Network Working Group
INTERNET-DRAFT

R. R. Stewart
Q. Xie
Motorola
K. Morneau
C. Sharp
Cisco
H. J. Schwarzbauer
Siemens
T. Taylor
Nortel Networks
I. Rytina
Ericsson

expires in six months

June 25,1999

MULTI_NETWORK DATAGRAM TRANSMISSION PROTOCOL
<draft-ietf-sigtran-mdtp-06.txt>

Status of This Memo

Abstract

This Internet Draft discusses a new protocol, namely the Multi-network Datagram Transmission Protocol (MDTP), that is intended to provide fault-tolerant reliable data transfer between communicating entities over IP networks [1].

MDTP is proposed as an application-level protocol that is designed to support redundant networks and transparent fault management. MDTP also provides timing control and configuration flexibilities to meet the stringent timing requirements often found in telephony signaling protocols. The motivation of developing MDTP is to support Internet-based high reliability applications such as signaling and call control for Internet telephony.

TABLE OF CONTENTS

1. Introduction

This Internet Draft discusses a new protocol, namely the Multi-network Datagram Transmission Protocol (MDTP). The intention of developing MDTP is to provide a fault-tolerant, real-time reliable data transfer mechanism between communicating endpoints over IP networks [1].

MDTP is proposed as an application-level protocol that is designed to support redundant networks and transparent fault management. MDTP also provides timing control and configuration flexibilities to meet the stringent timing requirements often found in telephony signaling protocols. The motivation of developing MDTP is to support Internet-based high reliability applications such as signaling and call control for Internet telephony.

MDTP is also designed to be scalable in order to support different signaling transport requirements for different interfaces to a telephony network.

For example, the transportation of signaling protocols such as ISDN PRI may not require redundant networks, and hence only a subset of MDTP will need to be implemented. On the other hand, redundant networks may be mandated when transporting SS7 signaling messages amongst different components in a carrier-grade telephony core network. In such cases, the transparent support for redundant networks, load sharing, and fault management defined in MDTP become essential.

Many of the fundamental concepts that have made TCP such a useful protocol are reused in MDTP, and some of the advantages of UDP are also merged into the design.

1.1 Terminology

The following terms are defined and used in this document:

- Redundant networks:

  An endpoint may be able to transmit or receive on more than one IP address/UDP port. RFC 1122 refers to this as multi-homing. This constitutes a redundant local network (for MDTP) relative to the endpoint. MDTP makes no attempt to assure routing diversity within the Internet connecting two endpoints. Each endpoint attempts to send to its peer endpoint using all the IP addresses and UDP ports its peer has open (within the constraints of any application specified restrictions). The choice of which local socket to send

upon is an implementation detail (it is possible only one socket is available and bound to all of the local networks to which the machine is connected). The O/S also will play a role in the multi-homing/redundancy. MDTP attempts a best effort at spreading the traffic across a

destination's available interfaces. It is assumed by MDTP that the network (if fault tolerance is desired) is engineered for diversity and MDTP's best effort will play only a small role in that diversity.

- Endpoint:

Representation of the logical point where MDTP datagrams can be sent to or received from. Moreover, an MDTP endpoint shall be defined as a set of IP address/port combinations in order to support redundant networks. For example, an endpoint on a multi-homed host connected with N IP networks can be represented as:

```
   [IP addr1/port1,
     ...
    IP addrN/portN]
```

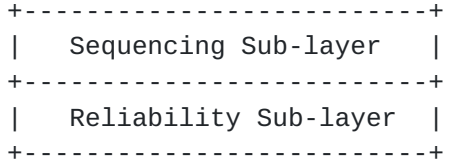
where the port numbers or IP addresses may not be unique, but their combinations shall be guaranteed unique by the underneath IP networks.

- Association:

Representation of an ongoing logical communication channel between two MDTP endpoints.

- Sub-layering:

Conceptually MDTP is subdivided into two sub-layers, as shown below:



This is introduced to achieve a clear separation between:
1) the reliable transport on a per association basis, and
2) the in-sequence delivery on a per stream basis to avoid blocking between independent streams.

- Reliability Sub-layer:

This Sub-layer copes only with functions to guarantee the

delivery of a datagram at its peer. At this sub-layer there is no subdivision into different streams.

- Transmission Sequence Number (TSN):

  A TSN is assigned to every datagram sent that transports user data. The TSN is used by the peer Reliability Sub-layer to detect any missing or duplicate user data. The TSN is processed by the Reliability Sub-layer only. Its value and presence is not known by the Sequencing Sub-layer

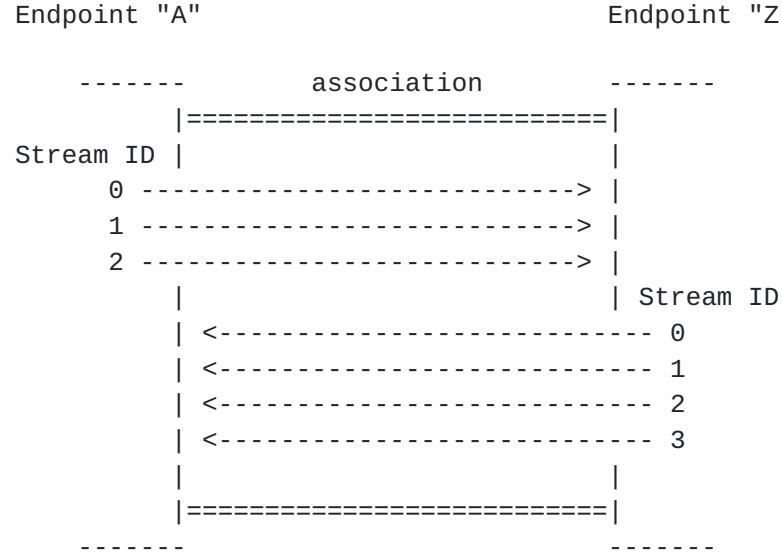
- Sequencing Sub-layer

  This sub-layer copes only with ordered delivery of datagrams belonging to a certain stream. It is based on the fact that the Reliability Sub-layer has ensured the guaranteed delivery of datagrams.

- Stream:

  Defined as a unidirectional logical sub-channel within an existing association (see the example below).

  Each stream shall be identified by a stream ID that is unique within the association and with regard to the endpoint that opens the stream.



  Datagrams sent through a stream shall be reliably transmitted and delivered independent to datagrams from other streams.

  As an implementation consideration, both the sender and receiver sides may need to dedicate resources, e.g., data queues, for each existing stream.

- Stream Sequence Number (SSN):

A Stream Sequence Number is associated with every datagram having a TSN. The SSN is valid only within the stream where the datagram belongs to. The SSN is processed by the Sequencing Sub-layer on a per stream basis.

Stream 0xffff is reserved and shall not be used. Stream 0x0 is open per default upon initiating an association and is not to be terminated.

- Sequence-number Attack:

  As defined in RFC 1948 [10].

- CRC Usage Policy:

  The minimum level of data integrity is provided using the checksum mechanism of the underlying transport protocol. It is therefore required that this mechanism is always enabled when transferring MDTP datagrams.

  In order to meet higher data integrity, as required for transporting of certain SCN signaling protocols, an additional 16 bit CRC value can optionally be carried in an MDTP datagram.

  See ITU-T Recommendation Q.703 [11] for details of how to calculate a 16 bit CRC.

1.2 Design Requirements of MDTP

The following are some of the design requirements of MDTP to make MDTP capable of supporting real-time call control environments that may employ redundant networks:

A) High communication fan-out: an endpoint may need to be in simultaneous communication with hundreds or thousands of endpoints performing various call processing functions. These endpoints may be codec converters, SS7 to IP translation applications, or, in the case of mobile networks, data selector and combiner applications.

B) Stringent timer control: an endpoint needs to have a very fine control over the timing for delivering a datagram. The timing should be easily adjusted depending on the message type and the destination. For example, after a few seconds of non-delivery the call which the message is about may not exist anymore.

C) Support multiple network paths: an endpoint communicating with a peer should be able to take advantage of the multiple network paths and

multi-homing in a transparent way. Therefore, the protocol must be able to take advantage of local multi-homed hosts and remote multi-homed hosts to provide resilient data delivery. This means that the application or upper layer protocols need not to be involved in the network fault management. Instead, when network failure occurs MDTP should be able to automatically transmit out-bound datagrams to an alternate destination network interface (if one exists) without intervention from the application.

D) Reliable transport: datagrams might be lost or discarded while traveling in the IP network towards the destination. The protocol must handle the re-transmission of lost messages in an autonomous way without any intervention from the upper layer. Also, sometimes datagrams may arrive in duplicate copies, in such cases MDTP must be able to detect and remove the duplicates automatically.

E) Support both ordered and unordered delivery: MDTP must support both ordered and unordered delivery. In the case of ordered delivery, the receiver shall detect out-of-order datagrams and re-order them before dispatching them to the upper layer. In the unordered case, received datagrams shall be dispatched without any effort of re-ordering.

F) Support stream sequencing: on the demand of the upper layer protocols or applications, MDTP should be able to support sequenced delivery with regard to each individual stream, i.e., the delay caused by the loss and retransmission of a datagram should be isolated to only the stream to which the datagram belongs. This is particularly important in some call control applications, where a loss of a message should only affect the call whom the message belongs to.

1.3 Interface to MDTP

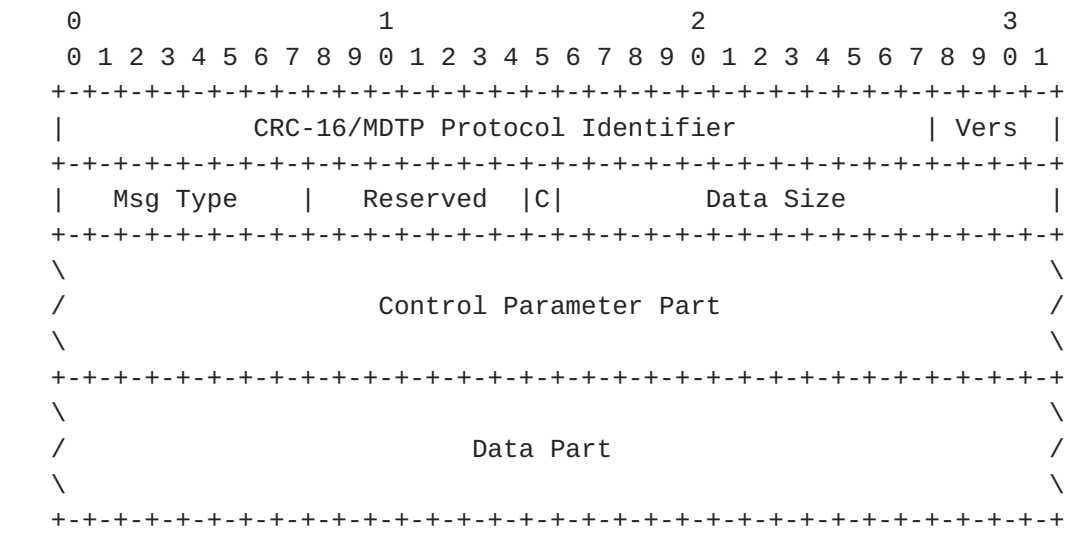
The application programs or upper layer protocols interface with MDTP through a set of primitives (see section 9).

Towards the IP networks, it is assumed that UDP is used for the transport layer. No special interfaces or changes are assumed within UDP or at the UDP/MDTP interface. MDTP maintains its own queuing and endpoint association. When MDTP runs on a router or on a gateway-enabled host, it will place no special constraints on the lower layer protocol implementations other than those described in the Router Requirements and Host Requirements RFCs.

2. MDTP Datagram Format

A MDTP datagram consists of a common header and possibly a control parameter part, a data part, or both.

MDTP Datagram Format



Note: Message Type and Data Size in the common header MUST be transmitted in network byte-order.

