Multipath transmission is regarded as an effective way to improve the quality of service of data delivery. This document defines a multipath message transport protocol based on application level relay (MPMTP-AR), which is a special profile of multipath transport protocol suite defined in Multipath Transport System Based on Application-level Relay (MPTS-AR). MPMTP-AR provides reliable data delivery service over multiple paths. This document discusses related message format and mechanism to support MPMTP-AR.
extracted from this document must include Simplified BSD License text as described in section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
Table of Contents

1. Introduction ............................................. 6
2. Terminology .............................................. 7
3. Definition ................................................ 7
4. Goals ..................................................... 8
5. Overview .................................................. 9
6. Usage Scenarios .......................................... 10
7. MPMTP-AR Agent Behavior ................................. 11
   7.1 Consistent Behaviors ................................ 11
   7.2 Modified Behaviors .................................. 11
   7.3 Additional Behaviors ................................ 12
   7.3.1 Session management ............................... 12
   7.3.2 Flow partitioning and scheduling ............... 12
   7.3.3 Subflow packaging ............................... 12
   7.3.4 Subflow reporting ............................... 13
   7.4 New Behaviors ....................................... 13
   7.4.1 Sequenced Delivery ................................ 13
   7.4.2 Acknowledgement and Congestion Avoidance .... 14
8. MPMTP-AR Packet Format ................................. 14
   8.1 Overview ............................................. 15
   8.2 MPMTP-AR Packet Header ............................. 15
   8.3 Fixed Header Fields of MPMTP-AR Data Packet .... 15
   8.4 Fixed Header Fields of MPMTP-AR Control Packet .. 17
      8.4.1 Sender Report (SR) ............................. 19
      8.4.2 Receiver Report (RR) ......................... 20
      8.4.3 Keep Alive (KA) ............................... 21
      8.4.4 Session Initiation (SINIT) ................... 21
      8.4.5 Session Initiation Acknowledgement (SINIT ACK) .. 23
      8.4.6 State Cookie (COOKIE ECHO) ................ 24
      8.4.7 Cookie Acknowledgement (COOKIE ACK) ....... 25
      8.4.8 Subflow INIT Request (SubINIT) .............. 25
      8.4.9 Subflow INIT Acknowledgement (SubINIT ACK) .. 25
      8.4.10 Selective Acknowledgement (SACK) ............ 26
      8.4.11 Subflow Shutdown (SubSHUTDOWN) ............. 29
      8.4.12 Subflow Shutdown Acknowledgement (SubSHUTDOWN ACK) .. 30
      8.4.13 Session Shutdown (SSHUTDOWN) ............... 30
      8.4.14 Session Shutdown Acknowledgement (SSHUTDOWN ACK) .. 30
      8.4.15 Session Shutdown Complete (SSHUTDOWN COMPLETE) .. 31
      8.4.16 Abort (ABORT) .................................. 31
      8.4.17 Operation Error (ERROR) ..................... 31
      8.4.18 Path Feedback (PATH FEEDBACK) ............... 37
      8.4.19 Path Probe (PATH PROBE) .................... 38
9. MPMTP-AR Session State Diagram ....................... 39
10. Session Initialization ................................. 42
15.2.3.2 Blind Masquerade ....................................... 73
15.2.3.3 Improper Monopolization of Services ................... 73
16. Reference .......................................................... 74
  16.1 Normal Reference ............................................ 74
  16.2 Information Reference ....................................... 74
Authors’ Addresses ..................................................... 75
1. Introduction

This document defines a multipath message transport protocol based on application-level relay (MPMTP-AR). The MPMTP-AR works on framework of multipath transport system based on application-level relay (MPTS-AR)\[1\], and provides reliable end-to-end multipath transmission services.

As Internet evolves, the demands of Internet resources are ever-increasing, but these resources (in particular, bandwidth) cannot be fully utilized due to protocol constraints. If these resources could be used concurrently, network resource utilization and end user experience could be improved significantly.

Although current protocols such as TCP, MPTCP and SCTP support reliable or multipath transport, they have some limitations described as following:

1) TCP provides reliable and sequential data delivery service. But it cannot support multipath transport.

2) MPTCP \[4\] supports multipath transport. At least one of participates has to be multi-homed. The transmission uses multiple IP pairs (Source IP, Destination IP) simultaneously. Transmission between each IP pair keeps same with regular TCP \[6\]. Multi-homed host limits the range of usage.

3) SCTP is an alternative transport protocol capable of supporting several IP addresses per connection. The first versions of SCTP used multiple addresses in failover scenarios, but recent extensions have enabled it to support the simultaneous use of several paths.

All those limitations motivated the development of MPMTP-AR. MPMTP-AR overcome those limitations by application level relay, select acknowledgement and retransmission mechanisms. Part of mechanisms reference to SCTP \[3\].

Multipath transport has several benefits as following:

1) To increase the efficiency of the resource usage, and thus increase the network capacity available to agent.

2) Increased Throughput: In some cases, the relay path has better performance than the default path. Total transport capacity of multiple paths may meet the requirements of the high bit-rate.

3) To increase the resilience of the connectivity by protecting path from the failure of one.
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 [2].

3. Definition

1) Cumulative TSN Ack Point: The TSN of the last DATA packet acknowledged via the Cumulative TSN Ack field of a SACK.

2) Session ID: The unique identify of the session.

3) Message Authentication Code (MAC): An integrity check mechanism based on cryptographic hash functions using a secret key. Typically, message authentication codes are used between two parties that share a secret key in order to validate information transmitted between these parties. In MPMTP-AR, it is used by an agent to validate the State Cookie information that is returned from the peer in the COOKIE ECHO chunk. The term "MAC" has different meanings in different contexts. MPMTP-AR uses this term with the same meaning as in [7].

4) MPMTP-AR application (MPMTP-AR user): The logical higher-layer application entity which uses the services of MPMTP-AR.

5) Ordered Message: A user message that is delivered in order with respect to all previous user messages sent within the session.

6) Outstanding TSN (at an MPMTP-AR agent): A TSN (and the associated DATA packet) that has been sent by the agent but for which it has not yet received an acknowledgement.

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

8) Slow-Start Threshold (ssthresh): An MPMTP-AR variable. This is the threshold that the agent will use to determine whether to perform slow start or congestion avoidance on a particular path. Ssthresh is in number of bytes.

9) Subflow: flow of data packets along a specific path, i.e., a subset of the packets belonging to a data flow. The combination of all subflows forms the complete original data flow. Typically, a subflow should map to a unique path.

10) Transmission Control Block (TCB): An internal data structure
created by an MPMTP-AR agent for each of its existing MPMTP-AR session to other MPMTP-AR endpoints. TCB contains all the status and operational information for the agent to maintain and manage the corresponding session.

11) Transmission Sequence Number (TSN): A 32-bit sequence number used internally by MPMTP-AR. One TSN is attached to each chunk containing user data to permit the receiving MPMTP-AR agent to acknowledge its receipt and detect duplicate deliveries.

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

13) Unordered Message: Unordered messages are "unordered" with respect to any other message; this includes both other unordered messages as well as other ordered messages.

4. Goals

Using MPMTP-AR for reliable transmission, original data belonging to a session is divided multiple subflows and carried over multiple paths. This may prove beneficial in case of reliable transmission.

1) Improve Throughput: MPMTP-AR MUST support the concurrent use of multiple paths. To meet the minimum performance incentives for deployment, a MPMTP-AR connection over multiple paths SHOULD achieve no worse throughput than regular default path connection over the best constituent paths.

2) Improve Resilience: MPMTP-AR MUST support the use of multiple paths interchangeably for resilience purposes, by permitting packets to be sent and re-sent on any available path. It follows that, in the worst case, the protocol MUST be no less resilient than regular default path.

In order to guarantee reliable and order transmission, MPMTP-AR has the following goals:

1) Load balancing: transmitting a message over multiple paths reduces the bandwidth usage on a single path, which in turn reduces the impact on other traffic on that path and then avoids congestion in network hot-spots. From the perspective of the whole network, the network reliability and network utilization can be significantly improved.
2) Acknowledgement and Retransmission: when the receiver receive data packet, it SHOULD indicate the sender about correctly received data packets by acknowledgement; if some data packets with transmission error, the sender MUST retransmit them by retransmission mechanism.

5. Overview

Figure 1: Illustrates the framework of the proposed multipath transmission system based on application-level relay (MPTS-AR).

MPTS-AR defines three kinds of logical entity including user agent, relay, and relay controller. It also defines a relay service control protocol named OpenPath protocol in control plane to manage relay paths, and a profile of multipath transport protocol suite in data plane to facilitate multipath data transport. Based on MPTS-AR, we define MPMT-AR as a profile to support reliable transmission.

![Diagram of the framework of the proposed multipath transmission system based on application-level relay.](image-url)

Figure 1: The framework of the proposed multipath transmission system based on application-level relay.
User agent is responsible for sending or receiving data packets. In the sender side, it needs to collect candidate paths, and establish a connective session before transmitting data. The framework defines two ways used for collecting candidate paths by the sender. Before using them, the sender performs connectivity checks on relay paths to ascertain reachability. According to transmission requirements of current session, the sender selects the default path and one or more available relay paths as active paths to transmit data to the receiver.

A multiple session may be setup at the beginning, or be upgraded from the default path.

The sender distributes the original data across the active paths based on a scheduling strategy up to send buffer, and encapsulates them as special structure defined in MPTP profile before delivery.

During the transmission, the sender adjust send buffer in line with the receiver buffer. The receiver combines the received subflows to recreate the original data by transmission sequence number.

In order to notify sender of the reception, receiver sends selective acknowledgement to the sender. It contains information about receive buffer, correct reception, out of order packets, and duplicate reception. The sender will sends error data packets immediately and wait for out of order packets for a special timer which can be set by application. The retransmission may choose the best performance path, or may be different from the path for which the timer expired or eroded.

The session provides for graceful close on request from the MPMTP-AR agents, and also allows ungraceful close either on request from the MPMTP-AR agents or as a result of an error condition detected.

6. Usage Scenarios

MPMTP-AR can provide end-to-end message transmission application, such as file transfer.

Figure 2 illustrates a end-to-end multipath session. There are three paths between sender and receiver, including a default path, one relay path via one relay, and another relay path via two relays. Applications running at sender side can choose a data partitioning method according to its particular requirements. Then, assigned data to paths. Receiver reassembles the received data packets using reorder buffer to retrieve the reconstructed data which is delivered to the above application.
As shown in fig.2, relay path is unidirectional and default path is bidirectional. All data from receiver to sender via default path.

