service_RSpec with a parameter number of 130.

3.2 Multiple TSPEC in a PATH Message

Here is the object from [RFC2210]. It is used as a SENDER_TSPEC in a PATH message:

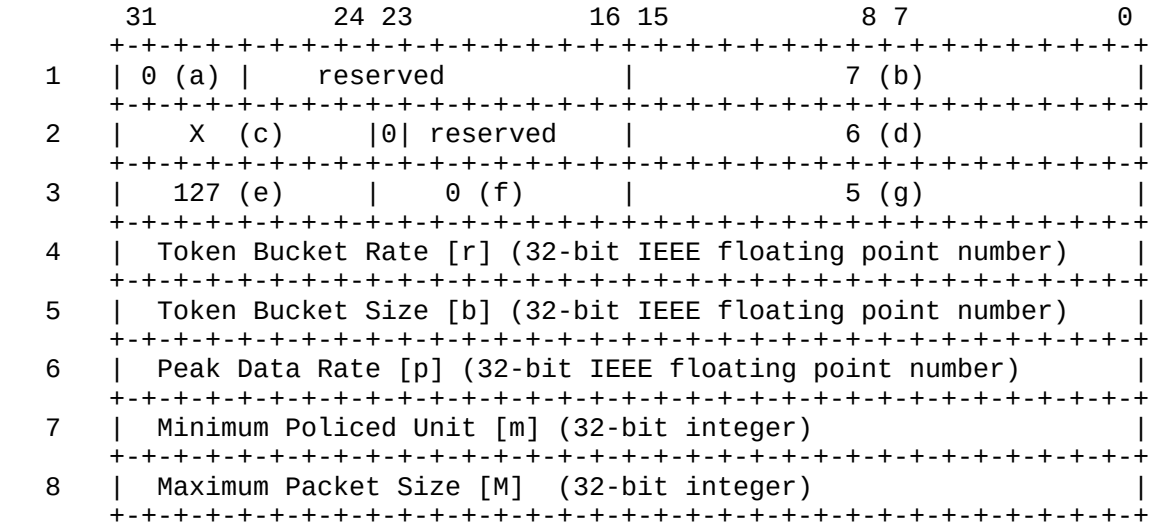
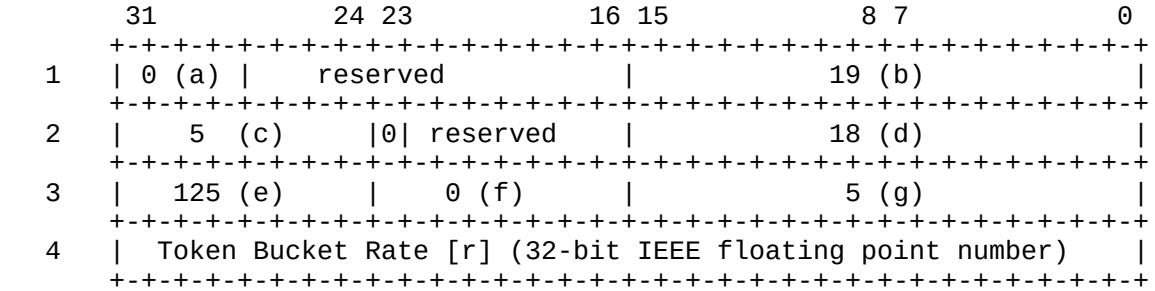


Figure 4. SENDER_TSPEC in PATH

(a) - Message format version number (0)
(b) - Overall length (7 words not including header)
(c) - Service header, service number
- '1' (Generic information) if in a PATH message;
(d) - Length of service data, 6 words not including per-service header
(e) - Parameter ID, parameter 127 (Token Bucket TSpec)
(f) - Parameter 127 flags (none set)
(g) - Parameter 127 length, 5 words not including per-service header

For completeness, Figure 4 is included in its original form for backwards compatibility reasons, as if there were only 1 TSPEC in the PATH. What is new when there are more than one TSPEC in this reservation message is the new MULTI_TSPEC object in Figure 5 containing, for example, 3 (Multiple_Token_Bucket_Tspec) TSPECs in a PATH message.



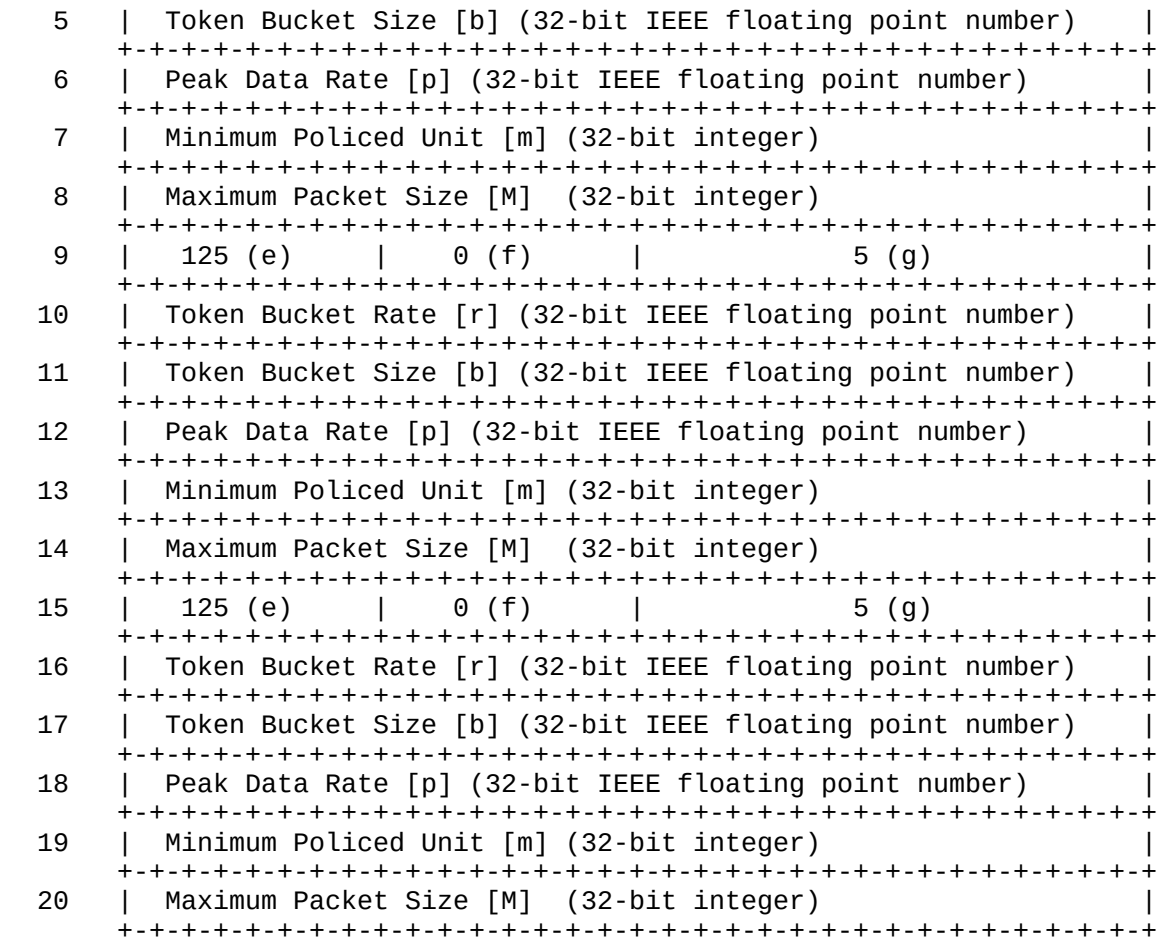


Figure 5. MULTI_TSPEC Object

(a) - Message format version number (0)
(b) - Overall length (19 words not including header)
(c) - Service header, service number 5 (Controlled-Load)
(d) - Length of service data, 18 words not including per-service header
(e) - Parameter ID, parameter 125 (Multiple Token Bucket TSpec)
(f) - Parameter 125 flags (none set)
(g) - Parameter 125 length, 5 words not including per-service header

Figure 5 shows the 2nd through Nth TSPEC in the PATH in the preferred order. The message format (a) remains the same for a second TSPEC and for other additional TSPECs.