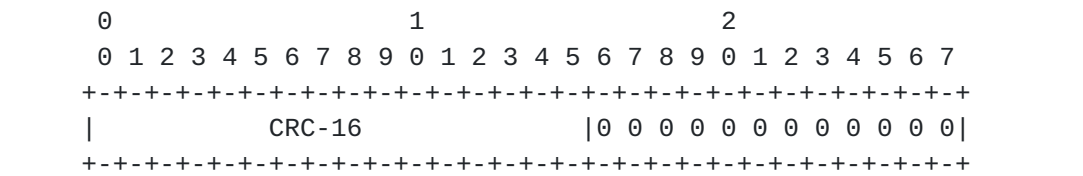
Note: when both the control part and data part are present in an MDTP datagram, the control part MUST be processed first.

2.1 MDTP Common Header Field Descriptions

CRC-16/MDTP Protocol Identifier: 28 bits

When the C Bit is NOT set, this field shall contain the 28 bit MDTP Protocol Identifier with a fixed value of 0xf787307. The receiver shall verify this Protocol Identifier before it consider the received datagram is a valid MDTP datagram.

When the C Bit is set, the most significant 16 bits of this field shall contain a CRC-16 value, and the other 12 bits shall be filled with '0' by the sender and ignored by the receiver, as illustrated below:



Version: 4 bits

This field represents the version number of the MDTP protocol, and shall be set to 0x3.

Message Type: 8 bits

When the value is non-zero, this shall indicate the type of control message present in the current MDTP datagram. A value of 0x0 indicates the control part is NOT present in the current datagram.

The value of Message Type is defined as the follows:

0x0 - indicating control part is NOT present

0x1 - Initiation
0x2 - Initiation Ack
0x3 - Extended Data Ack
0x4 - Advisory Ack Request
0x5 - Window-up
0x6 - Window-up Ack
0x7 - RTT-request
0x8 - RTT-ack
0x9 - Abort
0xa - Graceful Shutdown
0xb - Graceful Shutdown Ack
0xc - Stream Initiation
0xd - Stream Initiation Ack
0x10 - Stream Initiation Nack
0xe - Stream Termination
0xf - Stream Termination Ack

0x11 to 0xff - reserved and MUST NOT be used

Reserved: 7 bits

These bits are reserved for future use. The sender shall always set these bits to '0', and the receiver shall ignore there values.

C Bit: 1 bit

The CRC flag to indicate whether a CRC-16 value or the MDTP protocol identifier is present in the header, as described above.

Data Size: 16 bits

This value represents, in number of octets, the size of the user data present in the Data Part of the current datagram. If the Data Part is not present in the current datagram, it MUST be set to 0x0. This implies that no Data Part with zero size user data shall be allowed.

2.2 MDTP Control Parameter Part Definitions

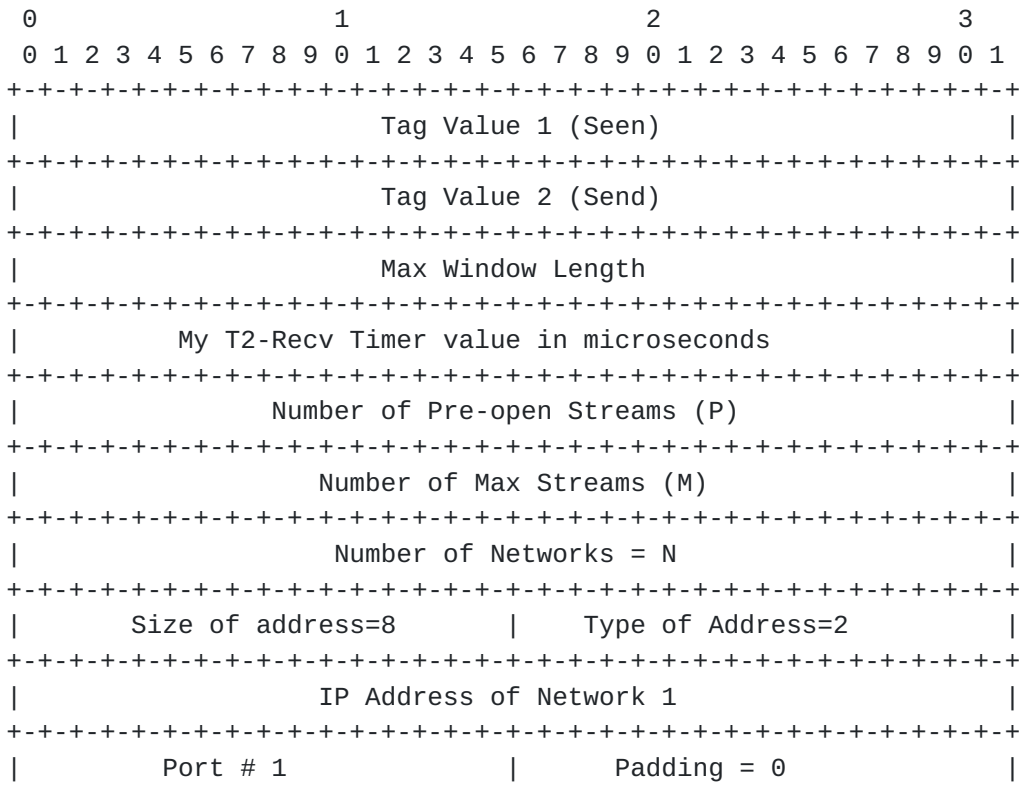
This section defines whether a control parameter part is present for each message type, and its format if a control parameter part is present.

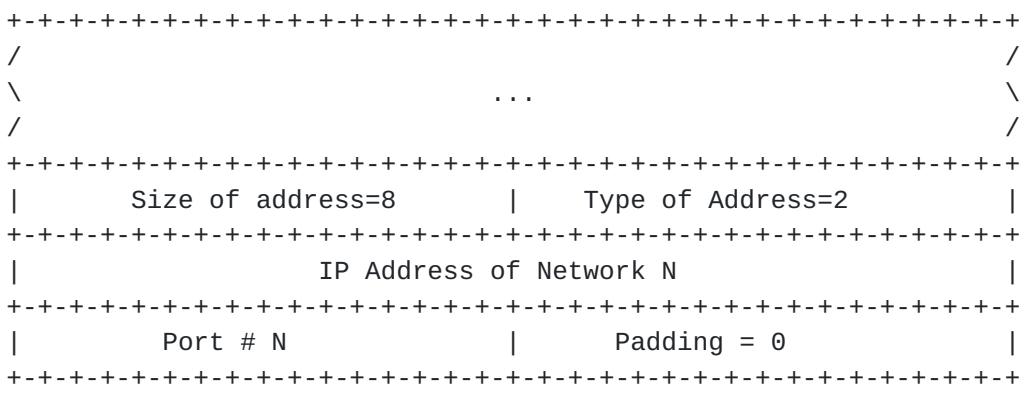
Note: integers in the control parameter part MUST be transmitted in network byte-order.

2.2.1 Initiation (0x1) and Initiation Ack (0x2):

The parameter field of the Initiation and Initiation Ack messages shall carry two initiation Tags, the maximal window length of the sender, the sender's T2-Receive timer value in microseconds, the number of pre-open outbound streams (P), the number of maximal inbound streams (M), and the sender's local network information. The network information informs the receiver the addresses that may be the source of datagrams for this association and are valid addresses that the receiver can use as a destination address. Note that the endpoint MAY be multi-homed.

The following defines the parameter format for carrying N IPv4 Network addresses (other network address formats can be carried by setting the size and type fields accordingly):

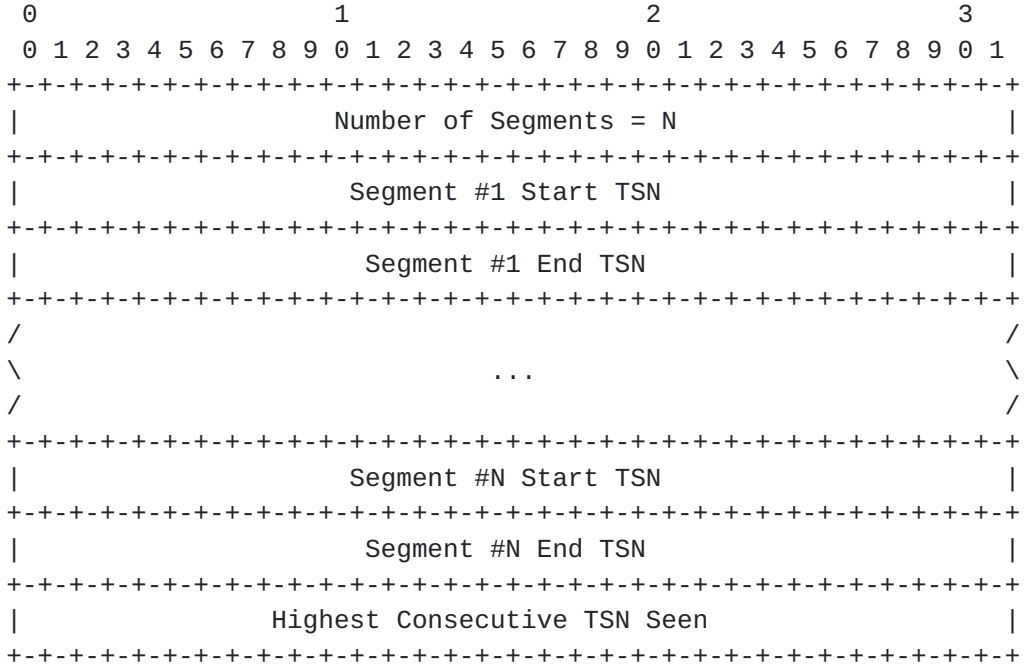




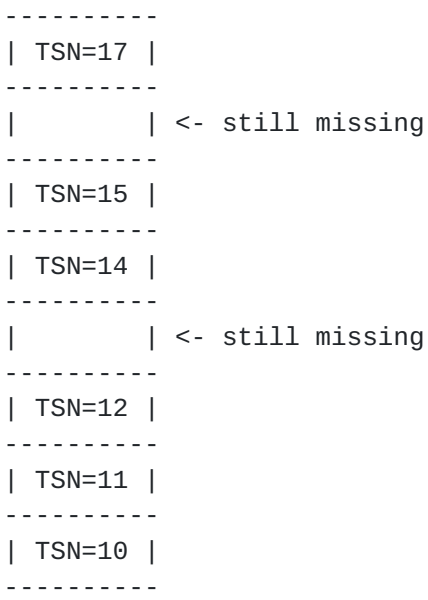
If there is any implementation-specific data needed to be exchanged at the setup of the association, it should be appended to the end of the above data structure. The format of the implementation-specific data should follow "Size/Type/Data Field" format as defined above. In case an endpoint does not support the implementation-specific data received, it shall ignore the additional fields.

2.2.2 Extended Data Ack (0x3):

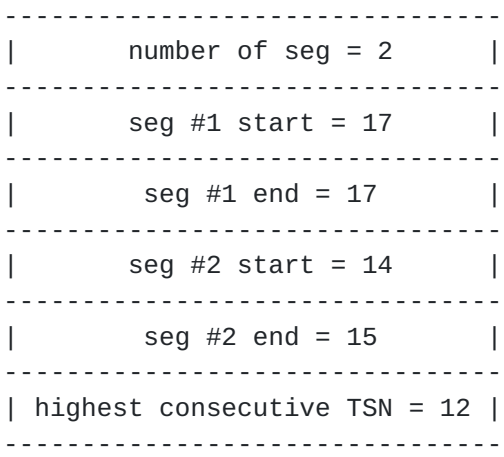
The parameter field contains 0 or more segment reports and the highest consecutive TSN received.



For example, assume the receiver has the following datagrams newly arrived at the time when it decides to send an Extended Data Ack,



the control parameter part of the Extended Data Ack shall be constructed as follows:



Note: when multiple segments are reported in a single Extended Data Ack, the order of the segments in the Extended Data Ack is not specified.

2.2.3 Advisory Ack Request (0x4):

No parameter field.

