ALMA MATER STUDIORUM - UNIVERSITY OF BOLOGNA

SCHOOL OF ENGINEERING AND ARCHITECTURE

Department of Computer Science and Engineering

Bachelor's degree in Computer Engineering

BACHELOR'S THESIS
in
Fundamentals of Telecommunications T

Analysis of ESA Bundle Protocol and LTP implementations

Candidate:
Martino Manaresi

Supervisor:
Prof. **Carlo Caini**
Co-supervisor:
Camillo Malnati

Academic Year 2023/2024

"Exploration is in our nature. We began as wanderers,
and we are wanderers still. We have lingered long
enough on the shores of the cosmic ocean. We are
ready at last to set sail for the stars"
Carl Sagan, Cosmos

ABSTRACT

.

The advent of the space age demands robust communication infrastructures to support operations in the harsh interplanetary space environment. The software infrastructure and network architecture required for space missions have been studied extensively since the 2000s. This research led to the development of the Delay/Disruption Tolerant Networking (DTN) architecture, with the Bundle Protocol (BP) at its core, and to associated new protocols, such as the Licklider Transmission Protocol (LTP). The current version of the Bundle Protocol is version 7, standardized in RFC 9171 and currently in the final phases of standardization by the Consultative Committee for Space Data Systems (CCSDS), an international organization composed of major space agencies specifically focused on space communication development. The objective of this thesis was to collaborate with the European Space Agency (ESA) to analyze and test its new implementations of BP and LTP. This software was made available to us by ESA developers at thesis start, thus the tests described in this thesis were most likely the first performed outside ESA. The aim of the analysis was to understand the general workings of ESA's implementations and to test their adherence to the standard. The analysis included and extended a range of interoperability tests with ION, one of the reference BP implementations; tests were later extended to other BP implementations, such as Unibo-BP, DTNME and μDTN. The test campaign not only provided ESA developers with independent feedback, including the discovery of few bugs, but also revealed some unexpected problems in well-established implementations. This highlights once more the necessity for independent interoperability testing, crucial to ensure the resilience required by critical infrastructures such as future space communication systems.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 CHALLENGED NETWORKS

A network is called "Challenged" if one or more of the following requirements is not met:

1. End-to-end connectivity, there must be a continuous path in space and time between sender and receiver.
2. Small RTT(Round Trip Time), so that intermediate nodes do not need to store copies of packets, in case of loss the sender will send a copy.
3. Packet losses due to errors introduced by the channel must be negligible and losses are mainly due to network congestion.

The stack TCP/IP is not suited for challenged networks as the requirements described above are necessary for its operations. Example of networks that do not meet these requirements are: space, military and submarine communications, ad-hoc networks of embedded systems and networks affected by natural disasters. A new architecture known as DTN (Delay-/Disruption-Tolerant Networking) was thus designed to overcome all mentioned challenges [RFC4838].

1.2 THE DTN ARCHITECTURE

In the 2000s the research on high delay networks, such as InterPlanetary Networks(IPN), was aggregated with the research on other types of challenged networks. The research result was the development of the DTN architecture. This architecture introduces a new layer above Transport and below Application called Bundle Layer. The nodes that support the Bundle Layer are called DTN Nodes or Bundle Nodes, the messages exchanged by these nodes are called Bundles and are exchanged through the Bundle Protocol (BP). DTN nodes implement a Store-and-Forward policy: Bundles are stored in secondary memory as an arbitrary amount of time can pass before the transmission link for the next hop is available. Between two DTN nodes it is possible to have multiple nodes that do not support the Bundle Layer.

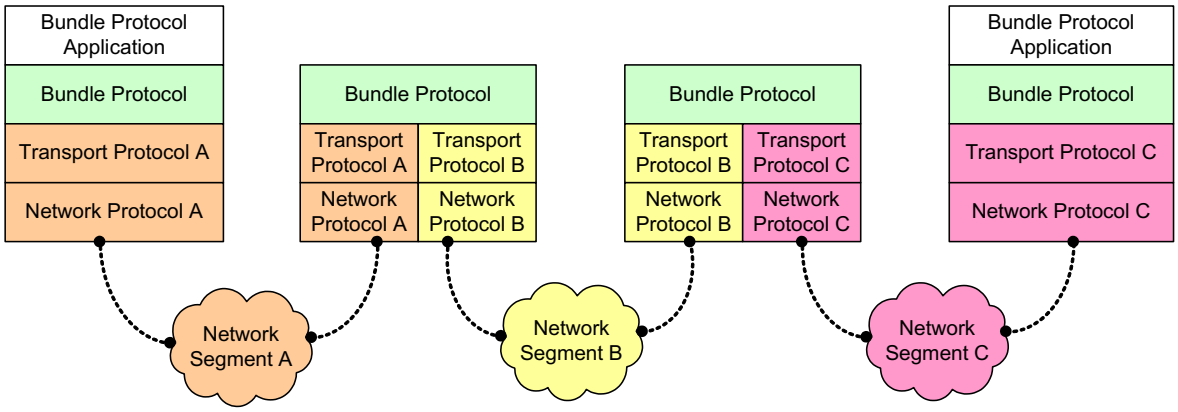

Figure 1.1 - The DTN Architecture

In fact, if the network segment, as depicted by the cloud in Figure 1.1, connecting two DTN nodes is the Internet network, it can indeed be made of basic Internet routers. BP version 6 was standardized by the Internet Research Task Force (IRTF) in 2007 in an experimental RFC [RFC5050] (issued in 2007). Version 7 was formalized in 2022 by the Internet Engineering Task Force (IETF) as a Proposed Standard RFC [RFC9171]. In parallel, the BP has been standardized also by Consultative Committee for Space Data System (CCSDS), a standardization body consisting of all major space Agencies, with a "Blue book" (standard document in CCSDS jargon) [CCSDS_BPV6]

1.3 CONVERGENCE LAYER

Convergence Layer is the name given to the entire protocol stack below the Bundle Layer while Convergence Layer Adapter (CLA) are the protocols located between the Bundle Layer and the Transport Layer, whose task is to abstract the differences and characteristics of the various transport protocols to present a consistent interface to the Bundle Layer. The most important adapters are the TCP Convergence Layer and LTP Convergence Layer (it is to be noted that although these two protocols are CLA their name lacks the "adapter" attribute, a nomenclature inconsistency)

The TCP Convergence Layer (TCPCL) comes in two versions: Version 3 [RFC7242], which is the most widely used and Version 4 [RFC9174]. These versions are not compatible. TCPCL is one of the CLAs that allow the BP to communicate via Internet.

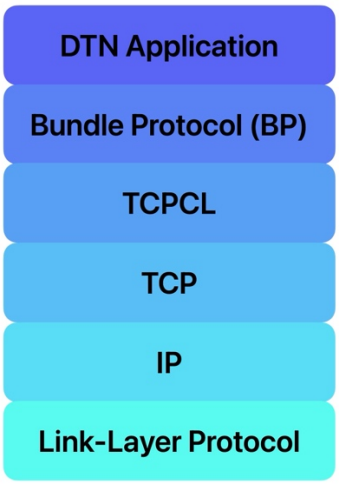

Figure 1.2 - A possible protocol stack using TCPCL.

LTPCL is the CLA that enables the BP to use the LTP(Licklider Transmission Protocol) [RFC5325][RFC5326][CCSDS_LTP], a protocol specifically engineered to be part of Converge Layer, since it is designed for extremely long RTTs and frequent interruption of connectivity, typical conditions of interplanetary communications.

