Delay Tolerant Networking Research Group
Internet Draft
<draft-irtf-dtnrg-ltp-00.txt>
May 2004
Expires November 2004

S. Burleigh
NASA/Jet Propulsion Laboratory
M. Ramadas
Ohio University
S. Farrell
Trinity College Dublin

Licklider Transmission Protocol

Status of this Memo

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). These long round-trip times may result from the use of half-duplex channels or from data propagation delays that are so lengthy as to simulate half-duplex transmission. Such environments are not well served by TCP, which depends on relatively short round-trip times for retransmission buffer management, timely flow control, and negotiation of other connection parameters.

Communication across interplanetary space is the most prominent example of this sort of environment, and LTP is in fact adapted from existing communication technologies designed to support deep space flight missions. It is principally aimed at supporting "long-haul" reliable transmission in the interplanetary space but might have applications in other long-RTT environments as well.

Table of Contents

1. Introduction 3
2. Motivation 5
2.1 IPN Operating Environment 5
2.2 Why not TCP? 7
3. Features 7
3.1 Massively Parallel State Retention 8
3.1.1 Out-of-order Delivery 9
3.1.2 Session IDs 10
3.1.3 Use of Non-volatile Storage 11
3.2 Absence of Negotiation 11
3.3 Laconic Acknowledgment 12
3.4 Adjacency 13
3.5 Optimistic and Dynamic Timeout Interval Computation 14
3.6 Deferred Transmission 15
4. Terminology 15
5. Overall Operation 19
5.1 Nominal Operation 19
5.2 Retransmission 20
5.2.1 Reception Reporting Rules 22
5.2.2 Design Rationale 22
5.3 Accelerated Delivery 24
5.4 Accelerated Retransmission 24
5.5 Session Cancellation 25
5.6 Unreliable Transmission 26
6. Functional Model 26
6.1 Deferred Transmission 27
6.2 Bandwidth Management 27
6.3 Timers 28
7. Segment Structure 30
7.1 Segment Header 30
7.1.1 Segment Type Flags 31
7.1.2 Segment Type Codes 32
7.1.3 Segment Class Masks 32
7.2 Segment Content 33
7.2.1 Data Segment 33

7.2.2 Report Segment 34
7.2.3 Report Acknowledgment Segment 36
7.2.4 Session Management Segments 36
8. Requests from Client Service 37
8.1 Transmission Request 37

8.2 Cancellation Request 38
9. Internal Procedures 39
9.1 Start Transmission 39
9.2 Start Checkpoint Timer 40
9.3 Start RS Timer 40
9.4 Stop Transmission 40
9.5 Suspend Timers 40
9.6 Resume Timers 41
9.7 Retransmit Checkpoint 42
9.8 Retransmit RS 42
9.9 Signify Segment Arrival 42
9.10 Signify Block Reception 43
9.11 Send Reception Report 43
9.12 Signify Transmission Completion 44
9.13 Retransmit Data 44
9.14 Stop RS Timer 45
9.15 Start Cancel Timer 45
9.16 Retransmit Cancellation Segment 45
9.17 Acknowledge Cancellation 46
9.18 Stop Cancellation Timer 46
9.19 Cancel Session 47
9.20 Close Session 47
10. Notices to Client Service 47
10.1 Session Start 47
10.2 Data Segment Arrival 47
10.3 Block Reception 48
10.4 Transmission Completion 48
10.5 Transmission Cancellation 48
10.6 Reception Cancellation 49
11. Requirements from the Operating Environment 49
12. Security Considerations 49
12.1 Mechanisms and Layering Considerations 51
12.2 Denial of Service Considerations 52
12.3 Authentication header 53
12.4 Implementation Considerations 53
12.5 Miscellaneous 54

13. Tracing LTP back to CFDP 54
14. IANA Considerations 56
15. Acknowledgments 56
16. References 57
17. Author's Addresses 57

1. Introduction

The Licklider Transmission Protocol (LTP) described in this memo is designed to provide retransmission-based reliability over links characterized by extremely long message round-trip times. These long round-trip times may result from the use of half-duplex channels or

from data propagation delays that are so lengthy as to simulate half-duplex transmission. Such environments are not well served by TCP, which depends on relatively short round-trip times for retransmission buffer management, timely flow control, and negotiation of other connection parameters.

As 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 the Interplanetary Internet (IPN) [IPN].

Since 1982 the principal source of standards for space communications has been the Consultative Committee for Space Data Systems (CCSDS)[CCSDS]. Engineers of CCSDS member agencies have developed communication protocols that function within the constraints imposed by operations in deep space. These constraints differ sharply from those within which the Internet typically functions:

- Extremely long signal propagation delays, on the order of seconds, minutes, or hours rather than milliseconds.
- Frequent and lengthy interruptions in connectivity.
- Low levels of communication traffic coupled with high rates of transmission error.
- Meager bandwidth and highly asymmetrical data rates.

The CCSDS File Delivery Protocol (CFDP) [CFDP], in particular, automates reliable file transfer across interplanetary distances by detecting data loss and initiating the requisite retransmission without mission operator intervention.

CFDP by itself is sufficient for operating individual missions, but its built-in networking capabilities are rudimentary. In order to operate within the IPN environment it must rely on the routing and incremental retransmission capabilities of the Bundling protocol defined for Delay-Tolerant Networking [BP][DTNRG]. LTP is intended to serve as a reliable "convergence layer" protocol underlying Bundling in DTN regions whose links are characterized by very long round-trip times. Its design notions are directly descended from the retransmission procedures defined for CFDP.

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. Motivation

2.1 IPN Operating Environment

There are a number of fundamental differences between the environment for terrestrial communications and the operating environments envisioned for the IPN.

The most challenging difference between communication among points on Earth and communication among planets is round-trip delay, of which there are two main sources, both relatively intractable: natural law and economics.

The more obvious type of delay imposed by nature is signal propagation time. Our inability to transmit data at speeds higher than the speed of light means that, while round-trip times in the terrestrial Internet range from milliseconds to a few seconds, minimum round-trip times to Mars range from 8 to 40 minutes, depending on the planet's position. Round-trip times between Earth and Jupiter's moon Europa run between 66 and 100 minutes.

Less obvious and more dynamic is the delay imposed by occultation.

Communication between planets must be by radiant transmission, which is usually possible only when the communicating entities are in line of sight of each other. An entity residing on a planetary surface will be periodically taken out of sight by the planet's rotation (it will be "on the other side of" the planet); an entity that orbits a planet will be periodically taken out of sight by orbital motion (it will be "behind" the planet); and planets themselves lose mutual visibility when their trajectories take them to opposite sides of the sun. During the time that communication is impossible, messages must wait in a queue for later transmission. Delivery is necessarily retarded.

Round-trip times and occultations can at least be readily computed given the ephemerides of the communicating entities. Additional delay that is less easily predictable is introduced by discontinuous transmission support, which is rooted in economics.

Communicating over interplanetary distances requires expensive special equipment: large antennas, high-performance receivers, etc. For most deep-space missions, even non-NASA ones, these are currently provided by NASA's Deep Space Network (DSN). The communication resources of the DSN are currently oversubscribed and will probably remain so for the foreseeable future. While studies have been done as to the feasibility of upgrading or replacing the current DSN, the number of deep space missions will probably continue to grow faster than the terrestrial infrastructure that supports them, making over-

subscription a persistent problem. Radio contact via the DSN must therefore be carefully scheduled and is often severely limited.

This over-subscription means that the round-trip times experienced by packets will be affected not only by propagation delay and occultation, but also by the scheduling and queuing delays imposed by management of Earth-based resources: packets to be sent to a given destination may have to be queued until the next scheduled contact period, which may be hours, days, or even weeks away. While queuing and scheduling delays are generally known well in advance except when missions need emergency service (such as during landings and maneuvers), the long and highly variable delays make the design of timers, and retransmission timers in particular, quite difficult.

Another significant difference between deep space and terrestrial

communication is bandwidth availability. The combined effects of large distances (resulting in signal attenuation), the expense and difficulty of deploying large antennas to distant planets, and the difficulty of generating electric power in space all mean that the available bandwidth for communication in the IPN will likely remain modest compared to terrestrial systems. Maximum data rates on the order of a few tens of megabits per second will probably be the norm for the next few decades.

Moreover, the available bandwidths are highly asymmetrical: data are typically transmitted at different rates in different directions on the same link. Current missions are usually designed with a much higher data "return" rate (from spacecraft to Earth) than "command" rate (from Earth to spacecraft). The reason for the asymmetry is simple: nobody ever wanted a high-rate command channel, and, all else being equal, it was deemed better to have a more reliable command channel than a faster one. This design choice has led to data rate asymmetries in excess of 100:1, sometimes approaching 1000:1. A strong desire for a very robust command channel will probably remain, so any transport protocol designed for use in the IPN will need to function with a relatively low-bandwidth outbound channel to spacecraft and landers.

The difficulty of generating power on and around other planets will also result in relatively low signal-to-noise ratios and thus high bit error rates. Current deep-space missions operate with raw bit error rates on the order of 10^-1, or one error in ten bits; heavy coding is used to reduce these rates, and of course this coding further reduces the residual bandwidth available for data transmission.

Propagation delay is the only truly immutable characteristic that distinguishes the IPN from terrestrial communications (unless and

until we find a way to transmit information faster than the speed of light). Queuing and scheduling delays, low data rates, intermittent connectivity, and high bit error rates can all be mitigated or eliminated by adding more infrastructure. But this additional infrastructure is likely to be provided (if at all) only in the more highly developed core areas of the IPN. We see the IPN growing outwards from Earth as we explore more and more planets, moons, asteroids, and possibly other stars. This suggests that there will

always be a "fringe" to the fabric of the IPN, an area without a rich communications infrastructure. The delay, data rate, connectivity, and error characteristics mentioned above will probably always be an issue somewhere in the IPN.