2.2.4 Window-up (0x5):

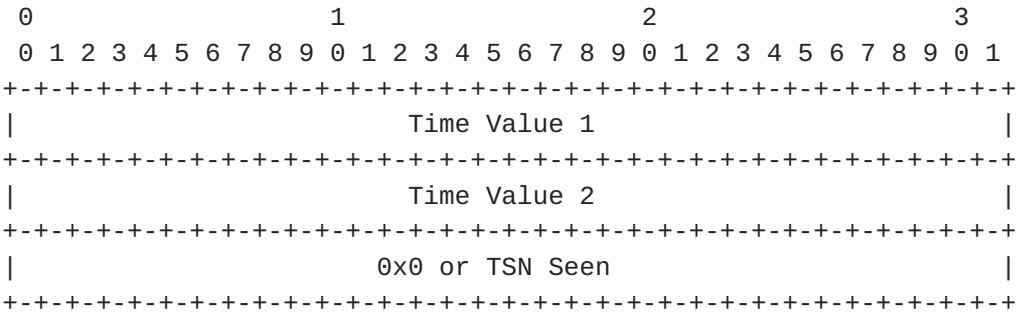
No parameter field.

2.2.5 Window-up Ack (0x6):

Same as that of Extended Data Ack.

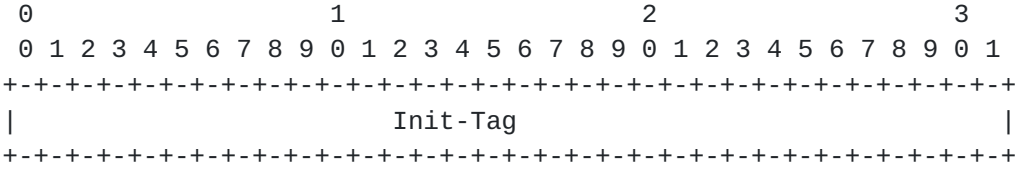
2.2.6 RTT-request (0x7) and RTT-ack (0x8):

The parameter field shall contain the time value that is used for RTT calculation (see section 6.2), and optionally an acknowledgment Seen value.



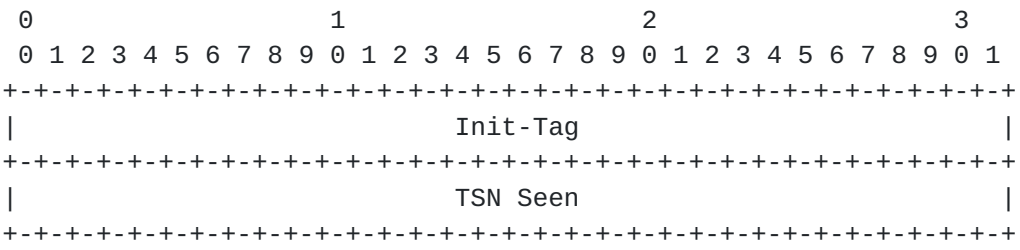
2.2.7 Abort (0x9):

The Abort message shall carry the initiation Tag of the destination endpoint as a measure of security.



2.2.8 Graceful Shutdown (0xa):

The destination endpoint initiation Tag shall be carried as a measure of security.



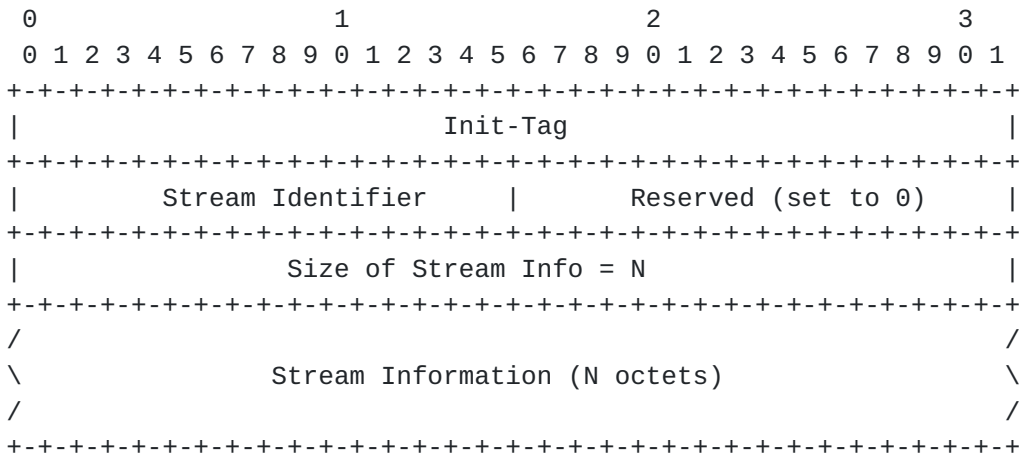
2.2.9 Graceful Shutdown Ack (0xb):

No parameter field.

2.2.10 Stream Initiation (0xc):

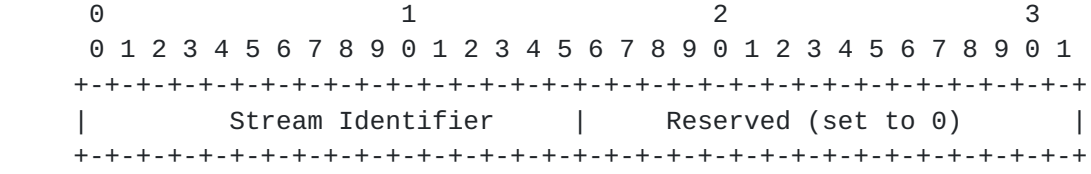
The parameter field shall contain the initiation Tag of the destination endpoint (see section 3.1) and the Stream Identifier.

Also, there shall be a "Size of Stream Info" and "Stream Information" fields that may contain an opaque user data structure specific to the stream being opened. The "Stream Information" field should be padded with '0's to 32 bit word boundary before transmission.



2.2.11 Stream Initiation Ack (0xd):

The parameter field shall contain the Stream Identifier.

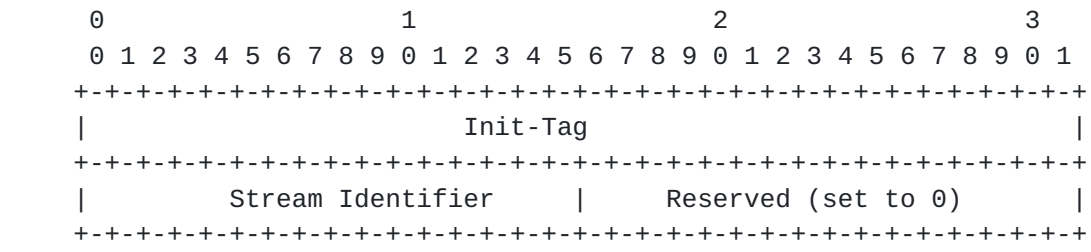


2.2.12 Stream Initiation Nack (0x10):

Same as that of Stream Initiation Ack.

2.2.13 Stream Termination (0xe):

The parameter field shall contain the initiation Tag value (see section 3.1) and the Stream Identification

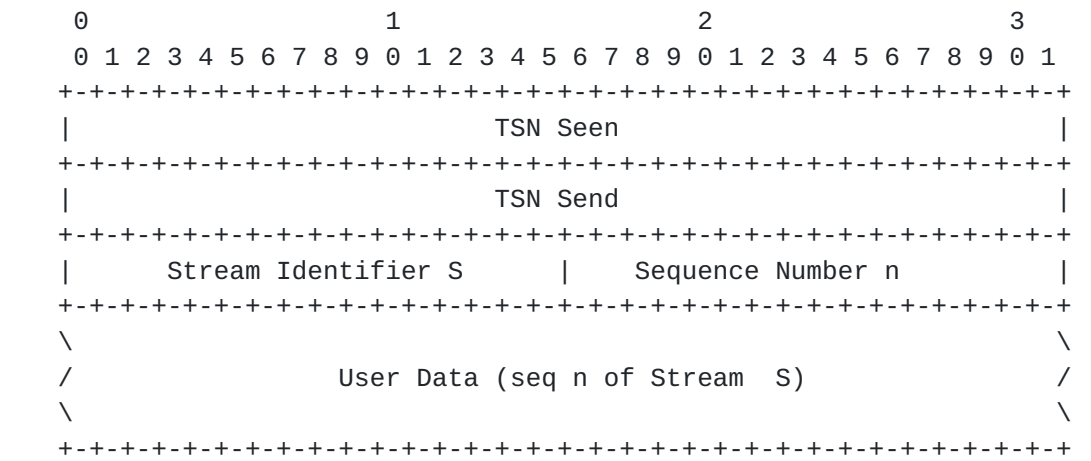


2.2.14 Stream Termination Ack (0xf):

Same as that of Stream Initiation Ack.

2.3 MDTP Data Part Definitions

The following format shall be used for MDTP datagram Data Part:



Note: TSN Seen, TSN Send, Stream Identifier, and Sequence Number MUST be transmitted in network byte-order.

TSN Seen: 32 bits

This is a piggy-backed acknowledgment, indicating the reception of datagrams up to this TSN.

TSN Send: 32 bits

This value represents the TSN of the user data carried in this datagram.

Stream Identifier S: 16 bits

Identify the stream to which the following user date belongs.

Sequence Number n: 16 bits

This value presents the sequence number of the following user data within the stream.

Sequence number 0x0 indicates that the following user data shall be treated as unordered, and shall be dispatched to the upper layer by the receiver without any attempt of re-ordering.

User Data: variable length

This is the payload user data. The size of the user data shall be specified in the Data Size field. The implementation may optionally have some '0' padded at the end of User Data field.

3. Endpoint Association Initialization

Before the first data transmission can take place from one endpoint ("A") to another endpoint ("Z"), the two endpoints must complete an initialization process in order to set up an association between them.

The upper layer may explicitly request MDTP to initialize an association to an endpoint, or implicitly open the association by sending the first datagram to that endpoint on stream 0.

Once the association is established, stream 0 is automatically opened and ready for datagram transmission in both directions. Moreover, if there are any pre-open streams specified by either side, they shall also be opened and ready for transmission from that side.

Other streams must be explicitly opened before data transmission can occur.

A tag-and-lock mechanism must be employed during the initialization in order to guard against security attacks as well as erroneous datagrams.

3.1 Initiation Message and Tag Lock

The initialization process consists of the following steps (assuming that MDTP endpoint "A" tries to set up an association with MDTP endpoint "Z"):

A) "A" shall first send an Initiation message to "Z", with Tag Seen field set to 0x0 and Tag Send field set to Tag_A, where Tag_A shall be a random number in the range of 0x80000000 to 0xffffffff (see

3.1.4 for Tag value selection), and enter the Tag-lock mode.

B) "Z" shall respond immediately with an Initiation Ack message, with Seen set to Tag_A and Send set to Tag_Z (same range as Tag_A), and enter the Tag-lock-new mode.

At this point, "Z" is ready to send user datagrams to "A" in stream 0. And upon the reception of the above Initiation Ack from "Z", "A" also becomes ready to send user datagrams to "Z" in stream 0.

Note: user data in other streams can not be sent until the respective streams are opened.

C) "Z" shall leave Tag-lock-new mode and enter Tag-lock mode only if a user datagram has been sent out from "Z" to "A".

Note: to guard against "man in the middle" attacks, an endpoint should impose a limit on the number of associations allowed to be in the Tag-lock-new mode; whenever this limit is reached, any further association Initiations received by the endpoint shall be silently discarded. Also, a timer shall be used on each association that is in the Tag-lock-new state; at the expiration of that timer, that association shall be shutdown by the endpoint by sending an Abort to the peer of that association.

Note: no user data shall be carried in both the Initiation and Initiation Ack messages, i.e. the Data Size field in the MDTP common header must be set to 0x0.

Note: if an endpoint receives an Initiation but decides not to establish the new association due to lack of resources, etc., it shall respond to the Initiation with an Abort message.

3.1.1 Passing Initiation Parameters

In addition to the Tags, both side must exchange their local network information, maximal window length, the sender's T2-Receive timer value in microseconds, number of pre-open outbound streams (P), and number of maximal inbound streams (M), in the Initiation and Initiation Ack messages. And the receiver shall process and store these initiation parameters.

The maximal window length from the peer will be used to validate the TSN range of the received datagrams (see section 4.6).

The sender's T2-receive timer will be used to adjust the T3-send timer (see section 4.1.1).

The number of maximal inbound streams (M) shall indicate the maximal number of concurrent streams the sender will accept from its peer (excluding stream 0). The sender will reject any new Stream Initiation request from its peer if this number is reached, unless some of the currently open streams are closed first by the peer.