The Overall Length (b) includes all the TSPECs within this object, plus the 2nd Word (containing fields (c) and (d)), which MUST NOT be repeated. The service header fields (e),(f) and(g) are repeated for

each TSPEC.

The Service header, here service number 5 (Controlled-Load) MUST remain the same.

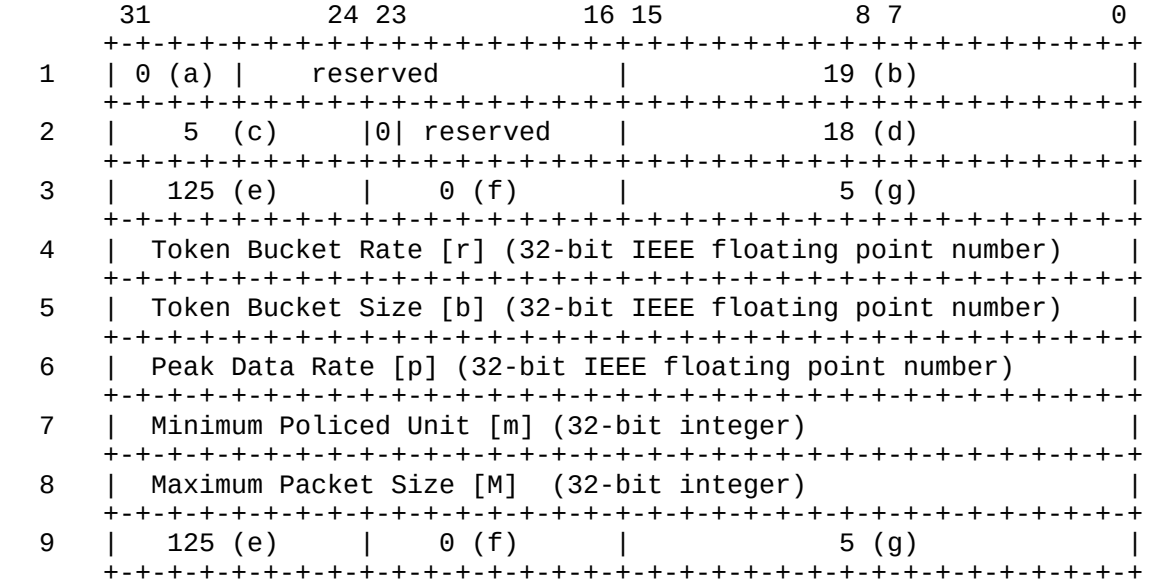
Each TSPEC is six 32-bit Words long (the per-service header plus the 5 values that are 1 Word each in length), therefore the length is in 6 Word increments for each additional TSPEC. Case in point, from the above Figure 5, Words 3-8 are the first TSPEC (2nd preferred), Words 9-14 are the next TSPEC (3rd preferred), and Words 15-20 are the final TSPEC (and 4th preferred) in this example of 3 TSPECs in this MULTI_TSPEC object. There is no limit placed on the number of TSPECs a MULTI_TSPEC object can have. However, it is RECOMMENDED to administratively limit the number of TSPECs in the MULTI_TSPEC object to 9 (making for a total of 10 in the PATH message).

The TSPECS are included in the order of preference by the message generator (PATH) and MUST be maintained in that order all the way to the Receiver. The order of TSPECs that are still grantable, in conjunction with the ADSPEC at the Receiver, MUST retain that order in the FLOWSPEC and MULTI_FLOWSPEC objects.

3.3 Multiple FLOWSPEC for Controlled-Load service

The format of an RSVP FLOWSPEC object requesting Controlled-Load service is the same as the one used for the SENDER_TSPEC given in Figure 4.

The format of the new MULTI_FLOWSPEC object is given below:



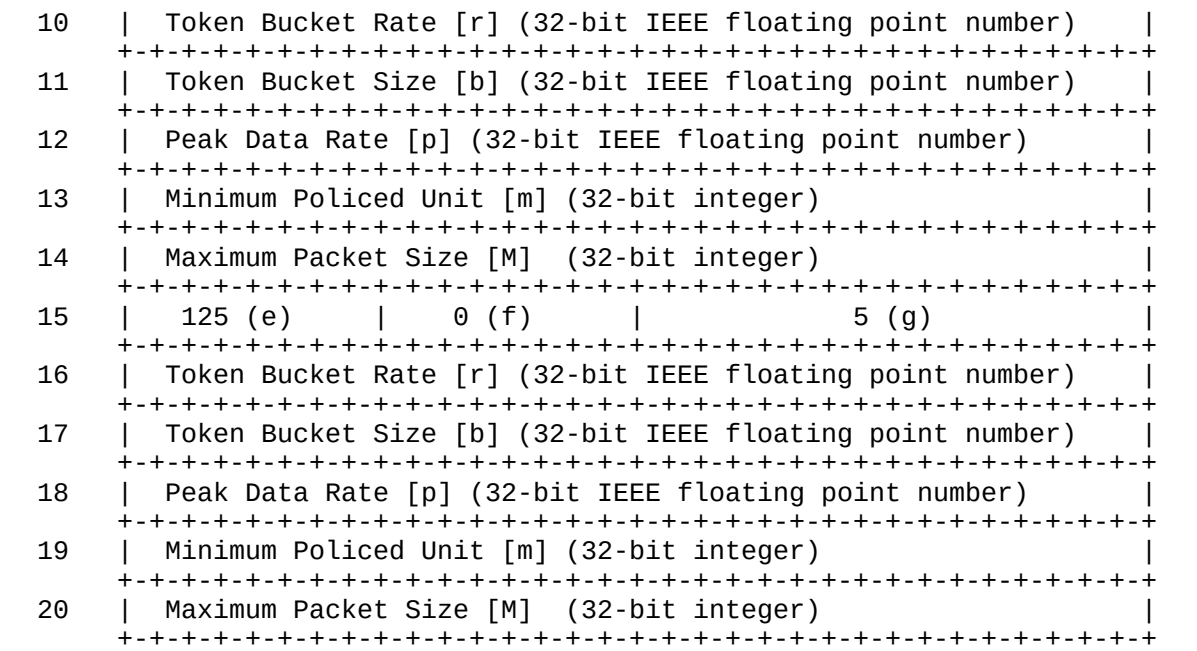


Figure 5. Multiple FLOWSPEC for Controlled-Load service

(a) - Message format version number (0)
(b) - Overall length (19 words not including header)
(c) - Service header, service number 5 (Controlled-Load)
(d) - Length of controlled-load data, 18 words not including per-service header
(e) - Parameter ID, parameter 125 (Multiple Token Bucket TSpec)
(f) - Parameter 125 flags (none set)
(g) - Parameter 125 length, 5 words not including per-service header

This is for the 2nd through Nth TSPEC in the RESV, in the preferred order.