7. MPMTP-AR Agent Behavior

Based on user agent behavior in MPTS-AR, MPMTP-AR agent behavior needs modification, extension and addition to support message reliable transmission. Those functions include session and path management, flow partitioning and scheduling, subflow packaging, sequenced delivery, subflow recombination, subflow reporting, and selection acknowledgement and congestion avoidance.

Compare with the agent behavior introduced in MPTS-AR, MPMTP-AR agent behaviors divided into four types: Consistent Behaviors, Modified Behaviors, Additional Behaviors, and New Behaviors.

7.1 Consistent Behaviors

7.1.1 Path management

As illustrated in MPTS-AR, sender need to manage path including obtaining candidate relay path list, periodically path probe, path keep-alive, and monitor quality of relay paths. Details refer to section of path management in MPTS-AR.

7.2 Modified Behaviors

7.2.1 Subflow recombination

Receiver collects receiving statistics for per-subflow using Path-ID and Subflow-Specific sequence number. Then decapsulates packets and puts data packets in receive buffer per subflow using Path-ID and orders them using Subflow-Specific Sequence Number.
7.3 Additional Behaviors

7.3.1 Session management

MPMTP-AR is a connection-oriented application-level protocol, so a connection needs to be established before data transmission. The MPMTP-AR session establishment uses a way of out-of-band signaling (OpenPath) to get path information. Session parameters negotiation and authentication should be done during signaling process. A session with multipath may be setup at the beginning, or may be upgraded from a session with a default path. The connection establishment uses a four-way-handshake.

MPMTP-AR provides for graceful close (i.e., shutdown) of an active session on request from agent. MPMTP-AR also allows ungraceful close (i.e., abort), either on request from the agent (ABORT primitive) or as a result of an error condition detected.

MPMTP-AR session is a set of multiple paths which are based on connection. The detailed description of session management messages will be specified in section 8.4.

7.3.2 Flow partitioning and scheduling

MPTS-AR defines a simple flow partitioning scheme, round-robin algorithm, to assign packets to multiple subflows. In MPMTP-AR, we introduce a performance-based algorithm.

Due to paths has different transmission characteristic, such as bandwidth, delay, jitter, and error rate, we should assign data packet to subflow according to their capacity to increase transmission quality. Since sender monitor active path quality, collects statistics (e.g. RTT, loss-rate, delay etc.), so sender assigned more data packets to the subflow with a better performance. Data packets distribution adjust dynamically and periodically.

Sender should associate a subflow with an active path according to scheduling strategy. MPTS-AR indicates that scheduling policy should take various factors into consideration, including path’s performance, and importance of subflow data and so on. Sender orders available path list according to the path quality by descending order. At the same time, the sender maintains active paths. If a path in available path list performs better than an active path for a time, then the path should take over the active path. This behavior is maintained throughout a session.

7.3.3 Subflow packaging
In a session, sender formats data chunks into MPMTP-AR data packets. Specific packet format will be described in section 8. Then sender dispatches MPMTP-AR data packets along corresponding paths.

Apart from two key fields including path identifier and subflow-specific sequence number defined in MPTS-AR, common header also has a unique session ID. It is generated by the sender and receiver respectively at the establishment of the session.

Control information and data are encapsulated into different chunks. The control chunk contain a common header and chunk value. All chunks put into payload field in chunk field of MPMTP-AR packet. Details of different chunks are defined in section 8.

7.3.4 Subflow reporting

As mentioned in MPTS-AR, both sender and receiver need to send subflow report to monitor transmission quality. In this section, we define when to generate reports and what reports contain.

Suggested constants are to be used for subflow report interval calculation. Sessions operating under this document may specify a separate parameter for the traffic bandwidth rather than using the default fraction of the session bandwidth. The recommended fraction of the session bandwidth used for reports be fixed at 5%. The recommendation is that 1/4 of the report bandwidth be dedicated to data sender; the recommended default values for these two parameters would be 1.25% and 3.75%, respectively. The reports bandwidth for data senders should be kept non-zero so that sender reports can still be sent for inter-media synchronization.

The content of report should be able to reflect sending and receiving status. They collect statistics, such as the number of sent chunks and received chunks for per subflow. Details of report format defined in section 8.4.

7.4 New Behaviors

7.4.1 Sequenced Delivery

MPMTP-AR supports reliable and ordered message transmission using Transmission Sequence Number, Subflow-Specific Sequence Number, and Path-ID.

MPMTP-AR uses two levels of sequence spaces to guarantee reliable and ordered transmission: a session-level sequence number (Transmission Sequence Number) and a subflow-level sequence number (Subflow-Specific Sequence Number, SSSN) for each DATA packet. TSN is initialed as a
randomly number, and SSSN is a number starting from zero.

In order to inform receiver to reassemble the received data, sender uses SSSN to mark location of chunk in a subflow.

One-to-one relationship is between path and subflow. MPMTP-AR uses path identity to distinguish multiple paths. This ensures path to be unique in the whole network. Sender is able to tell the relay to choose which path and tell the receiver which subflow the packet come from by Path-ID.

7.4.2 Acknowledgement and Congestion Avoidance

MPMTP-AR supports selected acknowledgements at session-level to acknowledge the reception.

MPMTP-AR assigns a Transmission Sequence Number (TSN) to each data chunk. The TSN is independent of any SSSN assigned at subflow level. SACK acknowledges all TSNs received, even if there are gaps in sequence. In this way, reliable delivery is kept functionally separate from sequenced delivery.

MPMTP-AR uses data sequence mapping and session-level SACKs to decide when a session-level segment was received. A session-level SACKs signal when receive window moves forward, sum correct receiving transmission sequence number, out of order blocks and retransmission blocks. It would mean that once a segment is SACKed, it can be discarded in send buffer. Re-transmission send buffer has higher priority. Transmission reports signal sender detail information of active path, such as correctly receiving ratio and RTT. Sender would send segments in re-transmission buffer over the best transmission quality path according to transmission report. MPMTP-AR agent message from application, and put it into send buffer. At the beginning, receiver tells sender the size of receive window via MPMTP-AR session establishment. During message transmission, receiver moves forward receive window and signals sender this change via SACKs. Sender adjusts send window in line to receive window.

Acknowledgement and congestion avoidance function is responsible for retransmission when timely acknowledgement has not been received. Packet retransmission is conditioned by congestion avoidance procedures similar to those used for TCP. Large-scale experiments are therefore required in order to determine the most appropriate retransmission strategy, and recommendations will be refined once more information is available.

8. MPMTP-AR Packet Format
This section defines MPMTP-AR packet format to meet the requirement of behaviour described in section 7.

8.1 Overview

As stated in MPTS-AR, although MPTP obtain a simple, unreliable datagram service from the underlying UDP [5], MPTP can meet reliable delivery requirements of upper application programs.

MPTP is only a profile suit that is not complete, reliable delivery requirement of upper application programs can be realised in specific application profile defined in this document named MPMPT-AR.

In order to support agent behavior illustrated in section 7, we define MPMTP-AR packet format containing data and control packet.

8.2 MPMTP-AR Packet Header

MPMTP-AR header inherits from MPTP packet Header, including eight octet-length common header. The eight octets except for the first four bits correspond to different fields for MPMTP-AR data packets and MPMTP-AR control packets.

8.3 Fixed Header Fields of MPMTP-AR Data Packet

Fixed header of MPMTP-AR data packet is defined below:

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=1|T|P|  AMT  |  TOS  |  Rsvd |             SSSN              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Path Identifier                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                Transmission Sequence Number                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|U|B|E|         Reserved        |            Length             |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
\                          Data Chunk                           /
 \                                                               
    +---------------------------------------------------------------+
```

The fields have the following meaning:

version (V): 2 bits

This field identifies the version of MPTP. The version defined by
MPMTP-AR packet type (T): 1 bit
This field identifies the type of a MPMTP-AR packet. If the MPMTP-AR packet type bit is set, this packet is a MPMTP-AR data packet; otherwise, this packet is a MPMTP-AR control packet.

padding (P): 1 bit
If the padding bit is set, the packet contains one or more additional padding octets at the end which are not part of the payload. The last octet of the padding contains a count of how many padding octets should be ignored, including itself. Padding may be needed by some encryption algorithms with fixed block sizes.

Application-specific MPMTP-AR type (AMT): 4 bits
This field identifies the type of application-specific MPTP. In this document, AMT = 2.

type of service (TOS): 4 bits
This field, similar to TOS field in internet packet header, is specified along the abstract parameters precedence, delay, throughput, and reliability. These abstract parameters embody the delivery requirements of upper application programs. The values of these abstract parameters should be specified in application-specific MPTP documents.

Precedence: An independent measure of the importance of the payload data.

Delay: Prompt delivery is important for the payload data with this indication.

Throughput: High data rate is important for the payload data with this indication.

Reliability: A higher level of effort to ensure delivery is important for the payload data with this indication.

Rsvd: 4 bits
These 4 bits are reserved, which can be used and described in application-specific MPTP documents.

Subflow-Specific Sequence Number (SSSN):
This field is the sequence of the MPMTP-AR data packet in the subflow. Each subflow has its own strictly monotonically increasing sequence number space.
Path Identifier:
This field is the identifier of the path that the MPMTP-AR packet is associated with. For a relay path, the path identifier is globally unique; for the default path, the path identifier is fixed to zero.

Transmission Sequence Number (TSN): 32 bits
This field is the sequence of the MPMTP-AR data packet in the original data. The original flow has the unique strictly monotonically increasing sequence number space.

U bit: 1 bit
The (U)nordered bit, if set to ‘1’, indicates that this is an unordered DATA chunk, and there is no Stream Sequence Number assigned to this DATA chunk. Therefore, the receiver MUST ignore the Stream Sequence Number field. After reassembly (if necessary), unordered DATA chunks MUST be dispatched to the upper layer by the receiver without any attempt to reorder. If an unordered user message is fragmented, each fragment of the message MUST have its U bit set to ‘1’.

B bit: 1 bit
The (B)eginning fragment bit, if set, indicates the first fragment of a user message.

E bit: 1 bit
The (E)nding fragment bit, if set, indicates the last fragment of a user message.

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

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

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

8.4 Fixed Header Fields of MPMTP-AR Control Packet

Fixed header of MPMTP-AR control packet is defined below:
The fields of V, T, P, AMT, Path Identifier, and Session ID have the same meaning as those fields in MPMTP-AR data packet. Other fields have the following meaning:

MPMTP-AR control packet type (CT): 8 bits
   This field identifies the type of MPMTP-AR control packet.

Length: 8 bits
   This field is the length of the encapsulated control data after the MPMTP-AR header in byte length.

Chunk: (variable length)
   This field contains data or control information.

Control data is packed into chunks. Following sections from 8.4.1 to 8.4.19 describe the format of control chunks.

The values of CT are defined as follows:
<table>
<thead>
<tr>
<th>ID Value</th>
<th>Chunk Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sender Report (SR)</td>
</tr>
<tr>
<td>2</td>
<td>Receiver Report (RR)</td>
</tr>
<tr>
<td>3</td>
<td>Keep Alive (KA)</td>
</tr>
<tr>
<td>4</td>
<td>Session Initiation (SINIT)</td>
</tr>
<tr>
<td>5</td>
<td>Session Initiation Acknowledgement (SINIT ACK)</td>
</tr>
<tr>
<td>6</td>
<td>Cookie Echo (COOKIE ECHO)</td>
</tr>
<tr>
<td>7</td>
<td>Cookie Acknowledgement (COOKIE ACK)</td>
</tr>
<tr>
<td>8</td>
<td>Subflow INIT Request (SubINIT)</td>
</tr>
<tr>
<td>9</td>
<td>Subflow INIT Acknowledgement (SubINIT ACK)</td>
</tr>
<tr>
<td>10</td>
<td>Selective Acknowledgement (SACK)</td>
</tr>
<tr>
<td>11</td>
<td>Subflow Shutdown (SubSHUTDOWN)</td>
</tr>
<tr>
<td>12</td>
<td>Subflow Shutdown Acknowledgement (SubSHUTDOWN ACK)</td>
</tr>
<tr>
<td>13</td>
<td>Session Shutdown (SHUTDOWN)</td>
</tr>
<tr>
<td>14</td>
<td>Session Shutdown Acknowledgement (SSHUTDOWN ACK)</td>
</tr>
<tr>
<td>15</td>
<td>Session Shutdown Complete (SSHUTDOWN COMPLETE)</td>
</tr>
<tr>
<td>16</td>
<td>Abort (ABORT)</td>
</tr>
<tr>
<td>17</td>
<td>Operation Error (ERROR)</td>
</tr>
<tr>
<td>18</td>
<td>Path Feedback (PATH FEEDBACK)</td>
</tr>
<tr>
<td>19</td>
<td>Path Probe (PATH PROBE)</td>
</tr>
</tbody>
</table>

### 8.4.1 Sender Report (SR)

The sender report packet consists of three sections.
The fields have the following meaning:

**Timestamp**: 32 bits  
Indicates the time when this report was sent.

**Sender’s packet count**: 32 bits  
The total number of MPMTP-AR data packets transmitted by the sender since starting transmission up until the time this SR packet was generated.

**Sender’s octet count**: 32 bits  
The total number of payload octets (i.e., not including header or padding) transmitted in RTP data packets by the sender since starting transmission up until the time this SR packet was generated. This field can be used to estimate the average payload data rate.

### 8.4.2 Receiver Report (RR)

The receiver report packet consists of four sections.

**Cumulative number of packets**: 32 bits  
The total number of MPMTP-AR data packets from sender that have been received since the beginning of reception.

**Highest sequence number received**: 32 bits
The field contains the highest transmission sequence number received in a MPMTP-AR data packet from the sender.

Last SR (LSR): 32 bits
This field indicates the time of receiving last sender report. If no SR has been received yet, this field is set to zero.

Delay since last SR (DLSR): 32 bits
The delay, expressed in units of 1/65536 seconds, between receiving the last SR packet from the sender and sending this reception report packet. If no SR packet has been received yet, the DLSR field is set to zero.

8.4.3 Keep Alive (KA)
This chunk contains only timestamp.

|                           Timestamp                           |
+---------------------------+----------------------------------+
| Timestamp                 +----------------------------------+
+---------------------------+----------------------------------+

8.4.4 Session Initiation (SINIT)
This chunk is used to initiate an MPMTP-AR session between two endpoints. The format of the SINIT chunk is shown below:

| Number of Outbound Subflow (N) |             MSN               |
+---------------------------+----------------------------------+
| Initial TSN               +----------------------------------+
\ Optional/Variable-Length Parameters /\ +----------------------------------+
\ /                              +----------------------------------+

The SINIT chunk contains the following parameters. Unless otherwise noted, each parameter MUST only be included once in the INIT chunk.
<table>
<thead>
<tr>
<th>Fixed Parameters</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Outbound Subflow</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Maximum Number of Subflow (MSN)</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Initial TSN</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Variable Parameters</td>
<td>Optional</td>
</tr>
</tbody>
</table>

IMPLEMENTATION NOTE: If an SINIT chunk is received with known parameters that are not optional parameters of the SINIT chunk, then the receiver SHOULD process the SINIT chunk and send back an SINIT ACK.

Number of Outbound Subflow (NOS): 16 bits (unsigned integer)
Defines the maximum number of outbound subflow the sender of this SINIT chunk wants to create in this session. The value of 0 is invalid.

Note: A receiver of an SINIT with the NOS value set to 0 SHOULD abort the session.

Maximum Number of Subflow (MSN): 16 bits (unsigned integer)
Defines the max number of subflows the sender of this SINIT chunk allows to use in this session.

Note: A receiver of an SINIT with the MSN value set to 0 SHOULD abort the session.

Initial TSN (I-TSN): 32 bits (unsigned integer)
This value defines the initial TSN that the sender will use. The valid range is from 0 to 2**32-1. This field MAY be set to the value of the Initiate TSN field.

1) Optional/Variable-Length Parameters in INIT
The format of optional parameters in the INIT ACK chunk is shown below:

```
+-----------------+-----------------+
|      Type       |     Length      |
+-----------------+-----------------+
| +----------------+-----------------+
|  Parameters Value |
+-----------------+-----------------+
```

Type: 16 bits (unsigned integer)
This field identifies the type of parameters in the optional field.

Length: 16 bits (unsigned integer)
This field indicates the length of the optional parameters in bytes from the beginning of the type field to the end of the parameters field excluding any padding. Optional parameters with one byte parameters will have Length set to 5 (indicating 5 bytes).

Details of variable-length parameters would be discussed in the future.

8.4.5 Session Initiation Acknowledgement (SINIT ACK)

The SINIT ACK chunk is used to acknowledge the initiation of an MPMTP-AR session.

The part of parameter of SINIT ACK is formatted similarly to the SINIT chunk. It uses two extra variable parameters: advertised receiver window credit and state cookie. The format of the SINIT ACK chunk is shown below:

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Number of Inbound Subflow (N) |             MSN               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Advertised Receiver Window Credit                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                             \  
/             Optional Parameters (State Cookie)                /  
\                                                             \  
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Advertised Receiver Window Credit (a_rwnd): 32 bits (unsigned integer)
This value represents the dedicated buffer space, in number of bytes, the receiver has reserved in session with this window. During the life of the session, this buffer space SHOULD NOT be lessened (i.e., dedicated buffers taken away from this session).

Number of Inbound Subflow (NIS): 16 bits (unsigned integer)
Defines the maximum number of subflow the receiver allows the sender to create in this session. The value 0 MUST NOT be used.

Note: There is no negotiation of the actual number of subflow but instead the two endpoints will use the min (requested, offered).
Note: The sender receives SINIT ACK with the NIS value set to 0
SHOULD destroy the session discarding its TCB.

MSN (Maximum Number of Subflow): 16 bits (unsigned integer)
This field defines the max number of subflows the receiver allows
to use in this session. Valid value range is 1 to 31.

State Cookie:
Type Value: 1
Length: Variable size, depending on size of Cookie
Parameter Value:
This parameter value MUST contain all the necessary state and
parameter information required to create the session, along
with a Message Authentication Code (MAC).

8.4.6 State Cookie (COOKIE ECHO)

This chunk is used only during the initialization of a session. It is
sent by the initiator of an session to its peer to complete the
initialization process. This chunk MUST precede any DATA chunk sent
within the session.

IPN (Init Path Number): 16 bit
Chunk Flags in this chunk called IPN indicates the number of
initialize path which the sender wants to use at the beginning of
the session. Valid value range is 1 to 255. It MUST NOT be bigger
than MSN in INIT ACK.

Length: 16 bits (unsigned integer)
Set to the size of the chunk in bytes, including the 4 bytes of
the chunk header and the chunk body.

Path-ID: 32 bits (unsigned integer)
This field indicates the path identification of subflow which the sender will use in the session.

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

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

Note: A Cookie Echo does NOT contain a State Cookie parameter; instead, the data within the State Cookie’s Parameter Value becomes the data within the Cookie Echo’s Chunk Value. This allows an implementation to change only the first 2 bytes of the State Cookie parameter to become a COOKIE ECHO chunk.

8.4.7 Cookie Acknowledgement (COOKIE ACK)

This chunk is used to end a session establishment and to acknowledge the receipt of a COOKIE ECHO chunk. This chunk is empty, so the packet just contains packet header.

8.4.8 Subflow INIT Request (SubINIT)

The sender should send this chunk to its peer agent to notify the use of a particular relay path negotiated in SINIT and SINIT ACK.

Path-ID: 32 bits (unsigned integer)
This field indicates the path identification of subflow which the sender want to use in the session.

8.4.9 Subflow INIT Acknowledgement (SubINIT ACK)

The receiver should send this chunk to its peer agent to notify the use of a particular relay path negotiated by SINIT and SINIT ACK.
8.4.10 Selective Acknowledgement (SACK)

This chunk is sent to the peer agent via default path to acknowledge received DATA packet and to inform the peer agent of gaps in the received subsequences of DATA packets as represented by their TSNs.

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

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

The SACK also contains zero or more Gap Ack Blocks. Each Gap Ack Block acknowledges a subsequence of TSNs received following a break in the sequence of received TSNs. By definition, all TSNs acknowledged by Gap Ack Blocks are greater than the value of the Cumulative TSN Ack.
Cumulative TSN Ack: 32 bits (unsigned integer)
This parameter contains the TSN of the last DATA packet received in sequence before a gap. In the case where no DATA packet has been received, this value is set to the peer’s Initial TSN minus one.

Advertised Receiver Window Credit (a_rwnd): 32 bits (unsigned integer)
This field indicates the updated receive buffer space in bytes of the sender of this SACK.

Number of Gap Ack Blocks: 16 bits (unsigned integer)
This field indicates the number of Gap Ack Blocks included in this SACK.

Number of Duplicate TSNs: 16 bit
This field contains the number of duplicate TSNs the agent has received. Each duplicate TSN is listed following the Gap Ack Block list.

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

Gap Ack Block Start: 16 bits (unsigned integer)
Indicates the Start offset TSN for this Gap Ack Block. To calculate the actual TSN number the Cumulative TSN Ack is added to this offset number. This calculated TSN identifies the first TSN in this Gap Ack Block that has been received.

Gap Ack Block End: 16 bits (unsigned integer)
Indicates the End offset TSN for this Gap Ack Block. To calculate the actual TSN number, the Cumulative TSN Ack is added to this offset number. This calculated TSN identifies the TSN of the last DATA packet received in this Gap Ack Block.

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