2.2 Why not TCP?

These environmental characteristics - long delays, low and asymmetric bandwidth, intermittent connectivity, and relatively high error rate - make using unmodified TCP for end to end communications in the IPN infeasible. Using the equations from Mathis, et al., [MSM97], we can calculate an upper bound on the sustainable throughput of a TCP connection, taking into account TCP's congestion avoidance mechanisms. Even if only 1 in 100 million packets are lost, a TCP connection to Mars is limited to just under 250kbps. If we assume that 1 in 5000 packets is lost (this figure was reported by Paxson as the packet corruption rate in the Internet [P97] then that number falls to around 1,600bps.

These values are upper bounds on steady-state throughput. Since the number of packets in an episode of connectivity will generally be under 10,000 due to the low available bandwidth, TCP performance would be dominated by its behavior during slow-start. This means that even when Mars is at its closest approach to Earth it would take a TCP nearly 100 minutes to ramp up to an Earth-Mars transmission rate of 20kbps.

Note: lab experiments using a channel emulator and standard applications show that even if TCP could be pushed to work efficiently at such distances, many applications either rely on several rounds of handshaking or have built-in timers that render them non-functional when the round-trip-time is over a couple of minutes. For example, it typically takes eight round trips for FTP to get to a state where data can begin flowing. Since an FTP server may time out and reset the connection after 5 minutes of inactivity, a conformant standard FTP server could be unusable for communicating even with the closest planets.

3. Features

The design of LTP differs from that of TCP in several significant

ways. The common themes running through these differences are two central design assumptions, both of which amount to making virtues of necessity.

First: given the severe innate constraints on interplanetary communication discussed above, we assume that the computational resources available to LTP engines will always be ample compared to the communication resources available on the link between them.

Certainly in many cases the computational resources available to a given LTP engine - such as one on board a small robotic spacecraft will not be ample by the standards of the Internet. But in those cases we expect that the associated communication resources (transmitter power, antenna size) will be even less ample, preserving the expected disproportion between the two.

Second, we note that establishing a communication link across interplanetary distance entails enacting several hardware configuration measures based on the presumed operational state of the remote communicating entity:

- orienting a directional antenna correctly;
- tuning a transponder to pre-selected transmission and/or reception frequencies;
- diverting precious electrical power to the transponder at the last possible moment, and for the minimum necessary length of time.

We therefore assume that the operating environment in which LTP functions is able to pass "link state cues" to LTP, telling it which remote LTP engine(s) should currently be transmitting to the local LTP engine and vice versa. The operating environment itself must have this information in order to configure communication link hardware correctly.

3.1 Massively Parallel State Retention

Like any reliable transport service, LTP is "stateful". In order to assure the reception of a block of data it has sent, LTP must retain for possible retransmission all portions of that block which might not yet have been received. In order to do so, it must keep track of which portions of the block are known to have been received so far, and which are not, together with any additional information needed for purposes of retransmitting part or all of that block.

Long round-trip times mean substantial delay between the transmission of a block of data and the reception of an acknowledgment from the block's destination, signaling arrival of the block. If LTP postponed transmission of additional blocks of data until it received acknowledgment of the arrival of all prior blocks, valuable opportunities to utilize what little deep space transmission bandwidth is available would be forever lost.

For this reason, LTP is based in part on a notion of massively parallel transmission.

Any number of requested transmissions may be concurrently "in flight" at various displacements along the link between two LTP engines, and the LTP engines must necessarily retain transmission status and retransmission resources for all of them. Moreover, if any of the data of a given block are lost en route, it will be necessary to retain the state of that transmission during an additional round trip while the lost data are retransmitted; even multiple retransmission cycles may be necessary.

In sum, LTP transmission state information persists for a long time because a long time must pass before LTP can be assured of transmission success - so LTP must retain a great deal of state information.

Since the alternatives are non-reliability on the one hand and severe under-utilization of transmission opportunities on the other, we believe such massive statefulness is cost-justified (though probably not in all applications).

3.1.1 Out-of-order Delivery

This design decision is reflected in a significant structural difference between LTP and TCP.

Both TCP and LTP provide mechanisms for multiplexing access by a variety of higher-layer services or applications: LTP's "client service IDs" correspond to TCP's port numbers. Also, both TCP and LTP implement devices for encapsulating threads of retransmission protocol: LTP's "sessions" functionally correspond to TCP connections. At any moment each such thread of retransmission protocol is engaged in conveying some single block of data from one protocol end point to another.

But while TCP permits only a single connection on a given port at any time, LTP permits an unlimited number of concurrent sessions for each

client service. And just as in TCP the vagaries of retransmission may cause data transmitted on one connection (on one port) to be

delivered after data that were subsequently transmitted on another connection (another port), so too in LTP is it possible for data transmitted in one session (for one client service) to be delivered after data that were subsequently transmitted in another session (for another - or possibly the same - client service).

In short, while TCP always delivers data in transmission order on a single port, LTP may well deliver data out of transmission order to a single client service. The contrasts may be summarized as follows:

	------TCP------	-------LTP-------
number of "ports"	65,535	unlimited; normally 1
"connections" per "port"	1	unlimited
maximum number of concurrent connection state machines	65,535	unlimited
blocks transmitted per "connection"	unlimited (one at a time)	1

Out-of-transmission-order delivery of transmitted blocks to client services averts two serious problems that could be raised by a single "bit hit" - the unrecoverable corruption of a single segment of one block - followed by the successful reception of some number of subsequently transmitted blocks while retransmission of the lost segment is requested and accomplished.

First, it ensures that delivery of the successfully received data is not unnecessarily postponed. LTP leaves up to the client service all decisions on what can and cannot be done with this data pending delivery of the undelivered block. [Note that this places on the client services all responsibility for establishing sequence relationships among transmitted blocks, e.g., embedding timestamps and/or sequence numbers within the blocks.]

Second, it enables LTP to release resources allocated to the

completed sessions' state information as rapidly as possible. This somewhat mitigates the burden of statefulness at the receiving engine.

3.1.2 Session IDs

In TCP, the delivery of data in transmission order on any single port, without gaps, enables the application that is receiving data on that port to use delivery order as the basis for reconstituting the

originally transmitted data items. LTP client services count on LTP to accomplish this reconstitution at the block level, but LTP itself cannot rely on data delivery order being similarly useful for this purpose. LTP instead attaches to each segment of application data the "session ID" that uniquely identifies the original transmission in which the data were issued. Session ID and block offset enable LTP to reassemble the originally transmitted data block from segments of data received out of offset order.

Note that, even so, the order of delivery of completed blocks may differ from the order in which the blocks were originally transmitted. Again, attaching the appropriate service-specific semantic significance to each delivered block is a client service responsibility.

3.1.3 Use of Non-volatile Storage

Another important implication of massive statefulness is that implementations of LTP should consider retaining transmission state information in non-volatile storage of some kind, such as magnetic disk or flash memory. Transport protocols such as TCP typically retain transmission state in dynamic RAM; if the computer on which the software resides is rebooted or powered down, then all transmissions currently in progress are aborted but the resulting degree of communication disruption is limited, because the number of concurrent transmissions is limited. Rebooting or power-cycling a computer on which an LTP engine resides could abort a much larger number of transmission sessions in various stages of completion, at a much larger total cost.

3.2 Absence of Negotiation

Implicit in the design of TCP is the assumption that the parameters of communication over a given connection can be bilaterally negotiated and renegotiated in a timely manner. Adjustment of transmission rate in particular is accomplished by the exchange of information in real time.

In the IPN, however, round-trip times may be so long and communication opportunities so brief that a negotiation exchange might not be completed before connectivity was lost. Even if connectivity were uninterrupted, waiting for negotiation to complete before revising data transmission parameters might well result in costly under-utilization of link resources.

For this reason, all LTP communication session parameters are established unilaterally, subject to application-level network management activity that may not take effect for hours, days, or

weeks.

3.3 Laconic Acknowledgment

Another respect in which LTP differs from TCP is that, while TCP connections are bidirectional (blocks of application data may be flowing in both directions on any single connection), LTP sessions are unidirectional. This design decision derives from the possible multiplicity of parallel sessions for any single client service, together with the fact that the flow of data in deep space flight missions is usually unidirectional. (Long round trip times make interactive spacecraft operation infeasible, so spacecraft are largely autonomous and command traffic is very light.)

One could imagine an LTP instance, upon being asked to transmit a block of data, searching through all existing sessions in hopes of finding one that was established upon reception of data from the new block's destination; transmission of the new block could be piggybacked on the acknowledgment traffic for that session. But the prevailing unidirectionality of space data communications means that such a search would frequently fail, and a new unidirectional session would have to be established anyway. Session bidirectionality therefore seemed to entail somewhat greater complexity unmitigated by any clear performance advantage, so we abandoned it. Bidirectional data transfer is supported, but it requires opening two individual

LTP sessions.

Because they are not piggybacked on data segments, LTP data acknowledgments - "reception reports" - are carried in a separate segment type. To minimize consumption of low and asymmetric transmission bandwidth in the IPN, these report segments are sent infrequently; each one contains a comprehensive report of all data received within some specified range of offsets from the start of the transmitted block. The expectation is that most data segments will arrive safely, so individual acknowledgment of each one would be expensive in information-theoretical terms: the real information provided per byte of acknowledgment data transmitted would be very small. Instead, report segments are normally sent only upon encountering explicit solicitations for reception reports - "checkpoints" - in the sequence of incoming data segments.

The aggregate nature of reception reports gives LTP transmission an episodic character:

- "Original transmissions" are sequences of data segments issued in response to transmission requests from client services.

- "Retransmissions" are sequences of data segments issued in

response to the arrival of report segments that indicate incomplete reception.