The sender shall use the number of pre-open outbound streams (P) to indicate to its peer that, in addition to the stream 0, the sender

wants to have that many more streams (from stream 1 to stream P) implicitly opened from the sender's side at the onset of the association. This allows the receiver to allocate and initialize necessary resources for the additional P inbound streams.

However, if the sender's P is greater than, or equal to, the receiver's M, the receiver shall replace the sender's P with M, and then only pre-open M inbound streams (from stream 1 to stream M). At the same time, the sender also must either settle with M, instead of P, pre-open outbound streams, or abort the association and report the resources shortage.

3.2 Tag Unlock and TSN Initialization

The first user datagram transmitted by "A" to "Z" shall have the TSN Seen value set to Tag_Z in the Data Part (see 2.3).

Similarly, the first user datagram transmitted by "Z" to "A" shall have the TSN Seen value set to Tag_A.

The reception of this first datagram with user data and with the correct Tag value in the TSN Seen field from its peer shall unlock the Tag and cause the endpoint to leave the Tag-lock or Tag-lock-new mode.

The receiver shall immediately send back an Extended Data Ack to acknowledge the reception of this first user datagram.

The TSN Send value carried in this first datagram with user data shall be used to establish the initial TSN of this peer, i.e., the sender of this datagram.

To strengthen the security, this initial TSN shall be randomly selected from the range between 0x1 and 0x7fffffff by the sender, by means such as those suggested in RFC 1750 [9].

Note: When an endpoint receives the first user datagram that causes it to leave the the Tag-lock or Tag-lock-new mode, it shall immediately send an Extended Data Ack to acknowledge the reception of this user datagram and shall NOT start a T2-recv timer. For all the subsequent user datagram receptions, the receiver shall follow the normal timer rules.

3.3 Datagram Processing during Tag Lock

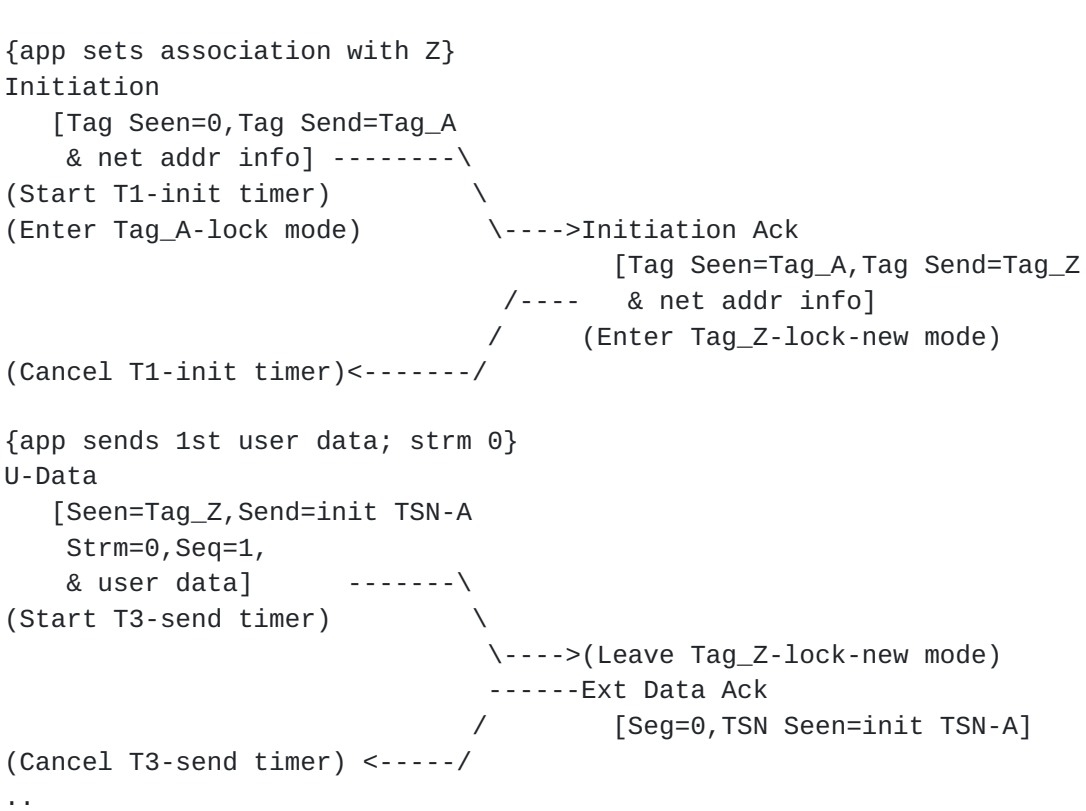
In Tag-lock or Tag-lock-new mode, an endpoint shall silently discard any user datagrams from the peer endpoint that does not carry the correct Tag value.

However, if there is a control part present in a discarded user datagram, the endpoint shall always process the control part even when the data part is being discarded.

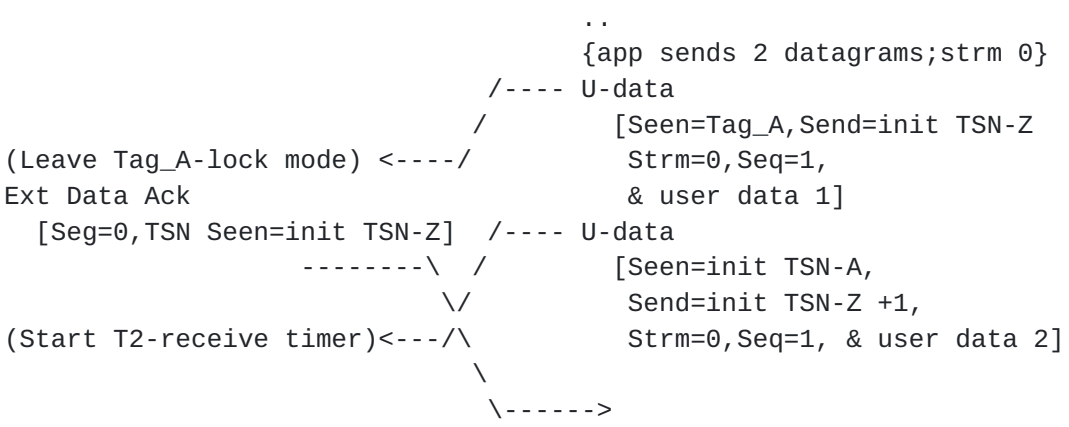
If another Initiation from "A" is received by "Z" after "Z" sent out its Initiation Ack, "Z" shall respond to this second Initiation by re-sending the Initiation Ack if the Tag Send field of this second Initiation has the same value as that of the original Initiation. Otherwise, "Z" shall respond by sending an Initiation of its own, with Tag Send field set to Tag_Z, so as to elicit an Initiation Ack from "A".

3.4 An Example of Association Initialization

In the following example, "A" initiates the association first and then sends a user datagram to "Z", then "Z" sends two user datagrams sometimes later:



..



If T1-init timer expires at "A" after the Initiation is sent, the same Initiation message with the same Tag_A value shall be retransmitted and the timer restarted. This shall be repeated Max.Init.Retransmit times before "A" considers "Z" unreachable and optionally reports the failure.

3.5 Other Initiation Issues

3.5.1 Selection of Tag Value

Tag values should be selected from the range of 0x80000000 to 0xffffffff. It is very important that the Tag value be randomized to guard against "man in the middle" and "sequence number" attacks. It is suggested that RFC 1750 [9] be used for the Tag randomization.

3.5.2 Initiation from behind a NAT

When a NAT is present between two endpoints, the endpoint that is behind the NAT, i.e., one that does not have a publicly available network address, shall take one of the following options:

A) Indicate that it has only one network by setting the 'Number of networks' field in the Initiation message to 0. This will make the endpoint that receives this Initiation message to consider the sender as only having that one address. This method can be used for a dynamic NAT, but any multi-homing configuration at the endpoint that is behind the NAT will not be visible to its peer, and thus not be taken advantage of.

B) Indicate all of its networks in the Initiation by specifying all the actual IP addresses and ports that the NAT will substitute for the endpoint. This method requires that the endpoint behind the NAT must have pre-knowledge of all the IP addresses and ports that the NAT will assign.

3.5.3 Initialization Collision

If two endpoints attempt to initialize an association with each other at about the same instance, a collision will occur. As a result, each side will receive an Initiation datagram from the other side after it transmitted its own. In such a case, both sides shall send an Initiation Ack datagram to the other side using the procedure described above.

3.5.4 Association Re-initialization

An endpoint shall be allowed to re-initialize an established association with the other endpoint.

Once an endpoint has left the Tag-lock or Tag-lock-new mode of the previous association initialization process, it shall treat any new Initiation message from its peer as a re-initialization event.

During a re-initialization, both endpoint shall follow the same procedure as defined in section 3.1. And a new Init-Tag must be used by the endpoint that receives the Initiation message, if it has already left the previous Tag-lock or Tag-lock-new mode.

Association re-initialization affects ongoing transmission and their resources. The receiver of the new Initiation may need to perform garbage-collection on its resources, including:

A) automatically terminating all existing streams within the current association and releasing the resources,

B) cancelling any running timers,

C) removing all outstanding datagrams of the current association from its retransmission queue, and

D) optionally, notifying the upper layer about the re-initialization.

4. Transfer User Datagram

The receiver of a user datagram shall always acknowledge the reception to the sender of the datagram. Normally, delayed acknowledgment shall be used. The delay shall be controlled by a T2-receive timer.

At the expiration of T2-receive timer, if there is out-bound user data, the ack should be piggy-backed on the data part of the out-bound user datagram, occupying the TSN Seen field (see section 2.3). Otherwise, a stand-alone Extended Data Ack shall be used to carry the acknowledgment.

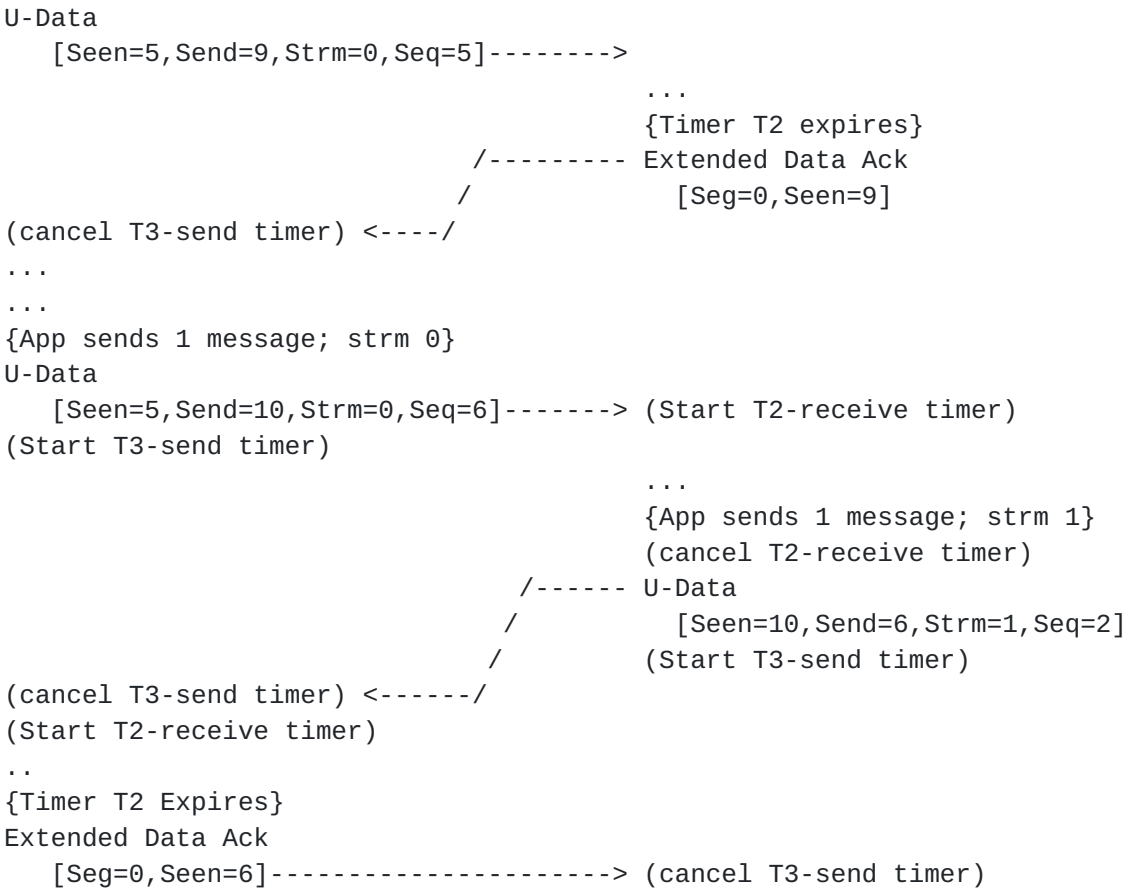
When Extended Data Ack is used, the sender shall fill the Highest

Consecutive TSN Seen field to indicate the highest TSN Send number it has received from the peer. Any received segments must also be reported (see sections 2.2.2 and 4.5).