The message format (a) remains the same for a second TSPEC and for additional TSPECs.

The Overall Length (b) includes the TSPECs, plus the 2nd Word (fields (c) and (d)), which MUST NOT be repeated. The service header fields (e),(f) and(g), which are repeated for each TSPEC.

The Service header, here service number 5 (Controlled-Load) MUST remain the same for the RESV message. The services, Controlled-Load and Guaranteed MUST NOT be mixed within the same RESV message. In other words, if one TSPEC is a Controlled Load service TSPEC, the remaining TSPECs MUST be Controlled Load service. This same rule also is true for Guaranteed Service - if one TSPEC is for Guaranteed

Service, the rest of the TSPECs in this PATH or RESV MUST be for Guaranteed Service.

The Length of controlled-load data (d) also increases to account for the additional TSPECs.

Each FLOWSPEC is six 32-bit Words long (the per-service header plus the 5 values that are 1 Word each in length), therefore the length is in 6 Word increments for each additional TSPEC. Case in point, from the above Figure 5, Words 3-8 are the first TSPEC (2nd preferred), Words 9-14 are the next TSPEC (3rd preferred), and Words 15-20 are the final TSPEC (and 4th preferred) in this example of 3 TSPECs in this FLOWSPEC. There is no limit placed on the number of TSPECs a particular FLOWSPEC can have.

Within the MULTI_FLOWSPEC, any SENDER_TSPEC that cannot be reserved - based on the information gathered in the ADSPEC, is not placed in the RESV or based on other information available to the receiver. Otherwise, the order in which the TSPECs were in the PATH message MUST be in the same order they are in the FLOWSPEC in the RESV. This is the order of preference of the sender, and MUST be maintained throughout the reservation establishment, unless the ADSPEC indicates one or more TSPECs cannot be granted, or the receiver cannot include any TSPEC due to technical or administrative constraints or one or more routers along the RESV path cannot grant a particular TSPEC. At any router that a reservation cannot honor a TSPEC, this TSPEC MUST be removed from the RESV, or else another router along the RESV path might reserve that TSPEC. This rule ensures this cannot happen.

Once one TSPEC has been removed from the RESV, the next in line TSPEC becomes the preferred TSPEC for that reservation. That router MUST generate a ResvErr message, containing an ERROR_SPEC object with a Policy Control Failure with Error code = 2 (Policy Control Failure), and an Error Value sub-code 102 (ERR_PARTIAL_PREEMPT) to the previous routers, clearing the now over allocation of bandwidth for this reservation. The difference between the previously accepted TSPEC bandwidth and the currently accepted TSPEC bandwidth is the amount this error identifies as the amount of bandwidth that is no longer required to be reserved. The ResvErr and the RESV messages are independent, and not normally sent by the same router. This aspect of this document is the extension to RFC 2205 (RSVP).

If a RESV cannot grant the final TSPEC, normal RSVP rules apply with regard to the transmission of a particular ResvErr.

3.4 Multiple FLOWSPEC for Guaranteed service

The FLOWSPEC object, which is used to request guaranteed service contains a TSPEC and RSpec. Here is the FLOWSPEC object from [RFC2215] when requesting Guaranteed service:

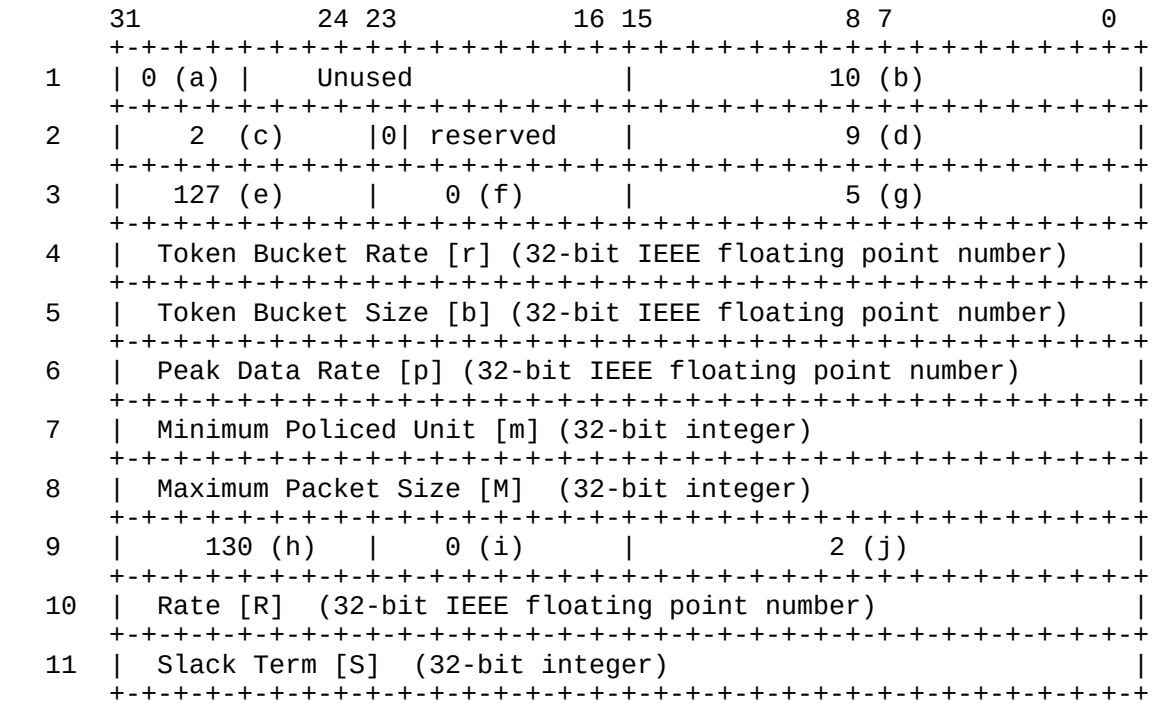
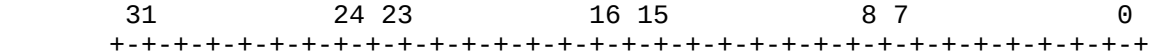