```
<table>
<thead>
<tr>
<th>TSN=36</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSN=34</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>TSN=33</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>TSN=29</td>
</tr>
</tbody>
</table>
```

then the parameter part of the SACK MUST be constructed as follows (assuming the new a_rwnd is set to 1500 by the sender):

```
<table>
<thead>
<tr>
<th>TSN=36</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSN=34</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>TSN=33</td>
</tr>
<tr>
<td>-------</td>
</tr>
</tbody>
</table>
```
Duplicate TSN: 32 bits (unsigned integer)
Indicates the number of times a TSN was received in duplicate
since the last SACK was sent.

Every time a receiver gets a duplicate TSN (before sending the SACK);
it adds it to the list of duplicates. The duplicate count is
reinitialized to zero after sending each SACK.

For example, if a receiver were to get the TSN 35 three times it
would list 35 twice in the outbound SACK. After sending the SACK, if
it received yet one more TSN 35 it would list 35 as a duplicate once
in the next outgoing SACK.

8.4.11 Subflow Shutdown (SubSHUTDOWN)

An agent in a session MUST use this chunk to initiate a graceful
close of the session with its peer. Either the sender or the receiver
can initialize this chunk. This chunk has the following format.

Path Number (N): 32 bits (unsigned integer)
This field indicates how many subflows will be closed.
Path ID: 32 bits (unsigned integer)
   This parameter contains the Path ID of the subflow which will be closed.

8.4.12 Subflow Shutdown Acknowledgement (SubSHUTDOWN ACK)

This chunk MUST be used to acknowledge the receipt of the SubSHUTDOWN chunk at the completion of the shutdown process. The format is similar with SubSHUTDOWN chunk.

```
+----------------------------------+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Path Number(N)       |      Length                   |
+----------------------------------+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Path ID #1                           |
+----------------------------------+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                 ...                              /
                                 /                                |
                              \                                       /                      |
                              \                                   /                        |
                          Path ID #N                          |
+----------------------------------+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

8.4.13 Session Shutdown (SSHUTDOWN)

This chunk is used to close a session. The chunk is empty, so the packet just contains header.

8.4.14 Session Shutdown Acknowledgement (SSHUTDOWN ACK)

This SSHUTDOWN ACK chunk MUST be used to acknowledge the receipt of the SSHUTDOWN chunk.

8.4.14 Session Shutdown Acknowledgement (SSHUTDOWN ACK)

This SSHUTDOWN ACK chunk MUST be used to acknowledge the receipt of the SSHUTDOWN chunk.
This chunk inform the sender to close path indicated by Path ID.

8.4.15 Session Shutdown Complete (SSHUTDOWN COMPLETE)

This chunk MUST be used to acknowledge the receipt of the SSHUTDOWN ACK chunk and complete the shutdown process. The chunk is empty, so the packet just contains header.

8.4.16 Abort (ABORT)

The ABORT chunk is used to close the session. The ABORT chunk may contain Cause Parameters to inform the receiver about the reason of the abort.

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

Length: 16 bits (unsigned integer)
Set to the size of the chunk in bytes, including the chunk header and all the Error Cause fields present.

See section 8.4.17 for Error Cause definitions.

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

0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Number     |   Reserved    |           Length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\  one or more Error Causes /
\                          /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Number: 8 bits (unsigned integer)
This field indicates the number of errors listed in next Error Causes field.

Length: 16 bits (unsigned integer)
Set to the size of the parameter in bytes, including the Number, Reserved, Length, and Error Cause fields.

Error causes are defined as variable-length parameters using the format described as below:

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

Cause Code: 16 bits (unsigned integer)
This value defines the type of error conditions being reported.
<table>
<thead>
<tr>
<th>Cause Code</th>
<th>Value</th>
<th>Cause Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Invalid Path-ID</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Missing Mandatory Parameter</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>Stale Cookie Error</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Out of Resource</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>Unrecognized Chunk Type</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>Invalid Mandatory Parameter</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>No User Data</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>Cookie Received While Shutting Down</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>User Initiated Abort</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>Protocol Violation</td>
</tr>
</tbody>
</table>

Cause Length: 16 bits (unsigned integer)
Set to the size of the parameter in bytes, including the Cause Code, Cause Length, and Cause-Specific Information fields.

Cause-Specific Information: variable length
This field carries the details of the error condition.

The following section 1) - 11) define error causes for MPMTP-AR.

1) Invalid Path-ID

Cause of error: Invalid Path ID indicates Relay received a DATA chunk sent to a nonexistent subflow.

<table>
<thead>
<tr>
<th>0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>+------------------------------------------+</td>
</tr>
<tr>
<td>Cause Code=1</td>
</tr>
<tr>
<td>Cause Length=8</td>
</tr>
<tr>
<td>+------------------------------------------+</td>
</tr>
<tr>
<td>Path ID</td>
</tr>
<tr>
<td>(Reserved)</td>
</tr>
<tr>
<td>+------------------------------------------+</td>
</tr>
</tbody>
</table>

Path ID: 16 bits (unsigned integer)
This field contains the Path Identifier of the error DATA chunk.

Reserved: 16 bits
This field is reserved. It is set to all 0 on transmit and ignored on receipt.

2) Missing Mandatory Parameter

Cause of error: Missing Mandatory Parameter. Indicating that one or more mandatory parameters are missing in a received SINIT or SINIT
ACK.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Cause Code=2              |      Cause Length=8+N*2       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Number of missing params=N                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Missing Param Type #1       |   Missing Param Type #2       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Missing Param Type #N-1     |   Missing Param Type #N       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Number of Missing params: 32 bits (unsigned integer)
   This field contains the number of parameters contained in the
   Cause-Specific Information field.

Missing Param Type: 16 bits (unsigned integer)
   Each field will contain the missing mandatory parameter number.