The following example illustrates both stand-alone and piggy-backed acknowledgments:

```
Endpoint A                                        Endpoint Z
{App sends 3 messages in strm 0}
U-Data
   [Seen=5,Send=7,Strm=0,Seq=3]--------> (Start T2-receive timer)
(Start T3-send timer)

U-Data
   [Seen=5,Send=8,Strm=0,Seq=4]-------->
```



4.1 Timer Management Rules

The the following rules shall be used to manage the timers during normal datagram transfer, unless otherwise stated for some special cases:

A) When a user datagram is received, the endpoint shall start a T2-receive timer if no T2-receive timer is currently running. Upon the expiration of the T2-receive timer, the endpoint shall acknowledge to the sender all the un-acked user datagrams it has received.

B) When a user datagram is sent out, the sending endpoint shall start a T3-send timer if no T3-send timer is currently running.

If the T2-receive timer is running, the endpoint shall first stop the T2 timer, piggy-back an ack (or Extended Data Ack) onto the out-bound datagram, and then start a T3-send timer.

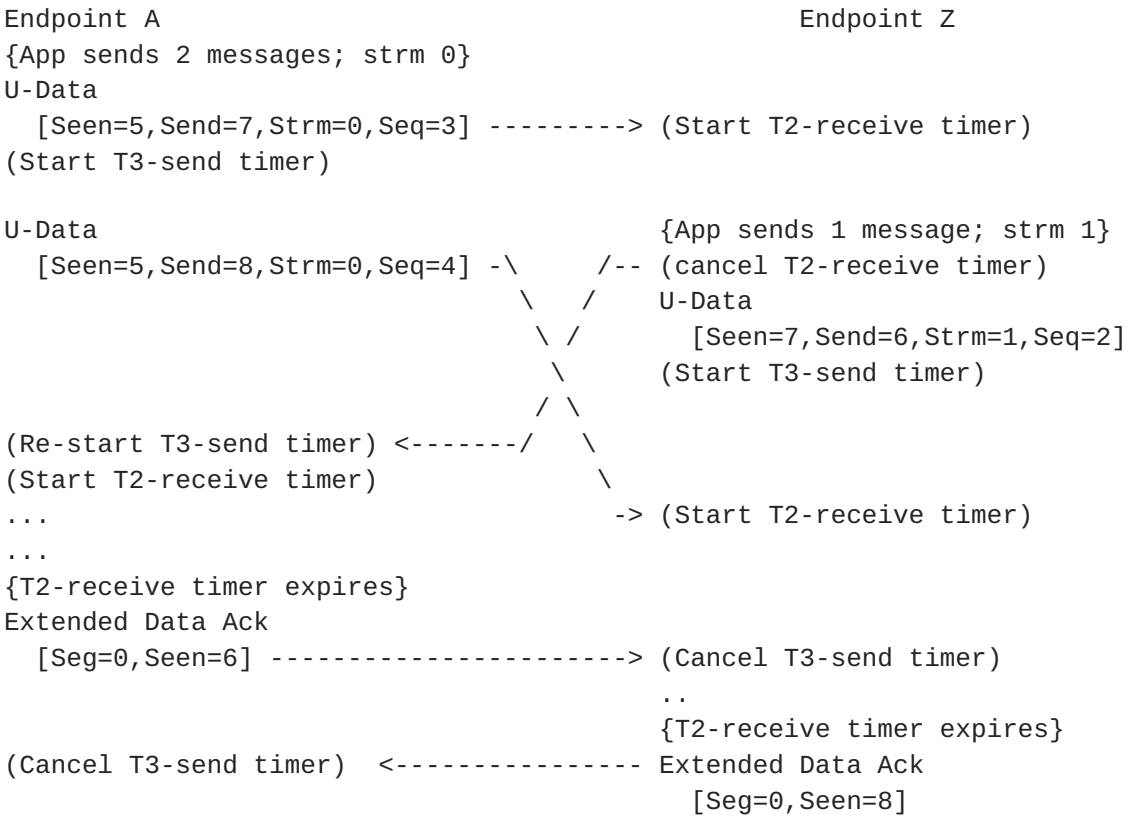
If the T3-send timer expires, the endpoint shall follow the rules described in 4.6 for possible re-transmission of the un-acked datagrams.

Moreover, whenever the T3-send timer is started the RTT estimate last calculated for that remote network address should be added to the base T3-send timer value (see sections 6.2 and 6.3 for RTT).

C) When all outstanding datagrams are acknowledged, the T3-send timer shall be stopped if one is still running.

D) If an endpoint has a T3-send timer running and receives a partial acknowledgment (one that acknowledges some of the outstanding datagrams), the endpoint shall restart the T3-send timer.

The following example shows the use of various timers.



4.1.1 T3-send Timer Adjustment with RTT

The sender shall keep track of the latest RTT measurement for the destination IP address (or addresses if the remote host is multi-homed) of its peer. Three procedures for obtaining RTT measurements are defined in sections 4.7, 6.2, and 6.3, respectively. And the calculation of RTT should follow the method described in [4].

Every time when a new datagram is sent for the first time (i.e., not for re-transmission), the following procedure shall be applied to determine the T3-send timer value:

1. TL3-value = 'TL3-default'

2. if TL3-value <= Receiver's T2-Recv + highest-RTT,
   TL3-value = TL3-value + highest-RTT

```
    end-if
```

3. T3-send = TL3-value + network-RTT

where, 'TL3-default' is a protocol parameter configurable by the endpoint, receiver's T2-Recv timer value is known during the association initiation (see section 3.1.1), the highest-RTT is the current highest RTT measurement across all the destination IP addresses available for transmission, and, the network-RTT is the current RTT measurement of the destination IP address this transmission is to take place (see section 4.2.1 for the determining of destination IP address).

However, if the previous T3-send timer expired and is being re-started for a re-transmission, the timer back-off rules defined in section 5.2 shall be used instead.

4.2 Multihoming Rotation

4.2.1 Remote Multihoming Rotation

When an endpoint is transmitting to a remote multi-homed endpoint, the transmitting endpoint shall rotate between destination IP addresses. Every time the application transmits a datagram, MDTP MUST keep track of the remote IP address to which it sent the datagram in the MDTP

protocol variable 'last.sent.intf'. MDTP should rotate each send in a round robin fashion amongst all available destination IP addresses on the remote multi-homed host and should update the protocol variable 'last.sent.intf' to indicate which destination IP address it last used.

If possible, acks should be transmitted to the same IP address from which the acked messages were received. When acknowledging multiple messages, this may not be possible. In the latter case, MDTP SHOULD rotate the transmission of acknowledgments to all remote IP addresses.

The MDTP implementation MUST allow an application to override this rotation by specifying the destination IP address to which to send a datagram. The implementation must also provide an interface to add and remove a remote IP address from rotation eligibility.

4.2.2 Local Multihoming Rotation

As discussed in section 3.3.4 of RFC 1122, an endpoint MAY rotate transmitted messages amongst all local network interfaces by specifying the local IP address and UDP port or it may allow the networking protocol to decide which local IP address (and network interface) to use to transmit a datagram..

If possible, acks should be transmitted from the same IP address over which the acked messages were received. When acknowledging multiple messages, this may not be possible. In the latter case, MDTP SHOULD rotate the transmission of acknowledgments from all configured IP address/port pairs.

4.3 Stream Sequence Number

The datagram stream sequence number shall always be set to 1 when the stream is opened.

Also, when the stream sequence number reaches the value 0xffff the next sequence number shall be set to 1. Sequence number '0' has special meaning (see section 4.4) and shall not be used in normal sequence number rotation..

4.4 Ordered and Un-ordered Delivery

Normally, the receiver shall ensure the user datagrams within any given stream be delivered to the upper layer according to the order of their stream sequence number. If there are datagram arrived out of order of their stream sequence number, the receiver must hold the received datagrams from delivery until they are re-ordered.

However, a sender can set the stream sequence number of a user datagram to 0, to indicate that no ordering shall be performed on that

datagram within that stream. Upon the reception of the datagram, the

receiver must by-pass the ordering mechanism and immediately delivery the datagram to the upper layer.

This provides an effective way to transmit "out-of-band" data in any given stream. Also, a stream can be used as an "un-ordered" stream by simply setting the stream sequence number of each out-bound user datagram to 0.

4.5 Report Missing Datagrams

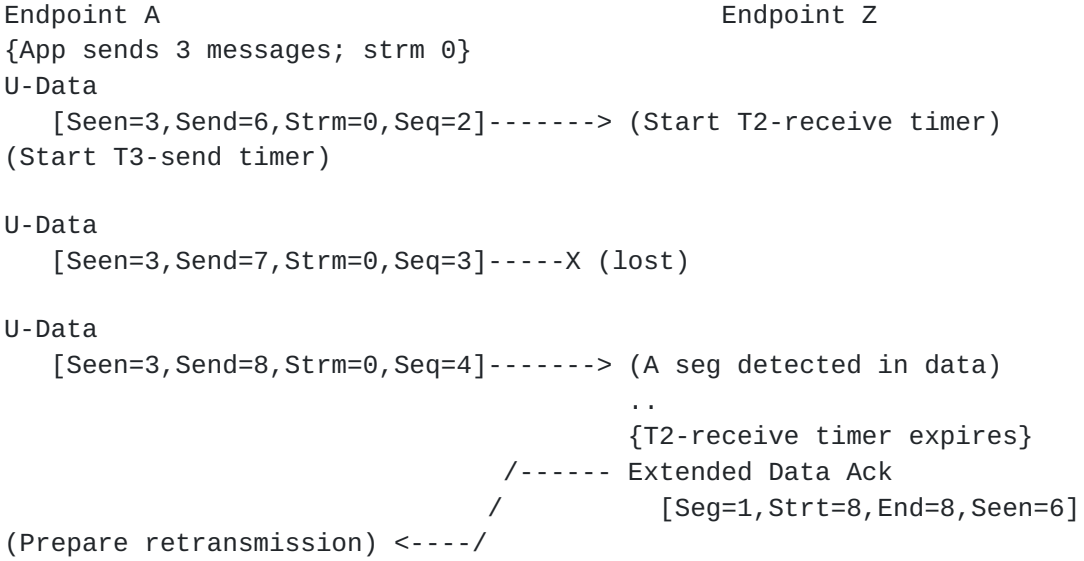
MDTP uses a receiver-based retransmission policy, where the sender attempts to elicit from the receiver information on the missing datagrams before the retransmission.

If a receiver detects holes in the received user datagram sequence (by examining TSN Send numbers), an Extended Data Ack with segment reports shall be sent back to inform the sender so that the sender can calculate and re-transmit the missing datagrams.

Multiple segments can be indicated in one single Extended Data Ack (see section 2.2.2).

If there is outbound user data, the endpoint shall piggy-back the Extended Data Ack with the user data in the same MDTP datagram, and the TSN Seen field in the data part shall not be used, i.e., the sender shall set the field to 0x0 and the receiver shall ignore it.

The following example shows the use of segment report in an Extended Data Ack.



In this example, when "Z" receives the third datagram from "A" it realizes that a gap exists in the received data. At the expiration of T2-receive timer, "Z" sends an Extended Data Ack with a segment report

to "A" to indicate the missing datagram.