2 BUNDLE PROTOCOL VERSION 7

The bundle protocol version 7, or BPv7 in short, is defined in [RFC9171]. Below will be recalled only the basic elements necessary to understand the work done in this thesis.

2.1 BUNDLE NODE

A bundle node is a fundamental unit in the bundle protocol architecture, capable of sending and receiving bundles. It consists of three primary components:

- BPA (Bundle Protocol Agent): responsible for executing the BP and managing the node's bundle processing operations.
- AA (Application Agent): composed by the Administrative Element and the Application-Specific Element. The first one handles administrative records, standard application data units used to implement some of the features of the BP, like bundle status reports. The second one handles application data units of the Application Layer.
- CLA: Optional components that enable the BPA to communicate with other nodes over various transport protocols or networks. A bundle node may have zero or more CLAs, depending on its communication requirements.

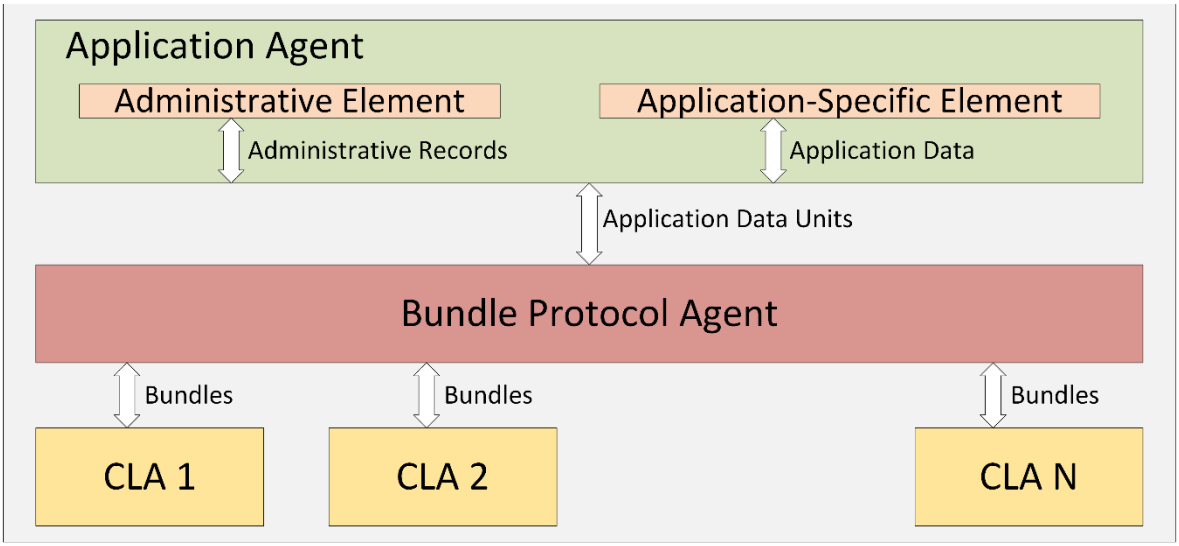

Figure 2.1 - Bundle Node

2.2 ENDPOINT ID

EID(Endpoint ID) is a name that conforms to the URI (Uniform Resource Identifier) [RFC3986] syntax and identifies a bundle endpoint, the bundle's source or destination. A bundle endpoint is a set of zero or more nodes; if the EID identifies a single bundle node it is called "Singleton". Each bundle node requires a singleton EID that uniquely identifies it, which is called Node ID.

The URI syntax, which is used by EIDs, is the following:

```
<scheme name>:<SSP(Scheme Specific Part)>
```

Each URI scheme is a set of semantic rules that specify how to parse the SSP. An example of URI is the URLs family used by the Web, e.g. http://unibo.it. The bundle protocol for historical reasons uses two URI schemes, "dtn" and "ipn".

2.2.1 Scheme “dtn”

The dtn scheme uses character strings that allow an arbitrary grade of expressivity. The SSP is divided in two parts: the *node name* and the *demux token*. All the demux token of singleton EIDs either have length zero or do NOT start with ~ (tilde), Node ID must have an empty demux token.

```
dtn://<node name>/<demux token>
```

The EID "*dtn:none*" represents the null EID in both the dtn and the ipn schemes.

2.2.2 Scheme “ipn”

The ipn scheme uses unsigned integers, for a compact binary representation. The SSP is divided in two parts: the *node number* and the *service number*. All ipn EIDs are singleton. The Node ID must have service number 0.

```
ipn:<node number>.<service number>
```

2.3 BUNDLE FORMAT

A bundle comprises a Primary Block, a Payload Block, and optionally one or more Extension Blocks. BPv7 bundles are serialized using the CBOR (Concise Binary Object

Representation) [RFC_8949]. The Primary Block, which represents the bundle header, has a different structure from the other blocks. It contains:

- The Bundle Protocol version.
- The Bundle Processing Control Flags, which indicate properties that apply to the entire bundle rather than to specific blocks.
- The CRC (Cyclic Redundancy Check) type: the possible values are NO-CRC, X25 CRC-16 [CRC_16], CRC32C (Castagnoli)
- The EID of the bundle destination's application element (e.g., "*ipn:1.1*"), this field can never be anonymous (e.g., "*dtn:none*").
- The Source Node ID, the EID of the bundle source's application element (e.g., "*ipn:2.1*"), which can be either a singleton or the null EID.
- The Report-to EID, i.e., the EID to which the status reports are destined; this may differ from the source EID or may be anonymous if no status reports are to be sent.
- The creation timestamp which includes the time at which the bundle is generated in DTN Time (milliseconds elapsed since the DTN Epoch, i.e., 2000-01-01 00:00:00 +0000 UTC) and the sequence number, which is an always increasing unsigned integer that can be optionally reset to 0; the aim of the sequence number is to make unique the timestamp assigned to a bundle by a source.
- The lifetime, representing a duration in milliseconds that, when added to the creation time, indicates the validity deadline of the bundle.
- The fragment offset, present only if the bundle is a fragment, represents the offset of the first byte of this fragment's payload in reference to the payload of the original bundle.
- The Total Application Data Unit Length, present only if the bundle is a fragment, represents the payload size of the original bundle.
- The CRC of the Primary Block (optional).

The Extension Blocks and the Payload Block are encoded in a common structure called Canonical Bundle Block. This structure contains the following fields:

- The Block Type, an 8-bit integer indicating the type of block; 1 is for the Payload Block.
- The Block Number, a number to unambiguously identify a bundle block. Mainly used by the companion bundle security protocol, BPSec [RFC9172].
- The Block Processing Control Flags, which indicate properties that apply only to this specific block.
- The CRC type: an integer that identifies the kind of CRC in use in the specific block; each block can have his own CRC type.
- The Block-Type-Specific Data, the length in bytes of the data and the data of this block (e.g. the Application Data Unit if this block is a Payload Block).
- The CRC of this Canonical Block (optional).

To uniquely identify a bundle, we need the triple: Source Node ID, creation timestamp time, and creation timestamp sequence number; if the bundle is a fragment also the fragment offset and the fragment length.

2.4 IMPLEMENTATIONS

There are a few existing BP implementations and other are coming, the most are released as free software. In the following we cite the most relevant. All of them are supported by Unified API [Bisacchi_2022B], and thus are compatible with all DTNsuite applications [DTNsuite].

2.4.1 ION

ION (Interplanetary Overlay Network) [Burleigh_2007] [ION] is the DTN protocol suite developed by NASA-JPL (Jet Propulsion Laboratory). It includes both BPv6 and BPv7 (in alternative), it is written in C and it is one of the BP implementations currently used on the ISS (International Space Station). It includes a few convergence layers such as LTPCL, UDPCL and TCPCLv3, and two implementations of SABR (Schedule-aware Bundle routing) [CCSDS_SABR], the latest version of CGR (Contact Graph Routing) [Araniti_2015]. The second implementation of SABR, with several experimental features, was developed by Unibo and then included in the official ION package [Unibo-CGR], [Caini_2021].

