논문 22-47-12-07 The Journal of Korean Institute of Communications and Information Sciences '22-12 Vol.47 No.12
https://doi.org/10.7840/kics.2022.47.12.2065

Structural Considerations for Generating and Handling LTP Report Segments from an Interoperability Testing

Cheol Hea Koo[◆°], Scott C. Burleigh[*]

ABSTRACT

Delay- and disruption-tolerant networking (DTN) technology is increasingly being considered for deep space exploration missions. In particular, DTN is being considered for trunk-line communications between ground and lunar elements of the Artemis program. Licklider transmission protocol (LTP) is a DTN protocol that supports reliable data transmission within a network and, as such, functions as a necessary "convergence-layer" protocol in the architecture. Korea Aerospace Research Institute (KARI) has recently developed an LTP reference implementation and performed an interoperability test with the National Aeronautics and Space Administration (NASA)'s Interplanetary Overlay Network (ION) DTN software. The test revealed that there can be significant variation in issuing report segments and data segment retransmission, which can lead to errors or malfunction during LTP transaction. This paper presents the results of the interoperability test between different implementations of LTP together with some implementation considerations suggested by the results to overcome the issues of the interoperability among different LTP implementations.

Key Words : Licklider transmission protocol, delay-tolerant networking, disruption-tolerant networking, DTN, convergence layer

Ⅰ. Introduction

Reliable transmission is vital in operational delay- and disruption-tolerant networking (DTN)[1] for space flight missions. Licklider transmission protocol (LTP) is a potential approach to providing this capability in the deep space environment where terrestrial internet technology cannot be applied.

Communications in the deep space environment suffer from high bit-error rates (BER), low signal-to-noise ratio (SNR), long signal propagation delay, and intermittent connectivity, which prevent the use of terrestrial internet technology in space communication[2]. LTP is a convergence-layer protocol and was invented by the Internet Research Task Force (IRTF) in collaboration with the Consultative Committee for Space Data Systems (CCSDS) as well as several space communication experts from various space agencies, companies, and institutes. The protocol seeks to address the aforementioned problems concerning space communication.

LTP is a point-to-point convergence-layer protocol working immediately above the link layer, providing reliable transmission between Bundle Protocol (BP) nodes, as shown in Fig. 1. However, LTP can also be used in non-DTN applications. For example, an LTP unit data transfer adapter can be deployed under a "class-1" implementation of CCSDS File Delivery Protocol (CFDP)[3] to support reliable point-to-point file delivery service between communication nodes, e.g., between a ground station

※ This work was supported by the IRAD project (1711170905) of Korea Aerospace Research Institute of Korea
◆° First and Corresponding Author : Korea Aerospace Research Institute, chkoo@kari.re.kr, 정회원
* InterPlanetary Networking Special Interest Group, sburleig.sb@gmail.com
논문번호 : 202209-199-B-RE, Received September 2, 2022; Revised October 14, 2022; Accepted October 20, 2022

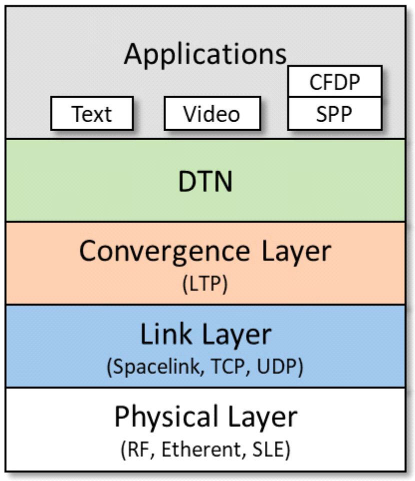


Fig. 1. Layer architecture of DTN

and a spacecraft in deep space.

The fundamental mechanism of LTP is automatic repeat request (ARQ). Successful operation of the ARQ procedures of LTP relies on proper negotiation between control segments at the closing of each LTP transaction.

The protocol data units in LTP are of five types; 1) data segment, 2) report segment, 3) report acknowledgment segment, 4) cancel segment, and 5) cancel acknowledgment segment. An LTP engine delivers its data segment reception status to a source LTP engine in a report comprising one or more report segment(s) generated upon receipt of a checkpoint signal from the source LTP engine.

The LTP specification document describes the fundamental behavior of generating and handling report segments[4,5]. Multiple implementations of LTP have been developed to these specifications. However, the LTP specification is silent on some implementation details that can significantly impact operations.

An interoperability test was performed between two different LTP implementations to verify their correct operation and check their compatibility with the LTP specification document. Unanticipated report segment structure shapes that perfectly complied with the specification but introduced operational anomalies were observed during the test. Successful interoperation was achieved only after software modifications were applied. This paper presents the results of this interoperability testing, along with some implementation considerations derived on the basis of the results.

Ⅱ. Related Works

Some prominent LTP implementations are summarized in Table 1.

The most widely used LTP implementation is found in the Interplanetary Overlay Network[6,7] (ION) implementation developed by NASA's Jet Propulsion Laboratory. ION is available on an open-source license. ION is an extensive DTN software suite, which includes aspects such as implementations of LTP, BP, CCSDS File Delivery Protocol (CFDP), Asynchronous Message Service

Table 1. LTP reference implementations

Year	Name	Developer	Distribution	Characteristic
2006	LTP-RI	Ohio	OCP license	runs on Java 1.5 (site closed)
2006	LTPlib	TCD	Open source	Ported to various OS and DTN2 can operate this distribution
2007	ION	JPL	Open source	included in ION's DTN protocol suite supports multi-platform OS, operable by script supports contact graph routing, BER management, link speed management
2019	-	ESA	ESA license	written in Java, BP (including LTP implementation) Daemon demonstrated with OPS-SAT
2021	i3DTN	KARI	private	focusing to replace space link protocol to LTP operable by script supports table-based contact schedule, BER management, link speed management

(AMS), and Asynchronous Management Protocol (AMP). ION has many commanding and configuring options, which can be called by script. It also includes numerous test cases that help design interoperability tests. In addition to the LTP implementation, ION provides an LTP convergence-layer adapter for BP, which can be used for interoperability testing.

Ohio University had developed implementations of LTP and BP in Java.

European Space Agency (ESA) has developed BP and LTP implementations written in Java. This BP implementation worked as a Daemon and was demonstrated with the OPS-SAT ground test environment[8].

i3DTN, an implementation of DTN developed by the Korea Aerospace Research Institute, is internationally interoperable, interoperable with overlay networks, and interoperable with multiple platforms; BP is not yet included. The LTP implementation of i3DTN, LTPimpl[9], provides an interoperable LTP implementation and has been tested with ION. The driving force for developing i3DTN has been to study DTN as a flight mission standard. The LTP implementation of i3DTN is designed to be as lightweight and compact as possible to be used on the ground and in spacecraft while still performing the same essential functions as ION. In particular, the CCSDS Space Packet protocol can be integrated with LTP to provide a reliable communication service. i3DTN is written in C and was developed for the Linux operating system using the POSIX pthread library. It does not require any third-party library support; all functions are self-contained, including script functions.

Trinity College Dublin (TCD) developed an LTP implementation, LTPlib[10], which can be operated with the DTN2 implementation[11] of BP.