When the peer endpoint is multi-homed, the Extended Data Ack should be sent out to the destination IP address specified in the MDTP protocol variable 'last.good.intf'. The value of 'last.good.intf' is always updated to point to the source IP address from which the last datagram from the peer endpoint arrived.

4.6 Range Check on TSN

For security reasons, the receiver must check the range of the TSN Send value in each received user datagrams.

Assume that the highest TSN received from a peer is T and the maximal window length of the same peer is W (exchanged during association initiation, see section 3.1). When the next user datagram arrives from this peer, the receiver shall silently discard the datagram if the TSN Send value carried in the datagram is greater than T+W (calculation rounds up at 0x7fffffff to 0x1).

4.7 Advisory Ack Request

An endpoint may use Advisory Ack Requests to improve bandwidth utilization, in combination of the window control (see section 5.1).

Advisory Ack Request shall always be piggy-backed on an outbound user datagram.

The endpoint should send an Advisory Ack Request to its peer when:

A) it reaches half of its window length with the sending of the current user datagram, or

B) it detects that the next send will reach the full window length with the sending of the current user datagram.

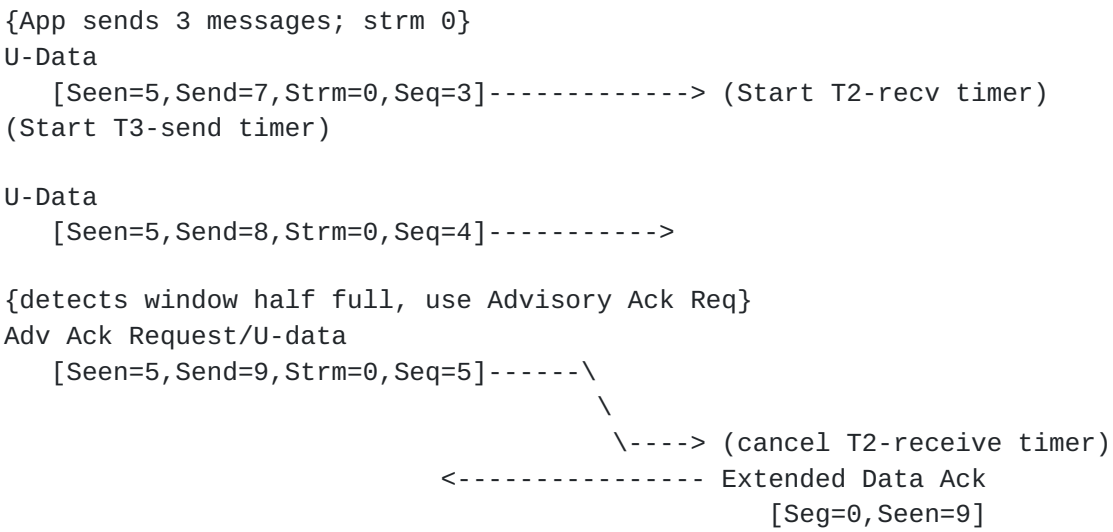
After the receiver detects the Advisory Ack Request in the control part of the datagram, it should handle it with the following rules:

A) The receiver may choose to ignore the peer's Advisory Ack Request for any reasons, such as flow control, etc, and move on to process the data part.

B) If the receiver chooses to respond, it should, at the end of processing the data part, immediately send an Extended Data Ack to acknowledge all the un-acked datagrams (including the one it just processed), and cancel its T2-receive timer if one is still running.

The following diagram shows an example of using Advisory Ack Request:

Endpoint A Endpoint Z



An endpoint sending an Advisory Ack Request may also use this request for its RTT calculation. The sending endpoint may note the time mark when sending the datagram with the Advisory Ack Request. When the peer endpoint responds with an Extended Data Ack, the sender of the Advisory Ack Request may use the time mark of the arriving Extend Data Ack and the stored time mark to calculate the RTT as defined in [4]. However, the sender of the Advisory Ack Request shall abandon the RTT calculation if more datagrams are sent to its peer and no Extended Data Ack is received.

4.8 CRC-16 Utilization

When sending a datagram, the sender can choose to strengthen the data integrity
of the transmission by including a CRC-16 value of the datagram.

After the datagram is constructed, the sender shall:

1) set the C Bit to '1' and fill the 28 bit CRC-16/MDTP Protocol Identifier field with '0', and the 4 bit Version field to the current MDTP version number (binary 0011).

2) calculate a CRC-16 value of the whole datagram, including the MDTP common header, the Control Parameter Part if present, and the Data Part if present,

3) put the resultant CRC-16 value into the most significant 16 bits of the CRC-16/MDTP Protocol Identifier, and leave the rest of the bits unchanged.

When a datagram is received, the receiver must first check the C Bit. If the C Bit is set, the receiver shall:

1) store the received CRC-16 value (the most significant 16 bits of the first word of the datagram),

2) replace the 16 bit CRC-16/MDTP Protocol Identifier field with '0' and calculate a CRC-16 value of the whole received datagram,

3) verify that the calculated CRC-16 value is the same as the received CRC-16 value, and

4) handle the datagram as an invalid MDTP datagram if the CRC-16 values mismatch .

If the C Bit is not set, the receiver shall check the MDTP Protocol Identifier instead, and handle the datagram as an invalid MDTP datagram if the check fails.

The default procedure of handling invalid MDTP datagrams is to silently discard them.

5 Congestion Controls

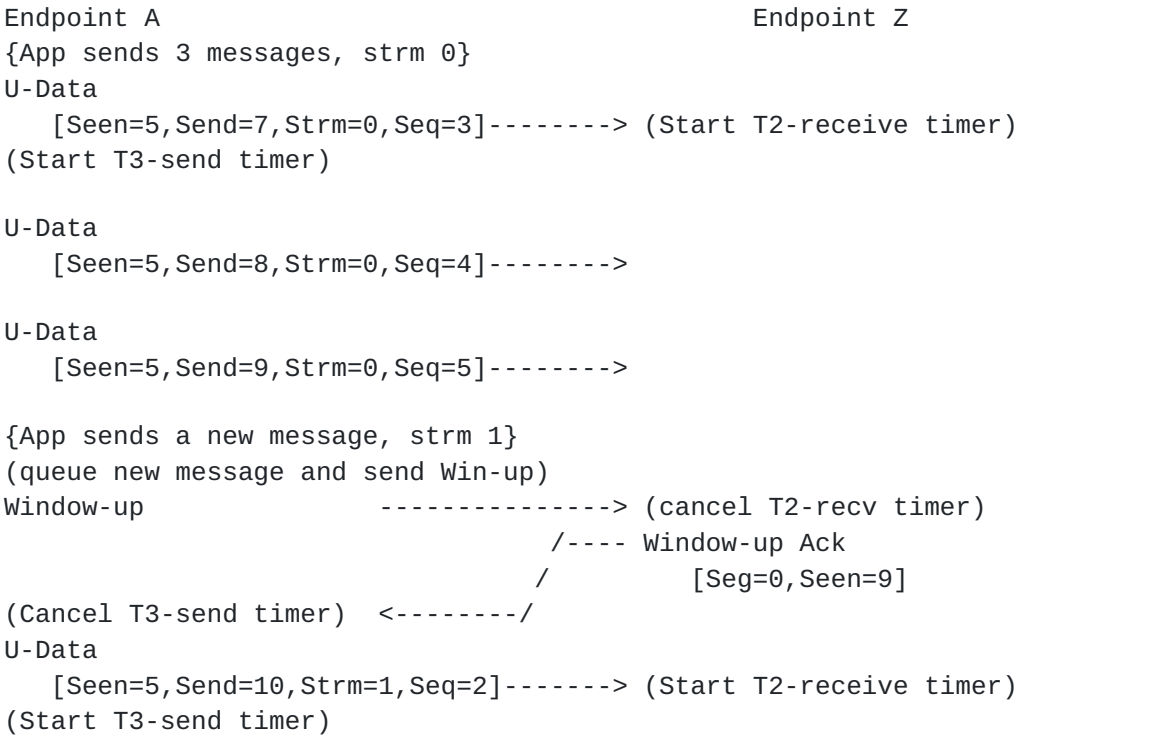
Several different mechanisms shall be used jointly to achieve congestion control in MDTP. These mechanisms are always used in regard to the association, not a individual stream.

5.1 Send with Window Control

The sending endpoint shall use a transmission window to control the number of outstanding datagrams, i.e., datagrams that have been sent, but yet to be acknowledged. The length of the window is defined as the maximal number of outstanding datagrams a sending endpoint can allow. This length is adjusted dynamically, depending on the current number of successful transmissions as well as the number of lost datagrams or retransmissions.

When the number of outstanding datagrams reaches the current window length, the endpoint shall still accept send requests from its upper layer, but shall transmit no more datagrams until some or all of the outstanding datagrams are acknowledged. The endpoint may also elect to queue only a specified number of datagram when the window is full. When this maximal number of queued datagrams is reached the endpoint shall return an error to its upper layer.

Moreover, when the window length is reached, the next send request from the upper layer will trigger a Window-up message to be transmitted. Upon receiving this Window-up the receiver must respond with a Window-up Ack, as illustrated by the following example (assuming current window length is 3):



In the above example, after the transmission of the first three datagrams, "A" reached its window length. The next message from the user triggered a Window-up that was sent to "Z". The Window-up shall contain no user data. In response, "Z" cancelled timer T2 and immediately sent a Window-up Ack. The arrival of this Window-up Ack effectively resolved all the outstanding datagrams at "A", thus allowing "A" to send out the next datagram.

5.1.1 Window Length Adjustment

The window length shall be initially set to 2, and shall then be dynamically adjusted based on datagram loss and acknowledgment.

If the current window length is less than or equal to 4, every time when the number of consecutive outstanding datagrams acknowledged in a single ack is equal to or greater than half the current window length, the sender's window length shall be raised by 1, until it reaches 'Max.Outstanding.dg' (which should be a user configurable parameter).

If the current window length is greater than 4, every time when the number of consecutive outstanding datagrams acknowledged in a single ack is equal to or greater than 4, the sender's window length shall be raised by 1, until it reaches 'Max.Outstanding.dg'.

In the following circumstances, the sender's window length shall be decreased. However, when the window length reaches 2 it shall not be decreased any further.

Firstly, if the sender receives a stand-alone Extended Data Ack with a Seen TSN that equals to the highest consecutive acked TSN, the sender should consider this as a duplicate ack and lower its window size by 4.

The peer endpoint may report reception gaps which may correspond to multiple datagram losses (indicated by an Extended Data Ack or

Window-up Ack). If between 1 to 3 datagrams are lost, the window length shall be decreased by 1. If between 4 to 7 datagrams are lost, the window length shall be decreased by 2. If 8 or more datagrams are lost, the window length shall be decreased by 4.

Any time a Window-up Ack is received by an endpoint, as a response to a previous Window-up it sent, the endpoint shall decrease its window by 1 if the window has not advanced from the time at which the Window-up was sent out.

Also, if a timeout forces a retransmission the sender's window length shall be reduced to half of its currently value.

The following table summarizes these rules:

duplicate ack received by sender	Adjust down by 4
8 or more datagrams lost	Adjust down by 4
4 to 7 datagrams lost	Adjust down by 2
1 to 3 datagrams lost	Adjust down by 1
Timeout forced retransmission	Adjust down by 1/2 of the current window.
Window-up Ack received and the window has not advanced.	Adjust down by 1
4 or more consecutive datagrams acknowledged (window length > 4)	Adjust up by 1
1/2 Window length or more acked (window length <=4)	Adjust up by 1

5.2 Send Timer Back-off at Re-transmission

Whenever a T3-send timer expires, the endpoint shall re-transmit the un-acked datagram that has the highest TSN Send value and re-start the T3-send timer, unless:

A) If the current window length is reached, a Window-up message shall be sent out (see section 5.1), or

B) If the current window length is not reached and there is still user data pending for transmission, a new datagram with user data shall be sent out and T3-send timer shall be restarted.