Checkpoints are mandatory only at the end of each original transmission or retransmission. For applications that require accelerated retransmission (and can afford the additional bandwidth consumption entailed), reception reporting can be more aggressive. Additional checkpoints may optionally be inserted at other points in an original transmission, and additional reception reports may optionally be sent on an asynchronous basis.

3.4 Adjacency

TCP reliability is "end to end": traffic between two TCP endpoints may traverse any number of intermediate network nodes, and two successively transmitted segments may travel by entirely different routes to reach the same destination. The underlying IP network-layer protocol accomplishes this routing. Although TCP always

delivers data segments to any single port in order and without gaps, the IP datagrams delivered to TCP itself may not arrive in the order in which they were transmitted.

In contrast, LTP reliability is "point to point": traffic between two LTP engines is expected not to traverse any intermediate relays. Point-to-point topology is innate in the nature of deep space communication, which is simply the exchange of radiation between two mutually visible antennae. No underlying network infrastructure is presumed, so no underlying network-layer protocol activity is expected; the underlying communication service is assumed to be a point-to-point link-layer protocol such as CCSDS Telemetry/Telecommand [TM][TC] (or, for terrestrial applications, PPP). The contents of link-layer frames delivered to LTP are always expected to arrive in the order in which they were transmitted, though possibly with any number of gaps due to data loss or corruption.

Note that building an interplanetary network infrastructure - the DTN-based architecture of the IPN - *on top of* LTP does not conflict with LTP design principles. The Bundling protocol functions at a new hyper-network level, and LTP bears essentially the same relationship to Bundling that a reliable link protocol would bear to IP. The design of LTP relies heavily on this topological premise, in at least two ways:

If two successively transmitted segments could travel by materially different routes to reach the same destination, then the assumption of rough determinism in timeout interval computation discussed below would not hold. Our inability to estimate timeout intervals with any

accuracy would severely compromise performance.

If data arrived at an LTP engine out of transmission order, then the assumptions on which the rules for reception reporting are based would no longer hold. A more complex and/or less efficient retransmission mechanism would be needed.

3.5 Optimistic and Dynamic Timeout Interval Computation

TCP determines timeout intervals by measuring and recording actual round trip times, then applying statistical techniques to recent RTT

history to compute a predicted round trip time for each transmitted segment.

The problem is at once both simpler and more difficult for LTP:

- The massively parallel nature of LTP changes the granularity at which timer accuracy is required. Confirmation of the reception of one block, or one segment, does not retard transmission of the next, so timeout values that would be intolerably optimistic in TCP don't constrain LTP's bandwidth utilization.

- But the reciprocal half-duplex nature of LTP communication makes it infeasible to use statistical analysis of round-trip history as a means of predicting round-trip time. The round-trip time for transmitted segment N could easily be orders of magnitude greater than that for segment N-1.

Since statistics derived from round-trip history cannot safely be used as a predictor of LTP round-trip times, we have to assume that round-trip timing is at least roughly deterministic - i.e., that sufficiently accurate RTT estimates can be computed individually in real time from available information.

This computation is performed in two stages.

We calculate a first approximation of RTT by simply doubling the known one-way light time to the destination and adding an arbitrary margin for possible processing delay at both ends of the transmission. The margin value is typically a small number of whole seconds. Although such a margin is enormous by Internet standards, it is insignificant compared to the two-way light time component of round-trip time in deep space. We choose to risk tardy retransmission, which will postpone delivery of one block by a relatively tiny increment, in preference to premature retransmission, which will unnecessarily consume precious bandwidth and thereby degrade overall performance.

Then, to account for the additional delay imposed by interrupted connectivity, we dynamically suspend timers during periods when the relevant remote LTP engines are known to be unable to transmit responses. This knowledge of the operational state of remote

entities is assumed to be provided by link state cues from the operating environment, as discussed earlier.

3.6 Deferred Transmission

Link state cues also notify LTP when it is and isn't possible to transmit segments by passing them to the underlying communication service.

Continuous duplex communication is the norm for TCP operations in the Internet; when communication links are not available, TCP simply does not operate. In deep space communications, however, at no moment can there ever be any expectation of two-way connectivity. It is always possible for LTP to be generating outbound segments - in response to received segments, timeouts, or requests from client services - that cannot immediately be transmitted. These segments must be queued for transmission at a later time when a link has been established, as signaled by a link state cue.

4. 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 DTN Bundling protocol, the convergence layer adapter mediating between 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 (typically the bundling layer) to be transmitted via LTP from one client service instance to another.

(3) Block Prefix

Any subset of a block that begins at the start of the block.

(4) Session

A thread of LTP protocol activity conducted for the purpose of transmitting a block.

(5) 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. An LTP segment is either a data segment, a report segment, a report-acknowledgment segment, a cancel segment, or a cancel-acknowledgment segment.

(6) Scope

A subset of a block. Scope comprises two numbers, upper bound and lower bound.

For a data segment, lower bound is the offset of the segment's client service data from the start of the block, while upper bound is the sum of the offset and length of the segment's client service data. For example, a segment with block offset 1000 and length 500 would have a lower bound 1000 and upper bound 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 yet 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 eligible for retransmission.

Reception reports (which may comprise multiple report segments) also have scope, as defined in Section 5.2.1 below.

(7) 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.

(8) Checkpoint

A data segment soliciting a reception report from the receiving LTP engine. All checkpoints other than the EOB segment that are NOT themselves issued in response to a reception report, are discretionary checkpoints, sent unreliably. The EOB segment and all

checkpoints issued in response to reception reports are mandatory

checkpoints, sent reliably.

(9) Reception Report

A sequence of one or more report segments reporting on all block data reception (within some scope) since the start of the block's transmission session.

(10) Synchronous Reception Report

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

(11) Asynchronous Reception Report

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

(12) 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 EOB for a session.

(13) 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.

(14) 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.

A key strategic element in the design is the use of self-delimiting numeric values (SDNVs) that are similar in design to the Abstract Syntax Notation One [ASN1] encoding. An SDNV can be used to encode a variable length number from 1 to 128 bytes long, and is of two basic types, SDNV-8 and SDNV-16.

The first octet of an SDNV - the "discriminant" - fully characterizes

the SDNV's value.

SDNV-8

If the first bit of the discriminant is zero, the length of the SDNV-8 is 1 octet and the contents of the remaining 7 bits of the discriminant viewed as an unsigned integer is the value encoded. An integer in the range of 0 to 127 can be encoded this way.

Otherwise (if the first bit of the discriminant is 1), the remaining 7 bits encode the length of the encoded number. If the content of the remaining 7 bits viewed as an unsigned integer has the value N, the encoded number is N+1 octets long and has the value found by concatenating octets 2 through N+2 of the SDNV-8, viewed as an unsigned integer. For example, if N were 0, the following single octet would have the value of the SDNV-8; if N were 127, the following 128 octets would have the encoded unsigned number.

SDNV-16

If the first bit of the discriminant is zero, then the length of the SDNV-16 is 2 octets and the contents of the remaining 7 bits of the discriminant concatenated with the following octet, viewed as a 15-bit unsigned integer, is the value encoded. An integer in the range of 0 to 32767 can be encoded this way.

Otherwise (if the first bit of the discriminant is 1), the encoding is similar to SDNV-8. If the content of the remaining 7 bits viewed as an unsigned integer has the value N, the encoded number is N+1 octets long, and has the value found by concatenating octets 2 through N+2 of the SDNV-16, viewed as an unsigned integer.

An SDNV can therefore be used to represent both very large and very small integer values. For example, the maximum size of an SDNV - a 1024-bit number - is large enough to contain a fairly safe encryption key, while any whole-degree Celsius temperature in the range that humans tolerate can be represented in a single-octet SDNV-8.

In the LTP header specification that follows, various fields in the header are defined to be of types SDNV-8 or SDNV-16 depending on the typical range of values expected for the field. If a field in the header carries a number that typically requires 16 bits to encode, but under certain infrequent conditions may grow longer and require, say 32 bits to encode, it might be optimal to specify it as an SDNV-16 instead of specifying the field as a fixed 32 bit integer.

However, 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 an additional 8 bits of discriminant may not be justified. For example, an SDNV-8 is a poor way to represent an integer whose value typically falls in the range 128 to 255.

In general, though, we believe that SDNV representation of selected numeric values in LTP segments yields the smallest segment sizes without sacrificing scalability.

5. Overall Operation

5.1 Nominal Operation

The nominal sequence of events in an LTP transmission session is as follows.

Operation begins when a client service instance asks an LTP engine to transmit a block to a remote client service instance. The sending engine opens a Sending State Record (SSR) for a new session, thereby starting the session, and it notifies the client service instance that the session has been started. The sending engine then initiates an original transmission: it queues for transmission as many data segments as are necessary to transmit the entire block, within the constraints on maximum segment size imposed by the underlying

communication service. The last such segment is marked both as a checkpoint indicating that the receiving engine must issue a reception report upon receving the segment, and as an EOB indicating that the receiving engine can calculate the size of the block by summing the offset and length of the data in the segment.

At the next opportunity, subject to allocation of bandwidth to the queue into which the block data segments were written, the enqueued segments are transmitted to the LTP engine serving the remote client service instance. A timer is started for the EOB, so that it can automatically be retransmitted if no response is received.

On reception of the first data segment for the block, the receiving engine opens a Receiving State Record (RSR) for the new session, and it notifies the local instance of the relevant client service that the session has been started. In the nominal case it receives all segments of the original transmission without error. Therefore on reception of the EOB data segment it responds by (a) queuing for transmission to the sending engine a report segment indicating complete reception and (b) delivering the received block to the local instance of the client service.

At the next opportunity, the enqueued report segment is immediately transmitted to the sending engine and a timer is started so that the report segment can be retransmitted automatically if no response is received.