In 2015, an interoperability test between ION and a Python LTP library developed by MITRE was conducted in support of the verification of CCSDS LTP specification requirements[12].

In 2021, we performed an interoperability test[9] between ION and i3DTN. In the course of the test, we discovered that the report segments for a given LTP ARQ report could be generated in a variety of different patterns that all complied with the LTP protocol specification; a compliant LTP receiving engine needed to be able to handle all of these patterns to support interoperability. Such flexibility demanded careful design of the data structures used to manage the LTP session state. During testing, we observed that variations in the generation and (as needed) retransmission of report segments could significantly impact software performance, depending on the data structures used to manage the LTP session state.

Managing the LTP session state can be complex and challenging because any single LTP report may comprise multiple report segments (when link characteristics such as high bit-error rate, high packet-error rate, or intermittent connectivity result in high rates of data loss), and those report segments do not necessarily arrive in order. If report segments are lost and must be retransmitted, the order of arrival of the retransmitted segments is entirely unpredictable.

We performed a detailed investigation as to how LTP report segments may be generated and retransmitted. In light of this investigation, we herein propose some implementation practices, focusing on the details of generating and handling report segments so as to enhance LTP software design and development.

Ⅲ. Introduction to LTP report segment and Experimental setup

The basic description of overall LTP operation is provided by RFC-5326, the LTP specification document[4,13]. This section summarizes key aspects of LTP behavior that underlie the design considerations that are proposed subsequently. Revisiting aspects of the protocol other than the report segments is not within the scope of this study.

3.1 Basic operation of LTP report segment

The report segment transfers an accounting of the received block data to the sender of an LTP block; this serves as an implicit request for retransmission

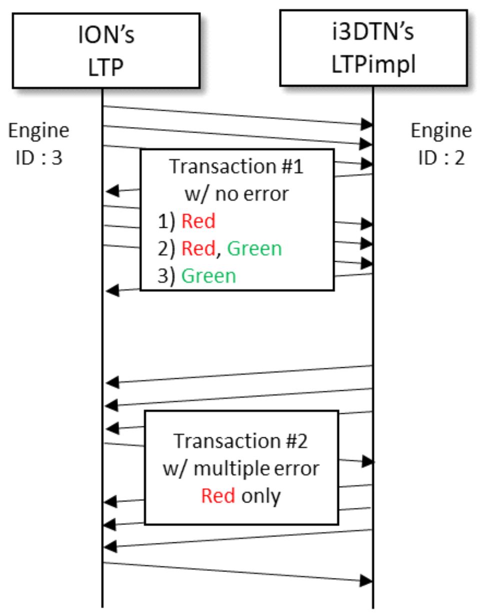


Fig. 2. Test sequences of the experiment

of the data not yet received. The detailed structure of the LTP report segment is depicted in Fig. 2. Reception of a data segment marked as a checkpoint is the event that causes transmission of a report comprising one or more report segments.

Upon reception of a report segment, the sending LTP engine must locate the unsuccessfully delivered data segments and retransmit those data segments. The data segments that must be transmitted are all segments for which

- the offset is no less than the report segment's lower bound;
- the extent, the sum of offset and length, is no more significant than the report segment's upper bound; and
- the offset and extent do not overlap with the offset and extent of any reception claim in the report segment.

There is no prescribed upper limit on the number of reception claims in a single segment. A report comprising 20 reception claims could be issued as a single report segment, two report segments, or 20 report segments.

Upon receiving a report segment, the sending LTP engine must retransmit the data segments for which reception was not claimed. The last retransmitted data segment must contain the report segment's serial number and must be marked as a "checkpoint" requesting transmission of a report on this retransmission.

3.2 Experimental setup for interoperability test

To gather various samples of report segments with varying numbers of reception claims, we performed an interoperability test of LTP between ION and i3DTN implementations and observed the recorded report segments via Wireshark[14]. ION can be configured to transmit and receive LTP segments containing data from BP or other LTP applications; an example of the latter can be found in the ION directory tests/ltp-retransmission. The ltpdriver utility for LTP testing in the DTN protocol suite of ION can be used for initiating an LTP transaction. The responding LTP engine can sleep to wait for an LTP transaction issued by the ltpdriver. The configuration of the interoperability test between ION and i3DTN is depicted in Fig. 3. LTPimpl was configured to emulate severe BER conditions, causing some LTP segments from i3DTN to ION to be discarded.

For test setup #1, ION's ltpdriver sent a block of 1,000,000 bytes three times with no BER, requiring ARQ for varying portions of the block: 1) red part

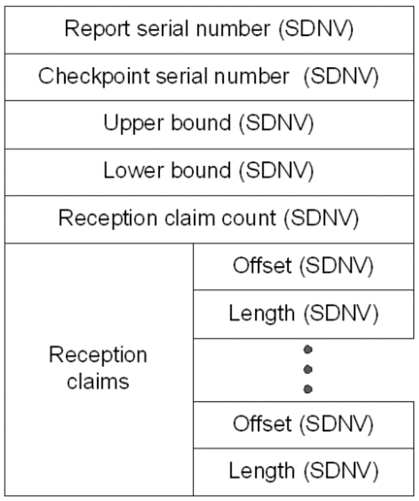


Table 2. Interoperability test configuration

Test	From	To	Remark
#1	ION	i3DTN	BER = 0
#2	i3DTN	ION	BER = 2×10^{-5}

only (i.e., with ARQ required for the entire block), 2) red part + green part, 3) green part only (i.e., no ARQ required at all).

For test setup #2, i3DTN transmitted 1,408,576 bytes to ION in all-red blocks, and the max size of each block was 150,000 bytes (1,200,000 bits). Max length of each LTP data segment was set to 1,500. The BER for emulating link error was set to 2×10^{-5}; as a result, more than 20 segments were lost per session transaction. The test configuration is summarized in Table 2.

Packet transmissions during the test were recorded via Wireshark, and each transaction was observed. Some transactions resulted in straightforward report segment exchanges, while others showed extremely complex report segment exchange patterns.

Upon observing the recorded activity, we identified several patterns of report segment exchange; every LTP engine needs to be able to handle all of those patterns properly so that it does not enter an unexpected control loop and wait an unbounded time, causing the transaction of the session to be aborted.

Notably, it is easy to set up a similar test environment without i3DTN. As performed in the previous interoperability test between i3DTN-LTPimpl and ION[9], ION can be configured to perform an LTP node-to-node test by itself.

Ⅳ. Experiment results and Considerations

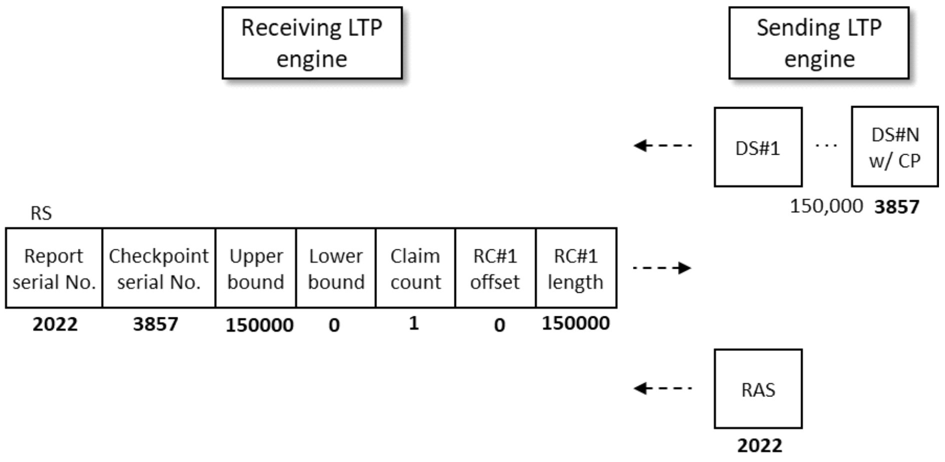
This section presents the results of the interoperability test performed between ION and i3DTN and the development of the LTP implementation design guidelines. Inferences obtained from the test results are also presented.