2.4.2 DTNME

DTNME (DTN Marshall Enterprise Implementation) [DTNME] is the DTN protocol suite developed by NASA-MSFC (Marshall Space Flight Center) as a fork of DTN2 [DTN2], the so called "reference implementation" of the BP. DTN2 was focused on terrestrial application of the DTN architecture but NASA had to make and keep it interoperable with ION as this was a requirement on ISS experiments. As DTN2 stopped to be updated in 2013, NASA MSFC eventually performed the fork. DTNME is written in C++ and adds support for BPv7 to DTN2. As ION, DTNME includes native implementations of LTP and TCPCLv3. Support for scheduled contacts and CGR-SABR is not native but it can be added thanks to two software packages developed at Unibo [Unibo-DTNME], [Unibo-CGR], [Gori_2020]. DTNME as ION is supported by the Unified-API.

2.4.3 µD3TN

µD3TN [µD3TN] is a recent BP implementation developed by D3TN GmbH and it supports BPv6, BPv7, and several convergence layers such as MTCP (Minimal TCP Convergence Layer) [MTCP], TCPCLv3 and SPP (Space Packet Protocol) [CCSDS_SPP]. This implementation is mainly written in C and Python[Python] and currently supports only POSIX-compliant systems. Former versions of the software also included support for microcontrollers such as STM32. It lacks support of both LTP and CGR/SABR.

2.4.4 Unibo-BP

Unibo-BP [Caini_2024] is a BPv7 only implementation developed by UniBo (University of Bologna) for research purposes. It includes a brand new implementation of TCPCLv3, and Unibo-CGR, the Unibo implementation of CGR/SABR. It is also compatible with preexisting UniBo implementations of LTP [Unibo-LTP] and ECLSA [ECLSA], originally developed as ION plug-ins. In brief, Unibo-BP represents the heart of Unibo-DTN, the DTN suite developed and maintained by Unibo, (BPv7, LTP, TCPCLv3, CGR/SABR, DTNsuite, ECLSA).

3 LTP

The Licklider Transmission Protocol (LTP) is designed to cope with long-delays and intermittent scheduling, thus being the perfect choice as a convergence layer for many BP applications, especially space-based ones. It has been standardized by both IRTF [RFC_5325], [RFC_5326] and CCSDS [CCSDS_LTP]. There is also an extended variant, Multicolor-LTP, which has recently been proposed by the University of Bologna and DLR researchers in [Bisacchi_2022A]. The protocol is named in honor of ARPA and Internet pioneer J.C.R. Licklider, of whom we remember the jokingly but visionary "*Memorandum for Members and Affiliates of the Intergalactic Computer Network*" [Licklider_1963].

3.1 MOTIVATION

The one-way propagation delay between Earth and Mars can vary between 3 and 22 minutes with an average of 10 minutes, equivalent to a RTT (Round Trip Time) range from 6 to 44 and average of 20 minutes. If TCP were to be used to communicate between Earth and Mars, even by considering the case with the least delay, to achieve a bandwidth of a mere 10 Mbps according to the formula in [RFC_3448], we would need a Bit Error Rate (BER) of 4.68 x 10^(-12), which is not achievable in space communications. Other defects of TCP that make it unsuitable for communication in environments with long delays include:

- The handshake to establish the connection before starting transmission, which in the best case requires 2 RTTs.
- The use of mechanisms such as Slow Start and Additive Increase Multiplicative Decrease to negotiate the transmission speed, since these mechanisms only work with feedback and would involve an exorbitant wait.
- The calculation of RTOs (Retransmission Time Outs) is carried out through statistical analysis, which, however, would be greatly influenced by transient loss, and can only be performed after the handshake.

More generally, TCP heavily relies on feedback mechanisms for almost all its components (congestion control, flow control, RTO estimation, loss recovery) as it was

designed for terrestrial wired networks, where the maximum RTT is in the order of 200ms and thus prompt feedback is granted even in the worst case. TCP performance is already severely degraded when RTT increase to 600 ms, the typical value of GEO satellite connections, which can be considered as the threshold of TCP usability.

Moreover, TCP requires an always active End-to-End path, a requirement that is not guaranteed in challenged networks.

LTP has been designed to be an alternative to both TCP and UDP in space, offering both a reliable and an unreliable service. LTP design tries hard to reduce the reliance on feedback mechanisms, or, in simple words, the "chattiness" of the protocol, by relying on contact and range information included in the contact-plan (a list of contacts and ranges known a priori, issued by a central authority to all nodes of a network). To this end, it:

- Does not have a connection establishment phase (the 3-way hand-shake of TCP); the sender simply starts transmitting when allowed by a contact.
- Does not have a congestion control; the sender simply transmits at the nominal rate that is declared in a contact.
- Flow control is based on a maximum number of "sessions" (see below)
- RTO is calculated a priori from range information.

Another difference between LTP and TCP is that the receiver can only reply with signaling segments, and no data, which is possible in TCP, thus sessions are not bidirectional and are categorized in Tx (Transmission) and Rx (Reception) sessions. If bidirectional data communication between two nodes is require, the receiver will start a Tx session with the sender.

When the reliable service is used, feedbacks are however obviously necessary to know if data have successfully arrived or not and thus whether they need to be retransmitted. This represents a residual but unavoidable dependence on RTT. To reduce this impact, transmission of data is carried out through parallel independent sessions, so that losses on a session do not delay the transmission of other data on other session.

When LTP is used as a Converge Layer for BP, one or more Bundles are aggregated by the LTPCLA into one LTP block, without adding any additional information. The bundles are aggregated into one block and ceases after the block size exceeds a threshold called "aggregation size limit" or the "aggregation time limit" is reached. Note that bundle aggregation is performed only for bundle requiring a reliable service.

Once a block is created, it is segmented into (usually many) LTP segments, each of which is then encapsulated into either an UDP segment or a CCSDS protocol, depending on the environment. All these aspects will be better clarified below.

3.2 BLOCKS AND CONTACTS

LTP blocks are divided into a Red Part and a Green Part, to be delivered respectively in a reliable and unreliable manner. In terms of LTP communication semantics, LTP Red is an at-most-once type semantic while LTP Green is a maybe/best-effort type semantic. It is possible for blocks to be completely monochromatic, which is usually the case. In some LTP implementations, like Multicolor-LTP [Bisacchi_2022A], only monochromatic blocks are possible, which offers the advantage of a reduced complexity and, more important, of enforcing a one-to-one correspondence between a class of service and blocks, as a block can only be either red or green. As a session is the independent transmission of a block, the result is that we can associate a color, i.e. a class of service, to sessions. Last, it is worth noting that Multicolor LTP offers a third color, Orange, associated to the class of service “notified”.

To resolve the issue that in TCP is addressed by the handshake and the negotiation of transmission speeds through the sliding window, LTP relies on the contact plan, where Contacts and Ranges are specified. A Contact represents a transmission opportunity at the BP level from node A to node B and specifies the time interval and the nominal transmission speed during the interval. A Range, on the other hand, specifies the propagation time (and possibly other delay factors) between node A and node B, during a given interval. The contact plan is used by BP for the GCR routing algorithm and by LTP to schedule data transmission. An LTP “engine”, a node in the specific LTP language, can send data to an adjacent engine only if there is an active contact at that time. Contacts are also used to determine the transmissions speed, ranges and the

calculation of the RTO. If the contact ends while a session is ongoing, all RTO timers will freeze, and restart at the next contact. In ION, however, it is also possible to abort the session, as an alternative.