The sending engine receives the report segment, turns off the timer for the EOB, enqueues for transmission to the receiving engine a report-acknowledgment segment, notifies the local client service instance that the block has been successfully transmitted, and closes the SSR for the session.

At the next opportunity, the enqueued report-acknowledgment segment is immediately transmitted to the receiving engine.

The receiving engine receives the report-acknowledgment segment, turns off the timer for the report segment, and closes the RSR for the session.

Closing both the SSR and RSR for a session terminates the session.

5.2 Retransmission

Loss or corruption of transmitted segments causes the operation of LTP to deviate from the nominal sequence of events described above.

Loss of one or more data segments other than the EOB triggers data retransmission:

Rather than returning a single reception report indicating complete reception, the receiving engine returns a reception report comprising as many report segments as are needed in order to report in detail on all data reception for this session (other than data reception that was previously reported in response to any discretionary checkpoints), within the constraints on maximum segment size imposed by the underlying communication service. [Still, only one report segment is normally returned; multiple report segments are needed only when a large number of segments comprising non-contiguous intervals of block data are lost.] A timer is started for *each* report segment.

On reception of each report segment the sending engine responds as follows:

It turns off the timer for the checkpoint referenced by the report segment, if any.

It enqueues a reception-acknowledgment segment acknowledging the report segment (to turn off the report retransmission timer at the

receiving engine). This segment is sent immediately at the next transmission opportunity.

If the reception claims in the report segment indicate that not all data within the scope have been received, it normally initiates a retransmission by re-enqueuing all data segments not yet received. The last such segment is marked a checkpoint and contains the report serial number of the report segment to which the retransmission is a response. These segments are likewise sent at the next transmission opportunity, but subject to allocation of bandwidth to the queue into which the retransmission data segments were written. A timer is started for the

checkpoint, so that it can automatically be retransmitted if no responsive report segment is received.

However, if the number of checkpoints issued for this session has reached a predefined limit (established by network management), then the receiving engine instead cancels the session as described later.

If, on the other hand, the reception claims in the RS indicate that all data within the scope of the RS have been received, and moreover the union of all reception claims received so far in this session indicate that all data in the block have been received, then the sending engine notifies the local client service instance that the block has been successfully transmitted and closes the SSR for the session.

On reception of a checkpoint segment with a non-zero report serial number, the receiving engine first turns off the timer for the referenced report segment. Then it returns a reception report comprising as many report segments as are needed in order to report in detail on all data reception within the scope of the referenced report segment, within the constraints on maximum segment size imposed by the underlying communication service; a timer is started for each report segment. If at this point all data in the block have been received, the receiving engine delivers the received block to the local instance of the client service and closes the RSR for the session; otherwise the data retransmission cycle continues.

However, if the number of reception reports issued for this session has reached a predefined limit (established by network management), then the receiving engine instead cancels the session as described later.

The detailed rules under which reception reports are produced are defined in Section 5.2.1 below.

Loss of an EOB or other checkpoint segment, or of the responsive report segment causes the checkpoint timer to expire. When this occurs, the sending engine normally retransmits the checkpoint segment. However, if the number of times this checkpoint has been retransmitted has reached a predefined limit (established by network

management), then the sending agent instead cancels the session.

Similarly, loss of a report segment or of the responsive report-acknowledgment segment causes the report segment's timer to expire. When this occurs, the receiving engine normally retransmits the report segment. However, if the number of times this report segment has been retransmitted has reached a predefined limit (established by network management), then the receiving agent instead cancels the session.

Reception of a previously received report segment that was retransmitted due to loss of an report-acknowledgment segment causes another responsive report-acknowledgment segment to be transmitted, but is otherwise ignored; if any of the data retransmitted in response to the previously received report segment were lost, further retransmission of those data will be requested by one or more new report segments issued in response to that earlier retransmission's checkpoint. Thus unnecessary retransmission is suppressed.

5.2.1 Reception Reporting Rules

The upper bound of a synchronous reception report is the upper bound of the checkpoint segment to which it responds.

The upper bound of an asynchronous reception report is the maximum upper bound value among all data segments received so far in the affected session.

The lower bound of a primary reception report is the upper bound of the previously issued primary reception report for the same session, if any; otherwise it is zero.

The lower bound of a secondary reception report is the lower bound of the report segment to which the triggering checkpoint was itself a response.

Asynchronous reception reports are never issued after the arrival of the EOB segment for a session.

A reception report comprises as many reception segments as are necessary to report on all data reception in the scope of the reception report, within the constraints on maximum segment size imposed by the underlying communication service. [Again, a reception

report normally comprises only a single reception segment; multiple report segments are needed only when a large number of segments for non-contiguous intervals of block data are lost.] The lower bound of the first report segment of a reception report is the reception report's lower bound; the upper bound of the last report segment of a reception report is the reception report's upper bound.

5.2.2 Design Rationale

Note that the responsibility for responding to segment loss in LTP is shared between the sender and receiver of a block: the sender retransmits checkpoint segments in response to checkpoint timeouts, and it retransmits non-checkpoint data segments in response to reception reports indicating incomplete reception, while the receiver additionally retransmits report segments in response to timeouts. An alternative design would have been to make the sender responsible for all retransmission, in which case the receiver would not expect report-acknowledgment segments and would not retransmit report segments. There are two disadvantages to this approach:

First, because of constraints on segment size that might be imposed by the underlying communication service, it is at least remotely possible that the response to any single checkpoint might be multiple report segments. An additional sender-side mechanism for detecting and appropriately responding to the loss of some proper subset of those reception reports would be needed. We believe the current design is simpler.

Second, an engine that receives a block needs a way to determine when the session can be closed. In the absence of explicit final report acknowledgment (which entails retransmission of the report in case of the loss of the report acknowledgment), the alternatives are (a) to close the session immediately on transmission of the report segment that signifies complete reception and (b) to close the session on receipt of an explicit authorization from the sender. In case (a), loss of the final report segment would cause retransmission of a checkpoint by the sender, but the session would no longer exist at the time the retransmitted checkpoint arrived; the checkpoint could reasonably be interpreted as the first data segment of a new block, most of which was lost in transit, and the result would be redundant retransmission of the entire block. In case (b), the explicit session termination segment and the responsive acknowledgment by the receiver (needed in order to turn off the timer for the termination segment, which in turn would be needed in case of in- transit loss or corruption of that segment) would somewhat complicate the protocol, increase bandwidth consumption, and retard the release of session state resources at the sender. Again we believe the current design is simpler and more efficient.

5.3 Accelerated Delivery

The receiving engine normally delivers block data content to the client service only at the moment when reception of the block is completed - that is, on reception of the last not-yet-received segment of the block. For some applications, however, it may be desirable to deliver block data content incrementally, upon arrival, because portions of the block may be individually useful to the client service.

When the client service requests accelerated delivery of a block, the arrival of each new data segment causes the receiving engine to deliver to the client service the data content of the segment together with the segment's offset within the block. The client service assumes all responsibility for reassembling the block; upon completion of reception, the receiving engine just delivers the final data segment's content and offset to the client service as usual but additionally indicates that reception is now complete.

5.4 Accelerated Retransmission

Data segment retransmission occurs only on receipt of a report segment indicating incomplete reception; report segments in turn, are normally transmitted only at the end of an original transmission or retransmission. For some applications it may be desirable to trigger data segment retransmission incrementally during the course of an original transmission so that the retransmitted segments arrive sooner. This can be accomplished in two ways:

Any data segment prior to the last one in the transmission can additionally be flagged as a checkpoint. Reception of each checkpoint causes the receiving engine to issue a reception report.

At any time during the original transmission of a session (that is, prior to reception of the EOB), the receiving engine can unilaterally issue additional "asynchronous" reception reports. [Note that the CFDP protocol's "Immediate" mode is an example of this sort of asynchronous reception reporting; see Section 12.] The reception reports generated for accelerated retransmission are processed in exactly the same way as the standard reception reports.

5.5 Session Cancellation

A transmission session may be canceled by either the sending or the receiving engine, in response either to a request from the local client service instance or to an LTP operational failure as noted

earlier. Session cancellation is accomplished as follows.

The canceling engine deletes all currently queued segments for the session and notifies the local instance of the affected client service that the session is canceled. If no segments for this session have yet been sent to or received from the corresponding LTP engine, then at this point the canceling engine simply closes its state record for the session and cancellation is complete. Otherwise, the canceling engine queues for transmission to the corresponding engine a session cancellation segment.

At the next opportunity, subject to allocation of bandwidth to the queue into which the cancellation segment was written, the enqueued cancellation segment is transmitted to the LTP engine serving the remote client service instance. A timer is started for the segment, so that it can automatically be retransmitted if no response is received.

The corresponding engine receives the cancellation segment, enqueues for transmission to the canceling engine a cancellation-acknowledgment segment, deletes all other currently queued segments for the indicated session, notifies the local client service instance that the block has been canceled, and closes its state record for the session.

At the next opportunity, the enqueued cancellation-acknowledgment segment is immediately transmitted to the canceling engine.

The canceling engine receives the cancellation-acknowledgment, turns off the timer for the cancellation segment, and closes its state record for the session.

Loss of a cancellation segment or of the responsive cancellation-acknowledgment causes the cancellation segment timer to expire. When this occurs, the canceling engine normally retransmits the

cancellation segment. However, if the number of times this cancellation segment has been retransmitted has reached a predefined limit (established by network management), then the canceling agent instead simply closes its state record for the session.

5.6 Unreliable Transmission

For operational conditions in which the massive statefulness of LTP reliability is unsupportable or unnecessary, LTP can perform unreliable transmission. In unreliable mode all retransmission and session cancellation capabilities are disabled, but LTP's block segmentation, bandwidth management, interface to the underlying communication service, and incremental data delivery may still make

it useful to client services.

