Networking Working Group
Request for Comments: 5326
Category: Experimental

M. Ramadas
ISTRAC, ISRO
S. Burleigh
NASA/Jet Propulsion Laboratory
S. Farrell
Trinity College Dublin

September 2008

Licklider Transmission Protocol - Specification

Status of This Memo

This memo defines an Experimental Protocol for the Internet community. It does not specify an Internet standard of any kind. Discussion and suggestions for improvement are requested. Distribution of this memo is unlimited.

IESG Note

This RFC is not a candidate for any level of Internet Standard. It represents the consensus of the Delay Tolerant Networking (DTN) Research Group of the Internet Research Task Force (IRTF). It may be considered for standardization by the IETF in the future, but the IETF disclaims any knowledge of the fitness of this RFC for any purpose and in particular notes that the decision to publish is not based on IETF review for such things as security, congestion control, or inappropriate interaction with deployed protocols. See RFC 3932 for more information.

Abstract

This document describes the Licklider Transmission Protocol (LTP), designed to provide retransmission-based reliability over links characterized by extremely long message round-trip times (RTTs) and/or frequent interruptions in connectivity. Since communication across interplanetary space is the most prominent example of this sort of environment, LTP is principally aimed at supporting "long-haul" reliable transmission in interplanetary space, but it has applications in other environments as well.

This document is a product of the Delay Tolerant Networking Research Group and has been reviewed by that group. No objections to its publication as an RFC were raised.

Table of Contents

1. Introduction

This document serves as the main protocol specification of LTP and is part of a series of documents describing LTP. Other documents in this series include the motivation document [LTPMTV] and the protocol extensions document [LTPEXT]. We strongly recommend reading the protocol motivation document before reading this document, to establish sufficient background and motivation for the specification.

LTP does Automatic Repeat reQuest (ARQ) of data transmissions by soliciting selective-acknowledgment reception reports. It is stateful, and has no negotiation or handshakes.

In an Interplanetary Internet setting deploying the Bundle Protocol that is being developed by the Delay Tolerant Networking Research Group, LTP is intended to serve as a reliable "convergence layer" protocol operating in pairwise fashion between adjacent Interplanetary Internet nodes that are in direct radio frequency (RF) communication. In that operational scenario, and potentially in some other deployments of the Bundle Protocol, LTP runs directly over a data-link layer protocol; when this is the case, forward error correction coding and/or checksum mechanisms in the underlying data-link layer protocol must ensure the integrity of the data passed between the communicating entities.

Since no mechanisms for flow control or congestion control are included in the design of LTP, this protocol is not intended or appropriate for ubiquitous deployment in the global Internet.

When LTP is run over UDP, it must only be used for software development or in private local area networks. When LTP is not run over UDP, it must be run directly over a protocol (nominally a link-layer protocol) that meets the requirements specified in Section 5.

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 [B97].

2. Terminology

(1) Engine ID

A number that uniquely identifies a given LTP engine, within some closed set of communicating LTP engines. Note that when LTP is operating underneath the Delay-Tolerant Networking (DTN) [DTN] Bundle Protocol [BP], the convergence layer adapter mediating the two will be responsible for translating between DTN endpoint IDs and LTP engine IDs in an implementation-specific manner.

(2) Block

An array of contiguous octets of application data handed down by the upper layer protocol (typically Bundle Protocol) to be transmitted from one LTP client service instance to another.

Any subset of a block comprising contiguous octets beginning at the start of the block is termed a "block prefix", and any such subset of the block ending with the end of the block is termed a "block suffix".

(3) Red-Part

The block prefix that is to be transmitted reliably, i.e., subject to acknowledgment and retransmission.

(4) Green-Part

The block suffix that is to be transmitted unreliably, i.e., not subject to acknowledgments or retransmissions. If present, the green-part of a block begins at the octet following the end of the red-part.

(5) Session

A thread of LTP protocol activity conducted between two peer engines for the purpose of transmitting a block. Data flow in a session is unidirectional: data traffic flows from the sending peer to the

receiving peer, while data-acknowledgment traffic flows from the receiving peer to the sending peer.

(6) Sender

The data sending peer of a session.

(7) Receiver

The data receiving peer of a session.

(8) Client Service Instance

A software entity, such as an application or a higher-layer protocol implementation, that is using LTP to transfer data.

(9) Segment

The unit of LTP data transmission activity. It is the data structure transmitted from one LTP engine to another in the course of a session. Each LTP segment is of one of the following types: data segment, report segment, report-acknowledgment segment, cancel segment, cancel-acknowledgment segment.

(10) Reception Claim

An assertion of reception of some number of contiguous octets of application data (a subset of a block) characterized by: the offset of the first received octet, and the number of contiguous octets received (beginning at the offset).

(11) Scope

Scope identifies a subset of a block and comprises two numbers -- upper bound and lower bound.

For a data segment, lower bound is the offset of the segment's application data from the start of the block (in octets), while upper bound is the sum of the offset and length of the segment's application data (in octets). For example, a segment with a block offset of 1000 and length of 500 would have a lower bound of 1000 and upper bound of 1500.

For a report segment, upper bound is the end of the block prefix to which the reception claims in the report apply, while lower bound is the end of the (smaller) interior block prefix to which the reception claims in the report do *not* apply. That is, data at any offset equal to or greater than the report's lower bound but less than its

upper bound and not designated as "received" by any of the report's reception claims must be assumed not received, and therefore eligible for retransmission. For example, if a report segment carried a lower bound of 1000 and an upper bound of 5000, and the reception claims indicated reception of data within offsets 1000-1999 and 3000-4999, data within the block offsets 2000-2999 can be considered missing and eligible for retransmission.

Reception reports (which may comprise multiple report segments) also have scope, as defined in Section 6.11.

(12) End of Block (EOB)

The last data segment transmitted as part of the original transmission of a block. This data segment also indicates that the segment's upper bound is the total length of the block (in octets).

(13) End of Red-Part (EORP)

The segment transmitted as part of the original transmission of a block containing the last octet of the block's red-part. This data segment also indicates that the segment's upper bound is the length of the block's red-part (in octets).

(14) Checkpoint

A data segment soliciting a reception report from the receiving LTP engine. The EORP segment must be flagged as a checkpoint, as must the last segment of any retransmission; these are "mandatory checkpoints". All other checkpoints are "discretionary checkpoints".

(15) Reception Report

A sequence of one or more report segments reporting on all block data reception within some scope.

(16) Synchronous Reception Report

A reception report that is issued in response to a checkpoint.

(17) Asynchronous Reception Report

A reception report that is issued in response to some implementation-defined event other than the arrival of a checkpoint.

(18) Primary Reception Report

A reception report that is issued in response to some event other than the arrival of a checkpoint segment that was itself issued in response to a reception report. Primary reception reports include all asynchronous reception reports and all synchronous reception reports that are sent in response to discretionary checkpoints or to the EORP segment for a session.

(19) Secondary Reception Report

A reception report that is issued in response to the arrival of a checkpoint segment that was itself issued in response to a reception report.

(20) Self-Delimiting Numeric Value (SDNV)

The design of LTP attempts to reconcile minimal consumption of transmission bandwidth with

(a) extensibility to satisfy requirements not yet identified, and

(b) scalability across a very wide range of network sizes and transmission payload sizes.

The SDNV encoding scheme is modeled after the Abstract Syntax Notation One [ASN1] scheme for encoding Object Identifier values. In a data field encoded as an SDNV, the most significant bit (MSB) of each octet of the SDNV serves to indicate whether or not the octet is the last octet of the SDNV. An octet with an MSB of 1 indicates that it is either the first or a middle octet of a multi-octet SDNV; the octet with an MSB of 0 is the last octet of the SDNV. The value encoded in an SDNV is found by concatenating the 7 least significant bits of each octet of the SDNV, beginning at the first octet and ending at the last octet.

The following examples illustrate the encoding scheme for various hexadecimal values.

```
0xABC  : 1010 1011 1100
         is encoded as

         {100 1010 1} {0 011 1100}
          -            -

         = 10010101 00111100

0x1234 : 0001 0010 0011 0100
         =  1 0010 0011 0100
         is encoded as

         {10 1 0010 0} {0 011 0100}
          -             -

         = 10100100 00110100

0x4234 : 0100 0010 0011 0100
         =100 0010 0011 0100
         is encoded as

         {1000000 1} {1 00 0010 0} {0 011 0100}
          -           -             -

         = 10000001 10000100 00110100

0x7F   : 0111 1111
         =111 1111
         is encoded as

         {0 111 1111}
          -

         = 01111111
```

Note:

Care must be taken to make sure that the value to be encoded is padded with zeroes at the most significant bit end (NOT at the least significant bit end) to make its bitwise length a multiple of 7 before encoding.

While there is no theoretical limit on the size of an SDNV field, we note that the overhead of the SDNV scheme is 1:7, i.e., 1 bit of overhead for every 7 bits of actual data to be encoded. Thus, a

7-octet value (a 56-bit quantity with no leading zeroes) would be encoded in an 8-octet SDNV; an 8-octet value (a 64-bit quantity with no leading zeroes) would be encoded in a 10-octet SDNV. In general, an N-bit quantity with no leading zeroes would be encoded in a ceil(N/7) octet SDNV, where ceil is the integer ceiling function. Clearly, for fields that typically carry larger values such as RSA public keys, the SDNV overhead could become unacceptable. Hence, when adopting the SDNV scheme for other purposes related to this document, such as any protocol extensions, we RECOMMEND that if the typical data field value is expected to be larger than 8 octets, then the data field should be specified as a {LENGTH, VALUE} tuple, with the LENGTH parameter encoded as an SDNV followed by LENGTH octets housing the VALUE of the data field.

We also note that SDNV is clearly not the best way to represent every numeric value. When the maximum possible value of a number is known without question, the cost of additional bits may not be justified. For example, an SDNV is a poor way to represent an integer whose value typically falls in the range 128 to 255. In general, though, we believe that the SDNV representation of various protocol data fields in LTP segments yields the smallest segment sizes without sacrificing scalability.

3. Segment Structure

Each LTP segment comprises

(a) a "header" in the format defined below.

(b) zero or more octets of "content".

(c) zero or more octets of "trailer" as indicated by information in the "Extensions field" of the header.