3.3 SESSIONS

Once the block is ready to be sent, a session is created; each LTP session is independent to make the best use of bandwidth in the presence of losses and in ION the maximum number of Tx and Rx sessions that can be active simultaneously in a given engine is specified, thus performing a kind of flow control function. Each LTP block is then fragmented into segments, each segment with a header. The size of the segments depends on the underlying level, UDP in testbeds and CCSDS SPP [CCSDS_SPP] in space. Red segments must be delivered reliably to the destination using an ARQ mechanism, like TCP, while green segments are sent without any kind of retransmission, like UDP. In the following paragraph we consider only monochrome Red sessions, by large the most complex, following the treatment presented in [Bisacchi_2022A]. For green and orange sessions we refer the interested reader to the cited reference.

3.3.1 No Losses

In a Red Session were there are no losses all segments are sent, the last segment has the flags CP (Checkpoint) and EOB (End of Block) set (Figure 3.1). The arrival of the CP triggers an RS (Report Segment) which confirms what data (range of bytes) have arrived, in this first case all. The RS must be confirmed by an RA (Report Ack). In the absence of losses, and by assuming negligible the block radiation time with respect the propagation delay, as always in space, LTP delivers all data in 1/2 RTT which is the theoretical minimum for any type of communication. Note that in the figure the

radiation time i.e. the time necessary to send all segments, has been deliberately emphasized to make the figure readable.

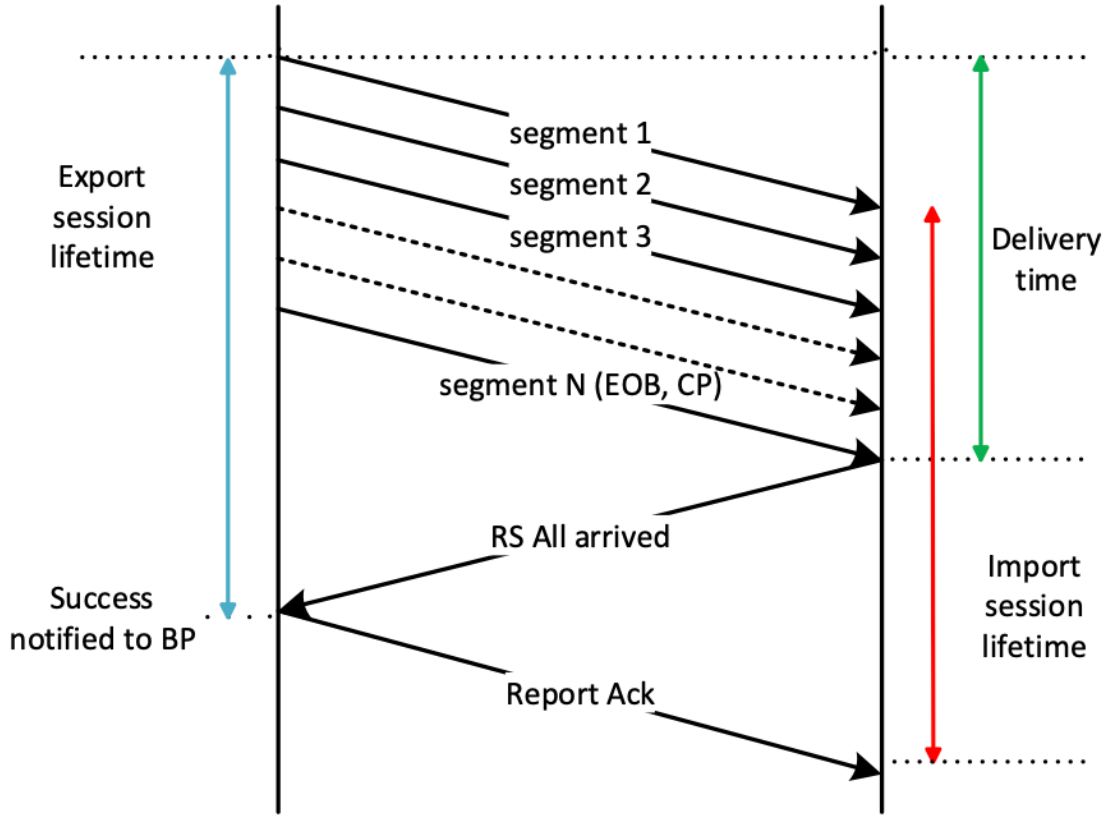

Figure 3.1 - Red LTP Session without any losses, from [Bisacchi_2022A] , the delay between segments and the transmission delay is not to scale

3.3.2 Losses on data segments

In this second example, the Red session incurs in the loss of 2 data segments (Figure 3.2). Once the last segment, containing EOB and CP, arrives, the receiving LTP Engine sends back an RS containing the "claims", i.e. the interval of contiguous bytes arrived. In this case segment 2 and 3 are contiguous, thus the RS contains only 2 claims, the former corresponding to segment 1, and the latter from segment 4 to the end. The RS is acknowledged by an RA as usual, and then segments 2 and 3 are retransmitted, with the last one being flagged as a CP. No further losses occur so the session concludes with an RS confirming all segment have arrived (one claim with all bytes of the block) and an RA acknowledging the RS. The complete transmission last in total 1 and half RTT.

The penalty for each incomplete data transmission is 1 RTT, independently of the number of losses, in sharp contrast with TCP, where, in the absence of the SACK option, each loss requires an RTT to be recovered.