4.1 LTP transaction with BER 0

Test configuration #1 provides error-free conditions in the link between the LTP engines. When there is no link disruption (i.e., BER is 0), the operation of report segment generation is straightforward, and there are no potential deviations. Fig. 4 shows the simplest example of an LTP transaction in which no missing data segments exist. When all data segments are received successfully, the sole report segment has a single claim that claims the reception of all data in the block from offset 0 to the end.

4.2 LTP transaction with BER, not 0

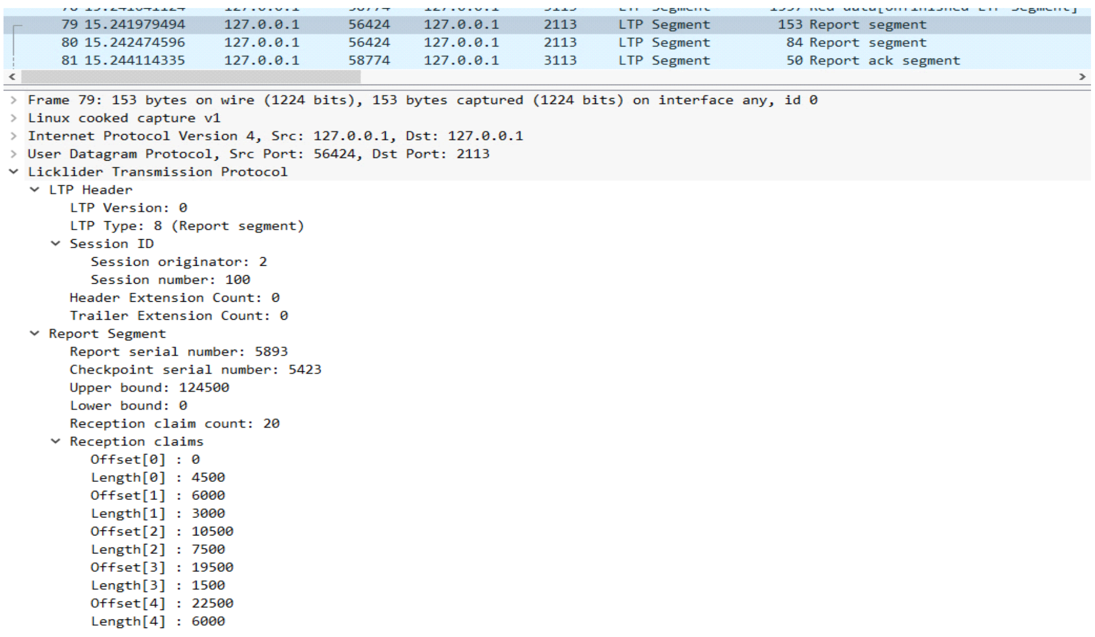
Test configuration #2 examines the link error status between nodes for an all-red-part transaction.



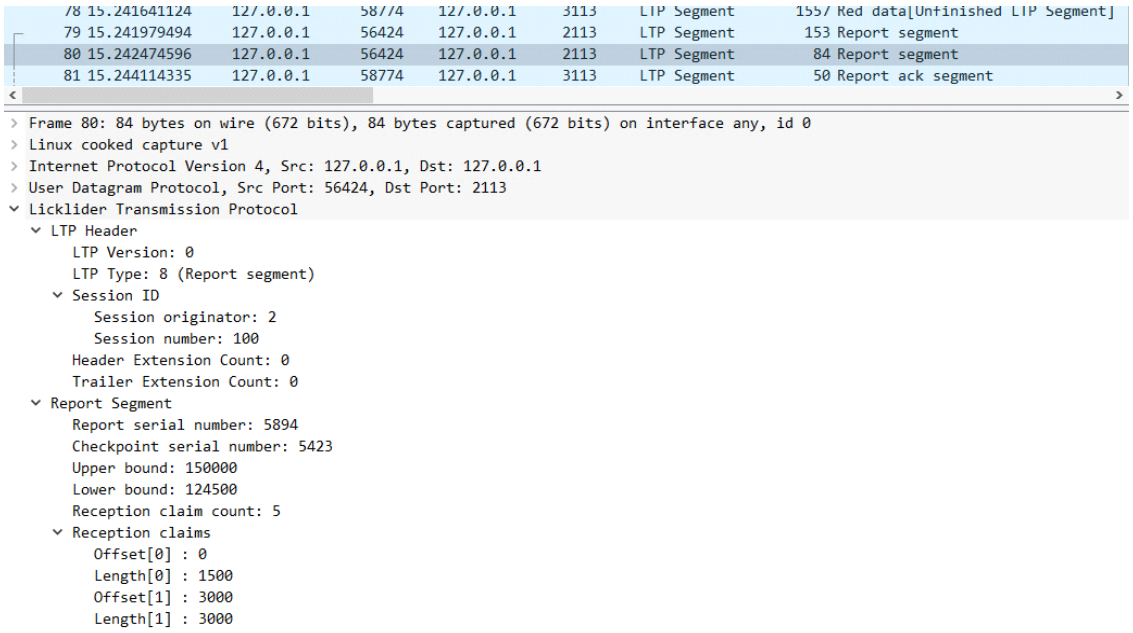
The packet loss rate is simulated according to the configured BER. In this test, BER was set to 2 × 10^{-5}; thus, a bit error occurs out of approximately every 50,000 bits. As a result, the affected data segment must be discarded at data-segment reception time. In this configuration, more than 20 data-segment losses occur. The test results are shown in Figs. 5-9. Fig. 5 shows an example of an LTP transaction when some initial data segments are lost, and the report segment claims 20 successful data receptions. As can be seen in Fig. 5 and Fig. 6, 26 reception claims were generated and delivered to the sender engine by two separate report segments. In contrast, another implementation might present these claims in a single segment or in more than the two report segments created in ION.

During the interoperability test, details of the LTP operations of ION were monitored to obtain samples of LTP raw transaction data; this aspect had not been thoroughly investigated previously. This investigation revealed that the LTP implementation of ION operates as follows when it encounters a retransmission request from a receiving LTP engine.

1) It retransmits in an original form all requested data segments except the last one. When it reaches the last data segment, it splits that segment into two parts, one containing all original data bytes except the last 1 and the other containing only the last original data byte. This ensures that the size of the last data segment (which must include the checkpoint and report serial numbers in addition to its data content) does not exceed the configured maximum data segment size. This practice requires the receiving LTP engine to handle the retransmission of a single original data segment that is split across two retransmission data segments. Modifying i3DTN to meet this requirement is time-consuming, and the procedure is especially complex when one of the two retransmission data segments is lost during transmission. A sending LTP engine must remember all block reception history until the current block transmission session completes successfully. This function must be reflected in the session state data structure design and development. A proposed design approach is described in Section 5.