3) Stale Cookie Error

Cause of error: Stale Cookie Error indicates the receipt of a valid
State Cookie that has expired.

```

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Cause Code=3              |       Cause Length=8          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Measure of Staleness (usec.)                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Measure of Staleness: 32 bits (unsigned integer)
   This field contains the difference, in microseconds, between the
current time and the time the State Cookie expired.

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

Leiwm, et al             Expires August 3, 2018                [Page 34]
4) Out of Resource

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

5) Unrecognized Chunk Type

Cause of error: Unrecognized Chunk Type. This error cause is returned
to the originator of the chunk if the receiver does not understand
the chunk.

Unrecognized Chunk: variable length
The Unrecognized Chunk field contains the unrecognized chunk from
the MPMTP-AR packet completing with Chunk Type, Chunk Flags, and
Chunk Length.

6) Invalid Mandatory Parameter

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

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

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Cause Code=7              |      Cause Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                               \   \
 \                                           /   \
 \                   Unrecognized Parameters   \   \
 +---------------------------------------------------------------------------+

Unrecognized Parameters: variable length

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

8) No User Data

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

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Cause Code=8              |      Cause Length=8           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                               \   \
 \                                           /   \
 \                   TSN Value               \   \
 +---------------------------------------------------------------------------+

TSN value: 32 bits (unsigned integer)

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

9) Cookie Received While Shutting Down

Cause of error: Cookie Received While Shutting Down. A COOKIE ECHO was received while the agent was in the session-level SSHUTDOWN-ACK-SENT state. This error is usually returned in an ERROR chunk followed with the retransmitted SSHUTDOWN ACK.
10) User-Initiated Abort

Cause of error: This error cause MAY be included in ABORT chunks that are sent because of an MPMT-AR user request. The MPMT-AR user can specify an Abort Reason that is transported by MPMT-AR transparently and MAY be delivered to the MPMT-AR user at the peer.

11) Protocol Violation

Cause of error: This error cause MAY be included in ABORT chunks that are sent because an MPMT-AR agent detects a protocol violation of the peer that is not covered by the error causes described in above section 1) to section 10). An implementation MAY provide additional information specifying what kind of protocol violation has been detected.

8.4.18 Path Feedback (PATH FEEDBACK)

This chunk is used to indicate the sender about path feedback information. The chunk includes time interval, subflow identification, max received transmission sequence number and correctly received byte of data packet.
### 8.4.19 Path Probe (PATH PROBE)

An agent should send this chunk to its peer agent to probe the path reachability and RTT. This chunk contains send time.

Last Feedback Time: 32 bits (unsigned integer)
- This field marks the time at which the receiver sends last PATH FEEDBACK.

Current Feedback Time: 32 bits (unsigned integer)
- This field marks the time at which the receiver sends this PATH FEEDBACK.

Received Max TSN: 16 bits (unsigned integer)
- This field indicates the max TSN of data packet received from the path marked by Path ID field.

Received Data Bytes: 16 bits (unsigned integer)
- This field indicates the bytes of data received from the path marked by Path ID between the Last Feedback Time and the Current Feedback Time.
Timestamp: 32 bits
This field records the send time of this chunk form the initiator of this chunk.

The receiver of this chunk delivers it to the sender via default path as soon as received and do not change anything. The initiator can calculate the RTT by send time and receive time.

9. MPMTP-AR Session State Diagram

During the life time of an MPMTP-AR session, the MPMTP-AR agent’s session progresses from one state to another in response to various events.

The state diagram in the figures below illustrates state changes, together with the causing events and resulting actions. Note that some of the error conditions are not shown in the state diagram. Full descriptions of all special cases are found in the text.

Note: Chunk names are given in all capital letters, while parameter names have the first letter capitalized, e.g., COOKIE ECHO chunk type vs. State Cookie parameter.
Notes:

1) If the State Cookie in the received COOKIE ECHO is invalid (i.e., failed to pass the integrity check), the receiver MUST silently discard the packet. Or, if the received State Cookie is expired (see section 10.1.4), the receiver MUST send back an ERROR chunk. In either case, the receiver stays in the CLOSED state.

2) If the T1-init timer expires, the agent MUST retransmit SINIT and restart the T1-init timer without changing state. This MUST be repeated up to ‘Max.Init Retransmits’ times. After that, the agent MUST abort the initialization process and report the error to the MPMTP-AR user.

3) If the T1-cookie timer expires, the agent MUST retransmit COOKIE ECHO and restart the T1-cookie timer without changing state. This MUST be repeated up to ‘MAX Init Retransmits’ times. After that, the agent MUST abort the initialization process and report the error to the MPMTP-AR user. After that, the agent MUST abort the initialization process and report the error to the MPMTP-AR user.

4) In the ESTABLISHMENT state, if the agent is the receiver of DATA packet and receives [SSHUTDOWN] from application or SSHUTDOWN from the peer agent, it should check outstanding DATA packet before sends a response.

5) In the SSHUTDOWN-SENT state, the agent MUST response any received control chunks without delay.

6) In the SHUTDOWN-RECEIVED state, the endpoint MUST NOT accept any new send requests from its SCTP user.

7) In the SSHUTDOWN-RECEIVED state, the agent MUST leave this state when all data in queue is ordered.

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

The CLOSED state is used to indicate that a session is not created (i.e., doesn’t exist).

10. Session Initialization

Before the first data transmission can take place from one MPMT-AR agent ("A") to another MPMT-AR agent ("Z"), the two endpoints must complete a session process in order to set up an MPMT-AR session between them.

The MPMT-AR user at an agent should use the SESSION primitive to initialize an MPMT-AR session to another MPMT-AR agent.

IMPLEMENTATION NOTE: From an MPMT-AR user’s point of view, a session may be implicitly opened, without a SESSION primitive being invoked, by the initiating agent’s sending of the first user data to the destination agent. The initiating MPMT-AR will assume default values for all mandatory and optional parameters for the SINIT/SINIT ACK.

Once the session is established, unidirectional path is open for data transfer (see section 10.1.1). Then according to the parameters negotiated at session, MPMT-AR agent A can use subflow after notifying agent Z.

10.1 Normal Establishment of a Session

The initialization process consists of the following steps (assuming that MPMT-AR agent "A" tries to set up a session with MPMT-AR agent "Z" and "Z" accepts the new session):

A) "A" first sends an SINIT chunk to "Z". In the SINIT, "A" must After sending the SINIT, "A" starts the T1-init timer and enters the COOKIE-WAIT state.

B) "Z" shall respond immediately with an SINIT ACK chunk. Moreover, "Z" MUST generate and send along with the SINIT ACK a State Cookie. See section 10.1.3 for State Cookie generation.

Note: After sending out SINIT ACK with the State Cookie parameter, "Z" MUST NOT allocate any resources or keep any states for the new session. Otherwise, "Z" will be vulnerable to resource attacks.

C) Upon reception of the SINIT ACK from "Z", "A" shall stop the T1-init timer and leave the COOKIE-WAIT state. "A" shall then send the State Cookie received in the SINIT ACK chunk in a COOKIE ECHO chunk, start the T1-cookie timer, and enter the COOKIE-ECHOED state.
D) Upon reception of the COOKIE ECHO chunk, agent "Z" will reply with a COOKIE ACK chunk after building a TCB and moving to the ESTABLISHED state.

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

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

An agent MUST send the SINIT ACK to the agent from which it received the SINIT.

Note: T1-init timer and T1-cookie timer shall follow the same rules given in section 11.3.

If an agent receives an SINIT, SINIT ACK, or COOKIE ECHO chunk but decides not to establish the new session due to missing mandatory parameters in the received SINIT or SINIT ACK, invalid parameter values, or lack of local resources, it SHOULD respond with an ABORT chunk. It SHOULD also specify the cause of abort, such as the type of the missing mandatory parameters, etc., by including the error cause parameters with the ABORT chunk.

Note that a COOKIE ECHO chunk that does NOT pass the integrity check is NOT considered an ‘invalid parameter’ and requires special handling; see section 10.1.4.

After the reception of the first DATA chunk in a session the agent MUST immediately respond with a SACK to acknowledge the DATA chunk. Subsequent acknowledgements should be done as described in section 11.2.

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

10.1.1 Handle Path Parameters

In the SINIT and SINIT ACK chunks, the sender of the chunk MUST indicate the number of outbound/inbound subflow and MSN it wishes to use in the session.

After receiving the session configuration information from the other side, MPMTP-AR agent MUST perform the following check: If the peer’s number outbound subflow (NOS) is less than the local maximum number
inbound subflow (MIS), meaning that the peer is incapable of supporting all the outbound subflow the agent wants to configure, the agent MUST use MIS and MAY report any shortage to the MPMTP-AR user. The MPMTP-AR user can then choose to abort the session if the resource shortage is unacceptable.

After the session is initialized, the valid outbound subflow identifier range for either agent shall be 0 to min (local MIS, remote NOS)-1.

10.1.2 Generating State Cookie

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

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

1) Create session TCB using information from both the received SINIT and the outgoing SINIT ACK chunk.

2) In the TCB, set the creation time to the current time of day, and the lifespan to the protocol parameter 'Valid.Cookie.Life'.

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

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

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

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

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

10.1.3 State Cookie Processing

When an agent (in the COOKIE-WAIT state) receives an SINIT ACK chunk with a State Cookie parameter, it MUST immediately send a COOKIE ECHO chunk to its peer with the received State Cookie.

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

10.1.4 State Cookie Authentication

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

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

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

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

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

5) Send a COOKIE ACK chunk to the peer acknowledging receipt of the COOKIE ECHO.

If a COOKIE ECHO is received from an agent with which the receiver of the COOKIE ECHO has an existing session, the procedures in section 10.2 should be followed.
10.1.5 Open Subflow

When an agent wants to use a new subflow, before data transmission it needs to notify the peer path information.

When an agent receives a SubINIT from another agent with which it associates an session, it shall take the following actions:

1) Check subflow parameter and session information;

2) Send response. If it is valid, the receiver sends SubINIT ACK to the sender; otherwise, the receiver MUST transmit an ERROR chunk with an error cause to the peer agent.

If the sender receives a SUBFLOW INIT ACK response, it can use this subflow to transmit data; otherwise it SHOULD terminal this subflow request.

10.2 Handle Duplication or Unexpected Issue

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

Note: An agent will not receive the chunk unless the chunk was from an MPMTP-AR agent associated with this agent. Therefore, the agent processes such a chunk as part of its current session.

The following scenarios can cause duplicated or unexpected chunks:

A) The peer has crashed without being detected, restarted itself, and sent out a new SINIT chunk trying to restore the session,

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

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

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

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

10.2.1 Unexpected SINIT in States Other than CLOSED, COOKIE-ECHOED, COOKIE-WAIT, and SSHUTDOWN-ACK-SENT
Unless otherwise stated, upon receipt of an unexpected SINIT for this session, the agent shall generate an SINIT ACK with a State Cookie. Before responding, the agent MUST check to see if the unexpected SINIT adds new parameters to the session. If new parameters are added to the session, the agent MUST respond with an ABORT. Parameters for the agent SHOULD be copied from the existing parameters of the session and be inserted into the SINIT ACK and cookie.

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

10.2.2 Unexpected SINIT ACK

If an SINIT ACK is received by an agent in any state other than the COOKIE-WAIT state, the agent should discard the SINIT ACK chunk. An unexpected SINIT ACK usually indicates the processing of an old or duplicated SINIT chunk.