When the T3-send timer is re-started at a retransmission, the following back-off rules shall be applied to determine the value of the new timer:

1. TL3-value = TL3-value * 2

2. T3-send = TL3-value + network-RTT

where, TL3-value is the protocol variable keeping the previous and current T3-send timer base value, and the network-RTT is the current RTT measurement of the destination IP address the re-transmission is to be sent to.

Note: the T3-send timer base value shall be restored to its default value 'TL3-default' when a datagram is received from the peer endpoint.

The total number of consecutive re-transmissions to all destination IP addresses in an association shall be recorded. If this value exceeds the limit defined in 'Max.Retransmit', the sending endpoint shall consider the peer endpoint unreachable and shall stop transmitting any more data to it. The sending endpoint MAY report the failure to the upper layer, including all datagrams in its out-bound buffer which have not been acknowledged. Whenever a datagram is received from a peer endpoint the total number of retransmissions shall be cleared.

6. Network Management

6.1 Failure Detection in Redundant Networks

When the peer endpoint is multi-homed, the re-transmission of a datagram should be attempted to the destination IP address specified in the MDTP protocol variable 'last.good.intf'. The value of 'last.good.intf' is always updated to point to the source IP address from which the last datagram from the peer endpoint arrived.

The number of consecutive T3-send timeout events is also recorded in a variable 'retran.count' for each destination IP address. This count should be incremented when a T3-send time-out event occurs for that destination IP address. Every time a datagram is received from a peer endpoint, the receiving endpoint shall reset to 0 the 'retran.count' corresponding to the source IP address .

If the value in 'retran.count' exceeds half of the value of the protocol parameter 'Max.Retransmit', the destination IP address shall be reported to the upper layer as out-of-service and shall be removed from eligibility for rotation. When re-transmitting a datagram, the re-transmission should use 'last.good.intf' as the preferred destination IP address to which to re-transmit, unless 'last.good.intf' points to the destination IP address on which the original T3-send time-out event occurred.

In the event that a datagram is received from an IP address that has been reported as out-of-service, the 'retran.count' shall be cleared as specified above, the destination IP address shall be reported as in-service to the upper layer, and the destination IP address shall be considered valid for rotation.

6.2 RTT Measurement

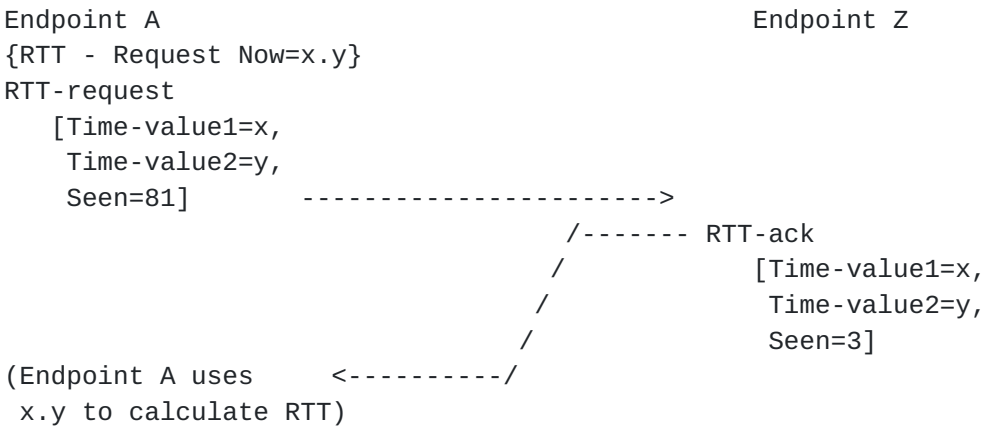
On occasions an endpoint of an association may need to perform an RTT measurement of the network (or one of the redundant networks) between itself and its peer.

RTT-request and RTT-ack messages shall be used to perform the RTT measurement. In the messages, two 32 bit long opaque integers are used in the control parameter field to carry the time value.

At the request of its upper layer, an endpoint shall initiate an RTT measurement by sending an RTT-request (to a specific network if redundant networks exist). The sender shall also place in Time value 1 and Time value 2 the value of the current time mark.

Upon the reception of this RTT-request message, the recipient shall immediately respond with a RTT-ack to the sender (over the same network on which the RTT-request arrives if the recipient is multi-homed), with the time mark carried in the original RTT-request copied into its own Time value fields.

Upon the reception of this reply, the sender shall use the time mark in the reply RTT-ack to calculate the RTT (to the specific destination IP address if redundant networks exist) as defined in [4].



6.3 Network Heart Beat

At the request of its upper layer, an endpoint shall enable heart beat to a specific peer with which it has an established association.

The RTT-request message defined in section 2.2 shall be used as the heart beat while the RTT-ack shall be used as the heart beat response.

After having heart beat enabled, the endpoint shall transmit a heart beat to that specific peer and start a T5-heartBeat timer. The peer shall immediately respond to the heart beat in the same manner as the

RTT measurement procedure described in section 6.2. This response, as well as the new RTT measurement, shall be stored by the endpoint.

When the T5-heartBeat timer expires, the endpoint shall first check if the previous heart beat has been responded to (on the same network it was sent in the case of multi-homed hosts). If not, the destination IP address to which the last heart beat was sent shall have the 'retran.count' incremented and checked following the rules described in section 6.1. Then, the endpoint shall send another heart beat and re-start the T5-heartBeat timer.

In the case where one or both endpoints are multi-homed, the sending of Heart beats shall follow the network rotation rules outlined in section 4.2.

If, before the expiration of T5-heartBeat timer, a datagram is received by the endpoint, the T5-heartBeat timer shall be stopped and restarted.

The suggested interval for T5-heartBeat timer is 4000 ms, and may be dynamically adjusted by adding the current RTT measurement if it is available.

7. Termination of Association

Before an endpoint terminates itself, it shall send an Abort message to each of its peer endpoints in all existing associations. The Abort shall be sent without requiring an acknowledgment from the peer endpoint. However, the sender of the Abort message MUST fill in the peer's Init-Tag.

When the peer endpoint receives the Abort, after verifying the Tag, the peer shall remove the sender from its record, and optionally report the termination of the sender to its upper layer. However if the Tag sent with the Abort message is incorrect, the peer must silently discard the Abort message.

The following shows an example of the termination of Endpoint A:

```
Endpoint A
{App indicates termination}
Abort
    [Tag-X]   ---------------------------------> to Endpoint X

Abort
    [Tag-Y]   ---------------------------------> to Endpoint Y

Abort
    [Tag-Z]   ---------------------------------> to Endpoint Z
```

7.1 Graceful Shutdown of an Association

An endpoint in an association may decide to "graceful shutdown" the association without completely closing it down. With graceful shutdown, both endpoints shall remove any record and pending datagrams associated with the association. Further communications between the two endpoints can be resumed by going through a re-initialization procedure (see section 3.5.4).

A Graceful Shutdown message shall be sent to the peer endpoint of the association, and the peer shall send back an acknowledgment. Note that it shall be the responsibility of the endpoint that sends the Graceful Shutdown message to assure that all the outstanding datagrams from its side have been resolved before it initiates the graceful shutdown procedure.

In the Graceful Shutdown message, the sender shall indicate the highest TSN Seen it has received from the peer, as well as the Init-Tag of the peer.

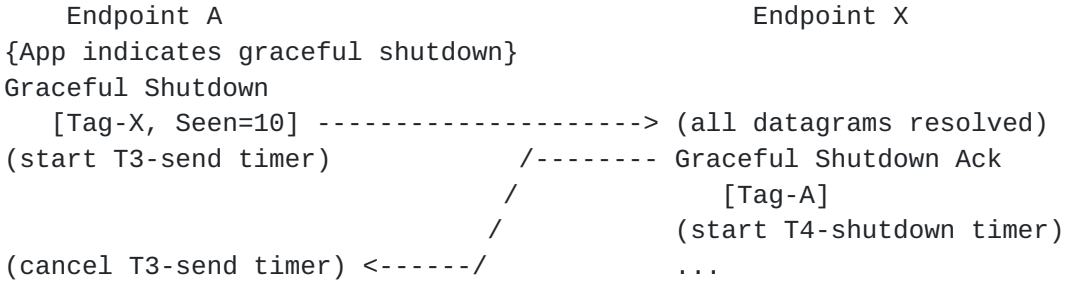
Upon the reception of the Graceful Shutdown, the peer shall first verify that Tag value contained in the Graceful Shutdown message is valid. If the Tag is invalid, the message must be silently discarded.

The peer then shall verify, by checking the Seen numbers from the Graceful Shutdown message, that all the out-bound datagrams have reached the destination. Otherwise, the peer shall re-transmit all lost datagrams.

After sending the Graceful Shutdown, if the endpoint receives any new user datagram it shall immediately respond with an Extended Data Ack and re-start its T3-send timer.

The peer shall send a Graceful Shutdown Ack when all the outstanding datagrams are acknowledged, then start a T4-shutdown timer. The endpoint, after receiving the Graceful Shutdown Ack, must also validate the Tag value contained in the message. If it does not match the Tag value that unlocked the association, the message should be silently discarded.

The following sequence shows an example of Graceful Shutdown:



(clean-up the association)

(T4-shutdown expires)
(clean-up the association)

Both endpoints shall reject any new data request from their upper layers while the graceful shutdown procedure is in progress.

8. Stream Operations

8.1 Stream Initiation

An MDTP association between the two endpoints must be established before any stream operation.

Except for the global stream (i.e, stream 0) and the pre-opened streams (see section 3.1.1), a stream shall be initiated (opened) by the sender before datagrams can be passed in that stream. When a stream is no longer used, it shall be terminated (closed) by the endpoint that opened the stream. Moreover, both sides of the association shall be able to initiate or terminate streams independently. Streams are unidirectional.

The sender initiates a stream by sending a Stream Initiation. In addition to specifying the Stream Identifier, the sender must set the Init-Tag field of the Stream Initiation to the Tag value of the peer endpoint.

The sender shall also attach the stream-specific data if any (usually provided by the upper layer), with the Stream Initiation. Otherwise, the Size of Stream Info field shall be set to 0x0.

After sending out the Stream Initiation, the sender shall start a T6-streamInit timer. If this timer expires, the sender shall re-transmit the Stream Initiation. The value and adjustment rules of T6-streamInit timer is the same as that of the T3-send timer (see sections 4.1.1 and 5.2).

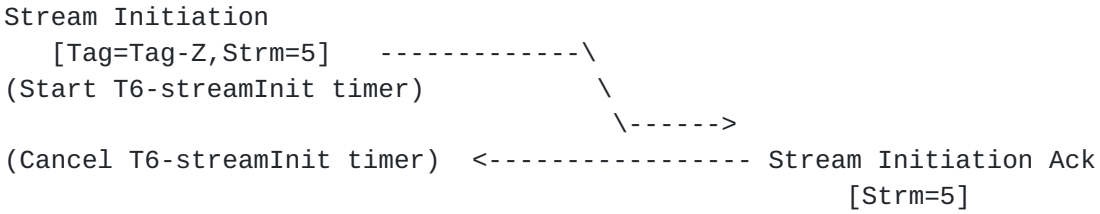
Upon the reception of the Stream Initiation, the peer must first verify that the correct Tag value is carried in the Init-Tag field of the Stream Initiation. The peer must silently discard the Stream Initiation if the tag value is found incorrect.

Then, the peer shall respond immediately with either a Stream Initiation Ack if it chooses to establish the requested stream, or a Stream Initiation Nack if it chooses to reject the request for reasons such as lack of resources.

The arrival of the Stream Initiation Ack or Nack shall cause the sender to cancel its T6-streamInit timer.