LTP segments are of four general types depending on the nature of the content carried:

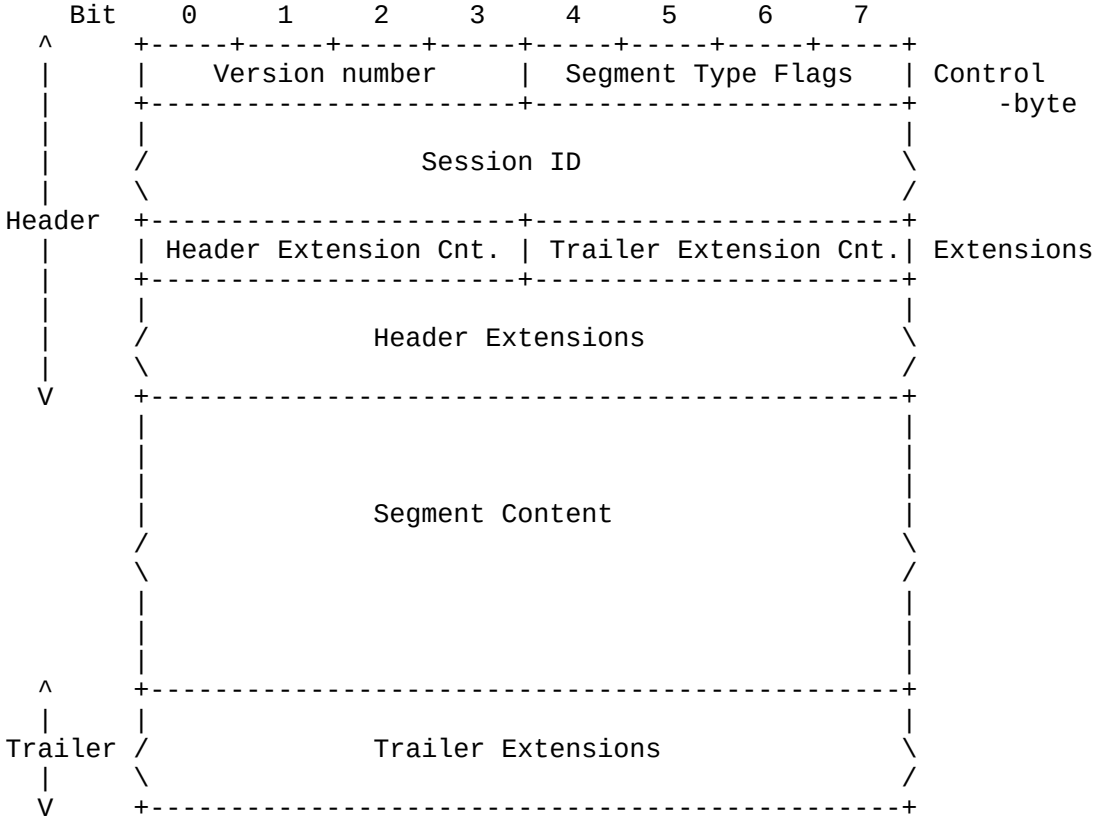
Data segments flow from the sender to the receiver and carry client service (application) data.

A report segment flows from the receiver to the sender and carries data reception claims together with the upper and lower bounds of the block scope to which the claims pertain.

A report-acknowledgment segment flows from the sender to the receiver and acknowledges reception of a report segment. It carries the serial number of the report being acknowledged.

Session management segments may be generated by both the sender and the receiver and are of two general sub-types: cancellation and cancellation-acknowledgment. A cancellation segment initiates session cancellation procedures at the peer and carries a single byte reason-code to indicate the reason for session cancellation. Cancellation-acknowledgment segments merely acknowledge reception of a cancellation segment and have no content.

The overall segment structure is illustrated below:



3.1. Segment Header

An LTP segment header comprises three data items: a single-octet control byte, the session ID, and the Extensions field.

Control byte comprises the following:

Version number (4 bits): MUST be set to the binary value 0000 for this version of the protocol.

Segment type flags (4 bits): described in Section 3.1.1.

Session ID uniquely identifies, among all transmissions between the sender and receiver, the session of which the segment is one token. It comprises the following:

Session originator (SDNV): the engine ID of the sender.

Session number (SDNV): typically a random number (for anti-DoS reasons), generated by the sender.

The format and resolution of session number are matters that are private to the LTP sender; the only requirement imposed by LTP is that every session initiated by an LTP engine MUST be uniquely identified by the session ID.

The Extensions field is described in Section 3.1.4.

3.1.1. Segment Type Flags

The last 4 bits of the control byte in the segment header are flags that indicate the nature of the segment. In order (most significant bit first), these flags are CTRL, EXC, Flag 1, and Flag 0.

A value of 0 in the CTRL (Control) flag identifies the segment as a data segment, while a value of 1 identifies it as a control segment. A data segment with the EXC (Exception) flag set to 0 is a red-part segment; a data segment with EXC set to 1 is a green-part segment. For a control segment, having the EXC flag set to 1 indicates that the segment pertains to session cancellation activity. Any data segment (whether red-part or green-part) with both Flag 1 and Flag 0 set to 1 indicates EOB. Any data segment (whether red-part or green-part) with both Flag 1 and Flag 0 set to 0 indicates data without any additional protocol significance. Any red-part data segment with either flag bit non-zero is a checkpoint. Any red-part data segment with Flag 1 set to 1 indicates the end of red-part.

Put another way:

```
if (CTRL flag = 0)
   segment is a data segment if (EXC flag = 0)
      segment contains only red-part data if (Flag 1 = 1)
         segment is a checkpoint segment is the last segment in the
         red part of the block if (Flag 0 = 1)
            segment is the last segment in the block
      else // segment is not end of red-part
         if (Flag 0 = 1)
            segment is a checkpoint
   else
      segment contains only green-part data if (Flag 1 = 1)
         if (Flag 0 = 1)
            segment is the last segment in the block
else
   segment is a control segment if (EXC flag = 0)
      segment pertains to report activity if (flag 0 = 0)
         segment is a report segment
      else
         segment is an acknowledgment of a report segment
   else
      segment pertains to session cancellation activity if (Flag 1 =
      0)
         segment pertains to cancellation by block sender if (Flag 0
         = 1)
            segment is a cancellation by sender
         else
            segment is an acknowledgment of a cancellation by sender
      else
         segment pertains to cancellation by block receiver if (Flag
         0 = 1)
            segment is a cancellation by receiver
         else
            segment is an acknowledgment of a cancellation by
            receiver
```

3.1.2. Segment Type Codes

Combinations of the settings of the segment type flags CTRL, EXC, Flag 1, and Flag 0 constitute segment type codes, which serve as concise representations of detailed segment nature.

CTRL	EXC	Flag 1	Flag 0	Code	Nature of segment
0	0	0	0	0	Red data, NOT {Checkpoint, EORP or EOB}
0	0	0	1	1	Red data, Checkpoint, NOT {EORP or EOB}
0	0	1	0	2	Red data, Checkpoint, EORP, NOT EOB
0	0	1	1	3	Red data, Checkpoint, EORP, EOB
0	1	0	0	4	Green data, NOT EOB
0	1	0	1	5	Green data, undefined
0	1	1	0	6	Green data, undefined
0	1	1	1	7	Green data, EOB
1	0	0	0	8	Report segment
1	0	0	1	9	Report-acknowledgment segment
1	0	1	0	10	Control segment, undefined
1	0	1	1	11	Control segment, undefined
1	1	0	0	12	Cancel segment from block sender
1	1	0	1	13	Cancel-acknowledgment segment to block sender
1	1	1	0	14	Cancel segment from block receiver
1	1	1	1	15	Cancel-acknowledgment segment to block receiver

3.1.3. Segment Class Masks

For the purposes of this specification, some bit patterns in the segment type flags field correspond to "segment classes" that are designated by mnemonics. The mnemonics are intended to evoke the characteristics shared by all types of segments characterized by these flag bit patterns.

CTRL	EXC	Flag 1	Flag 0	Mnemonic	Description
0	0	-	1		
	-- or --				
0	0	1	-	CP	Checkpoint
0	0	1	-	EORP	End of red-part; red-part size = offset + length
0	-	1	1	EOB	End of block; block size = offset + length
1	0	0	0	RS	Report segment; carries reception claims
1	0	0	1	RA	Report-acknowledgment segment
1	1	0	0	CS	Cancel segment from block sender
1	1	0	1	CAS	Cancel-acknowledgment segment to block sender
1	1	1	0	CR	Cancel segment from block receiver
1	1	1	1	CAR	Cancel-acknowledgment segment to block receiver
1	1	-	0	Cx	Cancel segment (generic)
1	1	-	1	CAx	Cancel-acknowledgment segment (generic)

3.1.4. Extensions Field

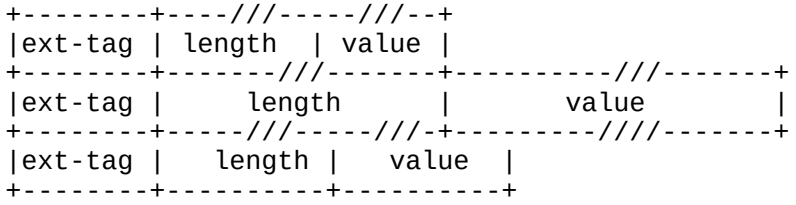
The Extensions field enables the inclusion of zero or more functional extensions to the basic LTP segment, each in type-length-value (TLV) representation as explained below.

The first octet of the Extensions field indicates the number of extensions present in the segment: the high-order 4 bits indicate the number of extension TLVs in the header (immediately following the extensions count octet and preceding the segment's content), while the low-order 4 bits indicate the number of extension TLVs in the trailer (immediately following the segment's content). That is, each segment may have from 0 to 15 extension TLVs in its header and from 0 to 15 extension TLVs in its trailer. In the absence of any extension TLVs, all bits of this extensions count octet MUST be set to zero.

Note that it is valid for header extensions to be immediately followed by trailer extensions; for example, since a CAx segment has no contents, it may have header extensions immediately followed by trailer extensions.

Each extension consists of a one-octet tag identifying the type of the extension, followed by a length parameter in SDNV form, followed by a value of the specified length.

The diagram below illustrates the extension TLVs as they may occur in the header or trailer.



The IANA maintains an LTP Extension Tag registry as shown below. See the IANA considerations section below for details of code point assignment in the Unassigned range.

Extension tag	Meaning
0x00	LTP authentication extension [LTPEXT]
0x01	LTP cookie extension [LTPEXT]
0x02-0xAF	Unassigned
0xB0-0xBF	Reserved
0xC0-0xFF	Private / Experimental Use

Note that since the last quarter of the extension-tag space is for experimental use, implementations should be aware that collisions for these tags are possible.

3.2. Segment Content

3.2.1. Data Segment (DS)

The content of a data segment includes client service data and the metadata enabling the receiving client service instance to receive and make use of that data.

Client service ID (SDNV)

The client service ID number identifies the upper-level service to which the segment is to be delivered by the receiver. It is functionally analogous to a TCP port number. If multiple instances of the client service are present at the destination, multiplexing must be done by the client service itself on the basis of information encoded within the transmitted block.

Offset (SDNV)

Offset indicates the location of the segment's client service data within the session's transmitted block. It is the number of bytes in the block prior to the byte from which the first octet of the segment's client service data was copied.