10.2.3 Handle a COOKIE ECHO when a TCB Exists

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

1) Compute a MAC as described in step 1 of section 10.1.4,

2) Authenticate the State Cookie as described in step 2 of section 10.1.4,

3) Compare the timestamp in the State Cookie to the current time. If the State Cookie is older than the lifespan carried in the State Cookie and the Session ID contained in the State Cookie do not match the current session’s Session ID, the packet should be discarded. The agent also MUST transmit an ERROR chunk with a "Stale Cookie" error cause to the peer agent.

4) If the State Cookie proves to be valid, unpack the TCB into a temporary TCB.

10.2.4 Handle Duplicate COOKIE-ACK

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

10.2.5 Handle Stale COOKIE Error

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

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

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

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

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

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

1) Send a new SINIT chunk to the agent to generate a new State Cookie and reattempt the setup procedure.

2) Discard the TCB and report to the MPMTP-AR user the inability to set up the session.

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

11. User Data Transfer

Data transmission MUST only happen in the ESTABLISHED, SHUTDOWN-PENDING states.

DATA packet MUST only be received according to the rules below in ESTABLISHED, SHUTDOWN-PENDING, and SHUTDOWN-SENT. A DATA packet received in any other state SHOULD be discarded.

A SACK MUST be processed in ESTABLISHED, and SHUTDOWN-PENDING. A SACK in the CLOSED state is out of the blue and SHOULD be processed according to the rules in section 13.3. A SACK packet received in any other state SHOULD be discarded.

An MPMTP-AR receiver MUST be able to receive a minimum of 1500 bytes
in one MPMT-AR packet. This means that an MPMT-AR agent MUST NOT indicate less than 1500 bytes in its initial a_rwnd sent in the SINIT ACK.

Notes: When converting user messages into DATA chunks, an agent will fragment user messages larger than the current session path MTU into multiple DATA chunks. The data receiver will normally reassemble the fragmented message from DATA chunks before delivery to the user (see section 11.7 for details).

11.1 Transmission of DATA Packet

This document is specified as if there is a single retransmission timer per subflow, but implementations MAY have a retransmission timer for each DATA packet.

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

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

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

Refer to section 11.2 on the receiver behavior when it advertises a zero window. The sender SHOULD send the first zero window to probe after 1 RTO when it detects that the receiver has closed its window and SHOULD increase the probe interval exponentially afterwards. Also note that the cwnd SHOULD be adjusted according to section 12.2.1. Zero window probing does not affect the calculation of cwnd.

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

B) At any given time, the sender MUST NOT transmit new data to the receiver if it has cwnd or more bytes of data outstanding to the
receiver.

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

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

\[
\text{if}((\text{flightsize} + \text{Max.Burst} \times \text{MTU}) < \text{cwnd}) \\
\quad \text{cwnd} = \text{flightsize} + \text{Max.Burst} \times \text{MTU}
\]

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

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

IMPLEMENTATION NOTE: When the window is full, the sender MUST transmit no more DATA packet until some or all of the outstanding DATA packet are acknowledged and transmission is allowed by rule A and rule B again.

Whenever a transmission or retransmission is made, the sender MUST start T3-rtx timer. If the timer is already running, the sender MUST restart the timer if the earliest (i.e., lowest TSN) outstanding DATA packet is being retransmitted. Otherwise, the data sender MUST NOT restart the timer.

When starting or restarting the T3-rtx timer, the timer value must be adjusted according to the timer rules defined in Sections 11.3.2 and 11.3.3.

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

11.2 Acknowledge on Reception of DATA Packets

The MPMTP-AR agent MUST always acknowledge the reception of each valid DATA packet when the DATA packet received is inside its receive window.

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

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

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

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

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

Acknowledgements MUST be sent in SACK chunks unless session-level shutdown was requested by user, in which case an agent MAY send an acknowledgement in the session-level SSHUTDOWN chunk. A SACK chunk can acknowledge the reception of multiple DATA packets. See section 8.4.10 for SACK chunk format. In particular, the MPMTP-AR agent MUST fill in the Cumulative TSN Ack field to indicate the latest sequential TSN (of a valid DATA packet) it has received. Any received DATA packets with TSN greater than the value in the Cumulative TSN Ack field are reported in the Gap Ack Block fields. The MPMTP-AR agent MUST report as many Gap Ack Blocks as can fit in a single SACK chunk limited by the current path MTU.

When a packet arrives with duplicate DATA packet, the agent MUST immediately send a SACK with no delay. The duplicate TSN number(s)
SHOULD be reported in the SACK as duplicate.

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

The data receiver is responsible for maintaining its receive buffers. The data receiver SHOULD notify the data sender in a timely manner of changes in its ability to receive data. How an implementation manages its receive buffers is dependent on many factors (e.g., operating system, memory management system, amount of memory, etc.). However, the data sender strategy defined in section 11.2.1 is based on the assumption of receiver operation similar to the following:

A) At initialization of the session, the agent notify the peer the number of receive buffer space allocated to the session by SINIT ACK. The agent sets a_rwnd to this value.

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

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

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

Under certain circumstances, the data receiver may need to drop DATA packets that it has received but hasn’t released from its receive buffers (i.e., delivered to the MPMTP-AR user). These DATA packets may have been acked in Gap Ack Blocks. For example, the data receiver may be holding data in its receive buffers while reassembling a fragmented user message from its peer when it runs out of receive buffer space. It may drop these DATA packets even though it has acknowledged them in Gap Ack Blocks. If a data receiver drops DATA packets, it MUST NOT include them in Gap Ack Blocks in subsequent SACKs until they are received again via retransmission. In addition, the agent should take into account the dropped data when calculating
An agent SHOULD NOT revoke a SACK and discard data. Only in extreme circumstances should an agent use this procedure (such as out of buffer space). The data receiver should take into account that dropping data that has been acked in Gap Ack Blocks can result in suboptimal retransmission strategies in the data sender and thus in suboptimal performance.

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

An agent SHOULD NOT send a DATA packet with no user data.

11.2.1 Processing a Received SACK

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

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

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

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

A) At the establishment of the session, the agent initializes the rwnd to the Advertised Receiver Window Credit (a_rwnd) the peer specified in the SINIT or SINIT ACK.

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

C) Any time a DATA packet is marked for retransmission, either via
T3-rtx timer expiration (section 11.3.3) or via Fast Retransmit (section 12.2.4); add the data size of those packets to the rwnd.

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

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

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

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

iii) If the SACK is missing a TSN that was previously acknowledged via a Gap Ack Block (e.g., the data receiver reneged on the data), then consider the corresponding DATA that might be possibly missing. Count one miss indication towards Fast Retransmit as described in section 12.2.4, and if no retransmit timer is running for the subflow to which the DATA packet was originally transmitted, then T3-rtx is started for that subflow.

iv) If the Cumulative TSN Ack matches or exceeds the Fast Recovery exitpoint (section 12.2.4), Fast Recovery is exited.

11.3 Management of Retransmission Timer

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

When a session is multi-subflows, the agent will calculate a separate RTO for each subflow.

The computation and management of RTO in MPMT-PAR follow closely how TCP manages its retransmission timer. To compute the current RTO, an agent maintains two state variables per subflow: SRTT (smoothed round-trip time) and RTTVAR (round-trip time variation).

11.3.1 RTO Calculation

The rules governing the computation of SRTT, RTTVAR, and RTO are as follows:
C1) Until an RTT measurement has been made for a packet sent to the given subflow, set RTO to the protocol parameter ‘RTO.Initial’.

C2) When the first RTT measurement R is made, set SRTT <- R, RTTVAR <- R/2, and RTO <- SRTT + 4 * RTTVAR.

C3) When a new RTT measurement R’ is made, set RTTVAR <- (1 - RTO.Beta) * RTTVAR + RTO.Beta * |SRTT - R’| and SRTT <- (1 - RTO.Alpha) * SRTT + RTO.Alpha * R’

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

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

C4) When data is in flight and when allowed by rule C5 below, a new RTT measurement MUST be made each round trip. Furthermore, new RTT measurements SHOULD be made no more than once per round trip for a given subflow. There are two reasons for this recommendation: First, it appears that measuring more frequently often does not in practice yield any significant benefit [ALLMAN99]; second, if measurements are made more often, then the values of RTO.Alpha and RTO.Beta in rule C3 above should be adjusted so that SRTT and RTTVAR still adjust to changes at roughly the same rate (in terms of how many round trips it takes them to reflect new values) as they would if making only one measurement per round-trip and using RTO.Alpha and RTO.Beta as given in rule C3. However, the exact nature of these adjustments remains a research issue.

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

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

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

C7) A maximum value may be placed on RTO provided it is at least RTO.max seconds.

There is no requirement for the clock granularity G used for computing RTT measurements and the different state variables, other
than:

G1) Whenever RTTVAR is computed, if RTTVAR = 0, then adjust RTTVAR <-
G. Experience [ALLMAN99] has shown that finer clock granularities
(<=100 msec) perform somewhat better than more coarse granularities.

11.3.2 Retransmission Timer Rules

The rules for managing the retransmission timer are as follows:

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

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

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

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

11.3.3 Handle T3-RTX Expiration

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

E1) For the subflow for which the timer expires, adjust its ssthresh
with rules defined in section 12.2.3 and set the cwnd <- MTU.

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

E3) Determine how many of the earliest (i.e., lowest TSN) outstanding
DATA packets for the subflow for which the T3-rtx has expired will
fit into a single packet, subject to the MTU constraint for the path
corresponding to the subflow to which the retransmission is being
sent (this may be different from the subflow for which the timer

Leiwm, et al Expires August 3, 2018 [Page 56]
expires; see section 11.4). Call this value K.

E4) Start the retransmission timer T3-rtx on the subflow to which the retransmission is sent, if rule R1 above indicates to do so. The RTO to be used for starting T3-rtx should be the one for the subflow to which the retransmission is sent, may be different from the subflow for which the timer expired (see section 11.4 below).

After retransmitting, once a new RTT measurement is obtained (which can happen only when new data has been sent and acknowledged, per rule C5, or for a measurement made from a HEARTBEAT; see section 11.3), the computation in rule C3 is performed, including the computation of RTO, which may result in "collapsing" RTO back down after it has been subject to doubling (rule E2).

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

E5) Whenever an agent switches from the current subflow to a different one, the current retransmission timers are left running. As soon as the agent transmits a packet containing DATA packet to the new subflow, start the timer on that subflow, using the RTO value of the subflow to which the data is being sent, if rule R1 indicates to do so.

11.4 Path Identifier and Specific-Subflow Sequence Number

Every DATA packet MUST carry a valid path identifier. If an agent receives a DATA packet with an invalid path identifier, it shall acknowledge the reception of the DATA packet following the normal procedure, immediately send an ERROR packet with cause set to "Invalid Path Identifier", and discard the DATA packet.