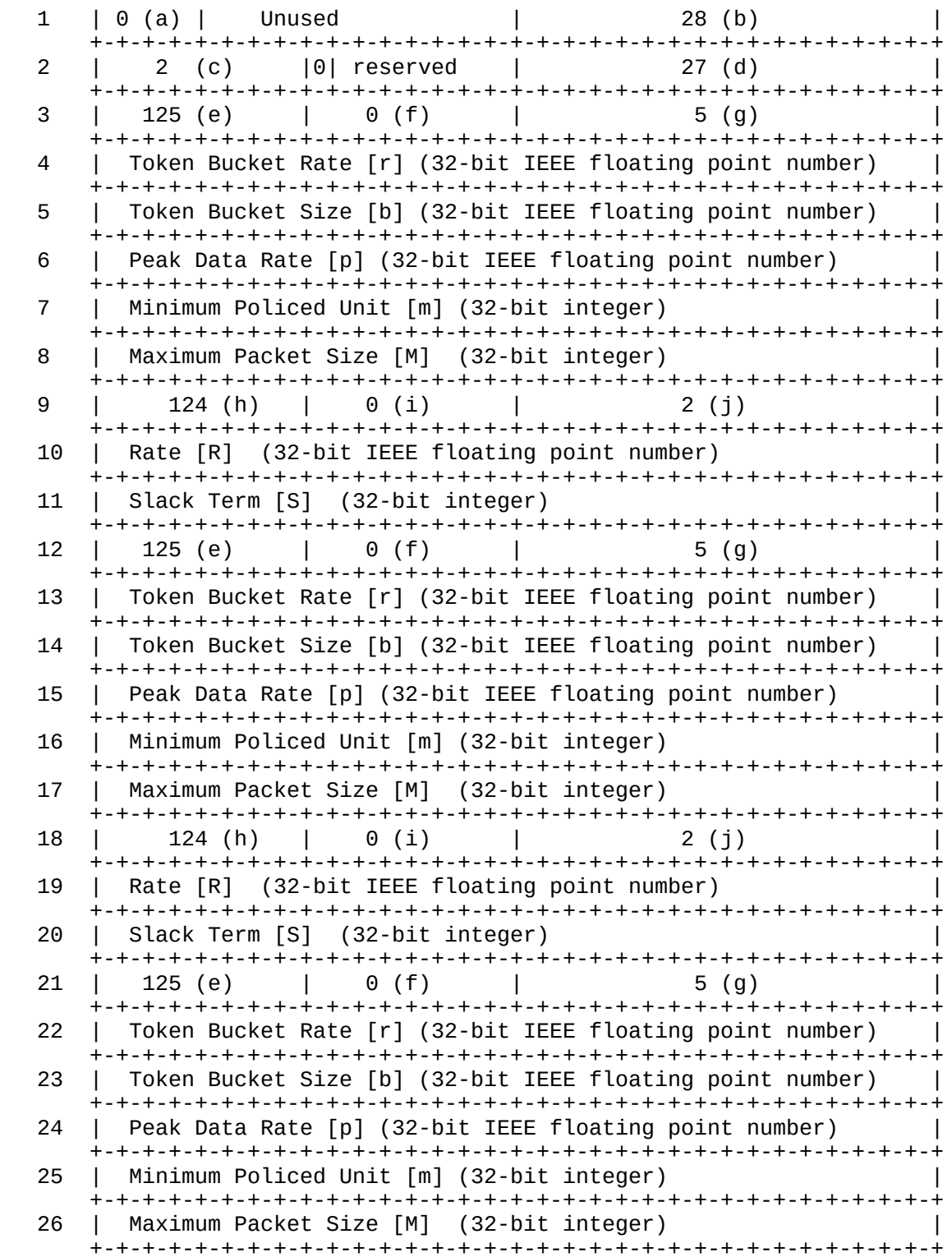
Figure 6. FLOWSPEC for Guaranteed service

(a) - Message format version number (0)
(b) - Overall length (9 words not including header)
(c) - Service header, service number 2 (Guaranteed)
(d) - Length of per-service data, 9 words not including per-service header
(e) - Parameter ID, parameter 127 (Token Bucket TSpec)
(f) - Parameter 127 flags (none set)
(g) - Parameter 127 length, 5 words not including parameter header
(h) - Parameter ID, parameter 130 (Guaranteed Service RSpec)
(i) - Parameter xxx flags (none set)
(j) - Parameter xxx length, 2 words not including parameter header

The difference in structure between the Controlled-Load FLOWSPEC and Guaranteed FLOWSPEC is the RSPEC, defined in [RFC2212].

For completeness, Figure 6 is included in its original form for backwards compatibility reasons, as if there were only 1 FLOWSPEC in the RESV. What is new when there is more than one TSPEC in the FLOWSPEC in a RESV message is the new MULTI_FLOWSPEC object in Figure 7 containing, for example, 3 FLOWSPECs requesting Guaranteed Service.




1 | 0 (a) | Unused | 28 (b) |
2 | 2 (c) |0| reserved | 27 (d) |
3 | 125 (e) | 0 (f) | 5 (g) |
4 | Token Bucket Rate [r] (32-bit IEEE floating point number) |
5 | Token Bucket Size [b] (32-bit IEEE floating point number) |
6 | Peak Data Rate [p] (32-bit IEEE floating point number) |
7 | Minimum Policed Unit [m] (32-bit integer) |
8 | Maximum Packet Size [M] (32-bit integer) |
9 | 124 (h) | 0 (i) | 2 (j) |
10 | Rate [R] (32-bit IEEE floating point number) |
11 | Slack Term [S] (32-bit integer) |
12 | 125 (e) | 0 (f) | 5 (g) |
13 | Token Bucket Rate [r] (32-bit IEEE floating point number) |
14 | Token Bucket Size [b] (32-bit IEEE floating point number) |
15 | Peak Data Rate [p] (32-bit IEEE floating point number) |
16 | Minimum Policed Unit [m] (32-bit integer) |
17 | Maximum Packet Size [M] (32-bit integer) |
18 | 124 (h) | 0 (i) | 2 (j) |
19 | Rate [R] (32-bit IEEE floating point number) |
20 | Slack Term [S] (32-bit integer) |
21 | 125 (e) | 0 (f) | 5 (g) |
22 | Token Bucket Rate [r] (32-bit IEEE floating point number) |
23 | Token Bucket Size [b] (32-bit IEEE floating point number) |
24 | Peak Data Rate [p] (32-bit IEEE floating point number) |
25 | Minimum Policed Unit [m] (32-bit integer) |
26 | Maximum Packet Size [M] (32-bit integer) |

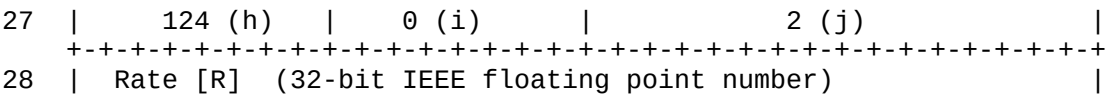


Figure 7. Multiple FLOWSPECs for Guaranteed service

(a) - Message format version number (0)
(b) - Overall length (9 words not including header)
(c) - Service header, service number 2 (Guaranteed)
(d) - Length of per-service data, 9 words not including per-service header
(e) - Parameter ID, parameter 125 (Token Bucket TSpec)
(f) - Parameter 125 flags (none set)
(g) - Parameter 125 length, 5 words not including parameter header
(h) - Parameter ID, parameter 124 (Guaranteed Service RSpec)
(i) - Parameter 124 flags (none set)
(j) - Parameter 124 length, 2 words not including parameter header

There MUST be 1 RSPEC per TSPEC for Guaranteed Service. Therefore, there are 5 words for Receiver TSPEC and 3 words for the RSPEC. Therefore, for Guaranteed Service, the TSPEC/RSPEC combination occurs in increments of 8 words.

4. Rules of Usage

The following rules apply to nodes adhering to this specification:

4.1 Backward Compatibility

If the recipient does not understand this extension, it ignores this MULTI_TSPEC object, and operates normally for a node receiving this RSVP message.