Length (SDNV)

The length of the ensuing client service data, in octets.

If the data segment is a checkpoint, the segment MUST additionally include the following two serial numbers (checkpoint serial number and report serial number) to support efficient retransmission. Data segments that are not checkpoints MUST NOT have these two fields in the header and MUST continue on directly with the client service data.

Checkpoint serial number (SDNV)

The checkpoint serial number uniquely identifies the checkpoint among all checkpoints issued by the block sender in a session. The first checkpoint issued by the sender MUST have this serial number chosen randomly for security reasons, and it is RECOMMENDED that the sender use the guidelines in [ESC05] for this. Any subsequent checkpoints issued by the sender MUST have the serial number value found by incrementing the prior checkpoint serial number by 1. When a checkpoint segment is retransmitted, however, its serial number MUST be the same as when it was originally transmitted. The checkpoint serial number MUST NOT be zero.

Report serial number (SDNV)

If the checkpoint was queued for transmission in response to the reception of an RS (Section 6.13), then its value MUST be the report serial number value of the RS that caused the data segment to be queued for transmission.

Otherwise, the value of report serial number MUST be zero.

Client service data (array of octets)

The client service data carried in the segment is a copy of a subset of the bytes in the original client service data block, starting at the indicated offset.

3.2.2. Report Segment (RS)

The content of an RS comprises one or more data reception claims, together with the upper and lower bounds of the scope within the data block to which the claims pertain. It also includes two serial numbers to support efficient retransmission.

Report serial number (SDNV)

The report serial number uniquely identifies the report among all reports issued by the receiver in a session. The first report issued by the receiver MUST have this serial number chosen randomly for security reasons, and it is RECOMMENDED that the receiver use the guidelines in [ESC05] for this. Any subsequent RS issued by the receiver MUST have the serial number value found by incrementing the last report serial number by 1. When an RS is retransmitted however, its serial number MUST be the same as when it was originally transmitted. The report serial number MUST NOT be zero.

Checkpoint serial number (SDNV)

The value of the checkpoint serial number MUST be zero if the report segment is NOT a response to reception of a checkpoint, i.e., the reception report is asynchronous; otherwise, it MUST be the checkpoint serial number of the checkpoint that caused the RS to be issued.

Upper bound (SDNV)

The upper bound of a report segment is the size of the block prefix to which the segment’s reception claims pertain.

Lower bound (SDNV)

The lower bound of a report segment is the size of the (interior) block prefix to which the segment's reception claims do NOT pertain.

Reception claim count (SDNV)

The number of data reception claims in this report segment.

Reception claims

Each reception claim comprises two elements: offset and length.

Offset (SDNV)

The offset indicates the successful reception of data beginning at the indicated offset from the lower bound of the RS. The offset within the entire block can be calculated by summing this offset with the lower bound of the RS.

Length (SDNV)

The length of a reception claim indicates the number of contiguous octets of block data starting at the indicated offset that have been successfully received.

Reception claims MUST conform to the following rules:

A reception claim's length shall never be less than 1 and shall never exceed the difference between the upper and lower bounds of the report segment.

The offset of a reception claim shall always be greater than the sum of the offset and length of the prior claim, if any.

The sum of a reception claim's offset and length and the lower bound of the report segment shall never exceed the upper bound of the report segment.

Implied requests for retransmission of client service data can be inferred from an RS's data reception claims. However, *nothing* can be inferred regarding reception of block data at any offset equal to or greater than the segment's upper bound or at any offset less than the segment's lower bound.

For example, if the scope of a report segment has lower bound 0 and upper bound 6000, and the report contains a single data reception claim with offset 0 and length 6000, then the report signifies successful reception of the first 6000 bytes of the block. If the total length of the block is 6000, then the report additionally signifies successful reception of the entire block.

If on the other hand, the scope of a report segment has lower bound 1000 and upper bound 6000, and the report contains two data reception claims, one with offset 0 and length 2000 and the other with offset 3000 and length 500, then the report signifies successful reception only of bytes 1000-2999 and 4000-4499 of the block. From this we can infer that bytes 3000-3999 and 4500-5999 of the block need to be retransmitted, but we cannot infer anything about reception of the first 1000 bytes or of any subsequent data beginning at block offset 6000.

3.2.3. Report Acknowledgment Segment

The content of an RA is simply the report serial number of the RS in response to which the segment was generated.

Report serial number (SDNV)

This field returns the report serial number of the RS being acknowledged.

3.2.4. Session Management Segments

Cancel segments (Cx) carry a single byte reason-code with the following semantics:

Reason-Code	Mnemonic	Semantics
00	USR_CNCLD	Client service canceled session.
01	UNREACH	Unreachable client service.
02	RLEXC	Retransmission limit exceeded.
03	MISCOLORED	Received either a red-part data segment at block offset above any green-part data segment offset or a green-part data segment at block offset below any red-part data segment offset.
04	SYS_CNCLD	A system error condition caused unexpected session termination.
05	RXMTCYCEXC	Exceeded the Retransmission-Cycles limit.
06-FF	Reserved	

The Cancel-acknowledgments (CAx) have no content.

Note: The reason we use different cancel segment types for the originator and recipient is to allow a loopback mode to work without disturbing any replay protection mechanism in use.

3.3. Segment Trailer

The segment trailer consists of a sequence of zero to 15 extension TLVs as described in Section 3.1.4 above.

4. Requests from Client Service

In all cases, the representation of request parameters is a local implementation matter, as are validation of parameter values and notification of the client service in the event that a request is found to be invalid.

4.1. Transmission Request

In order to request transmission of a block of client service data, the client service MUST provide the following parameters to LTP:

- Destination client service ID.
- Destination LTP engine ID.
- Client service data to send, as an array of bytes.
- Length of the data to be sent.
- Length of the red-part of the data. This value MUST be in the range from zero to the total length of data to be sent.

On reception of a valid transmission request from a client service, LTP proceeds as follows.

First, the array of data to be sent is subdivided as necessary, with each subdivision serving as the client service data of a single new LTP data segment. The algorithm used for subdividing the data is a local implementation matter; it is expected that data size constraints imposed by the underlying communication service, if any, will be accommodated in this algorithm.

The last (and only the last) of the resulting data segments must be marked as the EOB (end of block).

Note that segment type indicates that the client service data in a given LTP segment either is or is not in the red-part of the block. To prevent segment type ambiguity, each data segment MUST contain either only red-part data or only green-part data. Therefore, when the length of the block's red-part is N, the total length of the block is M, and N is not equal to M, the (N+1)th byte of the block SHOULD be the first byte of client service data in a green-part data segment. Note that this means that at the red-part boundary, LTP may send a segment of size lesser than the link MTU size. For bandwidth efficiency reasons, implementations MAY choose to instead mark the entire segment (within which the red-part boundary falls) as red-part, causing green-part data falling within the segment to also be treated as red-part.

If the length of the block's red-part is greater than zero, then the last data segment containing red-part data must be marked as the EORP (end of red-part) segment by setting the appropriate segment type flag bits (Section 3.1.2). Zero or more preceding data segments containing red-part data (selected according to an algorithm that is

a local implementation matter) MAY additionally be marked as a CP (Checkpoint), and serve as additional discretionary checkpoints (Section 3.1.2).

All data segments are appended to the (conceptual) application data queue bound for the destination engine, for subsequent transmission.

Finally, a session start notice (Section 7.1) is sent back to the client service that requested the transmission.

4.2. Cancellation Request

In order to request cancellation of a session, either as the sender or as the receiver of the associated data block, the client service must provide the session ID identifying the session to be canceled.

On reception of a valid cancellation request from a client service, LTP proceeds as follows.

First, the internal "Cancel Session" procedure (Section 6.19) is invoked.

Next, if the session is being canceled by the sender (i.e., the session originator part of the session ID supplied in the cancellation request is the local LTP engine ID):

- If none of the data segments previously queued for transmission as part of this session have yet been de-queued and transmitted -- i.e., if the destination engine cannot possibly be aware of this session -- then the session is simply closed; the "Close Session" procedure (Section 6.20) is invoked.

- Otherwise, a CS (cancel by block sender) segment with the reason-code USR_CNCLD MUST be queued for transmission to the destination LTP engine specified in the transmission request that started this session.

Otherwise (i.e., the session is being canceled by the receiver):

- If there is no transmission queue-set bound for the sender (possibly because the local LTP engine is running on a receive-only device), then the session is simply closed; the "Close Session" procedure (Section 6.20) is invoked.

- Otherwise, a CR (cancel by block receiver) segment with reason-code USR_CNCLD MUST be queued for transmission to the block sender.

5. Requirements from the Operating Environment

LTP is designed to run directly over a data-link layer protocol.

LTP MUST only be deployed directly over UDP, for software development purposes or for use in private local area networks, for example, in a sparse sensor network where the link, when available, is only used for LTP traffic.

In either case, the protocol layer immediately underlying LTP is referred to as the "local data-link layer" for the purposes of this specification.

When the local data-link layer protocol is UDP, (a) the content of each UDP datagram MUST be an integral number of LTP segments and (b) the LTP authentication [LTPEXT] extension SHOULD be used unless the end-to-end path is one in which either the likelihood of datagram content corruption is negligible or the consequences of receiving and processing corrupt LTP segments are insignificant (as during software development). In addition, the LTP authentication [LTPEXT] extension SHOULD be used to ensure data integrity unless the end-to-end path is one in which either the likelihood of datagram content corruption is negligible (as in some private local area networks) or the consequences of receiving and processing corrupt LTP segments are insignificant (as perhaps during software development).

When the local data-link layer protocol is not UDP, the content of each local data-link layer protocol frame MUST be an integral number of LTP segments.

The local data-link layer protocol MUST be a protocol that, together with the operating environment in which that protocol is implemented, satisfies the following requirements:

- It is required to inform LTP whenever the link to a specific LTP destination is brought up or torn down. Similarly, it is required to inform the local LTP engine whenever it is known that a remote LTP engine is set to begin or stop communication with the local engine based on the engines' operating schedules.

- It is required to provide link state cues to LTP upon transmission of the CP, RS (report), EORP, EOB, and Cx (cancel) segments so that timers can be started.

- It is required to provide, upon request, the current distance (in light seconds) to any peer engine in order to calculate timeout intervals.

A MIB (Management Information Base) with the above parameters, updated periodically by the local data-link layer and the operating environment, should be made available to the LTP engine for its operations. The details of the MIB are, however, beyond the scope of this document.

The underlying data-link layer is required to never deliver incompletely received LTP segments to LTP. In the absence of the use of LTP authentication [LTPEXT], LTP also requires the underlying local data-link layer protocol to perform data integrity checking of the segments received. Specifically, the local data-link layer protocol is required to detect any corrupted segments received and to silently discard them.