The nominal sequence of events in an unreliable transmission session is much simplified.

Operation begins when a client service instance asks an LTP engine to transmit a block unreliably to a remote client service instance. The sending engine queues for transmission as many data segments as are necessary to transmit the entire block, within the constraints on maximum segment size imposed by the underlying communication service. The last such segment is marked an EOB signifying that the receiving engine can calculate the size of the block by summing the offset and length of the data in this segment. Note that this segment is an EOB but not a checkpoint.

At the next opportunity, subject to allocation of bandwidth to the queue into which the block data segments were written, the enqueued segments are transmitted to the LTP engine serving the remote client service instance.

The arrival of each data segment causes the receiving engine to deliver to the client service the data content of the segment together with the segment's offset within the block. The client service assumes all responsibility for reassembling the block.

Upon arrival of the EOB segment, the receiving engine just delivers that final data segment's content and offset to the client service as usual but additionally indicates that reception of the block is now

complete.

Loss or corruption of transmitted data segments is not recoverable in this mode. Loss of the EOB is particularly troublesome: the receiving client service instance cannot readily distinguish between actual data loss and very severe queuing delay in this case, and the total size of the block can never be known. But for some applications (e.g., continuous telemetry streaming), or in deployment over a reliable link-layer protocol, this deficiency may be unimportant.

6. Functional Model

The functional model underlying the specification of LTP is one of deferred, opportunistic transmission, with access to the active transmission link apportioned among multiple outbound traffic queues. The accuracy of LTP retransmission timers depends in large part on a faithful adherence to this model.

6.1 Deferred Transmission

Every outbound LTP segment is appended to one of several (conceptual) queues of traffic bound for that segment's destination. Production of a segment and the subsequent actual transmission of that segment are in principle wholly asynchronous.

In the event that (a) a transmission link to the destination is currently active and (b) the queue to which a given outbound segment is appended is otherwise empty and (c) this queue is determined to have the highest transmission priority among all outbound traffic queues associated with that destination, the segment will be transmitted immediately upon production. Transmission of a newly appended segment is necessarily deferred in all other circumstances.

Conceptually, the de-queuing of segments from traffic queues bound for a given destination is initiated upon reception of a link state cue indicating that the underlying communication system is now transmitting to that destination, i.e., the link to that destination is now active. It ceases upon reception of a link state cue indicating that the underlying communication system is no longer transmitting to that destination, i.e., the link to that destination is no longer active.

Note: in the following discussion, the de-queuing of a segment always implies delivering it to the underlying communication system for immediate transmission.

6.2 Bandwidth Management

We believe it is necessary for LTP to provide a mechanism for apportioning access to the active transmission link, possibly unevenly, among multiple classes of client service data traffic, and at the same time to provide a minimum-latency control channel for acknowledgment traffic. To accomplish these ends, the LTP functional model is based on the use of N outbound traffic queues, N > 1, for each destination with which the LTP engine can communicate.

One such queue is strictly reserved for LTP internal operations: it contains only report and acknowledgment segments (collectively, "acknowledging segments"), which must be transmitted promptly to protect timer accuracy. A second queue is reserved for segments produced in sessions designated as "priority" sessions. Any other queues supported by a given LTP engine are for segments produced in non-priority sessions, typically of varying levels of urgency. The client service specifies the queue to be used for transmitting a given block - either the priority session queue or one of the non-priority session queues - at the time transmission of the block is requested.

While the link to a given destination is active, continuous iteration of the following algorithm governs the de-queuing of segments from the N traffic queues bound for that destination:

If any segments are currently in the internal operations queue, then de-queue the oldest such segment.

Otherwise, if any segments are currently in the priority session queue, then de-queue the oldest such segment.

Otherwise, if there are any other non-empty queues, invoke an implementation-specific algorithm to select the next queue to transmit from and then de-queue the oldest segment in that queue.

6.3 Timers

LTP relies on accurate calculation of expected arrival times for report and acknowledgment segments in order to know when proactive retransmission is required. If a calculated time were even slightly early, the result would be costly unnecessary retransmission. On the other hand, calculated arrival times may safely be several seconds late: the only penalties for late timeout and retransmission are slightly delayed data delivery and slightly delayed release of session resources.

The following discussion is the basis for LTP's expected arrival time calculations.

The total time consumed in a single "round trip" (transmission and reception of the original segment, followed by transmission and reception of the acknowledging segment) has the following components:

Protocol processing time consumed in issuing the original segment, receiving the original segment, generating and issuing the acknowledging segment, and receiving the acknowledging segment.

Outbound queuing delay: delay at the sender of the original segment while that segment is in a queue waiting for transmission, and delay at the sender of the acknowledging segment while that segment is in a queue waiting for transmission.

Radiation time: the time that passes while all bits of the original segment are being radiated, and the time that passes while all bits of the acknowledging segment are being radiated. (This is significant only at extremely low data transmission rates.)

Round-trip light time: propagation delay at the speed of light, in both directions.

Inbound queuing delay: delay at the receiver of the original segment while that segment is in a reception queue, and delay at the receiver of the acknowledging segment while that segment is in a reception queue.

Delay in transmission of the acknowledging segment due to loss of connectivity - that is, interruption in outbound link activity at the

sender of the acknowledging segment due to occultation, scheduled end of tracking pass, etc.

In this context, where errors on the order of seconds or even minutes may be tolerated, processing time at each end of the session is assumed to be negligible.

Inbound queuing delay is also assumed to be negligible because, even on small spacecraft, LTP processing speeds are high compared to data transmission rates.

Two mechanisms are used to make outbound queuing delay negligible:

The expected arrival time of an acknowledging segment is not calculated until the moment the underlying communication system notifies LTP that radiation of the original segment has begun. All outbound queuing delay for the original segment has already been incurred at that point.

Acknowledging segments (reports and acknowledgments) are always appended to the internal operations queue. This limits outbound queuing delay for an acknowledging segment to the time needed to de-queue and radiate all other acknowledging segments that are currently in that queue. Since acknowledging segments are sent infrequently and are normally very small, outbound queuing delay for a given acknowledging segment is likely to be minimal.

Radiation delay at each end of the session is simply segment size divided by transmission data rate. It is insignificant except when data rate is extremely low (e.g., 10 bps), in which case the use of LTP may well be inadvisable for other reasons. Therefore radiation delay is normally assumed to be negligible.

And we assume that one-way light time to the nearest second can always be known (e.g., provided by the operating environment).

So the initial expected arrival time for each acknowledging segment is computed as simply the current time at the moment that radiation of the original segment begins, plus twice the one-way light time, plus 2*N seconds of margin to account for processing and queuing delays and for radiation time at both ends. N is a parameter set by

network management for which 2 seconds seem to be a reasonable default value.

This leaves only one unknown, the additional round trip time introduced by loss of connectivity. To account for this, we again rely on external link state cues. Whenever interruption of transmission at a remote LTP engine is signaled by a link state cue, we suspend the countdown timers for all acknowledging segments expected from that engine. Upon a signal that transmission has resumed at that engine, we resume those timers after (in effect) adding to each expected arrival time the length of the timer suspension interval.

7. Segment Structure

Each LTP segment comprises (a) a "header" in a standard format and (b) zero or more octets of "content". LTP segments are of four general types, depending on the nature of the content carried.

Data segments carry client service (application) data, together with metadata enabling the receiving client service instance to receive and make use of that data.

Report segments carry data reception claims together with the upper and lower bounds of the data block scope to which the claims pertain.

Report acknowledgment segments carry only the serial number of the report being acknowledged.

Session management segments are of two general subtypes: Cancellation and Cancellation acknowledgment. The Cancellation segments carry a single byte reason-code to indicate the reason for the cancellation. Cancellation acknowledgment segments have no content.

7.1 Segment Header

<< Recommendations of SDNV-8 / SDNV-16 for fields in the segment header as recommended in this section, are under discussion. Future versions of the draft may recommend fields to be of one SDNV type instead of the other (SDNV-8 in place of SDNV-16, for example), if found to be more appropriate. >>

An LTP segment header comprises three data items: a single-octet control byte, a session ID, and an expansion zone.

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 below.

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

Session originator: the engine ID of the LTP engine that initiated the session, in SDNV-8 representation.

Session number: a number in SDNV-16 representation, typically a timestamp, generated by the LTP engine identified as the session originator.

The format and resolution of session number are matters that are private to the session-originating engine; the only requirement imposed by LTP is that every session initiated by an LTP engine MUST be uniquely identified by the session ID. For example, if timestamp resolution is not sufficient, an LTP implementation may choose to append an 8 or 16 bit sequence number to the timestamp to guard against the possibility of multiple sessions starting at the same system time.

Expansion zone is a numeric value in SDNV-8 representation intended for future expansion of LTP capabilities. In the absence of any expansion features, it MUST be a single octet whose binary value is zero.

7.1.1 Segment Type Flags

The last four 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 as follows.

Control flag (CTRL)

A value of 0 indicates that the segment carries data and is a data segment, while a value of 1 indicates that the segment carries control information for the protocol (and not data).

Exception flag (EXC)

A value of 1 in a data segment indicates that the segment is being transmitted unreliably. In a control segment (CTRL flag set), this indicates that the segment pertains to session cancellation activity.

Request flag (REQ)

If set, this flag signifies a request for some specific response from the receiver. The nature of that response depends on the values of the other flags as described below.

Closure flag (CLOS)

When set, this flag signifies the termination of some element of protocol activity. The nature of the activity being terminated again depends on the values of the other flags as described below.