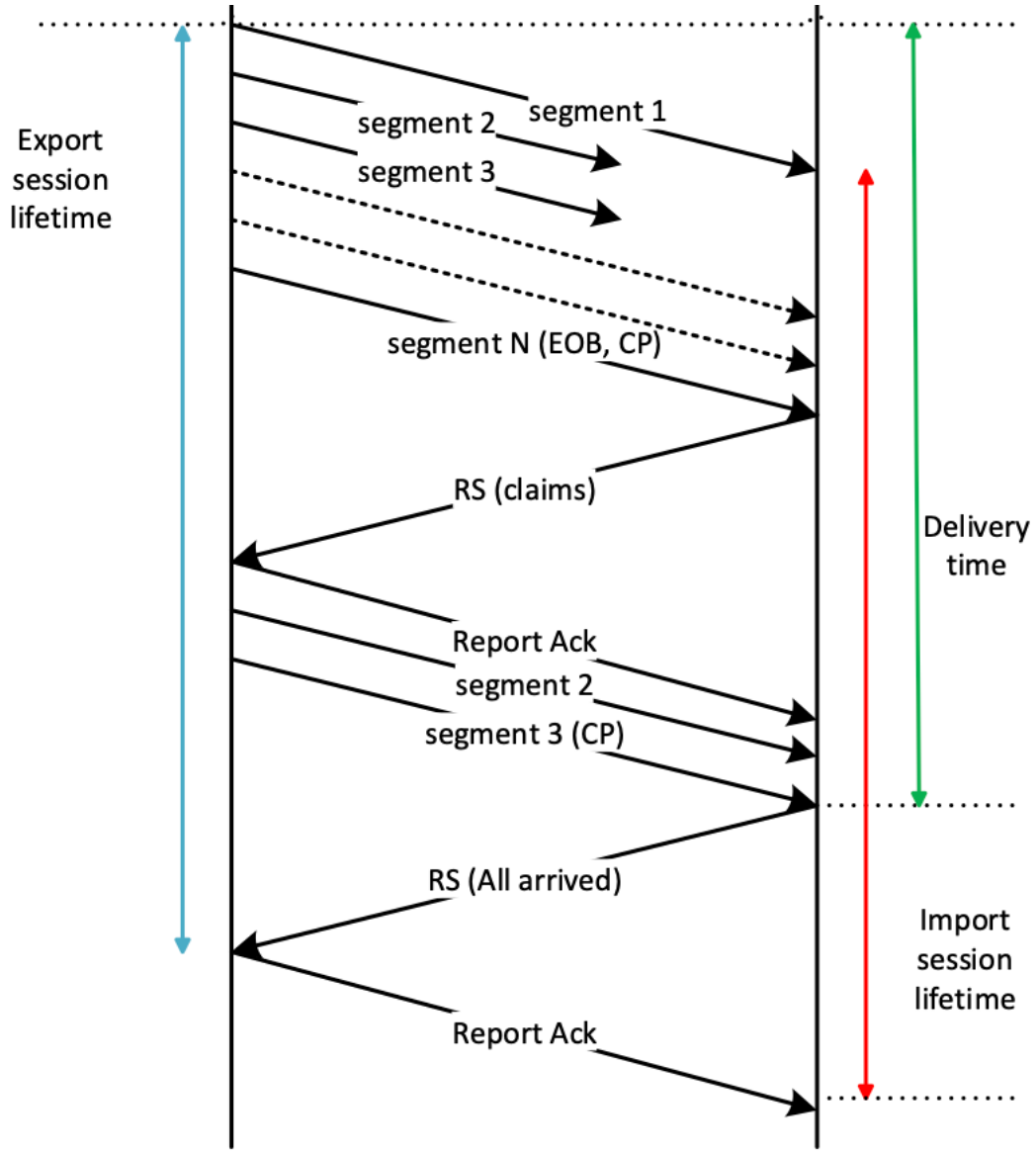

Figure 3.2 - Red LTP Session with losses on data segments, from [Bisacchi_2022A], the delay between segments and the transmission delay is not to scale.

3.3.3 Losses on data and signaling segments.

In this third example, the Red session incurs in the loss on both data segments and signaling segments, specifically data segment 2 and 3 are lost together with the last segment, that flagged with the first CP. After sending this segment, as for all CPs, RSs

CSs, CRs segments [RFC5327], the sender starts an RTO timer based on the contact plan, however the receiver does not respond with an RS because it has not received the CP; the CP RTO eventually elapses and the sender resends the CP. This time it arrives to destination and is acknowledged by an RS, and everything goes on as before. The penalty due to the loss of the CP (but it would be the same for the loss of an RS) is one RTO, which adds to the penalty due to losses on "pure" data segments. In total the data are the delivered to the receiver in 1 and half RTT plus the RTOs penalty, which is slightly greater than an RTT.

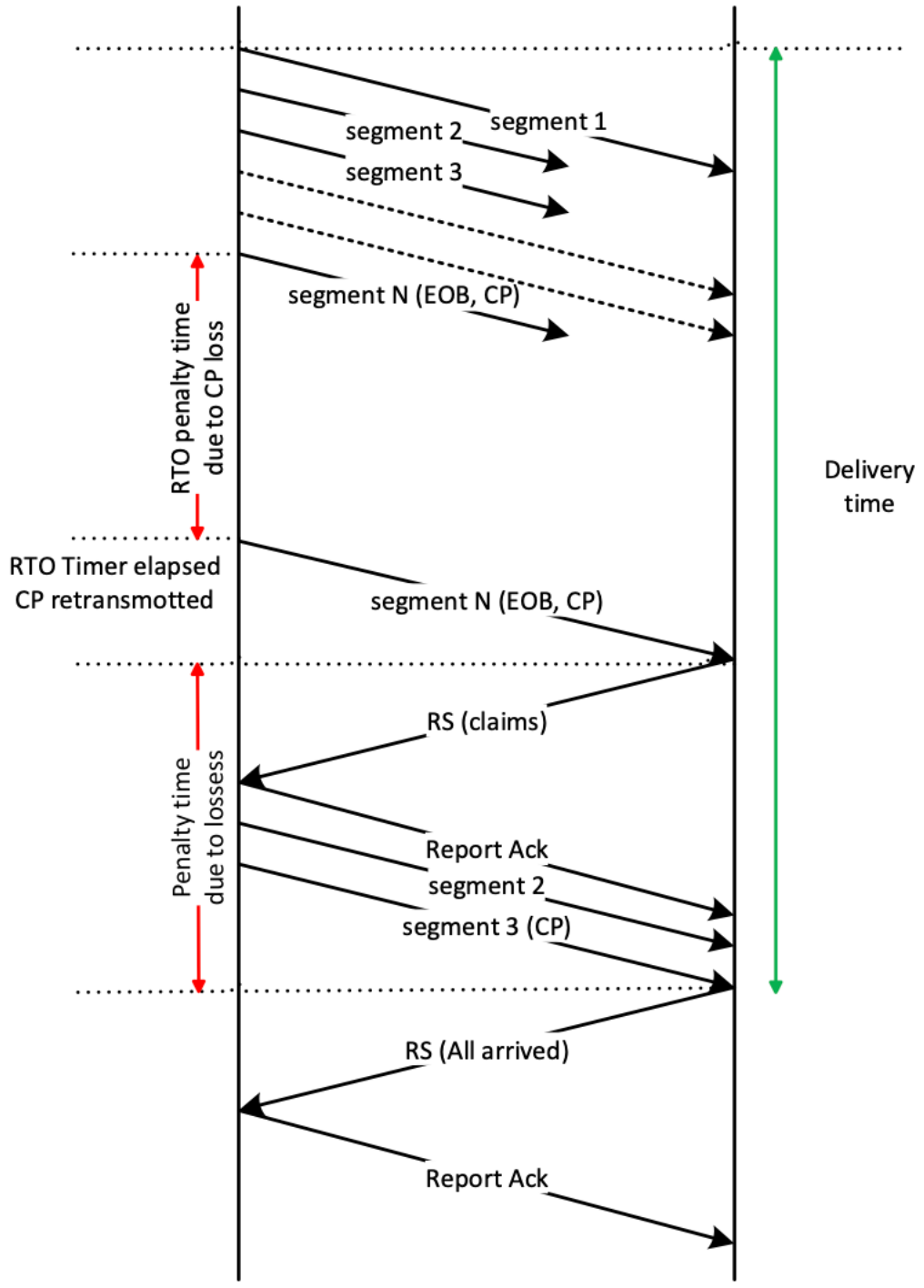

Figure 3.3 - Red LTP Session with data segments and signaling segments losses, from [Bisacchi_2022A], the delay between segments and the transmission delay is not to scale.

3.3.4 Session Cancellation

It may occur that during a Red Session either the sender or the receiver wants to cancel the on-going session, for example when a retransmission count reaches a certain threshold. These threshold(s) can usually be set while configuring the LTP engine.

If the sender wants to cancel the session, he will send a CS (Cancel Segment by Sender) and it will expect a CAS (Cancel Segment by Sender Ack), then it will close the TX session. Conversely, if the receiver wants to cancel the session, he will send a CR (Cancel Segment by Receiver) and it will expect a CAR (Cancel Segment by Receiver Ack). Each Cancel Segment contains a code indicating the reason for session cancellation. CS and CR are also subjected to a maximum number of retransmissions after which the session is forcedly closed.