6. Internal Procedures

This section describes the internal procedures that are triggered by the occurrence of various events during the lifetime of an LTP session.

Whenever the content of any of the fields of the header of any received LTP segment does not conform to this specification document, the segment is assumed to be corrupt and MUST be discarded immediately and processed no further. This procedure supersedes all other procedures described below.

All internal procedures described below that are triggered by the arrival of a data segment are superseded by the following procedure in the event that the client service identified by the data segment does not exist at the local LTP engine:

- If there is no transmission queue-set bound for the block sender (possibly because the local LTP engine is running on a receive-only device), then the received data segment is simply discarded.

- Otherwise, if the data segment contains data from the red-part of the block, a CR with reason-code UNREACH MUST be enqueued for transmission to the block sender. A CR with reason-code UNREACH SHOULD be similarly enqueued for transmission to the data sender even if the data segment contained data from the green-part of the block; note however that (for example) in the case where the block receiver knows that the sender of this green-part data is functioning in a "beacon" (transmit-only) fashion, a CR need not be sent. In either case, the received data segment is discarded.

6.1. Start Transmission

This procedure is triggered by the arrival of a link state cue indicating the start of transmission to a specified remote LTP engine.

Response: the de-queuing and delivery of segments to the LTP engine specified in the link state cue begins.

6.2. Start Checkpoint Timer

This procedure is triggered by the arrival of a link state cue indicating the de-queuing (for transmission) of a CP segment.

Response: the expected arrival time of the RS segment that will be produced on reception of this CP segment is computed, and a countdown timer is started for this arrival time. However, if it is known that the remote LTP engine has ceased transmission (Section 6.5), then this timer is immediately suspended, because the computed expected arrival time may require an adjustment that cannot yet be computed.

6.3. Start RS Timer

This procedure is triggered by the arrival of a link state cue indicating the de-queuing (for transmission) of an RS segment.

Response: the expected arrival time of the RA (report acknowledgment) segment in response to the reception of this RS segment is computed, and a countdown timer is started for this arrival time. However, as in Section 6.2, if it is known that the remote LTP engine has ceased transmission (Section 6.5), then this timer is immediately suspended, because the computed expected arrival time may require an adjustment that cannot yet be computed.

6.4. Stop Transmission

This procedure is triggered by the arrival of a link state cue indicating the cessation of transmission to a specified remote LTP engine.

Response: the de-queuing and delivery to the underlying communication system of segments from traffic queues bound for the LTP engine specified in the link state cue ceases.

6.5. Suspend Timers

This procedure is triggered by the arrival of a link state cue indicating the cessation of transmission from a specified remote LTP engine to the local LTP engine. Normally, this event is inferred from advance knowledge of the remote engine's planned transmission schedule.

Response: countdown timers for the acknowledging segments that the remote engine is expected to return are suspended as necessary based on the following procedure.

The nominal remote engine acknowledge transmission time is computed as the sum of the transmission time of the original segment (to which the acknowledging segment will respond) and the one-way light time to the remote engine, plus N seconds of "additional anticipated latency" (AAL) encompassing anticipated transmission delays other than signal propagation time. N is determined in an implementation-specific manner. For example, when LTP is deployed in deep-space vehicles, the one-way light time to the remote engine may be very large while N may be relatively small, covering processing and queuing delays. N may be a network management parameter, for which 2 seconds seems like a reasonable default value. As another example, when LTP is deployed in a terrestrial "data mule" environment, one-way light time latency is effectively zero while N may need to be some dynamically computed function of the data mule circulation schedule.

If the nominal remote engine acknowledge transmission time is greater than or equal to the current time (i.e., the acknowledging segment may be presented for transmission during the time that transmission at the remote engine is suspended), then the countdown timer for this acknowledging segment is suspended.

6.6. Resume Timers

This procedure is triggered by the arrival of a link state cue indicating the start of transmission from a specified remote LTP engine to the local LTP engine. Normally, this event is inferred from advance knowledge of the remote engine's planned transmission schedule.

Response: expected arrival time is adjusted for every acknowledging segment that the remote engine is expected to return, for which the countdown timer has been suspended. First, the transmission delay interval is calculated as follows:

- The nominal remote engine acknowledge transmission time is computed as the sum of the transmission time of the original segment (to which the acknowledging segment will respond) and the one-way light time to the remote engine, plus N seconds of AAL Section 6.5.

- If the nominal remote engine acknowledge transmission time is greater than the current time, i.e., the remote engine resumed transmission prior to presentation of the acknowledging segment for transmission, then the transmission delay interval is zero.

- Otherwise, the transmission delay interval is computed as the current time less the nominal remote engine acknowledge transmission time.

The expected arrival time is increased by the computed transmission delay interval for each of the suspended countdown timers, and the timers are resumed.

6.7. Retransmit Checkpoint

This procedure is triggered by the expiration of a countdown timer associated with a CP segment.

Response: if the number of times this CP segment has been queued for transmission exceeds the checkpoint retransmission limit established for the local LTP engine by network management, then the session of which the segment is one token is canceled: the "Cancel Session" procedure (Section 6.19) is invoked, a CS with reason-code RLEXC is appended to the (conceptual) application data queue, and a transmission-session cancellation notice (Section 7.5) is sent back to the client service that requested the transmission.

Otherwise, a new copy of the CP segment is appended to the (conceptual) application data queue for the destination LTP engine.

6.8. Retransmit RS

This procedure is triggered by either (a) the expiration of a countdown timer associated with an RS segment or (b) the reception of a CP segment for which one or more RS segments were previously issued -- a redundantly retransmitted checkpoint.

Response: if the number of times any affected RS segment has been queued for transmission exceeds the report retransmission limit established for the local LTP engine by network management, then the session of which the segment is one token is canceled: the "Cancel Session" procedure (Section 6.19) is invoked, a CR segment with

reason-code RLEXC is queued for transmission to the LTP engine that originated the session, and a reception-session cancellation notice (Section 7.6) is sent to the client service identified in each of the data segments received in this session.

Otherwise, a new copy of each affected RS segment is queued for transmission to the LTP engine that originated the session.

6.9. Signify Red-Part Reception

This procedure is triggered by the arrival of a CP segment when the EORP for this session has been received (ensuring that the size of the data block's red-part is known; this includes the case where the CP segment itself is the EORP segment) and all data in the red-part of the block being transmitted in this session have been received.

Response: a red-part reception notice (Section 7.3) is sent to the specified client service.

6.10. Signify Green-Part Segment Arrival

This procedure is triggered by the arrival of a data segment whose content is a portion of the green-part of a block.

Response: a green-part segment arrival notice (Section 7.2) is sent to the specified client service.

6.11. Send Reception Report

This procedure is triggered by either (a) the original reception of a CP segment (the checkpoint serial number identifying this CP is new) (b) an implementation-specific circumstance pertaining to a particular block reception session for which no EORP has yet been received ("asynchronous" reception reporting).

Response: if the number of reception problems detected for this session exceeds a limit established for the local LTP engine by network management, then the affected session is canceled: the "Cancel Session" procedure (Section 6.19) is invoked, a CR segment with reason-code RLEXC is issued and is, in concept, appended to the queue of internal operations traffic bound for the LTP engine that originated the session, and a reception-session cancellation notice (Section 7.6) is sent to the client service identified in each of the data segments received in this session. One possible limit on reception problems would be the maximum number of reception reports that can be issued for any single session.

If such a limit is not reached, a reception report is issued as follows.

If production of the reception report was triggered by reception of a checkpoint:

- The upper bound of the report SHOULD be the upper bound (the sum of the offset and length) of the checkpoint data segment, to minimize unnecessary retransmission. Note: If a discretionary checkpoint is lost but subsequent segments are received, then by the time the retransmission of the lost checkpoint is received the receiver would have segments at block offsets beyond the upper bound of the checkpoint. For deployments where bandwidth economy is not critical, the upper bound of a synchronous reception report MAY be the maximum upper bound value among all red-part data segments received so far in the affected session.

- If the checkpoint was itself issued in response to a report segment, then this report is a "secondary" reception report. In that case, the lower bound of the report SHOULD be the lower bound of the report segment to which the triggering checkpoint was itself a response, to minimize unnecessary retransmission. Note: For deployments where bandwidth economy is not critical, the lower bound of the report MAY instead be zero.

- If the checkpoint was not issued in response to a report segment, this report is a "primary" reception report. The lower bound of the first primary reception report issued for any session MUST be zero. The lower bound of each subsequent primary reception report issued for the same session SHOULD be the upper bound of the prior primary reception report issued for the session, to minimize unnecessary retransmission. Note: For deployments where bandwidth economy is not critical, the lower bound of every primary reception report MAY be zero.

If production of the reception report is "asynchronous" as noted above:

- The upper bound of the report MUST be the maximum upper bound among all red-part data segments received so far for this session.

- The lower bound of the first asynchronous reception report issued for any session for which no other primary reception reports have yet been issued MUST be zero. The lower bound of each subsequent asynchronous reception report SHOULD be the upper bound of the prior primary reception report issued for the

session, to minimize unnecessary retransmission. Note: For deployments where bandwidth economy is not critical, the lower bound of every asynchronous reception report MAY be zero.

In all cases, if the applicable lower bound of the scope of a report is determined to be greater than or equal to the applicable upper bound (for example, due to out-of-order arrival of discretionary checkpoints) then the reception report MUST NOT be issued. Otherwise:

As many RS segments must be produced as are needed in order to report on all data reception within the scope of the report, given whatever data size constraints are imposed by the underlying communication service. The RS segments are, in concept, appended to the queue of internal operations traffic bound for the LTP engine that originated the indicated session. The lower bound of the first RS segment of the report MUST be the reception report's lower bound. The upper bound of the last RS segment of the report MUST be the reception report's upper bound.

6.12. Signify Transmission Completion

This procedure is triggered at the earliest time at which (a) all data in the block are known to have been transmitted *and* (b) the entire red-part of the block -- if of non-zero length -- is known to have been successfully received. Condition (a) is signaled by arrival of a link state cue indicating the de-queuing (for transmission) of the EOB segment for the block. Condition (b) is signaled by reception of an RS segment whose reception claims, taken together with the reception claims of all other RS segments previously received in the course of this session, indicate complete reception of the red-part of the block.

Response: a transmission-session completion notice (Section 7.4) is sent to the local client service associated with the session, and the session is closed: the "Close Session" procedure (Section 6.20) is invoked.

6.13. Retransmit Data

This procedure is triggered by the reception of an RS segment.

Response: first, an RA segment with the same report serial number as the RS segment is issued and is, in concept, appended to the queue of internal operations traffic bound for the receiver. If the RS segment is redundant -- i.e., either the indicated session is unknown (for example, the RS segment is received after the session has been completed or canceled) or the RS segment's report serial number

matches that of an RS segment that has already been received and processed -- then no further action is taken. Otherwise, the procedure below is followed.