The Specific-Subflow Sequence Number of all subflows MUST start from 0 when the session is established. Also, when the Specific-Subflow Sequence Number reaches the value 65535 the next Specific-Subflow Sequence Number MUST be set to 0.

11.5 Ordered and Unordered Delivery

Within a session, an agent MUST deliver DATA packets received with the U flag set to 0 to the user according to the order of their Transmission Sequence Number. If DATA packets arrive out of order of their Transmission Sequence Number, the agent MUST hold the received DATA packets from delivery to the user until they are reordered.
However, an MPMTP-AR agent can indicate that no ordered delivery is required for a particular DATA packet transmitted within the session by setting the U flag of the DATA packet to 1.

When an agent receives a DATA packet with the U flag set to 1, it must bypass the ordering mechanism and immediately deliver the data to the user (after reassembly if the user data is fragmented by the data sender).

This provides an effective way of transmitting "out-of-band" data in a given subflow. Also, a message can be used as an "unordered" by simply setting the U flag to 1 in all DATA packets sent in the session.

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

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

Note: When transmitting unordered data, an agent does not increment its Transmission Sequence Number.

### 11.6 Report Gaps in Received DATA TSNs

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

Based on the Gap Ack Block from the received SACK, the agent can calculate the missing DATA packets and make decisions on whether to retransmit them (see section 11.2.1 for details).

Multiple gaps can be reported in one single SACK.

When this session uses multipath, the MPMTP-AR agent SHOULD always try to send the SACK to the sender from default path.

Upon the reception of a SACK, the agent MUST remove all DATA packets that have been acknowledged by the SACK’s Cumulative TSN Ack from its transmit queue. The agent MUST also treat all the DATA packets with TSNs not included in the Gap Ack Blocks reported by the SACK as "missing". The number of "missing" reports for each outstanding DATA
packet MUST be recorded by the data sender in order to make retransmission decisions. See section 12.2.4 for details.

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

11.7 Fragmentation and Reassembly

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

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

If its session uses multipath, the agent shall choose a size no larger than the session Path MTU. The session Path MTU is the smallest Path MTU of all subflows.

Note: Once a message is fragmented, it cannot be refragmented. Instead, if the path maximum transmission unit (PMTU) has been reduced, then IP fragmentation must be used. Please see section 12.3 for details of PMTU discovery.

When determining when to fragment, the MPMTP-AR implementation MUST take into account the MPMTP-AR packet header as well as the DATA packet header(s).

Fragmentation takes the following steps:

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

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

3) The transmitter MUST also set the B/E bits of the first DATA chunk in the series to ‘10’, the B/E bits of the last DATA chunk in the series to ‘01’, and the B/E bits of all other DATA chunks in the series to ‘00’.

An agent MUST recognize fragmented DATA chunks by examining the B/E bits in each of the received DATA chunks, and queue the fragmented DATA chunks for reassembly.

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

12. Congestion Control

Congestion control is one of the basic functions in MPMTP-AR. For some applications, it may be likely that adequate resources will be allocated to MPMTP-AR traffic to ensure prompt delivery of time-critical data, thus, it would appear to be unlikely, during normal operations, that transmission encounter several congestion conditions. However, MPMTP-AR must operate under adverse operational conditions, which can develop upon partial network failures or unexpected traffic surges. In such situations, MPMTP-AR must follow correct congestion control steps to recover from congestion quickly in order to get data delivered as soon as possible. In absence of network congestion, these preventive congestion control algorithms should show no impact on protocol performance.

IMPLEMENTATION NOTE: As far as its specific performance requirements are met, an implementation is always allowed to adopt a more conservative congestion control algorithm than the one defined below.

The congestion control algorithms used by MPMTP-AR are based on [RFC2581]. This section describes how the algorithms defined in [RFC2581] are adapted to use in MPMTP-AR. We first list differences in protocol designs between TCP and MPMTP-AR, and then describe MPMTP-AR’s congestion control scheme. The description will use the same terminology as in TCP congestion control whenever appropriate. MPMTP-AR congestion control is always applied to both the entire session and individual subflow.

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

The biggest difference between MPMTP-AR and TCP, however, is multiple paths. MPMTP-AR is designed to establish robust communication session between two endpoints more than one path. The treatment here of congestion control for multipath receivers is new with MPMTP-AR and may require refinement in the future. The current algorithms make the following assumptions:

1) The sender keeps a separate congestion control parameter set for each subflow it use. The parameters should decay if the subflow is not used for a long enough time period. The sender SHOULD adjust the entire session congestion control parameter according to all subflows’ congestion control parameter; the specific rules will be refined in the future.

2) For each subflow, an agent does slow start upon the first transmission to that subflow.

12.2 Slow-Start and Congestion Avoidance

The slow-start and congestion avoidance algorithms MUST be used by an agent to control the amount of data being injected into the network. The congestion control in MPMTP-AR is employed in regard not only to the session, but also to an individual subflow. In some situations, it may be beneficial for an MPMTP-AR sender to be more conservative than the algorithms allow; however, an MPMTP-AR sender MUST NOT be more aggressive than the following algorithms allow.

Like TCP, an MPMTP-AR agent uses the following three control variables to regulate its transmission rate.

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

Note: This variable is kept on the entire session.

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

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

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

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

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

Unlike TCP, an MPMTP-AR sender MUST keep a set of these control variables cwnd, ssthresh, and partial_bytes_acked for EACH subflow. Only one rwnd is kept for the whole session.

12.2.1 Slow-Start

Beginning data transmission into a network with unknown conditions or after a sufficiently long idle period requires MPMTP-AR to probe the network to determine the available capacity. The slow-start algorithm is used for this purpose at the beginning of a transfer, or after repairing loss detected by the retransmission timer.

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

2) The initial cwnd after a retransmission timeout MUST be no more than \( 1 \times MTU \).

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

4) Whenever cwnd is greater than zero, the agent is allowed to have cwnd bytes of data outstanding on that subflow.

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

In instances where its session is multipath, if an agent receives a SACK that advances its Cumulative TSN Ack Point, then it should update its cwnd (or cwnds). However, if the received SACK does not advance the Cumulative TSN Ack Point, the agent MUST NOT adjust the cwnd of any subflow.

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

6) When the agent does not transmit data on a given subflow, the cwnd of the subflow should be adjusted to max(cwnd/2, 4*MTU) per RTO.

12.2.2 Congestion Avoidance

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

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

1) partial_bytes_acked is initialized to 0.

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

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

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

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

12.2.3 Parameter Update

Upon detection of packet losses from SACK (see section 12.2.4), an agent should do the following:

\[
\text{ssthresh} = \max(\text{cwnd}/2, 4\times\text{MTU})
\]
\[
\text{cwnd} = \text{ssthresh}
\]
\[
\text{partial_bytes_acked} = 0
\]

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

When the T3-rtx timer expires on a subflow, MPMTP-AR should perform slow start by:

\[
\text{ssthresh} = \max(\text{cwnd}/2, 4\times\text{MTU})
\]
\[
\text{cwnd} = 1\times\text{MTU}
\]

and ensure that no more than one MPMTP-AR packet will be in flight for that subflow until the agent receives select acknowledgement for successful delivery of data to that subflow.

12.2.4 Fast Retransmit on Gap Reports

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

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

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

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

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

2) If not in Fast Recovery, adjust the ssthresh and cwnd of the according to the formula described in section 12.2.3.

3) Determine how many of the earliest (i.e., lowest TSN) DATA packets marked for retransmission, subject to constraint of the path MTU of the subflow to which the packet is being sent. When a Fast Retransmit is being performed, the sender SHOULD ignore the value of cwnd and SHOULD NOT delay retransmission for this packet.

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

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

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

Note: Before the above adjustments, if the received SACK also acknowledges new DATA packets and advances the Cumulative TSN Ack Point, the cwnd adjustment rules defined in section 12.2.1 and section 12.2.2 must be applied first.
A straightforward implementation of the above keeps a counter for each TSN hole reported by a SACK. The counter increases for each consecutive SACK reporting the TSN hole. After reaching 3 and starting the Fast-Retransmit procedure, the counter resets to 0. Because cwnd in MPMTP-AR indirectly bounds the number of outstanding TSN’s, the effect of TCP Fast Recovery is achieved automatically with no adjustment to the congestion control window size.

12.3 Path MTU Discovery

[ RFC4821 ], [ RFC1981 ], and [ RFC1191 ] specify "Packetization Layer Path MTU Discovery", whereby an agent maintains an estimate of the maximum transmission unit (MTU) along a given Internet path and refrains from sending packets along that path that exceed the MTU, other than occasional attempts to probe for a change in the Path MTU (PMTU). [ RFC4821 ] is thorough in its discussion of the MTU discovery mechanism and strategies for determining the current end-to-end MTU setting as well as detecting changes in this value.

An agent SHOULD apply these techniques, and SHOULD do so on a per-path basis.

There are two important MPMTP-AR-specific points regarding Path MTU discovery:

1) MPMTP-AR session can span multiple paths. An agent MUST maintain separate MTU estimates for each path.

2) The sender should track an session PMTU that will be the smallest PMTU discovered for all paths. When fragmenting messages into multiple parts this session PMTU should be used to calculate the size of each fragment. This will allow retransmission to be seamlessly sent to an alternate path without encountering IP fragmentation.

13. Fault Management

13.1 Endpoint Failure Detection

An agent shall keep a counter on the total number of consecutive retransmission to its peer (this includes retransmission to all paths). If the value of this counter exceeds the limit indicated in the protocol parameter ‘Session.Max.Retrans’, the agent shall consider the peer agent unreachable and shall stop transmitting any more data to it (and thus the session enters the CLOSED state). In addition, the agent MAY report the failure to the MPMTP-AR user and optionally report back all outstanding user data remaining in its outbound queue. The session is automatically closed when the peer agent becomes unreachable.
The counter shall be reset each time a DATA packet sent to that peer agent is acknowledged (by the reception of a SACK) from the peer agent.

13.2 Path Failure Detection

When the session employs multipath, an agent should keep an error counter for each path.

Each time the T3-rtx timer expires on any path, the error counter of that path will be incremented. When the value in the error counter exceeds the protocol parameter ‘Path.Max.Retrans’ of that path, the agent should mark the destination transport address as inactive, and a notification SHOULD be sent to the MPMTP-AR user.

When an outstanding TSN is acknowledged, the agent shall clear the error counter of the path to which the DATA packet was last sent. When the session is multipath and the last packet sent to it was a retransmission to an alternate path, there exists an ambiguity as to whether or not the acknowledgement should be credited to the address of the last packet sent. However, this ambiguity does not seem to bear any significant consequence to MPMTP-AR behavior. If this ambiguity is undesirable, the transmitter may choose not to clear the error counter if the last packet sent was a retransmission.

Note: When configuring the MPMTP-AR agent, the user should avoid having the value of ‘Session.Max.Retrans’ larger than the summation of the ‘Path.Max.Retrans’ of all paths for the remote agent. Otherwise, all paths may become inactive while the agent still considers the peer agent reachable. When this condition occurs, how MPMTP-AR chooses to function is implementation specific.