3.4 LTP SEGMENT

Each LTP segment is composed by:

- Header
- One or more bytes of data
- Zero or more trailer extensions (not covered in this thesis)

All the fields with variable dimension are encoded as SDNV (Self Delimiting Numerical Value) [ASN1] to minimize the use of bandwidth and to be flexible at the same time. In SDNV encoding the MSB (Most Significant Bit) indicates whether that is the last byte or not, the other 7 bit constitute the useful part of the byte (Figure 3.4). Note however that SDNV has the drawback of requiring the parsing of one byte at a time, which penalizes the processing times when LTP is implemented in hardware, as recently emphasized by ESA.

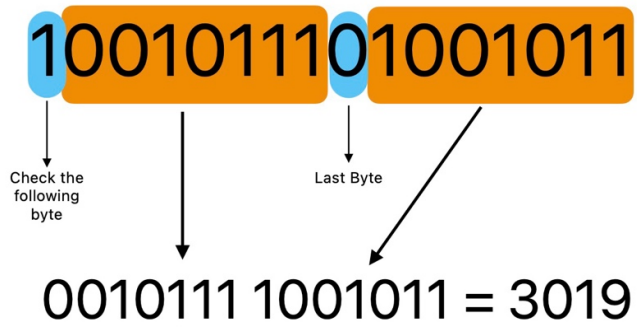

Figure 3.4 - SDNV Encoding Example

3.4.1 Header Structure

The header is composed by:

- 4 version bit(current version is 0)
- 4 bit to indicate the segment type
- Session ID, an unique identifier of the session composed by the Session Originator, the Engine Id of the Sender encoded as an SDNV, and the Session Number, a random number encoded as an SDNV
- 4 bit to indicate the number of header extensions
- 4 bit to indicate the number of trailer extensions
- 0 or more bytes of header extensions (not covered in this thesis)

All segment types are summarized in the following table.

Bit 4	Bit 3	Bit 2	Bit 1	Code	Segment Type
0	0	0	0	0	Red Data
0	0	0	1	1	Red Data, CP (Checkpoint)
0	0	1	0	2	Red Data, CP, EORP (End of Red Part)
0	0	1	1	3	Red Data, CP, EORP, EOB (End of Block)
0	1	0	0	4	Green Data,
0	1	0	1	5	Orange Data, Multicolor LTP only [Bisacchi_2022A]
0	1	1	0	6	Orange EOB, Multicolor LTP only
0	1	1	1	7	Green Data, EOB
1	0	0	0	8	RS (Report Segment)
1	0	0	1	9	RA (Report Ack)

1	0	1	0	10	Orange PN (Positive Notification) Multicolor LTP only
1	0	1	1	11	Orange NN (Negative Notification) Multicolor LTP only
1	1	0	0	12	CS (Cancel segment from Sender)
1	1	0	1	13	CAS (Cancel acknowledgement from Receiver)
1	1	1	0	14	CR (Cancel segment from Receiver)
1	1	1	1	14	CAR (Cancel acknowledgement from Sender)

Table 3.1 -LTP Segment Types

3.4.2 Segment Content

The data portion of LTP segments contains some metadata used for segment elaboration and the payload being delivered. All data segments, i.e. with a code lower or equal to 7, contain the following metadata:

- Client Service ID, encoded as a SDNV, indicates the upper layer service to which the segment's payload is to be delivered, this is like TCP ports, for the BP the value is 1.
- Offset, encoded as a SDNV, indicates the position of the segment's payload in the block being delivered.
- Length, encoded as a SDNV, indicates the length of segment's payload.

Additionally Red Data Segments containing a CP have the following metadata:

- Checkpoint Serial Number, encoded as a SDNV, number uniquely identifying the CP in a certain session, it is randomly chosen for the first CP segment and then increases by 1 each retransmission.
- Report Serial Number, encoded as a SDNV, only present if the CP is retransmitted in response to a RS and it is the unique identifier of such RS.

The RS instead contains the following metadata:

- Report Serial Number, encoded as a SDNV, number uniquely identifying the RS in a certain session, it is randomly chosen for the first RS segment and then increases by 1 each retransmission.
- Checkpoint Serial Number, encoded as a SDNV, the serial number of the CP that generated the RS.

- Upper Bound and Lower Bound, encoded as a SDNV, these two numbers serve as offsets for the claims, they are necessary as more than one report segment may be needed to transmit all the claims.
- Reception Claim Count, encoded as SDNV, the number of data reception claims in this report.
- Reception Claims, indicating successfully received batch of continuous data, each claim consists of an offset from Lower Bound and a length, the contiguous byte of data received, both these two fields are encoded as a SDNV.

Lastly the data section of RAs contains only the Report Serial Number of the RS which generated his transmission.

4 ESA BP OVERVIEW

ESA BP is a new implementation of the Bundle Protocol developed by ESA (European Space Agency) [ESA BP_SDD]. This implementation is completely written in [Java], it supports both BPv7 and BPv6 (not simultaneously) and BPSec[RFC_9172]. It comes with native implementations of TCPCLv3, UDPCL [RFC_9174] and LTPCL. The structure of the Bundle Node adopted by ESA is the following:

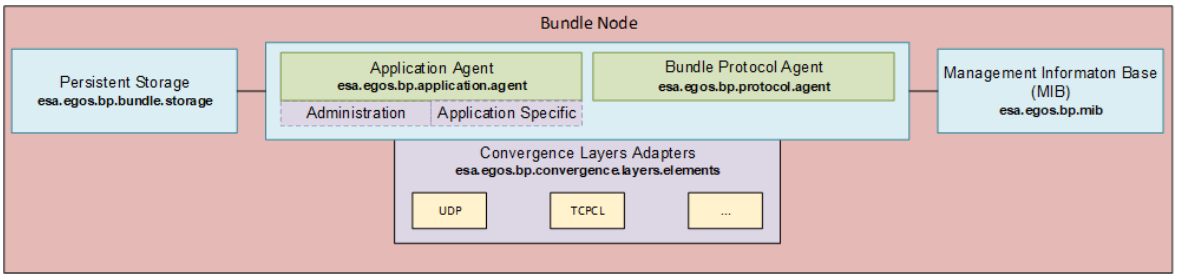

Figure 4.1 - Graphical Representation of ESA BP Bundle Node, from [ESA BP_SDD]

4.1 APPLICATION AGENT

The Application Agent (AA) is the component that handles the interaction between BP applications ("clients" in the following) and the Bundle Protocol Agent (BPA).

The AA exposes a daemon process to which the various BP clients can connect via ZeroMQ sockets [ZeroMQ][JeroMQ], an abstraction over datagram and stream sockets that present to the user an asynchronous message queue.

As shown in Figure 4.1 the AA is composed of an application-specific element and an administrative element. The application-specific element constructs, requests transmission of, accepts delivery of, and processes application data units (ADUs). The administrative-specific element construct and requests transmission of administrative records, it also accepts delivery and processes any custody signals that the node may receive (in BPv6 only)

4.2 BUNDLE PROTOCOL AGENT

The Bundle Protocol Agent (BPA) is the core component of the Bundle Node, as it implements the functionalities of the Bundle Protocol. The key activities of the BPA are:

- Handle user requests: register, deregister and change local endpoint connection, send bundle, cancel bundle requests, and send poll bundles (ping implementation at the Bundle Protocol level).
- Local Bundle delivery
- Bundle Handling: Forwarding, Reception, Custody (BPv6 only), Deletion

Without going into detail, it is interesting how the management of bundles is carried out by ESA BP: each Bundle is assigned one or more constraints that indicate the procedures to be followed to forward the bundle to another node or to deliver it to the AA. The set of all constraints represents the "status" of the bundle. If a bundle no longer has any constraints, then it is discarded. Other reasons for discarding a Bundle include user's request for cancellation, expiration of the bundle's lifetime, or failure in forwarding the bundle.