7.1.2 Segment Type Codes

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

CTRL	EXC	REQ	CLOS	Code	Nature of segment
0	0	0	0	0	Data, NOT a Checkpoint, NOT EOB
0	0	0	1	1	Undefined
0	0	1	0	2	Data, Checkpoint, NOT EOB
0	0	1	1	3	Data, Checkpoint, EOB
0	1	0	0	4	Data [unreliable transmission], not EOB
0	1	0	1	5	Data [unreliable transmission], EOB
0	1	1	0	6	Undefined
0	1	1	1	7	Undefined
1	0	0	0	8	Report segment
1	0	0	1	9	Report acknowledgment
1	0	1	0	10	Undefined
1	0	1	1	11	Undefined
1	1	0	0	12	Cancel segment

1	1	0	1	13	Cancel acknowledgment
1	1	1	0	14	Undefined
1	1	1	1	15	Undefined

7.1.3 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	REQ	CLOS	Mnemonic	Description
0	0	1	-	CP	Checkpoint
0	0	1	1	EOB	End of block;
0	1	0	1		block size = offset + length
1	0	0	0	R	Report segment; carries reception claims
1	0	0	1	RA	Report acknowledgment
1	1	0	0	C	Cancellation
1	1	0	1	CA	Cancellation acknowledgment

7.2 Segment Content

7.2.1 Data Segment

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

If the data segment is a checkpoint, the segment MUST additionally include the following two serial numbers (Checkpoint serial number, Report serial number) to support efficient retransmission. All non-checkpoint data segments MUST NOT have these two fields and MUST begin with the Client service ID field defined below as the first

element of the data segment.

Checkpoint serial number [SDNV-8]

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 [ECS94] for this. Any subsequent checkpoints issued by the sender MUST have the serial number value one more than the last checkpoint serial number issued. Any retransmission of the checkpoint segment MUST have the same serial number as the original transmission.

Report serial number [SDNV-8]

If the checkpoint was queued for transmission in response to the reception of an R segment [Sec 9.13], then its value MUST be the report serial number value of the R segment that caused the data

segment to be queued for transmission.

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

Client service ID [SDNV-8]

The client service ID number identifies the upper-level service to which the segment is to be delivered by the destination LTP engine. It is functionally analogous to a well-known 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.

At this time the only LTP client service we envision in the IPN is the DTN Bundling protocol. The client service ID value assigned to this client service is the number 1.

Offset [SDNV-16]

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-16]

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

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.

7.2.2 Report Segment

The content of an R segment 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-8]

The report serial number uniquely identifies the report among all reports issued by the block 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 [ECS94] for this. Any subsequent reports issued by the receiver MUST have the serial number value one more than the last report serial number issued. Any retransmission of the R segment MUST have the same serial number as the original transmission.

Checkpoint serial number [SDNV-8]

The value of 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 is the checkpoint serial number of the checkpoint that caused the R segment to be issued.

Upper bound [SDNV-16]

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-16]

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-8]

The number of data reception claims in this report segment.

Data reception claims

Each reception claim comprises two elements: offset and length.

Offset [SDNV-16]

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

Length [SDNV-16]

The length of a reception claim indicates the number of contiguous octets of block data starting at the indicated offset (within the scope of the report) that have been successfully received so far.

Reception claims MUST conform to the following rules:

A reception claim's offset shall never be less than zero and its length shall never be less than 1.

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 shall never exceed the difference between the upper and lower bounds of the report segment.

Implied requests for retransmission of client service data can be inferred from an R segment'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.

7.2.3 Report Acknowledgment Segment

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

Report serial number [SDNV-8]

This field returns the report serial number of the R segment being acknowledged.

7.2.4 Session Management Segments

The C segment carries a single byte reason-code with the following semantics.

Reason-Code Semantics

00	Client Service canceled session
01	Unreachable Client Service
02	Retransmission limit exceeded
03-FF	Undefined

The CA segments have no content.

8. 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.

8.1 Transmission Request

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

Client service ID

Destination LTP engine ID

Data to send, as an array of bytes.

Length of the data to be sent.

Quality of service required: reliable or unreliable transmission.

Flow-label, used to choose the queue within the queue-set for the LTP destination.

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 an EOB, with appropriate EOB segment flag bits set depending on reliable / unreliable transmission [Sec 7.1.2].

If the requested quality of service is reliable transmission, then all data segments resulting from subdivision of the data MUST have the EXC flag cleared. Moreover, at least the EOB segment MUST also be marked a checkpoint by having the REQ flag set; zero or more other data segments (selected according to an algorithm that is a local implementation matter) MAY additionally have the REQ flag set to act as checkpoints.

On the other hand if the requested quality of service is unreliable transmission, then all data segments resulting from subdivision of the data MUST have the EXC flag set and REQ flag cleared.

All data segments are appended to the appropriate (conceptual) transmission queue as specified in the transmission request.

Finally, a session start notice [Sec 10.1] is sent back to the client service that requested the transmission.

8.2 Cancellation Request

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

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

First the internal "Cancel session" procedure [Sec 9.19] is invoked.

Next, if the session is being canceled by the block 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 radiated, i.e., if the destination engine cannot possibly be aware of this session - then the session is simply closed; the "Close session" procedure [Sec 9.20] is invoked.
>
> Otherwise, a C segment with reason-code value 00 [Sec 7.2.4] MUST be appended to the internal operations queue of the queue-set bound for the destination LTP engine specified in the transmission request that started this session.

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

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 session is simply closed; the "Close session" procedure [Sec 9.20] is invoked.

Otherwise, a C segment with reason-code value 00 [Sec 7.2.4] MUST be appended to the internal operations queue of the queue-set bound for the block sender.

9. Internal Procedures

This section describes the internal procedures that are triggered by the occurrence of various events during the life-time of the 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.

The data segments transmitted in the course of any single LTP session MUST either all have segment type code less than 4 (reliable transmission) or else all have segment type code greater than 3 (unreliable transmission).

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 data segment is simply discarded. Otherwise, if the data segment was transmitted reliably (segment type code less than 4), a C segment with reason-code value 01 [Sec 7.2.4] MUST be added to the internal operations queue of the queue-set bound for the block sender; a C segment with reason-code value 01 SHOULD be similarly added to the internal operations queue bound for the data sender if the data segment was transmitted unreliably (segment type

code greater than 3) [For example, in the case where the block receiver knows that the sender is functioning in a "beacon" (transmit-only) fashion, a C segment need not be sent]. In either case the received data segment is discarded.

9.1 Start Transmission

This procedure is triggered by arrival of a link state cue indicating the start 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 begins.

9.2 Start Checkpoint Timer

This procedure is triggered by arrival of a link state cue indicating the de-queuing for transmission of a CP segment, provided that it is also the EOB for a session or if it was issued in response to an R segment i.e., the segment's report serial number is non-zero.

Response: the expected arrival time of the R segment that will be produced on reception of this CP 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 [Sec 9.5], then this timer is immediately suspended, because the computed expected arrival time may require an adjustment that cannot yet be computed.

9.3 Start RS Timer

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

Response: the expected arrival time of the RA segment that will be produced on reception of this R 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 [Sec 9.5], then this timer is immediately suspended, because the computed expected arrival time may require an adjustment that cannot yet be computed.

9.4 Stop Transmission

This procedure is triggered by 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.

9.5 Suspend Timers

This procedure is triggered by 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 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 margin to account for processing and queuing delay at the remote engine; N should be a network management parameter, for which 2 seconds seems to be a reasonable default value.

If the nominal 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.

9.6 Resume Timers

This procedure is triggered by 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. In each case, expected arrival time is increased by a transmission delay interval that is calculated as follows:

The nominal 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 margin to account for processing and queuing delay at the remote engine; again, N should be a network management parameter, for which 2 seconds seems to be a reasonable default value.

If the nominal 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 acknowledge transmission time.

After adjustment of expected arrival time, each of the suspended countdown timers is resumed.

9.7 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 [Sec 9.19] is invoked, a C segment with reason-code value 02 [Sec 7.2.4] is appended to the transmission queue specified in the transmission request that started this session, and a transmission cancellation notice [Sec 10.5] is sent back to the client service that requested the transmission.

Otherwise, a new copy of the CP segment is appended to the transmission queue specified in the transmission request that started this session.

9.8 Retransmit RS

This procedure is triggered by either (a) expiration of a countdown timer associated with an R segment or (b) reception of a CP segment whose checkpoint serial number is equal to that of one or more previously issued R segments for the same session -- an unnecessarily retransmitted checkpoint.

Response: if the number of times any affected R 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 [Sec 9.19] is invoked, a C segment with reason-code value 02 [Sec 7.2.4] is appended to the queue of internal operations traffic bound for the LTP engine that originated the session, and a reception cancellation notice [Sec 10.6] is sent to the client service identified in each of the data segments received in this session.

Otherwise, a new copy of each affected R segment is appended to the queue of internal operations traffic bound for the LTP engine that originated the session.

9.9 Signify Segment Arrival

This procedure is triggered by the arrival of a data segment, but only when either (a) the data segment is being transmitted unreliably or (b) segment arrival notification has been authorized for the local LTP engine by client service or network management.

Response: a segment arrival notice [Sec 10.2] is sent to the specified client service.

9.10 Signify Block Reception

This procedure is triggered by the arrival of a data segment, but only when either (a) the segment is also the EOB segment for a block being transmitted unreliably or (b) the segment is also a CP segment for a reliably transmitted block, and the EOB for this session has been received (including the case where this segment itself is the EOB segment) and the data block's size is known, and all data in the

block being transmitted in this session have been received.

Response: a block reception notice [Sec 10.3] is sent to the specified client service.

9.11 Send Reception Report

This procedure is triggered by either (a) reception of a CP segment whose checkpoint serial number is not equal to that of any previously issued R segment or (b) an implementation-specific circumstance pertaining to a particular block reception session for which no EOB has yet been received ("asynchronous" reception reporting). The response in either case is the same.