4.2 Applies to Only a Single Session

When there is more than one TSPEC object or more than one FLOWSPEC object, this MUST NOT be considered for more than one flow created. These are OR choices for the same flow of data. In order to attain three reservations between two endpoints, three different reservation requests are required, not one reservation request with 3 TSPECs.

4.3 No Special Error Handling for PATH Message

If a problem occurs with the PATH message - regardless of this

extension, normal RSVP procedures apply (i.e., there is no new PathErr code created within this extension document) - resulting in a PathErr message being sent upstream towards the sender, as usual.

4.4 Preference Order to be Maintained

When more than one TSPEC is in a PATH message, the order of TSPECs is decided by the Sender and MUST be maintained within the SENDER_TSPEC. The same order MUST be carried to the FLOWSPECs by the receiver. No additional TSPECS can be introduced by the receiver or any router processing these new objects. The deletion of TSPECs from a PATH message is not permitted. The deletion of the TSPECs when forming the FLOWSPEC is allowed by the receiver in the following cases:

- If one or more preferred TSPECs cannot be granted by a router as discovered during processing of the ADSPEC by the receiver, then they can be omitted when creating the FLOWSPEC(s) from the TSPECs.

- If one or more TSPECs arriving from the sender is not preferred by the receiver, then the receiver MAY omit any while creating the FLOWSPEC. A good reason to omit a TSPEC is if, for example, it does not match a codec supported by the receiver’s application(s).

The deletion of the TSPECs in the router during the processing of this MULTI_FLOWSPEC object is allowed in the following cases:

- If the original FLOWSPEC cannot be granted by a router then the router may discard that FLOWSPEC and replace it with the topmost FLOWSPEC from the MULTI_FLOWSPEC project. This will cause the topmost FLOWSPEC in the MULTI_FLOWSPEC object to be removed. The next FLOWSPECs becomes the topmost FLOWSPEC.

- If the router merges multiple RESV into a single RESV message, then the FLOWSPEC and the multiple FLOWSPEC may be affected

The preferred order of the remaining TSPECs or FLOWSPECs MUST be kept intact both at the receiver as well as the router processing these objects.

4.5 Bandwidth Reduction in Downstream Routers

If there are multiple FLOWSPECs in a single RESV message, it is quite possible that a higher bandwidth is reserved at a previous downstream device. Thus, any device that grants a reservation that is not the highest will have to inform the previous downstream routers to reduce the bandwidth reserved for this particular session.

The bandwidth reduction RFC [RFC4495] has the ability to partially

preempt existing reservations. However, it does not address the need that this draft addresses. RFC 4495 defines an ability to preempt part of an existing reservation so as to admit a new incoming reservation with a higher priority, in lieu of tearing down the whole reservation with lower priority. It does not specify the capability to reduce the bandwidth a RESV set up along the data path before the reservation is realized (from source to destination), when a subsequent router cannot support a more preferred FLOWSPEC contained in that RESV. This document will extend the RFC 4495 defined error to work for previous hops while a reservation is being established.

4.6 Merging Rules

RFC 2205 defines the rules for merging as combining more than one FLOWSPEC into a single FLOWSPEC. In the case of MULTI_FLOWSPECs, merging of the two (or more) MULTI_FLOWSPEC MUST be done to arrive at a single MULTI_FLOWSPEC. The merged MULTI_FLOWSPEC will contain all the flow specification components of the individual MULTI_FLOWSPECs in descending orders of bandwidth. In other words, the merged FLOWSPEC MUST maintain the relative order of each of the individual FLOWSPECs. For example, if the individual FLOWSPEC order is 1,2,3 and another FLOWSPEC is a,b,c, then this relative ordering cannot be altered in the merged FLOWSPEC.

A byproduct of this is the ordering between the two individual FLOWSPECs cannot be signaled with this extension. If two (or more) FLOWSPECs have the same bandwidth, they are to be merged into one FLOWSPEC using the rules defined in RFC 2205. It is RECOMMENDED that the following rules are used for determining ordering (in TSPEC and FLOWSPEC):

> For Controlled Load - in descending order of BW based on the Token Bucket Rate ’r’ parameter value
>
> For Guaranteed Service - in descending order of BW based on the RSPEC Rate ’R’ parameter value

The resultant FLOWSPEC is added to the MULTI_FLOWSPEC based on its bandwidth in descending orders of bandwidth.

As a result of such merging, the number of FLOWSPECs in a MULTI_FLOWSPEC object should be the sum of the number of FLOWSPECs from individual MULTI_FLOWSPEC that have been merged *minus* the number of duplicates.

4.7 Applicability to Multicast

An RSVP message with a MULTI_TSPEC works just as well in a multicast scenario as it does in a unicast scenario. In a multicast scenario, the bandwidth allotted in each hop is the lowest bandwidth that can

be admitted along the various path. For example:

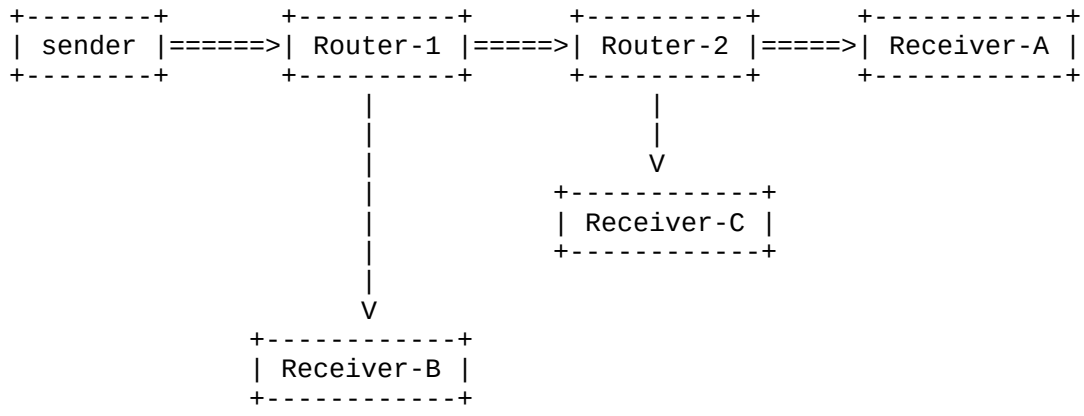


Figure 8. MULTI_TPSEC and Multicast

If the sender (in Figure 8) sends 3 TSPECs (i.e., 1 TSPEC Object, and 2 in the MULTI_TSPEC Object) of 12Mbps, 5Mbps and 1.5Mbps. Let us say the path from Receiver-B to Router-1 admitted 5Mbps, Receiver-C to Router-2 admitted 1.5Mbps and Receiver-A to Router-2 admitted 12Mbps.

When the Resv message is send upstream from Router-2, the combining of 1.5Mbps (to Receiver-C) and 12Mbps (to Receiver-A) will be resolved to 1.5Mbps (lowest that can be admitted). Only a Resv with 1.5Mbps will be sent upstream from Router-2. Likewise, at Router-1, the combining of 1.5Mbps (to Router-2) and 5Mbps (to Receiver-B) will be resolved to 1.5Mbps units.