4.3 CONVERGENCE LAYERS

The Bundle Protocol operates as an overlay network over (usually) the Transport layer, and along a bundle path to destination it is probable that different transport protocols will be used. Thus, the use of CLAs (Convergence Layer Adapter) that format bundles into suitable data units at each DTN hop, depending on the underlying transport protocol.

In ESA BP each CLA is composed of one or more CLE (Convergence Layer Elements), depending on the protocol below BP. For example, in case of TCP, the CLA is composed only of one TCPCL element, while in case of LTP by one LTPCL and one UDPLSA element.

ESA BP supports TCP, LTP and UDP as convergence layers, plus a few CCSDS protocols not covered in this thesis, as SPP, Frame [CCSDS_TM][CCSDS_TC][CCSDS_AOS] and SLE [CCSDS_SLE].

4.4 APIS AND COMMAND LINE INTERFACE

As all other BP implementations, but µD3TN, ESA BP provides an API to allow developers to implement BP applications interacting with the Bundle Node. By contrast

to them, however, ESA BP does not rely on a series of basic applications to send or receive bundles, such as the usual “send”, “receive”, “ping”, “echo”, etc., but it prefers to rely on a unique application, the “Command Line Interface” (alias “Client” in ESA BP documentation). This interface is used not only to send or receive bundles, but also to configure the Bundle Node. The CLI implements some of the interfaces defined in [ESA BP_ICD], based on jeroMQ messages[ZeroMQ][JeroMQ].

.

5 ESA BP CONFIGURATION

This chapter references [ESA BP_CIG] and aims to complement it by presenting a simple configuration example with only two nodes, namely Host (EID ipn:5.0) and VM2 (EID ipn:2.0), running BPv7 and connected to each other first via TCPCLv3 and later with LTP. This layout, however simple, is enough to explain the basics of ESA BP configuration can be used as a starting point for building more complex layouts. Figure 5.1 represents the network topology that will be setup during the guide.

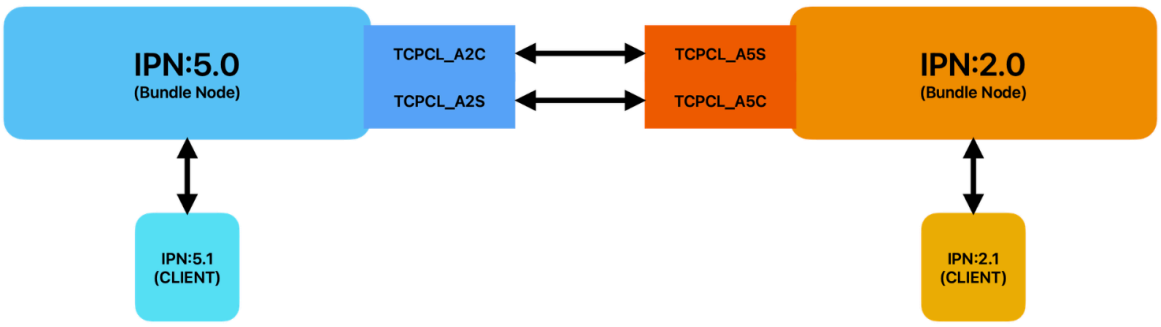

Figure 5.1 - Network Topology

This chapter is divided into 4 sections. The first deal with the Bundle node configuration, the second with the command line interface (or "Client") configuration, the third with LTP configuration (treated apart, as optional), the last with script files used to start the Bundle Node and the CLI.

5.1 BUNDLE NODE CONFIGURATION

In ESA BP the configuration of the Bundle Node is partitioned into many files, following the code structure, instead of being centralized in a unique file, which may be practical when programming, but an obstacle to new users. These files are organized into three directories (aac is a brief form for "esa.egos.bp.application.agent.core"):

- aac/resources/config
- aac/resources/mib
- aac/resources/routing-mib

5.1.1 "Config" files

(For more details see par. 6.2, p. 11 ESA_CIG). Usually, BP applications (Application Agent Client in ESA BP terminology) and BP daemons, i.e. Bundle Nodes implementing among other things the Application Client (Server) and the BP Agent, run on the same machine. However, when inter-process communications between them is performed by means of a TCP socket, as in ESA BP, it is also possible to run the BP application on a separate machine, which may be useful, for instance, when the BP application has to run on a simple hardware, where the installation of BP could be too heavy. For simplicity, and also because it should be the most common by large, we chose to run the Application Client in the same machine as the Bundle Node.

The Bundle Node must thus be connected to the local IP address so for both Host and VM2 in the file "aac/resources/config/config.properties", set "**zmq.address**" to localhost. After that your file should look like this:

```
zmq.secure.message=false
zmq.certificate.folder=$(BP_BN_RES)/daemon/.curve
zmq.server.certificate.name=server_cert.txt
zmq.connection=tcp://
zmq.address=localhost
zmq.port=5671
zmq.listener.rate=1000
zmq.send.timeout=100
zmq.socket.identity.file=$(BP_BN_RES)/daemon/identity_map.xml

mib.folder=$(BP_BN_RES)/mib/
routing.mib.folder=$(BP_BN_RES)/routing-mib/
request.model.persist.rate=200
```

Figure 5.2 - aac/resources/config/config.properties

The remaining Bundle Node configuration is split into MIB and routing configuration.

5.1.2 MIB files

(For more information see par. 6.2.2, p. 13 ESA_CIG). This sub-section describes how to configure MIB (Management Information Base) files in "aac/resources/mib/", the MIB contains the configuration for the Bundle Node adapters and the registered endpoints. In the figures, we will display Host and VM2 configurations consecutively.

5.1.2.1 Node State Definitions -NSD

The file “aac/resources/mib/NSD.sta” contains information about the node state, such as node identity and bundle processing information. In our case, the only part of the file that needs to be modified is "**nodeId"**, which contains the EID of the node, “ipn:5.0” and ipn:2.0 for Host and VM2 respectively. Note that service number “0” is reserved to node EIDs in the “ipn” scheme.

```
<identityInformation>
        <identyInfoItem infoId="nodeId">ipn:5.0</identyInfoItem>
        <identyInfoItem infoId="bpVersion">7</identyInfoItem>
        <identyInfoItem infoId="crcType">CRC_16_X25</identyInfoItem>
        <identyInfoItem infoId="bibeRetransDelay">60</identyInfoItem>
</identityInformation>
```

Figure 5.3 - Host aac/resources/mib/NSD.sta

```
<identityInformation>
        <identyInfoItem infoId="nodeId">ipn:2.0</identyInfoItem>
        <identyInfoItem infoId="bpVersion">7</identyInfoItem>
        <identyInfoItem infoId="crcType">CRC_16_X25</identyInfoItem>
        <identyInfoItem infoId="bibeRetransDelay">60</identyInfoItem>
</identityInformation>
```

Figure 5.4 - VM2 acc/resources/mib/NSD.sta

5.1.2.2 Registrations - REG

The “REG.sta” file contains endpoints that are permanently registered to the current Bundle Node; EID entries must be compatible with the node EID declared in the previous section. With the ipn scheme the only difference is that this time the service number must be different from “0”, e.g. “1”.

```
<mibREGsta>
    <staticConnections>
        <staticConnection eid="ipn:5.1" activityState="ACTIVE" failureAction="DEFER"/>
    </staticConnections>
</mibREGsta>
```

Figure 5.5 - Host aac/resources/mib/REG.sta

```
<mibREGsta>
    <staticConnections>
        <staticConnection eid="ipn:2.1" activityState="ACTIVE" failureAction="DEFER"/>
    </staticConnections>
</mibREGsta>
```

Figure 5.6 - VM2 acc/resources/mib/REG.sta

5.1.2.3 Elements - CLE

The “CLE.dyn” file contains settings related to convergence layers “elements”. These are peculiar of the ESA BP implementation, as the “adapters”, which will follow. Each element represents one or more protocols of the Convergence Layer stack, e. g. in ESA BP a single TCPCL element corresponds to an instance of the TCPCL protocol and to the underlying TCP socket. In ESA BP a TCPCL element can be either an INITIATOR or a RESPONDER: the Initiator starts the connection (CL client), while the Responder listens for incoming connections (CL server). Readers familiar with ION [ION] can think at Initiators as “outducts” and at Responders as “inducts”. For the sake of clarity, we adopted the following syntax rule for naming elements:

<transport protocol utilized>_E<destination node_number>[<S/C>].

e.g. TCPCL_E2S stands for: TCPCL element, used for communication with node 2, server.