Response: if the number of reception reports issued for this session exceeds the limit established for the local LTP engine by network management, then the affected session is canceled: the "Cancel session" procedure [Sec 9.19] is invoked, a C segment with reason-code value 02 [Sec 7.2.4] is appended to the queue of internal operations traffic bound for the LTP engine that originated the session, and a reception cancellation notice [Sec 10.6] is sent to the client service identified in each of the data segments received in this session. Otherwise, a reception report is issued as follows.

As many R segments are 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. They are appended to the queue of internal operations traffic bound for the LTP engine that originated the indicated session. If production of the reception report was triggered by reception of a checkpoint:

The upper bound of the report is the upper bound (the sum of the offset and length) of the checkpoint data segment.

If the checkpoint was itself issued in response to a report segment, then this report is a "secondary" reception report and

the lower bound of the report is that earlier report segment's lower bound. Otherwise, this report is a "primary" reception report and the lower bound of the report is the upper bound of the prior primary reception report issued for this session.

Otherwise, i.e., the reception report is asynchronous:

The upper bound of the report is the maximum upper bound among all data segments received so far for this session.

The lower bound of the report is the upper bound of the prior primary reception report issued for this session.

9.12 Signify Transmission Completion

This procedure is triggered by either (a) reception of an R segment whose reception claims taken together with the reception claims of all other report segments previously received in the course of this session indicate complete reception of an entire data block, or (b) arrival of a link state cue indicating the de-queuing (for transmission) of an EOB segment for a block transmitted unreliably.

Response: a transmission completion notice [Sec 10.4] is sent to the client service that requested the transmission identified in the segment header and the session is closed: the "Close session" procedure [Sec 9.20] is invoked.

9.13 Retransmit Data

This procedure is triggered by reception of an R segment.

Response: first, an RA segment with the same report serial number as the R segment is appended to the queue of internal operations traffic bound for the LTP engine that originated the indicated session. If the R segment is redundant i.e., either the indicated session is unknown (if for example, the R segment is received after the session has been completed or canceled), or the R segment's report serial number is equal to that of a previously received report segment for this session -- 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.

All retransmission buffer space occupied by data whose reception is claimed in the report segment can be released.

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

If the number of CP segments issued for this session exceeds the 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 [Sec 9.19] is invoked, a C segment with reason-code value 02 [Sec 7.2.4] is appended to the transmission queue specified in the transmission request that started this session, and a transmission cancellation notice [Sec 10.5] is sent back to the client service that requested the transmission.

Otherwise, new data segments encapsulating all block data whose non-reception is implied by the reception claims are appended to the transmission queue specified in the transmission request that started this session. The last - and only the last - such segment must me marked as a CP segment and must contain the report serial number of the received R segment.

9.14 Stop RS Timer

This procedure is triggered by reception of an RA segment.

Response: the countdown timer associated with the original R segment (identified by the report serial number of the RA segment) is deleted. If no other countdown timers associated with R segments existed for this session, then the session is closed: the "Close session" procedure [Sec 9.20] is invoked.

9.15 Start Cancel Timer

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

Response: the expected arrival time of the CA segment that will be produced on reception of this C 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 [Sec 9.5] then this timer is immediately suspended, because the computed expected arrival time may require an adjustment that cannot yet be computed.

9.16 Retransmit Cancellation Segment

This procedure is triggered by the expiration of a countdown timer

associated with a C segment.

Response: if the number of times this C 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 [Sec 9.20] is invoked.

Otherwise, a new copy of the C segment (retaining the same reason-code value) is appended to the appropriate transmission queue. If the segment is being sent by the block sender, then the appropriate transmission queue is the one specified in the transmission request that started this session; otherwise, the appropriate transmission queue is the queue of internal operations traffic bound for the LTP engine that originated the session.

9.17 Acknowledge Cancellation

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

Response: if the local segment is not the block sender for the block identified in the C segment and 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 no action is taken. Otherwise, a CA segment is appended to the queue of internal operations traffic bound for the LTP engine that sent the C segment.

It is possible that the C segment is being retransmitted because a previous CA 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 that session is locally canceled: the "Cancel session" procedure [Sec 9.19] is invoked and a reception cancellation notice [Sec 10.6] 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 [Sec 9.20] is invoked.

9.18 Stop Cancellation Timer

This procedure is triggered by reception of a CA segment.

Response: the session of which the segment is one token is closed, i.e., the "Close session" procedure [Sec 9.20] is invoked.

9.19 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. If the local LTP engine is the originator of the session, then all remaining data retransmission buffer space allocated to the session can be released. All countdown timers currently associated with the session are deleted.

9.20 Close Session

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

Response: any remaining countdown timers associated with the session (such as the timer associated with a C segment) are deleted. All other state information regarding the session is deleted; existence of the session is no longer recognized.

10. Notices to Client Service

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

10.1 Session Start

The LTP engine returns the Session ID of the new transmission session when a session start notice is delivered.

A session start notice informs the client service of the initiation of a transmission session in response to a transmission request from that client service. 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.

10.2 Data Segment Arrival

The parameters provided by the LTP engine when a data segment arrival notice is delivered are:

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

Source LTP engine ID

Data segment arrival notices deliver block data incrementally, as it is received. This enables the receiving client service instance to make use of partial data immediately, rather than potentially waiting hours or days for the retransmission of missing segments and the ultimate delivery of the completed block. Incremental block data delivery is mandatory for unreliable transmission, because there's never any guarantee that the EOB segment- which is required in order to deliver a complete block - will ever be received at all. Incremental delivery also enables the client service to cancel reception of a block, if necessary.

10.3 Block Reception

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

Session ID of the transmission session.

Array of client service data bytes that constitutes the block

Length of the block.

Source LTP engine ID.

A block reception notice delivers a complete data block to the client service.

10.4 Transmission Completion

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

A transmission completion notice informs the client service that the indicated session has successfully completed; the destination LTP engine has received the entire data block.

10.5 Transmission Cancellation

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

A transmission cancellation notice informs the client service that

the indicated session was terminated, either by decision of the destination client service instance or due to violation of a retransmission limit in the local LTP engine. There is no assurance that the destination client service instance received the data block.

10.6 Reception Cancellation

The sole parameter provided by the LTP engine when a reception cancellation notice is delivered is the Session ID of the transmission session.

A reception cancellation notice informs the client service that the indicated session was terminated, either by decision of the source client service instance or due to violation of a retransmission limit in the local LTP engine. The complete data block will not be delivered.

11. Requirements from the Operating Environment

LTP requires support from its operating environment (which includes network management activities) and link-state cues from the data-link layer for its operations.

The local data-link layer needs to inform LTP whenever the link to a specific LTP destination is brought up or torn down. Similarly, the operating environment needs to inform the local LTP engine whenever it is known that a remote LTP engine is set to begin communication with the local engine from the operating schedules. LTP also needs to be able to query the current speed-of-light to its peer engine from the operating environment to calculate its timers.

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

LTP also requires the underlying data-link layer to perform data integrity check of the segments received. Specifically, the data-link layer is expected to detect any segments received corrupted, and to silently discard them.

12. Security Considerations

<<This section is really only an initial set of notes resulting from discussions on and off the DTNRG list. Further analysis will

be required as LTP itself develops and is implemented.>>

LTP is designed as a hyper-datalink layer to primarily do ARQ of data transmissions over a point-to-point link exhibiting some or all of the following characteristics: high delays, high BERs and intermittent, possibly strictly scheduled connectivity.

However, how "point-to-point" the link is physically, depends on the underlying datalink. For a deep-space link using radio datalink, say we are talking from a Deep-Space-Network giant dish antenna to an orbiter on Mars, the signal could be received and processed by any other orbiter passing at the same time in the

Mars orbiter's reception area, which could potentially cover hundreds or thousands of square miles. Similarly a link transmitting data from an orbiter to a ground-rover could have a reception-coverage area of tens of square miles. So there is an inherent risk that of unintended receivers listening in on LTP transmissions over such datalinks.

Such promiscuous recipients of our LTP transmissions could potentially replay the transmissions sent, twiddle with control bits in the LTP header before they do so (more dangerous is the case when they make the bits claim the LTP segment to be a Cancel segment closing the session). Such problems are more severe for LTP compared to other terrestrial Internet protocols because LTP inherently does a lot of state retention (to handle the high delays and intermittent connectivity of its links), has very high time-out values and LTP nodes may be quite difficult to reset. In other words, by design, there is a long delay before LTP gives up on a session. Thus any such adversary listening in on the LTP transmissions has the potential to create severe DoS conditions for an LTP receiver.

To give a terrestrial example - were LTP to be used in a sparse sensor network, then denial of service attacks could be mounted which would result in nodes missing critical information, e.g. communications schedule updates. In such cases, a single DoS attack success could take a node entirely off the network until the node is physically visited and reset.

Even in the deep space cases, we do need to consider some terrestrial based attacks, in particular those involving insertion of a message into an on-going session (usually without having seen the exact bytes of the previous messages in the session). This could arise due to firewall failures at various points or due to Trojan software running on an authorized host.

Such attacks may depend on the attacker correctly "guessing" about

the state of LTP nodes, but it is worth noting that patterns of deep space communication may well be considered guessable from the Earth (e.g. a session to a Mars rover is only going to happen while that rover is visible from the Earthstation -- all public information) and the delay-tolerance inherent in LTP may decrease

the required accuracy of such guesses. Most of the attacks concerned here are DoS attacks.

12.1 Mechanisms and Layering Considerations

There is thus a real need to consider security of LTP transmissions, so we need to consider (to the extent possible) the appropriate layer(s) at which security mechanisms can best be deployed.

The Application layer (above-LTP)

This seems like a reasonable approach to protect LTP data, but would leave the LTP headers open. The headers carry information on where the transmission is coming from, which could be valuable in itself. Further, this approach provides little or no protection against DoS type attacks against LTP.