If the report's checkpoint serial number is not zero, then the countdown timer associated with the indicated checkpoint segment is deleted.

Note: All retransmission buffer space occupied by data whose reception is claimed in the report segment can (in concept) be released.

If the segment's reception claims indicate incomplete data reception within the scope of the report segment:

- If the number of transmission problems for this session exceeds a limit established for the local LTP engine by network management, then the session of which the segment is one token is canceled: the "Cancel Session" procedure (Section 6.19) is invoked, a CS with reason-code RLEXC is appended to the transmission queue specified in the transmission request that started this session, and a transmission-session cancellation notice (Section 7.5) is sent back to the client service that requested the transmission. One possible limit on transmission problems would be the maximum number of retransmission CP segments that may be issued for any single session.

- If the number of transmission problems for this session has not exceeded any limit, new data segments encapsulating all block data whose non-reception is implied by the reception claims are appended to the transmission queue bound for the receiver. The last -- and only the last -- data segment must be marked as a CP segment carrying a new CP serial number (obtained by incrementing the last CP serial number used) and the report serial number of the received RS segment.

6.14. Stop RS Timer

This procedure is triggered by the reception of an RA.

Response: the countdown timer associated with the original RS segment (identified by the report serial number of the RA segment) is deleted. If no other countdown timers associated with RS segments exist for this session, then the session is closed: the "Close Session" procedure (Section 6.20) is invoked.

6.15. Start Cancel Timer

This procedure is triggered by arrival of a link state cue indicating the de-queuing (for transmission) of a Cx segment.

Response: the expected arrival time of the CAx segment that will be produced on reception of this Cx segment is computed and a countdown timer for this arrival time is started. However, if it is known that the remote LTP engine has ceased transmission (Section 6.5), then this timer is immediately suspended, because the computed expected arrival time may require an adjustment that cannot yet be computed.

6.16. Retransmit Cancellation Segment

This procedure is triggered by the expiration of a countdown timer associated with a Cx segment.

Response: if the number of times this Cx segment has been queued for transmission exceeds the cancellation retransmission limit established for the local LTP engine by network management, then the session of which the segment is one token is simply closed: the "Close Session" procedure (Section 6.20) is invoked.

Otherwise, a copy of the cancellation segment (retaining the same reason-code) is queued for transmission to the appropriate LTP engine.

6.17. Acknowledge Cancellation

This procedure is triggered by the reception of a Cx segment.

Response: in the case of a CS segment where there is no transmission queue-set bound for the sender (possibly because the receiver is a receive-only device), then no action is taken. Otherwise:

- If the received segment is a CS segment, a CAS (cancel acknowledgment to block sender) segment is issued and is, in concept, appended to the queue of internal operations traffic bound for the sender.

- If the received segment is a CR segment, a CAR (cancel acknowledgment to block receiver) segment is issued and is, in concept, appended to the queue of internal operations traffic bound for the receiver.

It is possible that the Cx segment has been retransmitted because a previous responding acknowledgment CAx (cancel acknowledgment) segment was lost, in which case there will no longer be any record of the session of which the segment is one token. If so, no further action is taken.

Otherwise: the "Cancel Session" procedure (Section 6.19) is invoked and a reception-session cancellation notice (Section 7.6) is sent to the client service identified in each of the data segments received in this session. Finally, the session is closed: the "Close Session" procedure (Section 6.20) is invoked.

6.18. Stop Cancel Timer

This procedure is triggered by the reception of a CAx segment.

Response: the timer associated with the Cx segment is deleted, and the session of which the segment is one token is closed, i.e., the "Close Session" procedure (Section 6.20) is invoked.

6.19. Cancel Session

This procedure is triggered internally by one of the other procedures described above.

Response: all segments of the affected session that are currently queued for transmission can be deleted from the outbound traffic queues. All countdown timers currently associated with the session are deleted. Note: If the local LTP engine is the sender, then all remaining data retransmission buffer space allocated to the session can be released.

6.20. Close Session

This procedure is triggered internally by one of the other procedures described above.

Response: any remaining countdown timers associated with the session are deleted. The session state record (SSR|RSR) for the session is deleted; existence of the session is no longer recognized.

6.21. Handle Miscolored Segment

This procedure is triggered by the arrival of either (a) a red-part data segment whose block offset begins at an offset higher than the block offset of any green-part data segment previously received for the same session or (b) a green-part data segment whose block offset is lower than the block offset of any red-part data segment

previously received for the same session. The arrival of a segment matching either of the above checks is a violation of the protocol requirement of having all red-part data as the block prefix and all green-part data as the block suffix.

Response: the received data segment is simply discarded.

The Cancel Session procedure (Section 6.19) is invoked and a CR segment with reason-code MISCOLORED SHOULD be enqueued for transmission to the data sender.

Note: If there is no transmission queue-set bound for the sender (possibly because the local LTP engine is running on a receive-only device), or if the receiver knows that the sender is functioning in a "beacon" (transmit-only) fashion, a CR segment need not be sent.

A reception-session cancellation notice (Section 7.6) is sent to the client service.

6.22. Handling System Error Conditions

It is possible (especially for long-lived LTP sessions) that an unexpected operating system error condition may occur during the lifetime of an LTP session. An example is the case where the system faces severe memory crunch forcing LTP sessions into a scenario similar to that of TCP SACK [SACK] reneging. But unlike TCP SACK reception reports, which are advisory, LTP reception reports are binding, and reneging is NOT permitted on previously made reception claims.

Under any such irrecoverable system error condition, the following response is to be initiated: the Cancel Session procedure (Section 6.19) is invoked. If the error condition is observed on the sender, a CS segment with reason-code SYS_CNCLD SHOULD be enqueued for transmission to the receiver, and a transmission-session cancellation notice (Section 7.5) is sent to the client service; on the other hand, if it is observed on the receiver, a CR segment with the same reason-code SYS_CNCLD SHOULD be enqueued for transmission to the sender, and a reception-session cancellation notice (Section 7.6) is sent to the client service.

Note that as in (Section 6.21), if there is no transmission queue-set bound for the sender (possibly because the local LTP engine is running on a receive-only device), or if the receiver knows that the sender of this green-part data is functioning in a "beacon" (transmit-only) fashion, a CR segment need not be sent.

There may be other implementation-specific limits that may cause an LTP implementation to initiate session-cancellation procedures. One such limit is the maximum number of retransmission-cycles seen. A retransmission cycle at the LTP Sender comprises the two related events: the transmission of all outstanding CP segments from the sender, and the reception of all RS segments issued from the receiver in response to those CP segments. A similar definition would apply at the LTP Receiver but relate to the reception of the CP segments and transmission of all RS segments in response. Note that the retransmitted CP and RS segments remain part of their original retransmission-cycle. Also, a single CP segment may cause multiple RS segments to be generated if a reception report would not fit in a single data link-MTU-sized RS segment; all RS segments that are part of a reception report belong to the same retransmission cycle to which the CP segment belongs. In the presence of severe channel error conditions, many retransmission cycles may elapse before red-part transmission is deemed successful; an implementation may therefore impose a retransmission-cycle limit to shield itself from a resource-crunch situation. If an LTP sender notices the retransmission-cycle limit being exceeded, it SHOULD initiate the Cancel Session procedure (Section 6.19), queuing a CS segment with reason-code RXMTCYCEXC and sending a transmission-session cancellation notice (Section 7.5) to the client service.

7. Notices to Client Service

In all cases, the representation of notice parameters is a local implementation matter.

7.1. Session Start

The Session Start notice returns the session ID identifying a newly created session.

At the sender, the session start notice informs the client service of the initiation of the transmission session. On receiving this notice the client service may, for example, release resources of its own that are allocated to the block being transmitted, or remember the session ID so that the session can be canceled in the future if necessary. At the receiver, this notice indicates the beginning of a new reception session, and is delivered upon arrival of the first data segment carrying a new session ID.

7.2. Green-Part Segment Arrival

The following parameters are provided by the LTP engine when a green-part segment arrival notice is delivered:

Session ID of the transmission session.

Array of client service data bytes contained in the data segment.

Offset of the data segment's content from the start of the block.

Length of the data segment's content.

Indication as to whether or not the last byte of this data segment's content is also the end of the block.

Source LTP engine ID.

7.3. Red-Part Reception

The following parameters are provided by the LTP engine when a red-part reception notice is delivered:

Session ID of the transmission session.

Array of client service data bytes that constitute the red-part of the block.

Length of the red-part of the block.

Indication as to whether or not the last byte of the red-part is also the end of the block.

Source LTP engine ID.

7.4. Transmission-Session Completion

The sole parameter provided by the LTP engine when a transmission-session completion notice is delivered is the session ID of the transmission session.

A transmission-session completion notice informs the client service that all bytes of the indicated data block have been transmitted and that the receiver has received the red-part of the block.

7.5. Transmission-Session Cancellation

The parameters provided by the LTP engine when a transmission-session cancellation notice is delivered are:

Session ID of the transmission session.

The reason-code sent or received in the Cx segment that initiated the cancellation sequence.

A transmission-session cancellation notice informs the client service that the indicated session was terminated, either by the receiver or else due to an error or a resource quench condition in the local LTP engine. There is no assurance that the destination client service instance received any portion of the data block.

7.6. Reception-Session Cancellation

The parameters provided by the LTP engine when a reception cancellation notice is delivered are:

Session ID of the transmission session.

The reason-code explaining the cancellation.

A reception-session cancellation notice informs the client service that the indicated session was terminated, either by the sender or else due to an error or a resource quench condition in the local LTP engine. No subsequent delivery notices will be issued for this session.

7.7. Initial-Transmission Completion

The session ID of the transmission session is included with the initial-transmission completion notice.

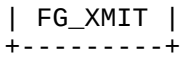
This notice informs the client service that all segments of a block (both red-part and green-part) have been transmitted. This notice only indicates that original transmission is complete; retransmission of any lost red-part data segments may still be necessary.