This is to allow the sender to transmit the flow at a rate that can be accepted by all devices along the path. Without this, if Router-2 receives a flow of 12Mbps, it will not know how to create a flow of 1.5Mbps down to Receiver-B. A differentiated reservation for the various paths along a multicast path is only possible with a Media-aware network device (MANE). The discussion of MANE and how it relates to admission control is outside the scope of this draft.

4.8 MULTI_TSPEC Specific Error

Since this mechanism is backward compatible, it is possible that a router without support for this MULTI_TSPEC extension will reject a reservation because the bandwidth indicated in the primary FLOWSPECs is not available. This means that an attempt with a lower bandwidth might have been successful, if one were included in a MULTI_TSPEC Object. Therefore, one should be able to differentiate between an admission control error where there is insufficient bandwidth when all the FLOWSPECs are considered and insufficient bandwidth when

only the primary FLOWSPEC is considered.

This requires the definition of an error code within the ERROR_SPEC Object. When a router does not have sufficient bandwidth even after considering all the FLOWSPEC provided, it issues a new "MULTI_TSPEC bandwidth unavailable " error. This will be an Admission Control Failure (error #1), with a subcode of 6. A router that does not support this MULTI_TSPEC extension will return the "requested bandwidth unavailable" error as defined in RFC 2205 as if there was no MULTI_TSPEC in the message.

4.9 Other Considerations

- RFC 4495 articulates why a ResvErr is more appropriate to use for reducing the bandwidth of an existing reservation vs. a ResvTear.

- Refreshes only include the TSPECs that were accepted. One SHOULD be sent immediately upon the Sender receiving the RESV, to ensure all routers in this flow are synchronized with which TSPEC is in place.

- Periodically, it might be appropriate to attempt to increase the bandwidth of an accepted reservation with one of the TSPECs that were not accepted by the network when the reservation was first installed. This SHOULD NOT occur too regularly. This document currently offers no guidance on the frequency of this bump request for a rejected TSPEC from the PATH.

4.10 Known Open Issues

Here are the know open issues within this document:

o Both the idea of MULTI_RSPEC and MULTI_FLOWSPEC need to be fleshed out, and IANA registered.

o Need to ensure the cap on the number of TSPECs and FLOWSPECs is viable, yet controlled.

5. Security considerations

The security considerations for this document do not exceed what is already in RFC 2205 (RESV) or RFC 2210 (IntServ), as nothing in either of those documents prevent a node from requesting a lot of bandwidth in a single TSPEC. This document merely reduces the signaling traffic load on the network by allowing many requests that fall under the same policy controls to be included in a single round-trip message exchange.

Further, this document does not increase the security risk(s) to

that defined in RFC 4495, where this document creates additional meaning to the RFC 4495 created error code 102.

A misbehaving Sender can include too many TSPECs in the MULTI_TSPEC object, which can lead to an amplification attack. That said, a bad implementation can create a reservation for each TSPEC received from within the Resv message. The number of TSPECs in the new MULTI_TSPEC object is limited, and the spec clearly states that only a single reservation is to be set up per Resv message.

6. IANA considerations

This document IANA registers the following new parameter name in the Integ-serv assignments at [IANA]:

Registry Name: Parameter Names
Registry:

Value	Description	Reference
125	Multiple_Token_Bucket_Tspec	[RFCXXXX]
124	Multiple_Guaranteed_Service_RSpec	[RFCXXXX]

Where RFCXXXX is replaced with the RFC number assigned to this Document.

This document IANA registers the following new error subcode in the Error code section, under the Admission Control Failure (error=1), of the rsvp-parameters assignments at [IANA]:

Registry Name: Error Codes and Globally-Defined Error Value Sub-Codes
Registry:
"Admission Control Failure"

Error		Subcode	meaning	Reference
6	=		MULTI_TSPEC bandwidth unavailable	[RFCXXXX]

7. Acknowledgments

The authors wish to thank Fred Baker, Joe Touch, Bruce Davie, Dave Oran, Ashok Narayanan, Lou Berger, Lars Eggert, Arun Kudur and Janet Gunn for their helpful comments and guidance in this effort.

And to Francois Le Faucheur, who provided text in this version.

8. References

8.1. Normative References

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

[RFC2205] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, S. Jamin, "Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification", RFC 2205, September 1997

[RFC2210] J. Wroclawski, "The Use of RSVP with IETF Integrated Services", RFC 2210, September 1997

[RFC2212] S. Shenker, C. Partridge, R. Guerin, "Specification of Guaranteed Quality of Service", RFC 2212, September 1997

[RFC2215] S. Shenker, J. Wroclawski, "General Characterization Parameters for Integrated Service Network Elements", RFC 2212, September 1997

[RFC4495] J. Polk, S. Dhesikan, "A Resource Reservation Protocol (RSVP) Extension for the Reduction of Bandwidth of a Reservation Flow", RFC 4495, May 2006

8.2. Informative References

[IANA] http://www.iana.org/assignments/integ-serv

Authors' Addresses

James Polk
3913 Treemont Circle
Colleyville, Texas, USA
+1.817.271.3552

mailto: jmpolk@cisco.com

Subha Dhesikan
Cisco Systems
170 W. Tasman Drive
San Jose, CA 95134 USA

mailto: sdhesika@cisco.com

Appendix A: Alternatives for Sending Multiple TSPECs

This appendix describes the discussion within the TSVWG of which approach best fits the requirements of sending multiple TSPECs within a single PATH or RESV message. There were 3 different options proposed, of which - 2 were insufficient or caused more harm

than other options.

Looking at the format of a PATH message [RFC2205] again:

```
        <PATH Message> ::= <Common Header> [ <INTEGRITY> ]

                                    <SESSION> <RSVP_HOP>

                                    <TIME_VALUES>

                                   [ <POLICY_DATA> ... ]

                                   [ <sender descriptor> ]

        <sender descriptor> ::= <SENDER_TEMPLATE> <SENDER_TSPEC>
                                                   ^^^^^^^^^^^^
                                   [ <ADSPEC> ]
```

For the PATH message, the focus of this document is with what to do with respect to the <SENDER_TSPEC> above, highlighted by the '^^^^' characters. No other object within the PATH message will be affected by this IntServ extension.

The ADSPEC is optional in IntServ; therefore it might or might not be in the RSVP PATH message. Presently, the SENDER_TSPEC is limited to one bandwidth associated with the session. This is changed in this extension to IntServ to multiple bandwidths for the same session. There are multiple options on how the additional bandwidths may be added:

Option #1 - creating the ability to add one or more additional (and complete) SENDER_TSPECs,

or

Option #2 - create the ability for the one already allowed SENDER_TSPEC to carry more than one bandwidth amount for the same reservation.

or

Option #3 - create the ability for the existing SENDER_TSPEC to remain unchanged, but add an optional <MULTI_TSPEC> object to the <sender descriptor> such as this:

```
        <sender descriptor> ::= <SENDER_TEMPLATE> <SENDER_TSPEC>

                                   [ <ADSPEC> ]  [ <MULTI_TSPEC> ]
                                                   ^^^^^^^^^^^^
```

Here is another way of looking at the option choices:

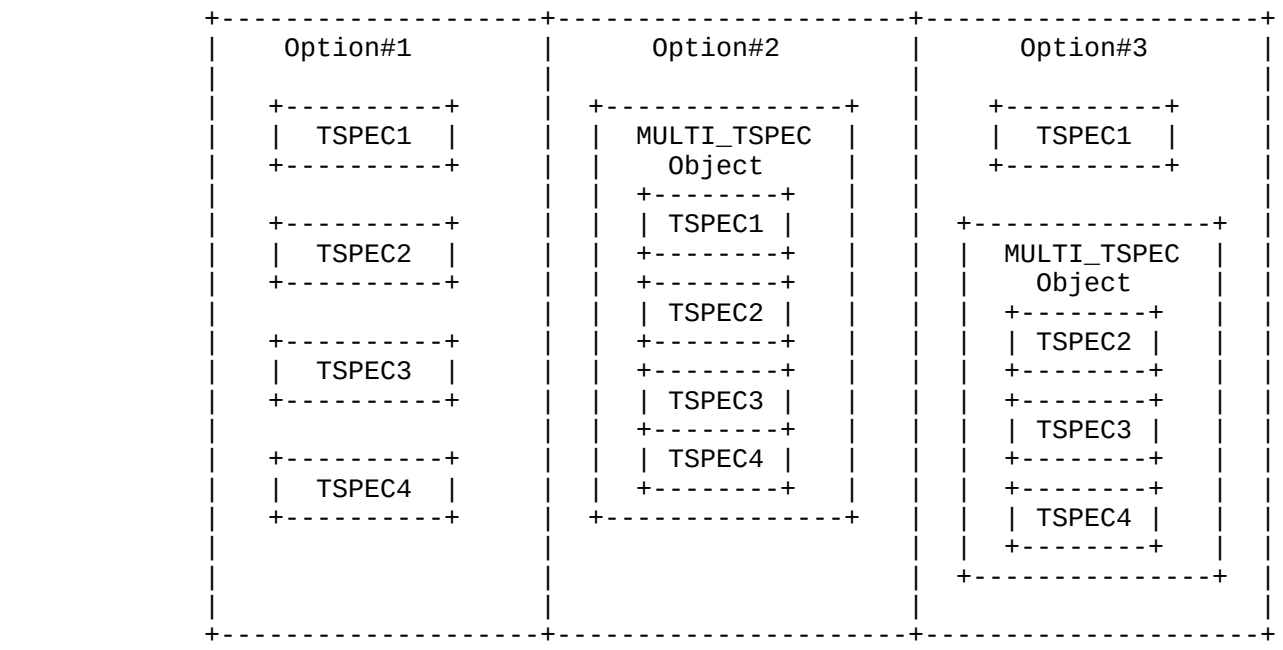


Figure 3. Concept of Option Choice

Option #1 and #2 do not allow for backward compatibility. If the currently used SENDER_TSPEC and FLOWSPEC objects are changed, then unless all the routers requiring RSVP processing are upgraded, this functionality cannot be realized. As it is unlikely that all routers along the path will have the necessary enhancements as per this extension at one given time, therefore, it is necessary this enhancement be made in a way that is backward compatible. Therefore, option #1 and option #2 has been discarded in favor of option #3, which had WG consensus in a recent IETF meeting.

Option #3: This option has the advantage of being backwards compatible with existing implementations of [RFC2205] and [RFC2210], as the multiple TSPECs and FLOWSPECs are inserted as optional objects and such objects do not need to be processed, especially if they are not understood.

Option#3 applies to the FLOWSPEC contained in the RESV message as well. In this option, the original SENDER_TSPEC and the FLOWSPEC are left untouched, allowing routers not supporting this extension to be able to process the PATH and the RESV message without issue. Two new additional objects are defined in this document. They are the MULTI_TSPEC and the MULTI_FLOWSPEC for the PATH and the RESV message, respectively. The additional TSPECs (in the new MULTI_TSPEC Object) are included in the PATH and the additional FLOWSPECS (in the new MULTI_FLOWSPEC Object) are included in the RESV message as new (optional) objects. These additional objects will have a class number of 11bbbbbb, allowing older routers to ignore the object(s)

and forward each unexamined and unchanged, as defined in section 3.10 of [RFC 2205].

We state in the document body that the top most FLOWSPEC of the new MULTI_FLOWSPEC Object in the RESV message replaces the existing FLOWSPEC when it is determined by the receiver (perhaps along with the ADSPEC) that the original FLOWSPEC cannot be granted. Therefore, the ordering of preference issue is solved with Option#3 as well.

NOTE: it is important to emphasize here that including more than one FLOWSPEC in the RESV message does not cause more than one FLOWSPEC to be granted. This document requires that the receiver arrange these multiple FLOWSPECs in the order of preference according to the order remaining from the MULTI_TSPECs in the PATH message. The benefit of this arrangement is that RSVP does not have to process the rest of the FLOWSPEC if it can admit the first one.

Additional details of these options can be found in the draft-polk-tsvwg-...-01 version of this appendix (which includes the RSVP bit mapping of fields in the TSPECs, if the reader wishes to search for that doc.

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 12, 2012

M. Tuexen
Muenster Univ. of Appl. Sciences
R. Stewart
Adara Networks
July 11, 2011

UDP Encapsulation of SCTP Packets
draft-tuexen-sctp-udp-encaps-07.txt

Abstract

This document describes a simple method of encapsulating SCTP Packets into UDP packets and its limitations. This allows the usage of SCTP in networks with legacy NAT not supporting SCTP. It can also be used to implement SCTP on hosts without directly accessing the IP-layer, for example implementing it as part of the application without requiring special privileges.

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 January 12, 2012.

Table of Contents

1. Introduction

This document describes a simple method of encapsulating SCTP packets into UDP packets. SCTP is defined in [RFC4960]. There are two main reasons for this:

- Allow SCTP traffic to pass legacy NATs, which do not provide native SCTP support as specified in [I-D.ietf-behave-sctpnat] and [I-D.ietf-tsvwg-natsupp].
- Allow SCTP to be implemented on hosts which do not provide direct access to the IP-layer. In particular, applications can use their own SCTP implementation if the operating system does not provide one.

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. Use Cases

This section discusses two important use cases for encapsulating SCTP into UDP.

3.1. Portable SCTP Implementations

Some operating systems support SCTP natively. For other operating systems implementations are available, but require special privileges to install and/or use them. In some cases no kernel implementation might be available at all. When proving an SCTP implementation as part of a user process, most operating systems require special privileges to access the IP layer directly.

Using UDP encapsulation makes it possible to provide an SCTP implementation as part of a user process which does not require any special privileges.

A crucial point for implementing SCTP in userland is controlling the source address of outgoing packets. This is not an issue when using all available addresses. However, this is not the case when also using the address management required for NAT traversal described in Section 4.7.

3.2. Legacy NAT traversal

Using UDP encapsulation allows an SCTP communication traversing legacy NATs not supporting SCTP as described in [I-D.ietf-behave-sctpnat] and [I-D.ietf-tsvwg-natsupp]. It is important to realize that for single homed associations it is only necessary that no IP addresses are listen in the INIT- and INIT-ACK chunks. Dynamic address reconfiguration to change the single address has to make use of wildcard addresses as described in [RFC5061].

For multi-homed SCTP association the address management as described in Section 4.7 MUST be performed.

4. SCTP over UDP

4.1. Architectural Considerations

An SCTP implementation supporting UDP encapsulation MUST store a UDP encapsulation port per destination address for each SCTP association.

4.2. Packet Format

To encapsulate an SCTP packet, a UDP header header as defined in [RFC0768] is inserted between the IP header and the SCTP common header.