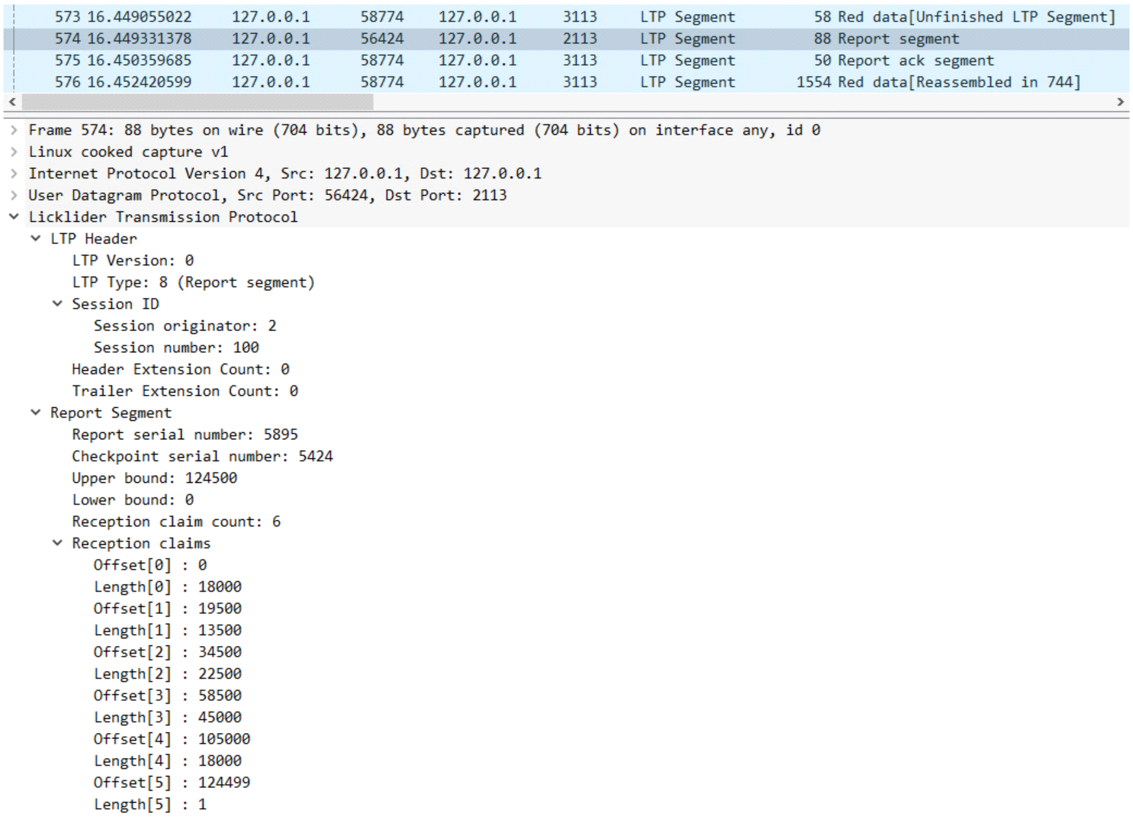


```
79 15.241979494   127.0.0.1   56424   127.0.0.1   2113   LTP Segment    153 Report segment
80 15.242474596   127.0.0.1   56424   127.0.0.1   2113   LTP Segment     84 Report segment
81 15.244114335   127.0.0.1   58774   127.0.0.1   3113   LTP Segment     50 Report ack segment

> Frame 79: 153 bytes on wire (1224 bits), 153 bytes captured (1224 bits) on interface any, id 0
> Linux cooked capture v1
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 56424, Dst Port: 2113
v Licklider Transmission Protocol
  v LTP Header
      LTP Version: 0
      LTP Type: 8 (Report segment)
    v Session ID
        Session originator: 2
        Session number: 100
      Header Extension Count: 0
      Trailer Extension Count: 0
  v Report Segment
      Report serial number: 5893
      Checkpoint serial number: 5423
      Upper bound: 124500
      Lower bound: 0
      Reception claim count: 20
    v Reception claims
        Offset[0] : 0
        Length[0] : 4500
        Offset[1] : 6000
        Length[1] : 3000
        Offset[2] : 10500
        Length[2] : 7500
        Offset[3] : 19500
        Length[3] : 1500
        Offset[4] : 22500
        Length[4] : 6000
```



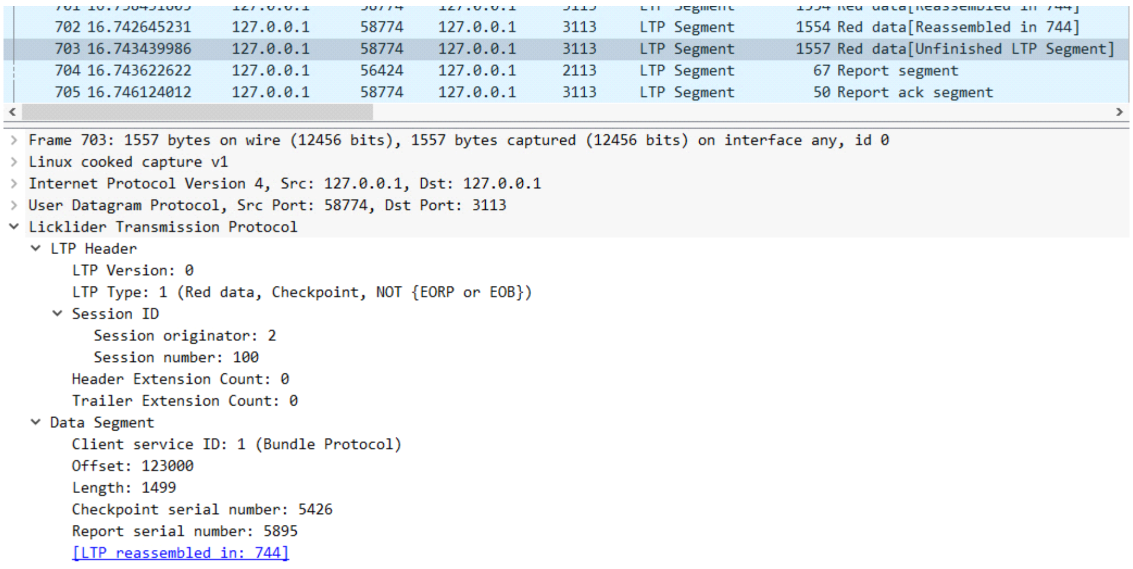
```
   78 15.241641124   127.0.0.1   58774   127.0.0.1   3113   LTP Segment   1557 Red data[Unfinished LTP Segment]
   79 15.241979494   127.0.0.1   56424   127.0.0.1   2113   LTP Segment    153 Report segment
   80 15.242474596   127.0.0.1   56424   127.0.0.1   2113   LTP Segment     84 Report segment
   81 15.244114335   127.0.0.1   58774   127.0.0.1   3113   LTP Segment     50 Report ack segment

> Frame 80: 84 bytes on wire (672 bits), 84 bytes captured (672 bits) on interface any, id 0
> Linux cooked capture v1
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 56424, Dst Port: 2113
v Licklider Transmission Protocol
  v LTP Header
      LTP Version: 0
      LTP Type: 8 (Report segment)
    v Session ID
        Session originator: 2
        Session number: 100
      Header Extension Count: 0
      Trailer Extension Count: 0
  v Report Segment
      Report serial number: 5894
      Checkpoint serial number: 5423
      Upper bound: 150000
      Lower bound: 124500
      Reception claim count: 5
    v Reception claims
        Offset[0] : 0
        Length[0] : 1500
        Offset[1] : 3000
        Length[1] : 3000
```