8. State Transition Diagrams

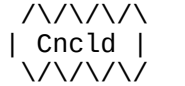
The following mnemonics have been used in the sender and LTP receiver state transition diagrams that follow:

TE	Timer Expiry
RDS	Regular Red Data Segment (NOT {CP|EORP|EOB})
GDS	Regular Green Data Segment (NOT EOB)
RL EXC	Retransmission Limit Exceeded
RP	Red-Part
GP	Green-Part
FG	Fully-Green

Note that blocks represented in rectangles, as in



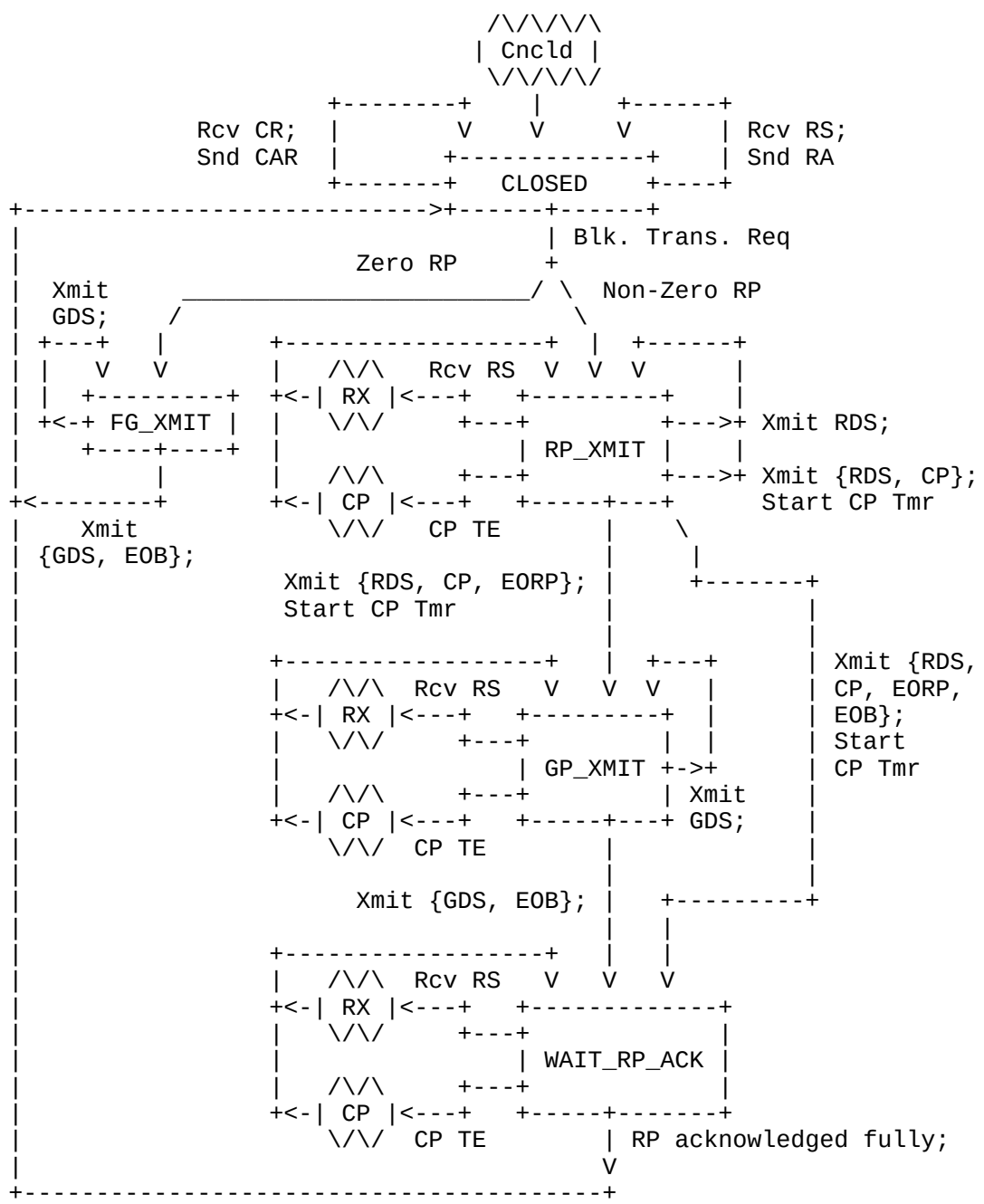
specify actual states in the state-transition diagrams, while blocks represented with jagged edges, as in



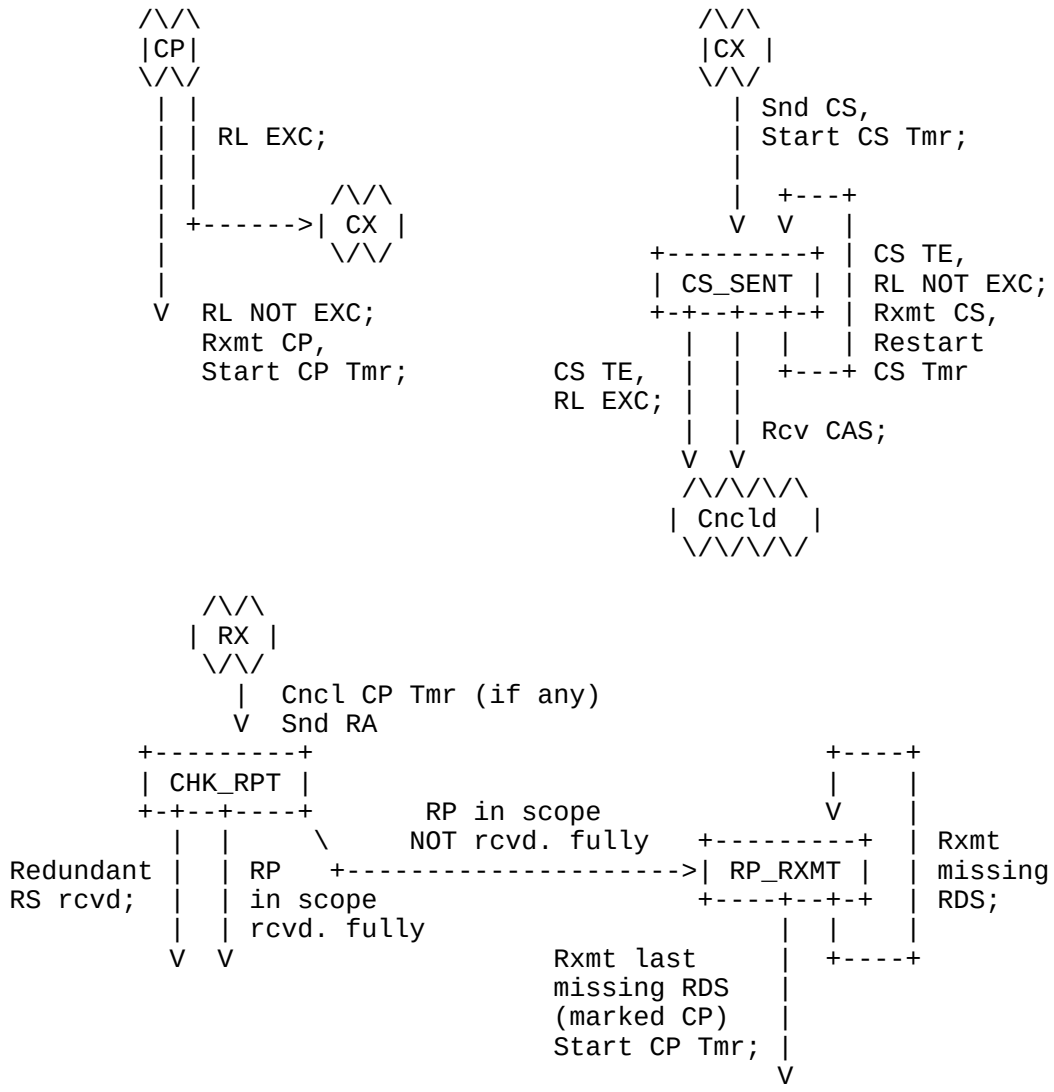
are either pointers to a state or place-holders for sequences of state transitions.

8.1. Sender

LTP Sender State Transition Diagram



LTP Sender State Transition Diagram (contd.)



Asynchronous cancel request may be received from the local client service while the LTP sender is in any of the states shown. If it was not already in the sequence of state transitions beginning at the CX marker, the internal procedure Cancel Session (Section 6.19) is followed, and the LTP sender moves from its current state into the sequence beginning at the CX marker initiating session cancellation with reason-code USR_CNCLD. From the CX marker, the CS segment with appropriate reason-code (USR_CNCLD or RLEXC depending on how the CX

sequence was entered) is queued for transmission to the LTP receiver and the sender enters the Cancel-from-Sender Sent (CS_SENT) state. The internal procedure Start Cancel Timer (Section 6.15) is started upon receiving a link state cue indicating the beginning of transmission of the CS segment. Upon receiving the acknowledging CAS segment from the receiver, the LTP sender moves to the CLOSED state (via the 'Cncld' pointer). If the CS timer expires, the internal procedure Retransmit Cancellation Segment (Section 6.16) is followed:

- If the network management set retransmission limit is exceeded, the session is simply closed and the LTP sender follows the Cncld marker to the CLOSED state. If the retransmission limit is not exceeded however, the CS segment is queued for a retransmission and the LTP sender stays in the CS_SENT state. The CS timer is started upon receiving a link state cue indicating the beginning of actual transmission according to the internal procedure Start Cancel Timer (Section 6.15).

Asynchronous cancel request may also be received from the receiver LTP in the form of a CR segment when the LTP sender is in any of the states. Upon receiving such a CR segment, the internal procedure Acknowledge Cancellation (Section 6.17) is invoked: The LTP sender sends a CAR segment in response and returns to the CLOSED state.

The LTP sender stays in the CLOSED state until receiving a Block Transmission Request (Blk. Trans. Req) from the client service instance. Upon receiving the request, it moves to either the Fully Green Transmission State (FG_XMIT) if no portion of the block was requested to be transmitted as red or to the Red-Part Transmission State (RP_XMIT) state if a non-zero block-prefix was requested to be transmitted red.

In the FG_XMIT state, the block is segmented as multiple green LTP data segments respecting the link MTU size and the segments are queued for transmission to the remote engine. The last such segment is marked as EOB, and the LTP sender returns to the CLOSED state after queuing it for transmission.

Similarly, from the RP_XMIT state, multiple red data segments are queued for transmission, respecting the link MTU size. The sender LTP may optionally mark some of the red data segments as asynchronous checkpoints; the internal procedure Start Checkpoint Timer (Section 6.2) is followed upon receiving a link state cue indicating the transmission of the asynchronous checkpoints. If the block transmission request comprises a non-zero green part, the LTP sender marks the last red data segment as CP and EORP, and after queuing it for transmission, moves to the Green Part Transmission (GP_XMIT) state. If the block transmission request was fully red however, the

last red data segment is marked as CP, EORP, and EOB and the sender LTP moves directly to the Wait-for-Red-Part-Acknowledgment (WAIT_RP_ACK) state. In both of the above state-transitions, the internal procedure Start Checkpoint Timer (Section 6.2) is followed upon receiving a link state cue indicating the beginning of transmission of the queued CP segments. In the GP_XMIT state, the green-part of the block is segmented as green data segments and queued for transmission to the LTP receiver; the last green segment of the block is additionally marked as EOB, and after queueing it for transmission the LTP sender moves to the WAIT_RP_ACK state.