The following example shows the opening of stream 5 by "A":

```
    Endpoint A                                  Endpoint Z
{App Initiates stream 5}
```



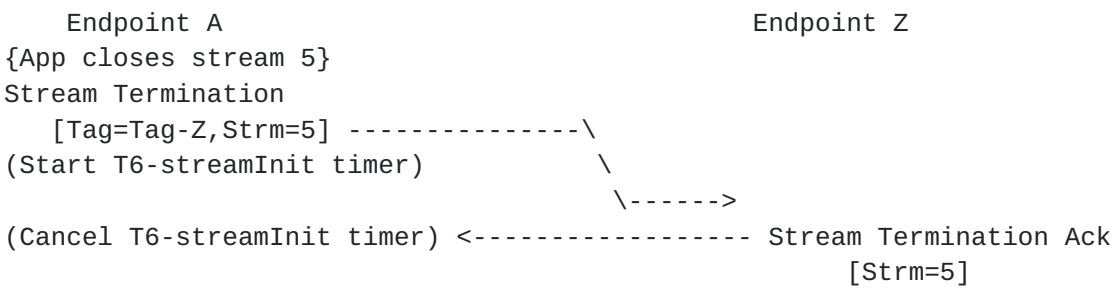
8.2 Stream Termination

An endpoint shall be allowed to terminate any one of the streams it opened, by sending a Stream Termination to its peer. However, stream 0 is not allowed to be terminated, and if an endpoint receives a termination message for stream 0 it must silently discard the message.

The same Tag verification process and timer rules used for stream initiation shall be applied to stream termination.

The peer shall immediately send a Stream Termination Ack in response to the Stream Termination.

The following example shows the termination of stream 5 by "A":



Received datagrams associated with a terminated stream shall be silently discarded. It is up to the endpoint to assure that all outstanding user datagrams in the stream are acknowledged before the stream termination.

8.3 Other Issues with Stream Operations

When an association is re-initialized (see section 3.5.4), all existing streams within that association will be automatically terminated.

The receiver shall silently discard any datagrams associated with a stream which has not yet been opened or has already been terminated.

9. Interface with Upper Layer

The upper layer protocols (ULP) shall request for services by passing primitives to MDTP and shall receive notifications from MDTP for various events.

The primitives and notifications described in this section should be used as a guideline for implementing MDTP.

A) Init.MDTP primitive

This primitive allows MDTP to initialize its internal data structures and allocate necessary resources for setting up its operation environment. Note that once MDTP is initialized, ULP can communicate directly with any other endpoints without re-invoking this primitive.

Mandatory attributes:

None.

Optional attributes:

The following types of attributes may be passed along with

the primitive:

- Timer selection and its operation syntax -- to indicate to MDTP an alternative timer the MDTP should use for its operation.
- Initial MDTP operation mode;
- IP port number, if ULP wants it to be specified;

B) Init.Association

This primitive allows the upper layer to initiate an association to a specific peer endpoint. The peer endpoint shall be specified by one of the IP address/port pairs which define the endpoint (see section 1.1).

Mandatory attributes:

- associationID - specified as one of the IP address/port pairs of the peer endpoint with which the association is to be established.

Optional attributes:

- eligibleNetList - a list of destination IP address/port pairs that the peer endpoint is allowed to use in its network rotation. By default, all destination IP address/port pairs on the peer are available.

C) Term.Association

Terminating an association.

Mandatory attributes:

- associationID - specified as one of the IP address/port pairs of the peer endpoint with which the association is to be terminated.

Optional attributes:

None.

D) Send.Data primitive

This is the main method to send datagrams via MDTP.

Mandatory attributes:

- data - This is the payload ULP wants to transmit;
- size - The size of the payload in number of octets;
- associationID - One of the IP address/port pair of the peer endpoint. Note that the actual destination address sent to will be determined by MDTP due to the network rotation, unless the current mode prohibits MDTP network rotation; in such a case the datagram will be sent to the IP address/port specified by associationID.

Optional attributes:

 o mode-flags - This indicates a new MDTP operation mode, taking effect
   immediately including the current datagram send;

- context - optional information that will be carried in the Send.Failure notification to the ULP if the transportation of this datagram fails.

- streamID - to indicate which stream to send the data on. By default, the global stream will be used.

E) Receive.Data primitive

This primitive shall return the first datagram in the MDTP in-queue to ULP, if there is one available. It may, depending on the specific implementation, also return other informations such as the sender's address, whether there are more datagrams available for retrieval, etc. The behavior is undefined if no datagram is available when this primitive is invoked.

Mandatory attributes:

- buffer - the memory location indicated by the ULP to store the received datagram.

Optional attributes:

- associationID - the storage to be filled with one of the IP address/port pair of the peer endpoint that sent this datagram.

F) Data.Arrive notification

MDTP shall invoke this notification on the ULP when a datagram is successfully received and ready for retrieval.

G) Send.Failure notification

If a datagram can not be delivered MDTP shall invoke this notification on the ULP.

The following may be optionally be passed with the notification:

- data - the location ULP can find the un-delivered datagram.
- context - optional information associated with this datagram (see D).
- associationID - One of the IP address/port pair of the peer this datagram was attempted to be sent to.

H) Network.Status.Change notification

When a endpoint-id is marked down (e.g., when MDTP detects a failure), or marked up (e.g., when MDTP detects a recovery), MDTP shall invoke this notification on the ULP.

The following shall be passed with the notification:

- endpoint-id - This indicates the IP address/port of the peer endpoint affected by the change;
- new-status - This indicates the new status.

I) Communication.Up notification

This notification is used when MDTP becomes ready to send or receive datagrams, or when a lost communication to an endpoint is restored.

The following shall be passed with the notification:

- status - This indicates what type of event that has occurred;
- associationID - An IP address/port to identify the peer endpoint;

J) Communication.Lost notification

When MDTP loses communication to an endpoint completely or detects that the endpoint has performed a abort or graceful shutdown operation, it shall invoke this notification on the ULP.

The following shall be passed with the notification:

- status - This indicates what type of event that has occurred;
- associationID - An IP address/port number to identify the peer endpoint;

The following may be optionally passed with the notification:

- packets-enqueue - The number and location of un-sent datagrams still holding by MDTP;
- last-acked - the sequence number last acked by that peer endpoint;
- last-sent - the sequence number last sent to that peer endpoint;

K) Change.Network.Rotation primitive

When the upper layer wants to inform MDTP to make a specific network eligible or ineligible for in network rotation, the upper layer will send this primitive to MDTP.

Mandatory attributes:

- action - This indicates if the network is to be made eligible or ineligible for network rotation.
- network-id - This is the IP address/port of the peer endpoint to be added or removed from network rotation consideration.

L) Open.Stream primitive

This should be used by the upper layer to open a new outbound stream.

Mandatory attributes:

- associationID - One of the IP address/port to identify the peer endpoint of the association to which the stream is to be opened. An

association must have existed at the time of stream open.

Optional attributes:

- streamInfo - The upper layer should use this field to pass any stream-specific data to the other endpoint of the association.

M) Open.Stream.Succeed notification

This should be used to report the successful opening of an new outbound stream.

Mandatory attributes:

- associationID - One of the IP address/port to identify the peer endpoint of the association to which the outbound stream has been successfully opened.

- streamID - The stream number of the outbound stream assigned by MDTP.

Optional attributes:

- streamInfo - The streamInfo used for opening this outbound stream.

N) Open.Stream.Rejected notification

This reports to the ULP that the open of an outbound stream is rejected by the peer endpoint.

Mandatory attributes:

- associationID - One of the IP address/port to identify the peer endpoint of the association by which the stream open is rejected.

Optional attributes:

- streamInfo - The info used in the failed attempt of the stream open.

O) Close.Stream notification

This should be used to report the successful closing of an outbound stream.

Mandatory attributes:

- associationID - One of the IP address/port to identify the peer endpoint of the association with which the stream is closed.
- streamID - The stream number of the closed stream.

P) Peer.Open.Stream notification

This notifies the ULP that a new inbound steam is opened by a peer endpoint.

Mandatory attributes:

- o associationID - One of the IP address/port to identify the peer endpoint of the association to which the stream is opened.
- o streamID - The stream number of the new inbound stream assigned by the peer.

Optional attributes:

- o streamInfo - The stream-specific Information passed from the peer endpoint.

Q) Peer.Close.Stream notification

This reports to the ULP the closing by a remote peer of an inbound stream.

Mandatory attributes:

- o associationID - One of the IP address/port to identify the peer endpoint of the association by which the inbound stream is closed.
- o streamID - The stream number of the closed inbound stream.

R) Close.Stream primitive

This shall be used by the upper layer to close an outbound stream.

Mandatory attributes:

- o associationID - One of the IP address/port to identify the peer endpoint of the association to which the outbound stream is to be closed.
- o streamID - The stream identifier to identify the stream to be closed (this should be the number returned by the Stream.Open primitive on this stream).

10. Suggested MDTP Timer and Protocol Parameter Values

The following are suggested timer values for MDTP:

```
T1-init Timer       -   160 ms
T2-receive Timer    -    20 ms
T3-send Timer       -   160 ms (TL3-default)
T4-shutdown Timer   -   300 ms
T5-heartBeat timer  -  4000 ms (TL5-default)
T6-streamInit timer -   same as T3-send
```

The following protocol parameters are recommended:

```
Max.Outstanding.dg      - 20 messages
Max.Retransmit          - 10 attempts
Max.Init.Retransmit     -  8 attempts
```

11. Abbreviations

MDTP - Multi-network Datagram Transmission Protocol.

NAT - Network Address Translation

RTT - Round Trip Time

TSN - Transport Sequence Number

ULP - Upper Layer Protocol

12. Acknowledgments

The authors wish to thank Brian Wyld, A. Sankar, Henry Houh, Gary Lehecka, Lyndon Ong, Greg Sidebottom, Lixia Zhang, Jarno Rajahalme, Heinz Prantner, Matt Holdrege, Kelvin Porter, Richard Band, and many others for their invaluable comments.

13. Authors' Addresses

Randall R. Stewart
Cellular Infrastructure Group
Motorola, Inc.
1475 W. Shure Drive, #2C-6
Arlington Heights, IL 60004
USA

Tel: +1-847-632-7438
EMail: stewrtrs@cig.mot.com

Qiaobing Xie
Cellular Infrastructure Group
Motorola, Inc.
1501 W. Shure Drive, #2309
Arlington Heights, IL 60004
USA
Tel: +1-847-632-3028
EMail: xieqb@cig.mot.com

Ken Morneau
Cisco Systems Inc.
13615 Dulles Technology Drive
Herndon, VA. 20171
Tel: +1-703-484-3323
EMail:kmorneau@cisco.com

Chip Sharp
Cisco Systems Inc.
7025 Kit Creek Road
Research Triangle Park, NC 27709
Tel: +1-919-851-2085
EMail:chsharp@cisco.com

Hanns Juergen Schwarzbauer
SIEMENS AG
Hofmannstr. 51
81359 Munich, Germany
Tel: +49-89-722-24236
EMail: HannsJuergen.Schwarzbauer@icn.siemens.de

Tom Taylor
Nortel Networks
1852 Lorraine Ave.
Ottawa Ontario Canada
K1H6Z8
Tel: +1-613-736-0961
EMail:taylor@nortelnetworks.com

Ian Rytina
Ericsson Australia
37/360 Elizabeth Street
Melbourne, Victoria 3000, Australia
Tel:
EMail:ian.rytina@ericsson.com

14. References

[1] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification", RFC 791, USC/Information Sciences Institute, September 1981.

[2] Postel, J., "User Datagram Protocol", RFC 768, USC/Information Sciences Institute, August 1980.

[3] Postel, J. (ed.), "Transmission Control Protocol", RFC 793, USC/Information Sciences Institute, September 1981.

[4] Jacobson V., "Congestion Avoidance and Control", Proceedings of SIGCOMM '88, pp 314-329, August 1988.

[5] Seth, T., etc. "Performance Requirements for Signaling in Internet

Telephony", Internet-Draft <draft-seth-sigtran-req-00.txt>, May 1999.

[6] Rytina, I., "Framework for Generic Common Signaling Transport Protocol", draft-rytina-sigtran-generic-framework-00.txt>, Feb. 1999.

[7] Ashworth, J., "The Naming of Hosts", RFC 2100, April 1997.

[8] Braden, R., "Requirements for Internet hosts - Application and Support", RFC 1122, October 1989.

[9] Eastlake 3rd, D., Crocker, S., and Schiller, J., "Randomness Recommendations for Security", RFC 1750, December 1994.

[10] Bellovin, S., "Defending Against Sequence Number Attacks", RFC1948, May 1996

[11] ITU-T Recommendation Q.703 "Q.703 - Signaling link", July 1996.

This Internet Draft expires in 6 months from June 1999.