Fig. 6. Second report segment to session number 100, which claims 5 items



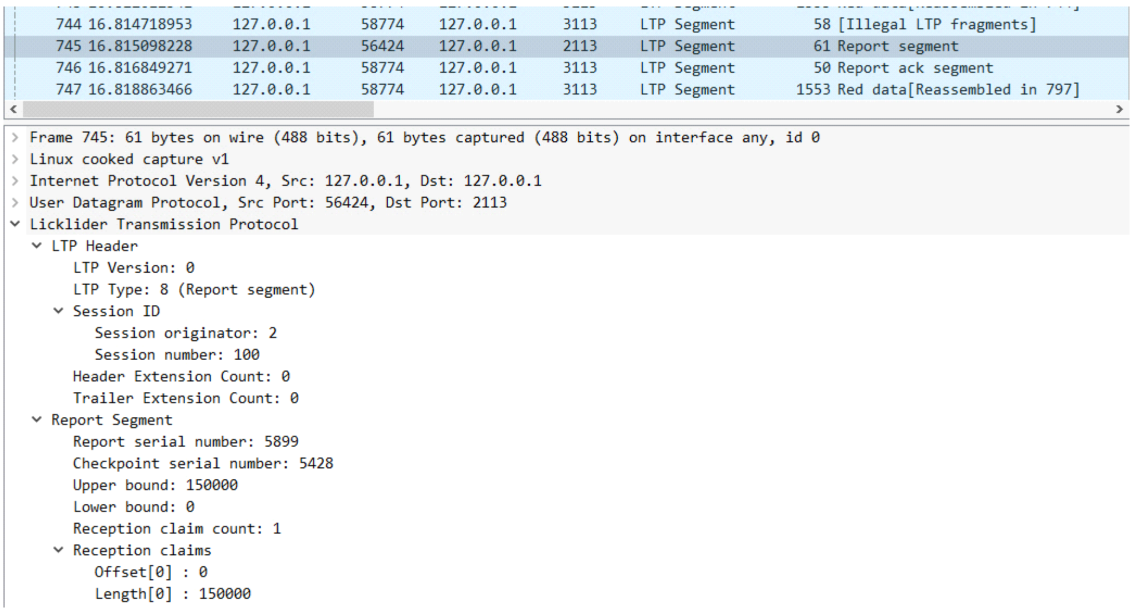
```
  573 16.449055022   127.0.0.1   58774   127.0.0.1   3113   LTP Segment     58 Red data[Unfinished LTP Segment]
  574 16.449331378   127.0.0.1   56424   127.0.0.1   2113   LTP Segment     88 Report segment
  575 16.450359685   127.0.0.1   58774   127.0.0.1   3113   LTP Segment     50 Report ack segment
  576 16.452420599   127.0.0.1   58774   127.0.0.1   3113   LTP Segment   1554 Red data[Reassembled in 744]

> Frame 574: 88 bytes on wire (704 bits), 88 bytes captured (704 bits) on interface any, id 0
> Linux cooked capture v1
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 56424, Dst Port: 2113
v Licklider Transmission Protocol
  v LTP Header
      LTP Version: 0
      LTP Type: 8 (Report segment)
    v Session ID
        Session originator: 2
        Session number: 100
      Header Extension Count: 0
      Trailer Extension Count: 0
  v Report Segment
      Report serial number: 5895
      Checkpoint serial number: 5424
      Upper bound: 124500
      Lower bound: 0
      Reception claim count: 6
    v Reception claims
        Offset[0] : 0
        Length[0] : 18000
        Offset[1] : 19500
        Length[1] : 13500
        Offset[2] : 34500
        Length[2] : 22500
        Offset[3] : 58500
        Length[3] : 45000
        Offset[4] : 105000
        Length[4] : 18000
        Offset[5] : 124499
        Length[5] : 1
```

Fig. 7. One of mid report segment to session number 100 which claims 6 items



```
702 16.742645231   127.0.0.1    58774   127.0.0.1    3113   LTP Segment    1554 Red data[Reassembled in 744]
703 16.743439986   127.0.0.1    58774   127.0.0.1    3113   LTP Segment    1557 Red data[Unfinished LTP Segment]
704 16.743622622   127.0.0.1    56424   127.0.0.1    2113   LTP Segment      67 Report segment
705 16.746124012   127.0.0.1    58774   127.0.0.1    3113   LTP Segment      50 Report ack segment

> Frame 703: 1557 bytes on wire (12456 bits), 1557 bytes captured (12456 bits) on interface any, id 0
> Linux cooked capture v1
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 58774, Dst Port: 3113
v Licklider Transmission Protocol
  v LTP Header
      LTP Version: 0
      LTP Type: 1 (Red data, Checkpoint, NOT {EORP or EOB})
    v Session ID
        Session originator: 2
        Session number: 100
      Header Extension Count: 0
      Trailer Extension Count: 0
  v Data Segment
      Client service ID: 1 (Bundle Protocol)
      Offset: 123000
      Length: 1499
      Checkpoint serial number: 5426
      Report serial number: 5895
      [LTP reassembled in: 744]
```

Fig. 8. One of mid data segment retransmission for offset(123000) and length(1499)



```
744 16.814718953   127.0.0.1    58774   127.0.0.1    3113   LTP Segment      58 [Illegal LTP fragments]
745 16.815098228   127.0.0.1    56424   127.0.0.1    2113   LTP Segment      61 Report segment
746 16.816849271   127.0.0.1    58774   127.0.0.1    3113   LTP Segment      50 Report ack segment
747 16.818863466   127.0.0.1    58774   127.0.0.1    3113   LTP Segment    1553 Red data[Reassembled in 797]

> Frame 745: 61 bytes on wire (488 bits), 61 bytes captured (488 bits) on interface any, id 0
> Linux cooked capture v1
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 56424, Dst Port: 2113
v Licklider Transmission Protocol
  v LTP Header
      LTP Version: 0
      LTP Type: 8 (Report segment)
    v Session ID
        Session originator: 2
        Session number: 100
      Header Extension Count: 0
      Trailer Extension Count: 0
  v Report Segment
      Report serial number: 5899
      Checkpoint serial number: 5428
      Upper bound: 150000
      Lower bound: 0
      Reception claim count: 1
    v Reception claims
        Offset[0] : 0
        Length[0] : 150000
```

Fig. 9. Final report segment to session number 100 which claims fully successful delivery