The LTP layer

Security at this layer offers nice authentication possibilities. For example, an authentication header (like that in IPSEC [AH]) can help to protect against replay attacks and bogus packets. However, an adversary can still see the LTP header of segments passing by in the ether. This approach also requires some key management infrastructure be in place in order to provide strong authentication.

The Datalink layer (below-LTP)

Providing encryption/authentication in the layer(s) below LTP has some nice properties, like being able to do encryption on-chip in hardware, making it fast. However, depending on the datalink we may be forced to use encryption / authentication on all LTP sessions which may not be required. A more flexible scheme might enable us to do encryption / authentication on only critical information sessions. For example we might want it only for commands that ask a rover to reinstall a new OS image and reboot; and may be not so much when we are transmitting a picture (though this can be hard to achieve without layering violations). Security provided by the datalink may result in unnecessary overhead and lessens flexibility, but may well be the optimal place to include

compression and encryption.

12.2 Denial of Service Considerations

Potential DoS attack points

Implementers SHOULD consider the following DoS attacks :

A fake C segment could be inserted thus bringing down a session.

Various ACK messages may be deleted causing timers to expire which could deny service if done with knowledge of the communications schedule. One possible way to achieve this would be to mount a denial of service attack on a lower layer device in order to prevent it sending an ACK, or perhaps to simply jam the transmission (all of which are more likely for terrestrial applications of LTP). An attacker might also flip some bits, which is (hopefully!) tantamount to deleting that message.

An attacker may flood a node with "internal" messages (using the terminology of section 6.2) which require processing, thus preventing "real" data segments from being transmitted.

An attacker could attempt to fill up the storage in some node 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.

- <<More TBD>>

Anti-DoS measures

<<We are considering including some or all (or none!) of the following anti-DoS measures in future versions of this specification:

A value in C segments that increases the likelihood that the message is genuine -- possibly using a hash over some previous data that is assumed to have gotten through.

Providing a mechanism whereby a node which considers that it is under DoS attack can use frequent checkpoints hopefully eliciting an earlier response from the real receiver or else timing out earlier due to the failure of the attacker to respond.

Mandating that serial numbers (checkpoint serial numbers, report serial numbers) begin each session anew using random numbers rather than from 0. Maintaining serial number ordering across multiple sessions as a further option (two sides both knowing this trick can get the benefit, without hurting a node that doesn't maintain this cross-session state.)

Add a Destination Engine ID field in segments for to provide some simple protection against replay attacks (the effectiveness of this countermeasure will depend upon lower layer data integrity.)

A node might provide an interface to its higher layers which allows the higher layers to indicate which traffic has highest priority. In the event of a resource scarcity (e.g. nearly full storage) a node could drop all other traffic. (This would depend for effectiveness upon some lower layer authentication and/or integrity mechanisms.)

>>

12.3 Authentication header

An LTP Authentication Header (LTP-AH) MAY be used to provide cryptographic authentication of the segment.

Implementations MAY support this header. If they do not support this header then they MUST ignore it. <<Check that ignoring is ok for all cases of response generation.>>

The LTP-AH uses the LTP expansion field <<Exactly how is TBD>> and contains the following fields <<Only abstract definitions for now.>>

- Ciphersuite: an eight bit integer value <<probably want two initially -- an AES-MAC and RSA-SIG>>
- KeyID: An SDNV <<with some mapping to relevant ID formats, e.g. OCTET STRING for AES-MAC, IssuerAndSerial for RSA-SIG>>
- AuthVal: An SDNV-16 value containing the authentication value <<Either a MAC or signature -- check if SDNV-16 allows long enough signatures for futureproofing>>

<<Have to say something about key management!!!>>

12.4 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 overall segment length.

Implementations SHOULD check that SDNV values are within suitable ranges where possible, e.g. <<TBD>>

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) should be initialised using random values. Good sources of randomness which are not easily guessable SHOULD be used [ECS94].

<<More TBD>>

12.5 Miscellaneous

An attacker could modify a message or insert a new message with the aim of making a session which should use reliable transport into one which uses unreliable transport and could also switch from normal to accelerated delivery. The end result could be that real data arrives out of order. Higher layer processing SHOULD take this into account if the LTP or lower layers do not preclude such attacks.

<<Other stuff that crops up.>>

13. Tracing LTP back to CFDP

LTP in effect implements most of the "core procedures" of the

CCSDS File Delivery Protocol (CFDP) specification, minus all features that are specific to operations on files and filestores (file systems); in the IPN architecture we expect that file and filestore operations will be conducted by file transfer application protocols -- notably CFDP itself -- operating on top of the DTN Bundling protocol. (Bundling in effect implements the CFDP "extended procedures" in a more robust and scalable manner than is prescribed by the CFDP standard.) The fundamental difference is that, while CFDP delivers named files end-to-end, LTP is used to transmit arbitrary, unnamed blocks of data point-to-point.

Some differences between LTP and CFDP are simply matters of

terminology; the following table summarizes the correspondences in language between the two.

LTP	CFDP
LTP engine	CFDP entity
Segment	Protocol Data Unit (PDU)
Reception Report	NAK
Client service request	Service request
Client service notice	Service indication

CFDP specifies four mechanisms for initiating data retransmission, called "lost segment detection modes". LTP effectively supports all four:

"Deferred" mode is implemented in LTP by the Request flag in the EOB data segment, which triggers reception reporting upon receipt of the EOB.

"Prompted" mode is implemented in LTP by turning on Request flags in data segments that precede the EOB; these additional checkpoints trigger interim reception reporting under the control of the source LTP engine.

"Asynchronous" mode is implemented in LTP by the autonomous production, under locally specified conditions, of additional reception reports prior to arrival of the EOB.

"Immediate" mode is simply a special case of asynchronous mode, where the condition that triggers autonomous reception reporting is detection of a gap in the incoming data.

CFDP uses a cyclic timer to iterate reception reporting until reception is complete. Because choosing a suitable interval for such a timer is potentially quite difficult, LTP instead flags the last data segment of each retransmission as a checkpoint, sent reliably; the cascading reliable transmission of checkpoint and RS segments assures the continuous progress of the transmission session.

CFDP's Suspend and Resume PDUs are functionally displaced in LTP by deferred transmission and automated bandwidth management.

As the following table indicates, most of the functions of CFDP

PDUs are accomplished in some way by LTP segments.

--------------LTP--------------	-------------CFDP------------
Data segments	File data and metadata PDUs
Closure flag on data segment	EOF (Complete)
Request flag on data segment	EOF (Complete), Prompt (NAK), Prompt (Keep Alive)
Report segment	ACK (EOF Complete), NAK, Keep Alive, Finished (Complete)
Report acknowledgment	ACK (Finished Complete)
Cancel segment	EOF (Cancel, Protocol Error) Finished (Cancel, Protocol Error)
Cancellation Acknowledgment	ACK (EOF (Cancel, Protocol Error), Finished (Cancel, Protocol Error))

But some CFDP PDUs have no LTP equivalent because in an IPN architecture they will likely be implemented elsewhere. CFDP's EOF (Filestore error) and Finished (Filestore error) PDUs would be implemented in an IPN application-layer file transfer protocol, e.g., CFDP itself. CFDP's Finished [End System] PDU is a feature of the Extended Procedures, which would in effect be implemented by the Bundling protocol.

14. IANA Considerations

At present there are none known. However if someone wanted to run LTP over IP (or even TCP or UDP), then we would want to allocate a port number. <<Considering this is TBD>>

15. 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. 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.

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.

16. References

[IPN] InterPlanetary Internet Special Interest Group web page, "http://www.ipnsig.org".

[CCSDS] Consultative Committee for Space Data Systems web page,

"http://www.ccsds.org".

[CFDP] CCSDS File Delivery Protocol (CFDP). Recommendation for Space Data System Standards, CCSDS 727.0-B-2 BLUE BOOK Issue 1, October 2002.

[BP] K. Scott, and S. Burleigh, "Bundle Protocol Specification", Work in Progress, October 2003.

[DTNRG] Delay Tolerant Networking Research Group web page, "http://www.dtnrg.org".

[MSM97] M. Mathis, J. Semke, and J. Mahdavi, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm", Computer Communications Review Vol.27(3), 1997.

[P97] V. Paxson, "Measurements and Analysis of End-to-End Internet Dynamics", Ph.D. Thesis., Computer Science Divisions, University of California, Berkeley, 1997.

[TM] Packet Telemetry Specification. Recommendation for Space Data System Standards, CCSDS 103.0-B-2 BLUE BOOK Issue 2, June 2001.

[TC] Telecommand Part 2 - Data Routing Service. Recommendation for Space Data System Standards, CCSDS 202.0-B-3 BLUE BOOK Issue 3, June 2001.

[ASN1] Abstract Syntax Notation One (ASN.1). Specification of Basic Notation. ITU-T Rec. X.680 (2002) | ISO/IEC 8824-1:2002.

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

[ECS94] D. Eastlake, S. Crocker, and J. Schiller, "Randomness

Recommendations for Security", RFC 1750, December 1994.

17. Author's Addresses

Scott C. Burleigh
Jet Propulsion Laboratory
4800 Oak Grove Drive

M/S: 179-206
Pasadena, CA 91109-8099
Telephone +1 (818) 393-3353
FAX +1 (818) 354-1075
Email Scott.Burleigh@jpl.nasa.gov

Manikantan Ramadas
Internetworking Research Group
301 Stocker Center
Ohio University
Athens, OH 45701
Telephone +1 (740) 593-1562
Email mramadas@irg.cs.ohiou.edu

Stephen Farrell
Distributed Systems Group
Computer Science Department
Trinity College Dublin
Ireland
Telephone +353-1-608-3070
Email stephen.farrell@cs.tcd.ie