Figure 1 shows the packet format of an encapsulated SCTP packet when IPv4 is used.

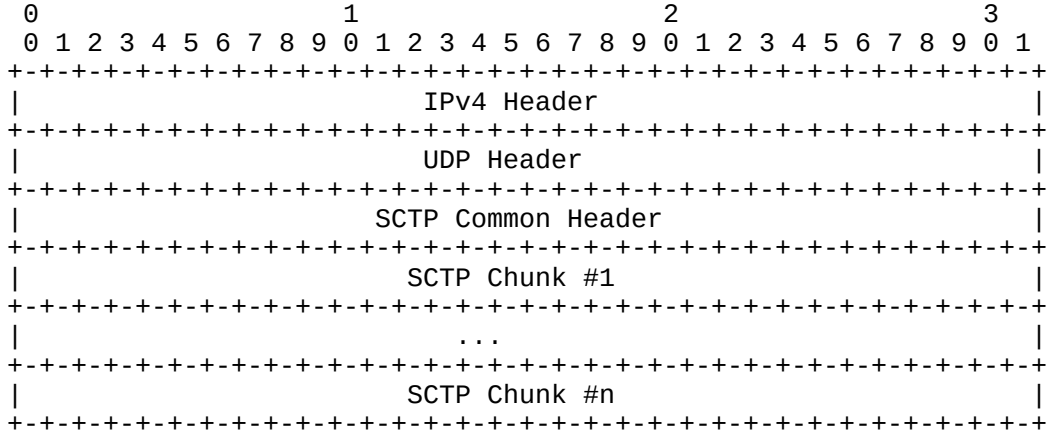


Figure 1

The packet format for an encapsulated SCTP packet when using IPv6 is shown in Figure 2. Please note the the number m of IPv6 extension headers can be 0.

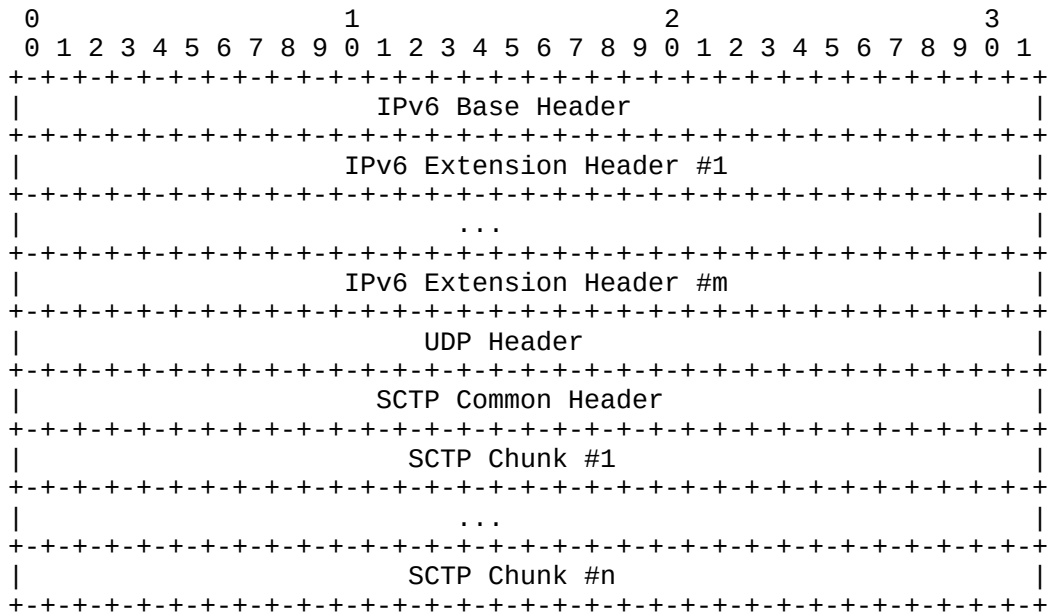


Figure 2

The UDP checksum MUST NOT be zero.

4.3. Encapsulation Procedure

When inserting the UDP header, the source port is 9899, the destination port is the one stored for the destination address the packet is sent to or 9899 if not destination address is stored.

The length of the UDP packet is the length of the SCTP packet plus the size of the UDP header.

The checksum MUST be computed.

4.4. Decapsulation Procedure

When an encapsulated packet is received, the UDP header is removed. Then a lookup is performed to find the association the received SCTP packet belongs to. The UDP source port is stored as the encapsulation port of the SCTP destination address the received SCTP packet is sent from.

4.5. ICMP considerations

When receiving ICMP or ICMPv6 response packet, there might not be enough bytes in the payload to identify the SCTP association which the SCTP packet triggering the ICMP or ICMPv6 packet belongs to. If a received ICMP or ICMPv6 packet can to be related to a specific SCTP association, it MUST be discarded silently.

4.6. Path MTU considerations

If an SCTP endpoint starts to encapsulate the packets of a path, it MUST decrease the path MTU of that path by the size of an UDP header. If it stops encapsulating them, the path MTU MUST be increased by the size of an UDP header.

When performing path MTU discovery as described in [RFC4820] it MUST take into account that it cannot rely on the feedback provided by ICMP or ICMPv6 due to the limitation laid out in Section 4.5.

4.7. Handling of Embedded IP-addresses

When using UDP encapsulation is used for legacy NAT traversal, IP address that might be translated MUST NOT be put into any SCTP packet.

This means that an SCTP association is setup singled homed and the protocol extension [RFC5061] is used to add multiple address. Only wildcard addresses are put into the SCTP packet.

When addresses are changed during the lifetime of the association [RFC5061] MUST be used with wildcard addresses only.

4.8. ECN considerations

TBD

5. IANA Considerations

This document does not require any actions from IANA.

6. Security Considerations

Encapsulating SCTP into UDP does not add any additional security considerations to the ones given in [RFC4960] and [RFC5061].

7. Acknowledgments

The authors wish to thank Irene Ruengeler for her invaluable comments.

8. References

8.1. Normative References

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

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

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[RFC4820] Tuexen, M., Stewart, R., and P. Lei, "Padding Chunk and Parameter for the Stream Control Transmission Protocol (SCTP)", RFC 4820, March 2007.

[RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, March 2007.

[RFC4895] Tuexen, M., Stewart, R., Lei, P., and E. Rescorla, "Authenticated Chunks for the Stream Control Transmission Protocol (SCTP)", RFC 4895, August 2007.

[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.

8.2. Informative References

[I-D.ietf-behave-sctpnat] Stewart, R., Tuexen, M., and I. Ruengeler, "Stream Control Transmission Protocol (SCTP) Network Address Translation", draft-ietf-behave-sctpnat-05 (work in progress), June 2011.

[I-D.ietf-tsvwg-natsupp]
Stewart, R., Tuexen, M., and I. Ruengeler, "Stream Control Transmission Protocol (SCTP) Network Address Translation Support", draft-ietf-tsvwg-natsupp-01 (work in progress), June 2011.

Authors' Addresses

Michael Tuexen
Muenster University of Applied Sciences
Stegerwaldstr. 39
48565 Steinfurt
DE

Email: tuexen@fh-muenster.de

Randall R. Stewart
Adara Networks
Chapin, SC 29036
USA

Email: randall@lakerest.net