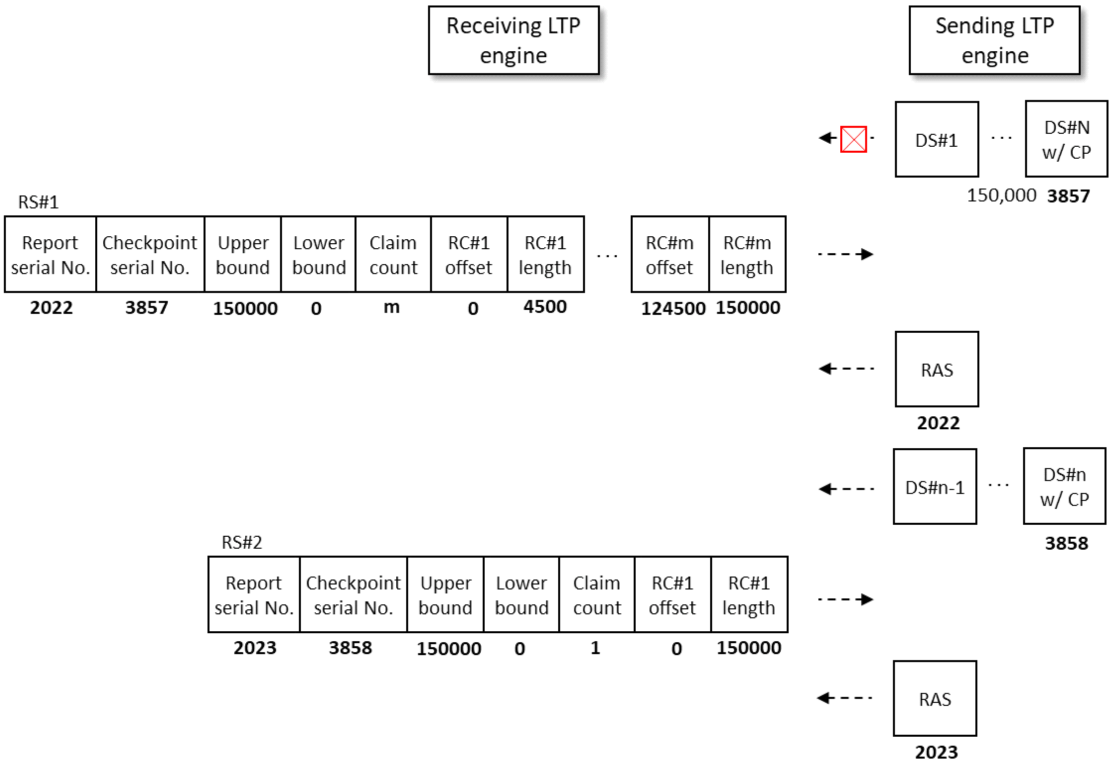
2) ION holds a maximum of 20 claims in a report segment. Multiple report segments must be generated if more than 20 claims are needed for the report that must be issued on the reception of a given checkpoint. Handling a report comprising multiple report segments is more complex than handling a report comprising a single report segment because the sending LTP engine must assess the overall completion status of the session when handling each report segment individually. Reception of multiple report segments in a session results in multiple retransmissions, each terminating in a distinct checkpoint; the sending LTP engine must anticipate potential data losses in each of those retransmissions.

Moreover, it is always possible for a checkpoint data segment to be lost. When this occurs (as signaled by the expiration of the checkpoint's retry timer), the sending LTP engine must retransmit the checkpoint and handle the responding report, possibly resulting in further retransmission. An important insight gained from this interoperability testing was that a sending LTP engine may need to conduct retransmission activity for multiple LTP block transmission sessions concurrently.

3) A receiving LTP engine is likewise required to handle out-of-order and potentially split data segments from a sending LTP engine. An LTP engine must be able to concatenate all segment data, whether positionally adjacent or misaligned, into a single aligned data block. This single block may be virtual, comprising multiple linked storage elements; however, for performance reasons, it is desirable that the storage element(s) be as large as possible. Until a session completes, management of reception claims information is necessary at both the sending and receiving LTP engines. In particular, it is required in order to generate report segments from a receiving LTP engine. ION uses a red/black tree algorithm to manage the ordered, aligned data segments of each LTP block.

Ⅴ. Full anatomy of LTP report segment and proposing handling practice

This section presents a practical mechanism for LTP report segment handling that enables the sending LTP engine to cope with various report segment reception patterns. A simple example of report segment handling in response to some link disruptions during the transaction is shown in Fig. 10. This can be the simplest LTP transaction under link disruption conditions because a single report