While the LTP sender is at any of the RP_XMIT, GP_XMIT, or WAIT_RP_ACK states, it might be interrupted by the occurrence of the following events:

1. An RS might be received from the LTP receiver (either in response to a previously transmitted CP segment or sent asynchronously for accelerated retransmission). The LTP sender then moves to perform the sequence of state transitions beginning at the RX marker (second part of the diagram), and retransmits data if necessary, illustrating the internal procedure Retransmit Data (Section 6.13):

   First, if the RS segment had a non-zero CP serial number, the corresponding CP timer is canceled. Then an RA segment acknowledging the received RS segment is queued for transmission to the LTP receiver and the LTP sender moves to the Check Report state (CHK_RPT). If the RS segment was redundantly transmitted by the LTP receiver (possibly because either the last transmitted RA segment got lost or the RS segment timer expired prematurely at the receiver), the LTP sender does nothing more and returns back to the interrupted state. Similarly, if all red data within the scope of the RS segment is reported as received, there is no work to be done and the LTP sender returns to the interrupted state. However, if the RS segment indicated incomplete reception of data within its scope, the LTP sender moves to the Red-Part Retransmit state (RP_RXMT) where missing red data segments within scope are queued for transmission. The last such segment is marked as a CP, and the LTP sender returns to the interrupted state. The internal procedure (Section 6.2) is followed upon receiving a link state cue indicating the beginning of transmission of the CP segment.

2. A previously set CP timer might expire. Now the LTP sender follows the states beginning at the CP marker (second part of the diagram), and follows the internal procedure Retransmit Checkpoint (Section 6.7):

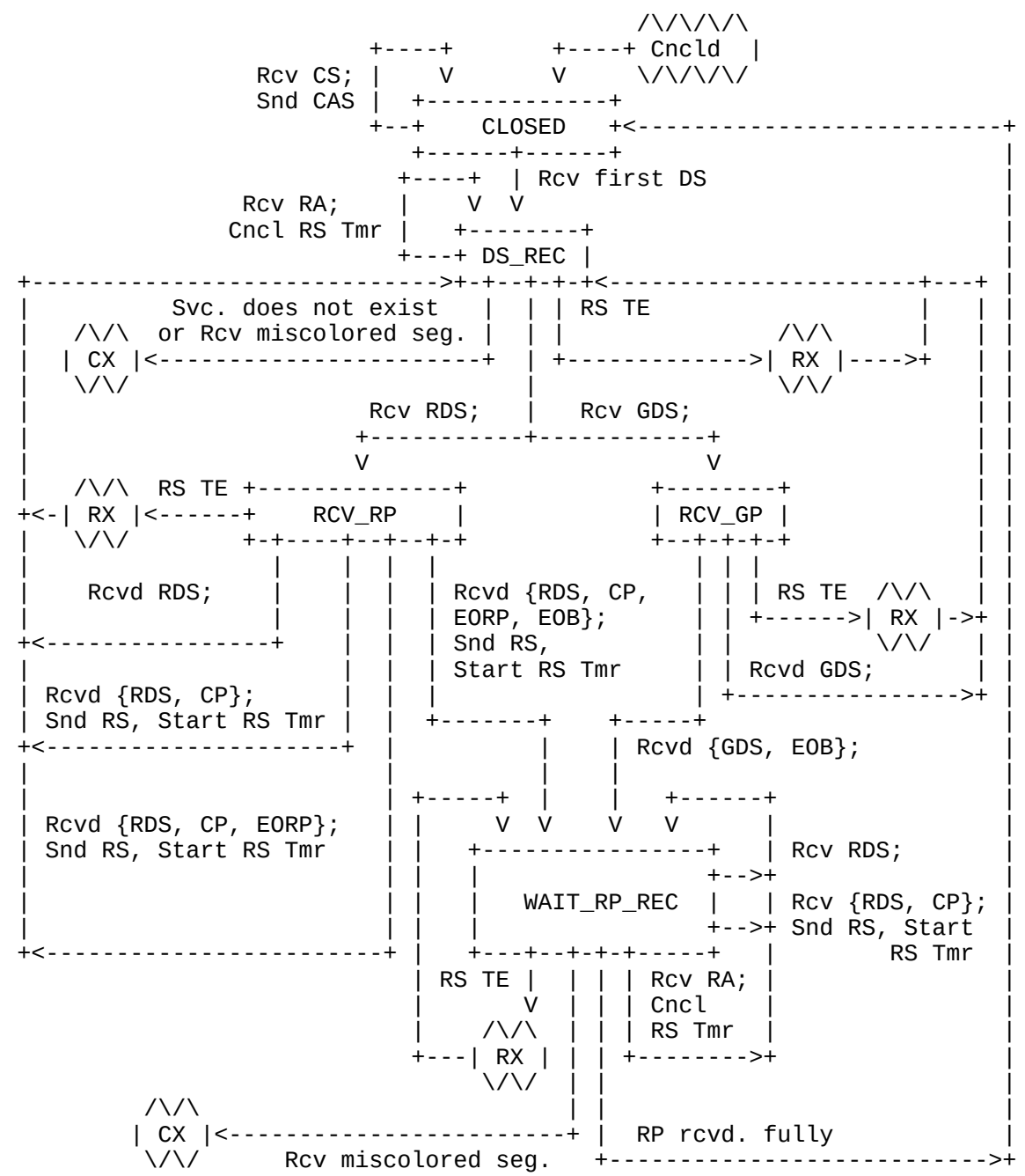
If the CP Retransmission Limit set by network management for the session has been exceeded, the LTP sender proceeds towards canceling the session (with reason-code RLEXC) as indicated by the sequence of state transitions following the CX marker. Otherwise (if the Retransmission Limit is not exceeded yet), the CP segment is queued for retransmission and the LTP sender returns to the interrupted state. The internal procedure Start Checkpoint Timer (Section 6.2) is started again upon receiving a link state cue indicating the beginning of transmission of the segment.

The LTP sender stays at the WAIT_RP_ACK state after reaching it until the red-part data is fully acknowledged as received by the receiver LTP, and then returns to the CLOSED state following the internal procedure Close Session (Section 6.20).

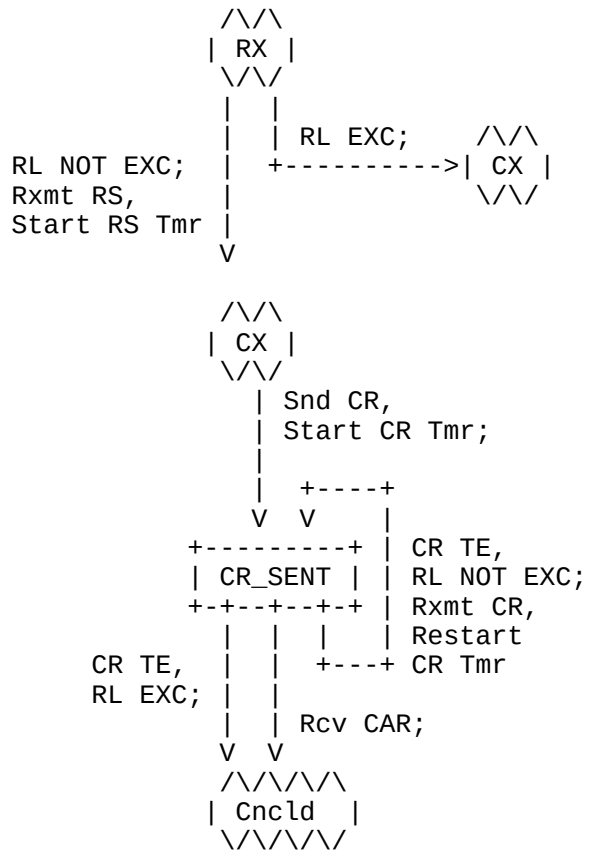
Note that while at the CLOSED state, the LTP sender might receive an RS segment (if the last transmitted RA segment before session close got lost or if the LTP receiver retransmitted the RS segment prematurely), in which case it retransmits an acknowledging RA segment and stays in the CLOSED state. If the session was canceled by the receiver by issuing a CR segment, the receiver may retransmit the CR segment (either prematurely or because the acknowledging CAR segment got lost). In this case, the LTP sender retransmits the acknowledging CAR segment and stays in the CLOSED state.

8.2. Receiver

LTP Receiver State Transition Diagram



Receiver State Transition Diagram (contd.)



Asynchronous cancel requests are handled in a manner similar to the way they are handled in the LTP sender. If the cancel request was made from the local client service instance and the LTP receiver was not already in the CR_SENT state, a CR segment with reason-code USR_CNCLD SHOULD be sent to the LTP sender following the sequence of state transitions beginning at the CX marker as described above. If the asynchronous cancel request is received from the LTP sender, a CAS segment is sent and the LTP receiver moves to the CLOSED state (independent of the state the LTP receiver may be in).

The LTP receiver begins at the CLOSED state and enters the Data Segment Reception (DS_REC) state upon receiving the first data segment. If the client service ID referenced in the data segment was non-existent, a Cx segment with reason-code UNREACH SHOULD be sent to the LTP sender via the Cancellation sequence beginning with the CX marker (second part of the diagram). If the received segment was

found to be miscolored, the internal procedure Handle Miscolored Segment (Section 6.21) is followed, and a CX segment with reason-code MISCOLORED SHOULD be sent to the LTP sender with the Cancellation sequence beginning with the CX marker.

Otherwise, the LTP receiver enters the Receive Red-Part state (RCV_RP) or the Receive Green-Part state (RCV_GP) depending on whether the segment received was red or green, respectively.

In the RCV_RP state, a check is made of the nature of the received red DS. If the segment was a regular red data segment, the receiver LTP just returns to the DS_REC state. For red data segments marked also as CP and as CP & EORP, a responding RS segment is queued for transmission to the sender following either the internal procedure Retransmit RS (Section 6.8) or Send Reception Report (Section 6.11) depending on whether the CP segment was a retransmission (an RS segment corresponding to the checkpoint serial number in the CP segment was previously issued) or not, respectively. The LTP receiver then returns to the DS_REC state. If the block transmission was fully red and the segment was marked as CP, EORP, and EOB, the LTP receiver enters the Wait-for-Red-Part-Reception state (WAIT_RP_REC). In all cases, the internal procedure Start RS Timer (Section 6.3) is followed upon receiving link state cues indicating the beginning of transmission of the RS segments.

In the RCV_GP state, if the received green data segment was not marked EOB, the LTP receiver returns to the DS_REC state. Otherwise, it enters the WAIT_RP_REC state to receive the red-part of the block fully.