When the primary path is marked inactive (due to excessive retransmission, for instance), the sender MAY automatically transmit new packets to alternate paths if they exist and are active. If more than one alternate path is active when the primary path is marked inactive, the sender may switch the load allocated to the inactive path to one or more active path.

13.3 Handle "Out of the Blue" Packets

An MPMTP-AR packet is called an "out of the blue" (OOTB) packet if it is correctly formed, but the receiver is not able to identify the session to which this packet belongs.

The receiver of an OOTB packet MUST do the following:

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

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

3) If the packet contains a COOKIE ECHO chunk, process it as described in section 10.1.

4) If the packet contains a SSHUTDOWN ACK chunk, the receiver should respond to the sender of the OOTB packet with a SSHUTDOWN COMPLETE.

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

6) If the packet contains a "Stale Cookie" ERROR or a COOKIE ACK, the MPMTP-AR packet should be silently discarded. Otherwise,

7) The receiver should respond to the sender of the OOTB packet with an ABORT. After sending this ABORT, the receiver of the OOTB packet shall discard the OOTB packet and take no further action.

14. Termination of Session

An agent should terminate its session when it exits from service. A session can be terminated by either abort or session-level shutdown. An abort of a session is abortive by definition in that any data pending of the session is discarded and not delivered to the peer. A session-level shutdown is considered as a graceful close where all data in queue is delivered to respective peers. When agent performs a session-level shutdown, the agent will stop accepting new data from MPMTP-AR user; if the agent is a sender, it only delivers data in all subflow queues when sending the SSHUTDOWN chunk.

14.1 Abort of a Session

When agent decides to abort an existing session, it MUST send an ABORT chunk to its peer agent.

An agent MUST NOT respond to any ABORT chunk (also see section 13.3).

After checking the Session ID, the receiving agent MUST remove the session from its record and SHOULD report the termination to MPMTP-AR user. If a User-Initiated Abort error cause is present in the ABORT chunk, the Abort Reason SHOULD be made available to the MPMTP-AR user.
14.2 Shutdown of a Session

Using the session-level SSHUTDOWN, the MPMTP-AR user in a session can gracefully close a session. This will allow all outstanding DATA packet to be delivered before the session terminate. According to the initiator, it includes sender-initialized and receiver-initialized.

Upon receipt of the SSHUTDOWN primitive from MPMTP-AR user:

For sender-initialized: The agent enters the SSHUTDOWN-PENDING state and remains there until all outstanding data has been acknowledged by its peer. The agent accepts no new data from user, but retransmits data to the peer agent if necessary to fill gaps. Once all its outstanding data has been acknowledged, the agent shall send a SSHUTDOWN chunk. It shall then start the T2-shutdown timer and enter the SSHUTDOWN-SENT state. If the timer expires, the agent must resend the SSHUTDOWN.

For receiver-initialized: Upon receipt of the SSHUTDOWN primitive from MPMTP-AR user, the agent enters the SSHUTDOWN-PENDING state and remains there until all data has been received correctly. Once all data has been received correctly, the agent shall send a SSHUTDOWN chunk. It shall then start the T2-shutdown timer and enter the SSHUTDOWN-SENT state. If the timer expires, the agent must resend the SSHUTDOWN.

The rules in section 11.3 MUST be followed to determine the proper timer value for T2-shutdown.

An agent should limit the number of retransmissions of the SSHUTDOWN chunk to the protocol parameter ‘Session.Max.Retrans’. If this threshold is exceeded, the agent should destroy the TCB and MUST report the peer agent unreachable to the MPMTP-AR user (and thus the session enters the CLOSED state).

Upon reception of the SSHUTDOWN, the agent shall enter the SSHUTDOWN-RECEIVED state.

Once an agent has reached the SSHUTDOWN-RECEIVED state, it should discard subsequent SSHUTDOWN chunks.

The sender of the SSHUTDOWN MAY also start an overall guard timer ‘T5-shutdown-guard’ to bound the overall time for the shutdown sequence. At the expiration of this timer, the sender SHOULD abort the session by sending an ABORT chunk. If the ‘T5-shutdown-guard’ timer is used, it SHOULD be set to the recommended value of 5 times ‘RTO.Max’.
The SSHUTDOWN receiver MUST send a SSHUTDOWN ACK and start a T2-shutdown timer of its own, entering the SHUTDOWN-ACK-SENT state. If the timer expires, the agent must resend the SSHUTDOWN ACK.

The sender of the SSHUTDOWN ACK should limit the number of retransmissions of the SSHUTDOWN ACK chunk to the protocol parameter ‘Session.Max.Retrans’. If this threshold is exceeded, the agent should destroy the TCB and may report the peer agent unreachable to the MPMTP-AR user (and thus the session enters the CLOSED state).

Upon the receipt of the SSHUTDOWN ACK, the agent shall stop the T2-shutdown timer, send a SSHUTDOWN COMPLETE chunk to its peer, and remove all record of the session.

After receiving the SSHUTDOWN COMPLETE chunk, the agent will verify that it is in the SSHUTDOWN-ACK-SENT state; if it is not, the chunk should be discarded; otherwise, the agent should stop the T2-shutdown timer and remove all acknowledge of the session (and thus the session enters the CLOSED state).

An agent SHOULD ensure that all its outstanding DATA packets have been sent or received correctly before sending SSHUTDOWN.

The sender of the SINIT or COOKIE ECHO should respond to the receipt of a SSHUTDOWN ACK with a stand-alone SSHUTDOWN COMPLETE in an MPMTP-AR packet. This is considered an Out of the Blue packet as defined in section 13.3. The sender of the SINIT lets T1-init continue running and remains in the COOKIE-WAIT or COOKIE-ECHOED state. Normal T1-init timer expiration will cause the SINIT or COOKIE chunk to be retransmitted and thus start a new session.

If a SSHUTDOWN is received in the COOKIE-WAIT or COOKIE ECHOED state, the SSHUTDOWN chunk SHOULD be silently discarded.

In sender-initialized scenario: An agent should reject any new data request from its user if it is in the SSHUTDOWN-PENDING, SSHUTDOWN-SENT state. If an agent is in the SSHUTDOWN-ACK-SENT state and receives an SINIT chunk (e.g., if the SSHUTDOWN COMPLETE was lost), it should discard the SINIT chunk and retransmit the SSHUTDOWN ACK chunk.

15. Security Consideration

15.1 Security Objectives

As a common transport protocol designed to reliably carry time-sensitive user messages, such as billing or signaling messages for telephony services, between two networked endpoints, MPMTP-AR has the
following security objectives.

- availability of reliable and timely data transport services
- integrity of the user-to-user information carried by MPMTP-AR

15.2 MPMTP-AR Responses to Potential Threats

MPMTP-AR may potentially be used in a wide variety of risk situations. It is important for operators of systems running MPMTP-AR to analyse their particular situations and decide on the appropriate counter-measures.

Operators of systems running MPMTP-AR should consult [RFC2196] for guidance in securing their site.

15.2.1 Countering Insider Attacks

The principles of [RFC2196] should be applied to minimize the risk of theft of information or sabotage by insiders. Such procedures include publication of security policies, control of access at the physical, software, and network levels, and separation of services.

15.2.2 Protecting Confidentiality

In most cases, the risk of breach of confidentiality applies to the signaling data payload, not to the MPMTP-AR or lower-layer protocol overheads. If that is true, encryption of the MPMTP-AR user data only might be considered. As with the supplementary checksum service, user data encryption MAY be performed by the MPMTP-AR user application. Alternately, the user application MAY use an implementation-specific API to request that the IP Encapsulating Security Payload (ESP) [RFC4303] be used to provide confidentiality and integrity.

Particularly for mobile users, the requirement for confidentiality might include the masking of IP addresses and ports. In this case, ESP SHOULD be used instead of application-level confidentiality. If ESP is used to protect confidentiality of MPMTP-AR traffic, an ESP cryptographic transform that includes cryptographic integrity protection MUST be used, because if there is a confidentiality threat there will also be a strong integrity threat.

Whenever ESP is in use, application-level encryption is not generally required.

Regardless of where confidentiality is provided, the Internet Key Exchange Protocol version 2 (IKEv2) [RFC4306] SHOULD be used for key
management.

Operators should consult [RFC4301] for more information on the security services available at and immediately above the Internet Protocol layer.

15.2.3 Protecting against Blind Denial-of-Service Attacks

A blind attack is one where the attacker is unable to intercept or otherwise see the content of data flows passing to and from the target MPMTP-AR node. Blind denial-of-service attacks may take the form of flooding, masquerade, or improper monopolization of services.

15.2.3.1 Flooding

The objective of flooding is to cause loss of service and incorrect behavior at target systems through resource exhaustion, interference with legitimate transactions, and exploitation of buffer-related software bugs. Flooding may be directed either at the MPMTP-AR node or at resources in the intervening IP Access Links or the Internet. Where the latter entities are the target, flooding will manifest itself as loss of network services, including potentially the breach of any firewalls in place.

In general, protection against flooding begins at the equipment design level, where it includes measures such as:

- avoiding commitment of limited resources before determining that the request for service is legitimate.
- giving priority to completion of processing in progress over the acceptance of new work.
- identification and removal of duplicate or stale queued requests for service.
- not responding to unexpected packets sent to non-unicast addresses.

Network equipment should be capable of generating an alarm and log if a suspicious increase in traffic occurs. The log should provide information such as the identity of the incoming link and source address(es) used, which will help the network or MPMTP-AR system operator to take protective measures. Procedures should be in place for the operator to act on such alarms if a clear pattern of abuse emerges.

The design of MPMTP-AR is resistant to flooding attacks, particularly
in its use of a four-way startup handshake, its use of a cookie to defer commitment of resources at the responding MPMT-AR node until the handshake is completed, and its use of a Session ID to prevent insertion of extraneous packets into the flow of an established session.

The IP Authentication Header and Encapsulating Security Payload might be useful in reducing the risk of certain kinds of denial-of-service attacks.

15.2.3.2 Blind Masquerade

Masquerade can be used to deny service in several ways:

- by tying up resources at the target MPMT-AR node to which the impersonated node has limited access. For example, the target node may by policy permit a maximum of one MPMT-AR session with the impersonated MPMT-AR node. The masquerading attacker may attempt to establish an session purporting to come from the impersonated node so that the latter cannot do so when it requires it.

- by deliberately allowing the impersonation to be detected, thereby provoking counter-measures that cause the impersonated node to be locked out of the target MPMT-AR node.

- by interfering with an established session by inserting extraneous content such as a SSHUTDOWN request.

MPMT-AR reduces the risk of blind masquerade attacks through IP spoofing by use of the four-way establishment handshake. Because the initial exchange is memory-less, no lockout mechanism is triggered by blind masquerade attacks. In addition, the SINIT ACK containing the State Cookie is transmitted back to the IP address from which it received the SINIT. Thus, the attacker would not receive the SINIT ACK containing the State Cookie. MPMT-AR protects against insertion of extraneous packets into the flow of an established session by use of the Verification Session ID.

Logging of received SINIT requests and abnormalities such as unexpected SINIT ACKs might be considered as a way to detect patterns of hostile activity. However, the potential usefulness of such logging must be weighed against the increased MPMT-AR startup processing it implies, rendering the MPMT-AR node more vulnerable to flooding attacks. Logging is pointless without the establishment of operating procedures to review and analyze the logs on a routine basis.

15.2.3.3 Improper Monopolization of Services
Attacks under this heading are performed openly and legitimately by the attacker. They are directed against fellow users of the target MPMTP-AR node or of the shared resources between the attacker and the target node. Possible attacks include the opening of a large number of associations between the attacker’s node and the target, or transfer of large volumes of information within a legitimately established session.

Policy limits should be placed on the number of associations per adjoining MPMTP-AR node. MPMTP-AR user applications should be capable of detecting large volumes of illegitimate or "no-op" messages within a given session and either logging or terminating the session as a result, based on local policy.

16. Reference

16.1 Normal Reference


16.2 Information Reference


Authors’ Addresses

Weimin Lei
Northeastern University
Institute of Communication and Information System
College of Computer Science and Engineering
Shenyang, China, 110819
P. R. China

Phone: +86-24-8368-3048
Email: leiweimin@mail.neu.edu.cn

Shaowei Liu
Northeastern University
Institute of Communication and Information System
College of Computer Science and Engineering
Shenyang, China, 110819
P. R. China

Email: liushaowei1988@163.com

Wei Zhang
Northeastern University
Institute of Communication and Information System
College of Computer Science and Engineering
Shenyang, China, 110819
P. R. China

Email: zhangwei1@mail.neu.edu.cn