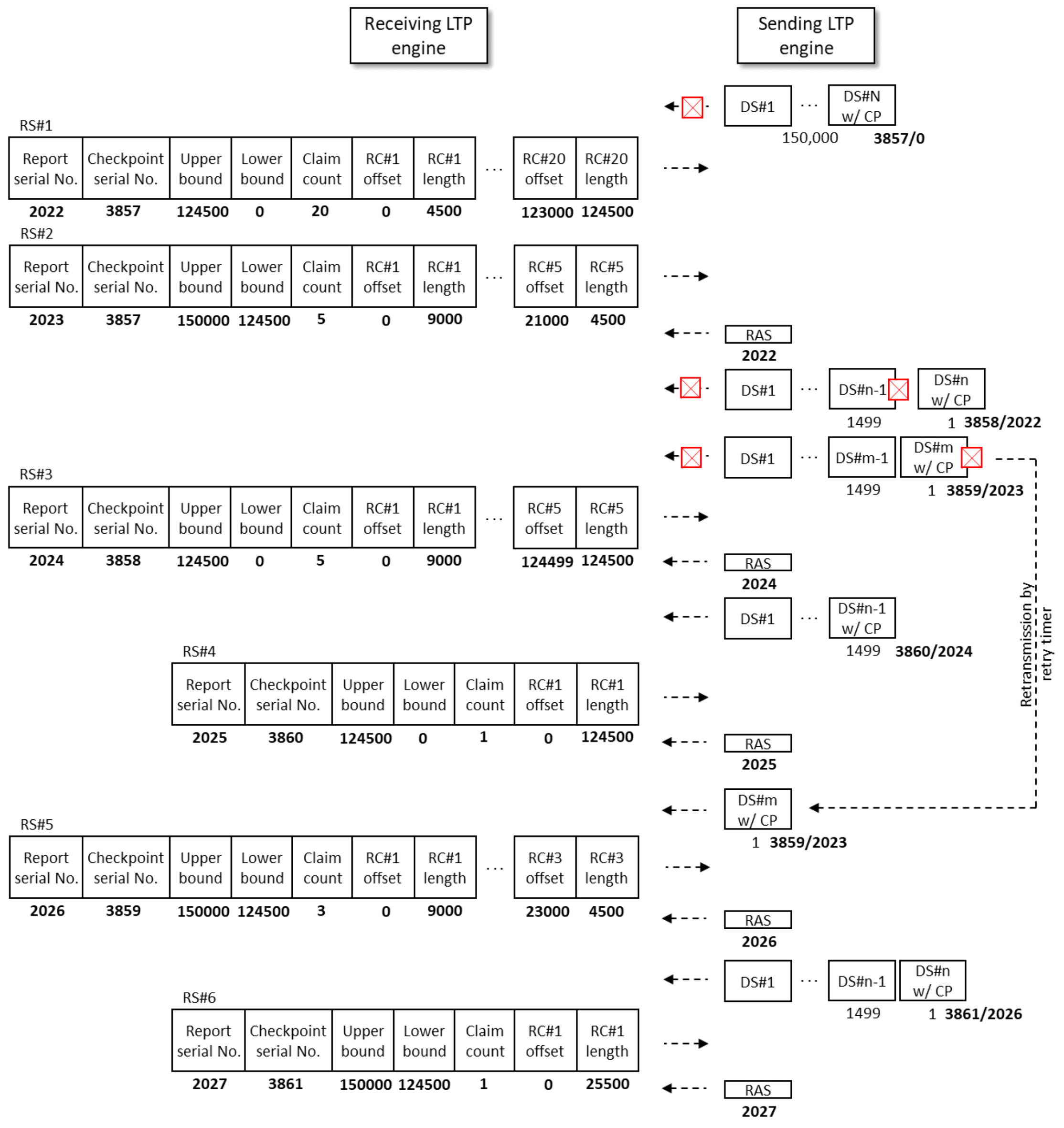

Receiving LTP engine
Sending LTP engine
DS#1 ··· DS#N w/ CP
150,000 3857/0
RS#1
Report serial No. | Checkpoint serial No. | Upper bound | Lower bound | Claim count | RC#1 offset | RC#1 length | ··· | RC#20 offset | RC#20 length
2022 | 3857 | 124500 | 0 | 20 | 0 | 4500 | | 123000 | 124500
RS#2
Report serial No. | Checkpoint serial No. | Upper bound | Lower bound | Claim count | RC#1 offset | RC#1 length | ··· | RC#5 offset | RC#5 length
2023 | 3857 | 150000 | 124500 | 5 | 0 | 9000 | | 21000 | 4500
RAS 2022
DS#1 ··· DS#n-1 DS#n w/ CP
1499 1 3858/2022
DS#1 ··· DS#m-1 DS#m w/ CP
1499 1 3859/2023
RS#3
Report serial No. | Checkpoint serial No. | Upper bound | Lower bound | Claim count | RC#1 offset | RC#1 length | ··· | RC#5 offset | RC#5 length
2024 | 3858 | 124500 | 0 | 5 | 0 | 9000 | | 124499 | 124500
RAS 2024
DS#1 ··· DS#n-1 w/ CP
1499 3860/2024
Retransmission by retry timer
RS#4
Report serial No. | Checkpoint serial No. | Upper bound | Lower bound | Claim count | RC#1 offset | RC#1 length
2025 | 3860 | 124500 | 0 | 1 | 0 | 124500
RAS 2025
DS#m w/ CP
1 3859/2023
RS#5
Report serial No. | Checkpoint serial No. | Upper bound | Lower bound | Claim count | RC#1 offset | RC#1 length | ··· | RC#3 offset | RC#3 length
2026 | 3859 | 150000 | 124500 | 3 | 0 | 9000 | | 23000 | 4500
RAS 2026
DS#1 ··· DS#n-1 DS#n w/ CP
1499 1 3861/2026
RS#6
Report serial No. | Checkpoint serial No. | Upper bound | Lower bound | Claim count | RC#1 offset | RC#1 length
2027 | 3861 | 150000 | 124500 | 1 | 0 | 25500
RAS 2027

Fig. 11. A more complex example of report segment with BER and multiple retransmissions

segment is generated rather than multiple report segments and a single retransmission of the segments in that report's single claim is enough. Multiple reception claims are not needed. However, generally, more complicated patterns may be expected as shown in Fig. 11, as seen from this interoperability test results. Owing to multiple data-segment disruptions, multiple report segments are issued. Therefore, a sender engine must be able to handle the separated report segments and respond to these separately. As shown in Fig. 11, a receiver engine can choose multiple successful report segments when issuing successful reception claims for the retransmitted data segments rather than issuing a single report segment that claims the whole area directly; this may be the simplest form for processing by a sender engine.

5.1 Flexible operation of SDR

We herein refer to the LTP data storage

management mechanism as a “simple data recorder” (SDR). In a given implementation, the SDR abstraction may entirely take the form of a database, file, shared memory region, or another storage medium.

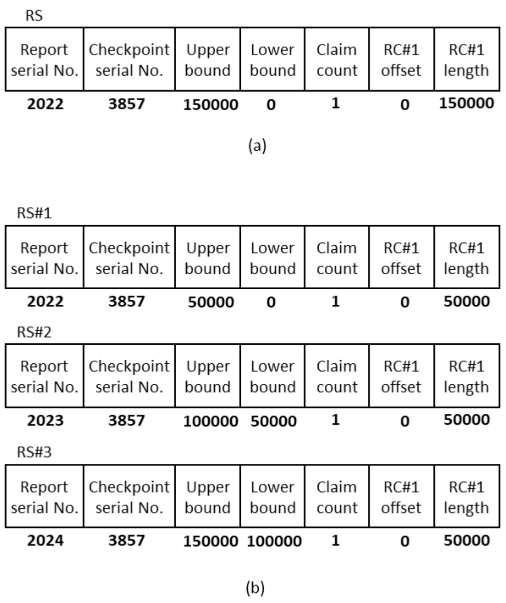
Data are stored in a unit of SDR storage. An SDR element is allocated to store a data segment when it is issued at a sending LTP engine, or it arrives at a receiving LTP engine. We may consider each SDR element as being configured to have a fixed capacity as determined by a current network environment characteristic such as maximum transmission unit. As described in Section 4, partial data in an original segmentation can be delivered separately. For example, 1500 bytes in an original data segment can be split into two segments containing 1499 bytes of original data and 1 byte of original data when retransmitted. When a receiving LTP engine receives two separate retransmitted data segments, it is recommended to reassemble the two separate data segments into a single data segment in an SDR element. This enables the antecedent SDR elements to be freed for resource reuse.

An interesting scenario occurs when these separate but positionally adjacent data segments arrive out-of-order, as reassembly of the original data segment cannot be performed until both fractional segments have arrived.

5.2 Decision of delivery completion status

When a receiving LTP engine receives all of the data segments from a sending LTP engine, usually the receiving LTP engine issues a report segment whose sole reception cites a single reception whose offset is 0 and whose length is the size of the block as indicated by the end of the block (EOB) flag in the ending checkpoint.

However, this is not the only acceptable approach for issuing a fully successful reception claim. A report of complete block reception can be split into multiple report segments, as shown in Fig. 12. Report segment(s) (a) and (b) in Fig. 12 both claim successful reception of all data from 0 to 150,000. However, the technical readiness level for preparing those two cases is much different. Case (b) requires



RS

Report serial No.	Checkpoint serial No.	Upper bound	Lower bound	Claim count	RC#1 offset	RC#1 length
2022	3857	150000	0	1	0	150000

(a)

RS#1

Report serial No.	Checkpoint serial No.	Upper bound	Lower bound	Claim count	RC#1 offset	RC#1 length
2022	3857	50000	0	1	0	50000

RS#2

Report serial No.	Checkpoint serial No.	Upper bound	Lower bound	Claim count	RC#1 offset	RC#1 length
2023	3857	100000	50000	1	0	50000

RS#3

Report serial No.	Checkpoint serial No.	Upper bound	Lower bound	Claim count	RC#1 offset	RC#1 length
2024	3857	150000	100000	1	0	50000

(b)

Fig. 12. Various ways to make full reception claim

sophisticated functionality to interpret and respond to the separated and partitioned report information. A sending LTP engine must be able to deduce from multiple full receptions the reception of the entire block.