A previously set RS timer may expire and interrupt the LTP receiver while in the DS_REC, RCV_RP, RCV_GP, or WAIT_RP_REC state. If so, the internal procedure Retransmit RS (Section 6.8) is followed as illustrated in the states beginning at the RX marker (shown in the second part of the diagram) before returning to the interrupted state:

- A check is made here to see if the retransmission limit set by the network management has been exceeded in the number of RSs sent in the session. If so, a CR segment with reason-code RLEXC SHOULD be sent to the LTP sender and the sequence indicated by the CX marker is followed. Otherwise, the RS segment is queued for retransmission and the associated RS timer is started following the internal procedure Start RS Timer (Section 6.3) upon receiving a link state cue indicating the beginning of its transmission.

The LTP receiver may also receive RA segments from the sender in response to the RS segments sent while in the DS_REC state. If so, then the RS timer corresponding to the report serial number mentioned in the RA segment is canceled following the internal procedure Stop RS Timer (Section 6.14).

The LTP receiver stays in the WAIT_RP_REC state until the entire red-part of the block is received, and moves to the CLOSED state upon full red-part reception. In this state, a check is made upon reception of every red-part data segment to see if it is at a block offset higher than any green-part data segment received. If so, the internal procedure Handle Miscolored Segment (Section 6.21) is invoked and the sequence of state transitions beginning with the CX marker is followed; a CX segment with reason-code MISCOLORED SHOULD be sent to the LTP sender with the Cancellation sequence beginning with the CX marker.

Note that if there were no red data segments received in the session yet, including the case where the session was indeed fully green or the pathological case where the entire red-part of the block gets lost but at least the green data segment marked EOB is received (the LTP receiver has no indication of whether the session had a red-part transmission), the LTP receiver assumes the "RP rcvd. fully" condition to be true and moves to the CLOSED state from the WAIT_RP_REC state.

In the WAIT_RP_REC state, the LTP receiver may receive the retransmitted red data segments. Upon receiving red data segments marked CP, it queues the responding RS segment for transmission based on either internal procedure Retransmit RS (Section 6.8) or Send Reception Report (Section 6.11) depending on whether the CP was found to be a retransmission or not, respectively. The internal procedure Start RS Timer is invoked upon receiving a link state cue indicating the beginning of transmission of the RS segment. If an RA segment is received, the RS timer corresponding to the report segment mentioned is canceled and the LTP receiver stays in the state until the entire red-part is received.

In the sequence of state transitions beginning at the CX marker, the CR segment with the given reason-code (depending on how the sequence is entered) is queued for transmission, and the CR timer is started upon reception of the link state cue indicating actual transmission following the internal procedure Start Cancel Timer (Section 6.15). If the CAR segment is received from the LTP sender, the LTP receiver returns to the CLOSED state (via the Cncld marker) following the internal procedure Stop Cancel Timer (Section 6.18). If the CR timer expires asynchronously, the internal procedure Retransmit Cancellation Segment (Section 6.16) is followed:

- A check is made to see if the retransmission limit set by the network management for the number of CR segments per session has been exceeded. If so, the LTP receiver returns to the CLOSED state following the Cncld marker. Otherwise, a CR segment is scheduled for retransmission with the CR timer being started following the internal procedure Start Cancel Timer (Section 6.15) upon reception of a link state cue indicating actual transmission.

The LTP receiver might also receive a retransmitted CS segment at the CLOSED state (either if the CAS segment previously transmitted was lost or if the CS timer expired prematurely at the LTP sender). In such a case, the CAS is scheduled for retransmission.

9. Security Considerations

9.1. Denial of Service Considerations

Implementers SHOULD consider the likelihood of the following Denial of Service (DoS) attacks:

- A fake Cx could be inserted, thus bringing down a session.

- Various acknowledgment segments (RA, RS, etc.) could be deleted, causing timers to expire, and having the potential to disable communication altogether if done with a knowledge of the communications schedule. This could be achieved either by mounting a DoS attack on a lower-layer service in order to prevent it from sending an acknowledgment segment, or by simply jamming the transmission (all of which are more likely for terrestrial applications of LTP).

- An attacker might also corrupt some bits, which is tantamount to deleting that segment.

- An attacker may flood an LTP engine with segments for the internal operations queue and prevent transmission of legitimate data segments.

- An attacker could attempt to fill up the storage in an engine by sending many large messages to it. In terrestrial LTP applications, this may be much more serious since spotting the additional traffic may not be possible from any network management point.

If any of the above DoS attacks is likely, then one or more of the following anti-DoS mechanisms ought to be employed:

- Session numbers SHOULD be partly random making it harder to insert valid segments.

- An engine that suspects that either it or its peer is under DoS attack could frequently checkpoint its data segments (if it were the sender) or send asynchronous RSs (if it were the receiver), thus eliciting an earlier response from its peer or timing out earlier due to the failure of an attacker to respond.

- Serial numbers (checkpoint serial numbers, report serial numbers) MUST begin each session anew using random numbers rather than from 0.

- The authentication header [LTPEXT].

9.2. Replay Handling

The following algorithm is given as an example of how an LTP implementation MAY handle replays.

1. On receipt of an LTP segment, check against a cache for replay. If this is a replay segment and if a pre-cooked response is available (stored from the last time this segment was processed), then send the pre-cooked response. If there is no pre-cooked response, then silently drop the inbound segment. This can all be done without attempting to decode the buffer.

2. If the inbound segment does not decode correctly, then silently drop the segment. If the segment decodes properly, then add its hash to the replay cache and return a handle to the entry.

3. For those cases where a pre-cooked response should be stored, store the response using the handle received from the previous step. These cases include:

   (a) when the inbound packet is a CP segment, the RS segment sent in response gets stored as pre-cooked,

(b) when the Incoming packet is an RS segment, the RA segment is stored as pre-cooked, and

(c) when the incoming packet is a Cx segment, the CAx segment sent in response gets stored pre-cooked.

4. Occasionally clean out the replay cache -- how frequently this happens is an implementation issue.

The downside of this algorithm is that receiving a totally bogus segment still results in a replay cache search and attempted LTP decode operation. It is not clear that it is possible to do much better though, since all an attacker would have to do to get past the replay cache would be to tweak a single bit in the inbound segment each time, which is certainly cheaper than the hash+lookup+decode combination, though also certainly more expensive than simply sending the same octets many times.

The benefit of doing this is that implementers no longer need to analyze many bugs/attacks based on replaying packets, which in combination with the use of LTP authentication should defeat many attempted DoS attacks.

9.3. Implementation Considerations

SDNV

Implementations SHOULD make sanity checks on SDNV length fields and SHOULD check that no SDNV field is too long when compared with the overall segment length.

Implementations SHOULD check that SDNV values are within suitable ranges where possible.

Byte ranges

Various report and other segments contain offset and length fields. Implementations MUST ensure that these are consistent and sane.

Randomness

Various fields in LTP (e.g., serial numbers) MUST be initialized using random values. Good sources of randomness that are not easily guessable SHOULD be used [ESC05]. The collision of random values is subject to the birthday paradox, which means that a collision is likely after roughly the square root of the space has been seen (e.g., 2^16 in the case of a 32-bit random value).

Implementers MUST ensure that they use sufficiently long random values so that the birthday paradox doesn't cause a problem in their environment.

10. IANA Considerations

10.1. UDP Port Number for LTP

The UDP port number 1113 with the name "ltp-deepspace" has been reserved for LTP deployments. An LTP implementation may be implemented to operate over UDP datagrams using this port number for study and testing over the Internet.

10.2. LTP Extension Tag Registry

The IANA has created and now maintains a registry for known LTP Extension Tags (as indicated in Section 3.1). The registry has been populated using the initial values given in Section 3.1 above. IANA may assign LTP Extension Tag values from the range 0x02-0xAF (inclusive) using the Specification Required rule [GUIDE]. The specification concerned can be an RFC (whether Standards Track, Experimental, or Informational), or a specification from any other standards development organization recognized by IANA or with a liaison with the IESG, specifically including CCSDS (http://www.ccsds.org/). Any use of Reserved values (0xB0-0xBF inclusive) requires an update this specification.

11. Acknowledgments

Many thanks to Tim Ray, Vint Cerf, Bob Durst, Kevin Fall, Adrian Hooke, Keith Scott, Leigh Torgerson, Eric Travis, and Howie Weiss for their thoughts on this protocol and its role in Delay-Tolerant Networking architecture.

Part of the research described in this document was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. This work was performed under DOD Contract DAA-B07-00-CC201, DARPA AO H912; JPL Task Plan No. 80-5045, DARPA AO H870; and NASA Contract NAS7-1407.

Thanks are also due to Shawn Ostermann, Hans Kruse, Dovel Myers, and Jayram Deshpande at Ohio University for their suggestions and advice in making various design decisions. This work was done when Manikantan Ramadas was a graduate student at the EECS Dept., Ohio University, in the Internetworking Research Group Laboratory.

Part of this work was carried out at Trinity College Dublin as part of the SeNDT contract funded by Enterprise Ireland's research innovation fund.

12. References

12.1. Normative References

[B97] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

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

[LTPMTV] Burleigh, S., Ramadas, M., and S. Farrell,"Licklider Transmission Protocol - Motivation", RFC 5325, September 2008.

[LTPEXT] Farrell, S., Ramadas, M., and S. Burleigh, "Licklider Transmission Protocol - Security Extensions", RFC 5327, September 2008.

12.2. Informative References

[ASN1] Abstract Syntax Notation One (ASN.1). ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER). ITU-T Rec. X.690 (2002) | ISO/IEC 8825-1:2002.

[BP] Scott, K. and S. Burleigh, "Bundle Protocol Specification", RFC 5050, November 2007.

[DTN] K. Fall, "A Delay-Tolerant Network Architecture for Challenged Internets", In Proceedings of ACM SIGCOMM 2003, Karlsruhe, Germany, Aug 2003.

[ESC05] D. Eastlake, J. Schiller and S. Crockerr, "Randomness Recommendations for Security", RFC 4086, June 2005.

[SACK] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgement Options", RFC 2018, October 1996.

Authors' Addresses

Manikantan Ramadas
ISRO Telemetry Tracking and Command Network (ISTRAC)
Indian Space Research Organization (ISRO)
Plot # 12 & 13, 3rd Main, 2nd Phase
Peenya Industrial Area
Bangalore 560097
India
Telephone: +91 80 2364 2602
EMail: mramadas@gmail.com

Scott C. Burleigh
Jet Propulsion Laboratory
4800 Oak Grove Drive
M/S: 301-490
Pasadena, CA 91109-8099
Telephone: +1 (818) 393-3353
Fax: +1 (818) 354-1075
EMail: Scott.Burleigh@jpl.nasa.gov

Stephen Farrell
Computer Science Department
Trinity College Dublin
Ireland
Telephone: +353-1-896-1761
EMail: stephen.farrell@cs.tcd.ie

Full Copyright Statement

Copyright (C) The IETF Trust (2008).

This document is subject to the rights, licenses and restrictions contained in BCP 78 and at http://www.rfc-editor.org/copyright.html, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.