Loss of a checkpoint data segment may present the most demanding retransmission problem. For example, a report segment has to be configured when the first and last data segments are lost during the first transaction try, as depicted in Fig. 13. The loss of the first or last data segment can present a challenging situation for processing requesting retransmission. Considering the case of loss of the last checkpoint data-segment block is beyond the scope of this study. However, it will be an interesting topic for future investigation.

When the arrival of the retransmitted last (checkpoint) segment causes the receiving engine to issue a report noting the successful reception of all segments of the block except the very first, the sending engine can reason that the successful retransmission of solely that first segment will enable all retransmission resources consumed by this block to be released immediately and that retransmission is therefore urgent. As the report

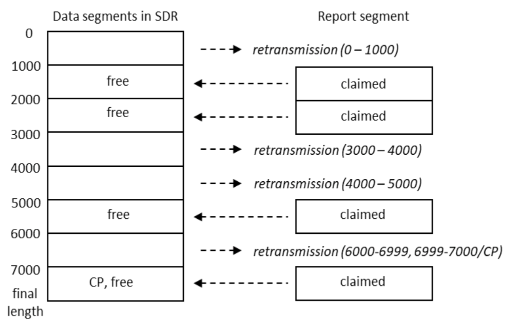


Fig. 13. A most sophisticated case in a report segment

segment of LTP comprises only positive acknowledgments, an intermediate report segment can be configured with a lower bound of either 0 or 1000 when indicating the loss of data segment from 0 to 1000, as clarified by the claims in the report. As (final length, 0) and (final length, 1000) are both legitimate expressions according to the LTP specification document, knowing the missing unclaimed region before reception of a subsequent report segment that explicitly identifies the missing region can be beneficial to a sending engine by reducing delay in the initiation of the necessary retransmission.

An alternative mechanism for improving efficiency in SDR utilization is for the sending engine to free all the data segments claimed by report segments immediately upon report segment reception. When all data segments are fully delivered, and all reception has been claimed, all SDR storage allocated to the block would have been removed; this would inform the sending LTP engine that all data segments have been delivered successfully. Thereafter, the sending LTP engine can close the session. This scheme is beneficial when the capacity of the SDR is limited. An LTP engine can remove all claimed data segments at the earliest possible time, making it easier to locate unclaimed data segments.

Ⅵ. Conclusion

This paper presents the results of LTP interoperability testing between ION and i3DTN and the insights gained from that testing. Error-free and errored conditions were configured to monitor and thoroughly analyze LTP segment exchange between i3DTN and ION. This test revealed that there can be significant variation in the patterns in which report segments are generated, and lost data segments are retransmitted. We concluded that it is challenging to process report segments in these varying patterns as enacted by different LTP implementations. Whenever a new LTP implementation is available, it is undoubtedly recommended to conduct interoperability tests to verify its compatibility with how other implementations handle these reporting and retransmission patterns.

LTP is vital in DTN technology and DTN protocol software suite, providing a reliable convergence layer between the link layer and BP. LTP session state management is key to the success of the protocol. Considering the complexity of report and retransmission handling at the beginning of SDR design can render interoperability testing considerably easy and ultimately reduce costs.

The test configuration constructed for this study was relatively simple; when the interplanetary network becomes larger and more complex and there may be multiple paths between the source and destination, duplicated, truncated, or overlapping LTP segments may arrive in parallel. Those challenging segmentation variations can introduce operational confusion including errors in analysis and evaluation of the segmentation data. Further investigation is warranted.

References

[1] F. Warthman, "*Delay Tolerant Networks (DTNs) A Tutorial*," DTN Research Group Internet Draft–2003, 2003.

[2] A. G. Voyiatzis, "A survey of delay-and disruption-tolerant networking applications," *J. Internet Eng.*, vol. 5, no. 1, pp. 331-344, 2012.

[3] CCSDS, "*CCSDS File Delivery Protocol (CFDP) - Draft Recommended Standard*," CCSDS 727.0-P-4.1, Pink Book, Tech. Rep., 2014.

[4] M. Ramadas, S. C. Burleigh, and S. Farrell, "*RFC 5326, Licklider Transmission Protocol Specification*," IRTF DTN Research Group, 2008. [Online] Available: https://tools.ietf.org/html/rfc5326

[5] CCSDS, "*Licklider Transmission Protocol (LTP) for CCSDS*," CCDS 734.1-B-1, Blue Book, Tech. Rep., 2015.

[6] S. Burleigh, "Interplanetary overlay network: An implementation of the DTN bundle protocol," in *4th IEEE CCNC 2007*, pp. 222-226, 2007. (http://dx.doi.org/10.1109/CCNC.2007.51)

[7] NASA, "*Interplanetary Overlay Network (ION) Design and Operation*," ver 3.7.2, JPL D-48259, Retrieved from https://sourceforge.net/projects/ion-dtn/

[8] F. Flentge, *ESA Operational Ground CFDP & DTN Implementations*(2019), Retrieved from https://indico.esa.int/event/323/contributions/5030/attachments/3729/5182/12.25a_-_DTN__CFDP.pdf

[9] C. H. Koo, "An implementation of LTP protocol opening a gate to space communication network from space packet communication," *J. KICS*, vol. 46, no. 12, pp. 2134-2143, 2021. (http://dx.doi.org/10.7840/kics.2021.46.12.2134)

[10] *TCD's LTPlib*, https://down.dsg.cs.tcd.ie/ltplib/

[11] *DTN2*, https://github.com/delay-tolerant-networking/DTN2

[12] CCSDS, *Licklider Transmission Protocol (LTP) for CCSDS Interoperability Test Report*, 2015. [Online] Available: https://cwe.ccsds.org/cesg/docs/Forms/DispForm.aspx?ID=2664

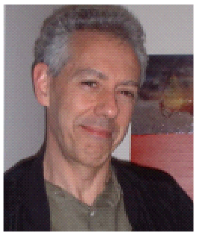
[13] N. Ansell, *Delay and Disruption Tolerant Networks ; Interplanetary and Earth-Bound – Architecture, Protocols, and Applications*, Ch 11. Licklider Transmission Protocol (LTP), CRC Press, pp. 345-400, 2019. (http://dx.doi.org/10.1201/9781315271156)

[14] *Wireshark*, www.wireshark.org

구 철 회 (Cheol Hea Koo)

1997년 2월 : 충남대학교 전자공학과 졸업
1999년 2월 : 충남대학교 의용전자공학 석사
2021년 2월 : 충남대학교 컴퓨터공학과 박사
2002년 3월~현재 : 한국항공우주연구원 책임연구원
<관심분야> 내장형 소프트웨어, 위성 통신, 심우주 통신, CFDP, 우주인터넷
[ORCID:0000-0002-7180-1476]

스 캇 (Scott C. Burleigh)



(전)Caltech/JPL Principal Engineer
현재 : IPNSIG vice Chair
<관심분야> 태양계인터넷, 우주인터넷, CFDP, LTP, BP
[ORCID:0000-0003-3768-5413]