Below we provide the Host configuration. We inserted two elements, one server and one client. On the server (the induct), we set the listening port to 4556, as this is the default port for TCPCL and therefore the natural solution when dealing interoperability tests, as later in this thesis. In the client element (the outduct) we inserted the IP address 10.0.0.12:4556, i.e. the IP address of VM2, where it is supposed that a server will be listening on the 4556 port. Note that BP EIDs are not inserted in CL elements.

As said, in ESA BP one CL may consists of either one protocol, as here for TCPCL, or a stack of protocols, as shown later for LTPCL. In the former case, the element is said to be a “singleton” This is why singleton is set to true in the file. Note that the use of the term singleton in this context is questionable, as in the DTN RFCs a singleton refers to an EID representing one DTN node, i.e. has a completely different meaning and should not have been redefined.

```
<mibCLEsta>

    <element id="TCPCL_E2S" status="B"
implClassName="esa.egos.bp.convergence.layers.elements.CLElementTCPCL" singleton="true"
         maxqs="20" poll="1000">
            <elementConfig>
                <role>RESPONDER</role>
                <port>4556</port>
                <magic>dtn!</magic>
               ...
        </elementConfig>
    </element>

    <element id="TCPCL_E2C" status="B"
implClassName="esa.egos.bp.convergence.layers.elements.CLElementTCPCL" singleton="true"
        maxqs="20" poll="1000">
            <elementConfig>
                <role>INITIATOR</role>
                <ip>10.0.0.12:4556</ip>
                <magic>dtn!</magic>
                ...
        </elementConfig>
    </element>

</mibCLEsta>
```

Figure 5.7 - Host aac/resources/mib/CLE.dyn

Below the equivalent configuration for VM2.

```
<mibCLEsta>

    <element id="TCPCL_E5S" status="B"
implClassName="esa.egos.bp.convergence.layers.elements.CLElementTCPCL" singleton="true"
        maxqs="20" poll="1000">
            <elementConfig>
                <role>RESPONDER</role>
                <port>4556</port>
                <magic>dtn!</magic>
               ...
        </elementConfig>
    </element>

    <element id="TCPCL_E5C" status="B"
implClassName="esa.egos.bp.convergence.layers.elements.CLElementTCPCL" singleton="true"
    maxqs="20" poll="1000">
            <elementConfig>
                <role>INITIATOR</role>
                <ip>10.0.0.1:4556</ip>
                <magic>dtn!</magic>
                ...
        </elementConfig>
    </element>

</mibCLEsta>
```

Figure 5.8 - VM2 acc/resources/mib/CLE.dyn

Users familiar with ION, DTNME, Unibo-BP and µD3TN will have noted that in ESA BP we need to configure one specific server (one induct) for each proximate node, instead of one server for all nodes, as in these implementations.

5.1.2.4 Adapters - CLA

The "CLA.sta" file contains the settings of CL "adapters". These adapters specify by which "elements" a bundle node can send bundles to another node and vice versa. Each adapter may have a list of elements defined, although in the example only one is used.

For each adapter, the values to modify are:

1) endpoint, the endpoint the adapter is devoted to ("it points to" if the element is client, "it is pointed from", if server);
2) element, the element(s) that the adapter can use;
3) active, if the adapter must be activated at the start of the Bundle Node.

Adapters can be easily activated and deactivated from the CLI (Command Line Interface). The syntax we adopted for naming adapters is:

<transport protocol utilized>_A<destination node_number>[<S/C>].

e.g. TCPCL_A2S stands for TCPCL adapter dedicated to node 2, containing an element of server kind. Once again, we stress that in ESA BP each server is devoted to one proximate node, thus we can say, as before, that it "points" to it. Note that at the start only one couple of adapter will be active, respectively the Host Client (TCPCL_A2C) and VM2 Server (TCPCL_A5S).

For the Host we have:

```
<mibCLAsta>

    <adapter id="TCPCL_A2S" endpoint="ipn:2.0" poll="1000" flush="1000" maxqs="20" maxbs="50000"
bibeEnabled="false"
        cbheEnabled="true" retransmissionLimit="5" active="false">
        <elements>
            <element>TCPCL_E2S</element>
        </elements>
    </adapter>

        <adapter id="TCPCL_A2C" endpoint="ipn:2.0" poll="1000" flush="1000" maxqs="20" maxbs="50000"
bibeEnabled="false"
        cbheEnabled="true" retransmissionLimit="5" active="true">
        <elements>
            <element>TCPCL_E2C</element>
        </elements>
    </adapter>

</mibCLAsta>
```

Figure 5.9 - Host acc/resources/mib/CLA.sta

Analogously for VM2.

```
<mibCLAsta>

    <adapter id="TCPCL_A5S" endpoint="ipn:5.0" poll="1000" flush="1000" maxqs="20" maxbs="50000"
bibeEnabled="false" cbheEnabled="true" retransmissionLimit="5" active="true">
        <elements>
            <element>TCPCL_E5S</element>
        </elements>
    </adapter>

<adapter id="TCPCL_A5C" endpoint="ipn:2.0" poll="1000" flush="1000" maxqs="20" maxbs="50000"
bibeEnabled="false" cbheEnabled="true" retransmissionLimit="5" active="false">
        <elements>
            <element>TCPCL_E5C</element>
        </elements>
    </adapter>

</mibCLAsta>
```

Figure 5.10 - VM2 aac/resources/mib/CLA.sta

5.1.3 Routing-mib

ESA BP implements only static routing, i.e. routing based on static rules (for more information see par. 6.3, p. 28 ESA_CIG). The “aac/resources/routing-mib/NHT.dyn” file contains the Next Hop Table which stores static routing information.

It depends on network topology, which in the example considered here consists of only two nodes with an ideal link (Figure 5.11). For the sake of symmetry, we have a TCPCL

client and a server on both sides, which means that two TCP connections can be established in parallel, one under control of each node. Once established each TCP connection can be used also in the reverse direction, as an “opportunistic link” (opportunistic as it is not under the control of the node, being instantiated and managed by the peer node).

Note that on the Host (ipn:5.0) file (Figure 5.12), we have a section for VM2 (ipn:2.0) and vice versa (Figure 5.13). Each section can be considered as one static rule. In simple words, it states that to get to destination X, one of the listed adapters must be used. In our case the destination is a proximate node, i.e. we have a client adapter pointing to it, as we have just one DTN hop; more generally, in case of multiple hops, the adapter would bring only to the first node of a longer path to destination.

A peculiar thing of ESA BP is that an adapter acting as server is devoted to only one adapter acting as client, as shown in the figure. This differs from all other BP implementations, which for each CL have one server for all possible clients, present and future (they could not be known in advance, but “appear” later).

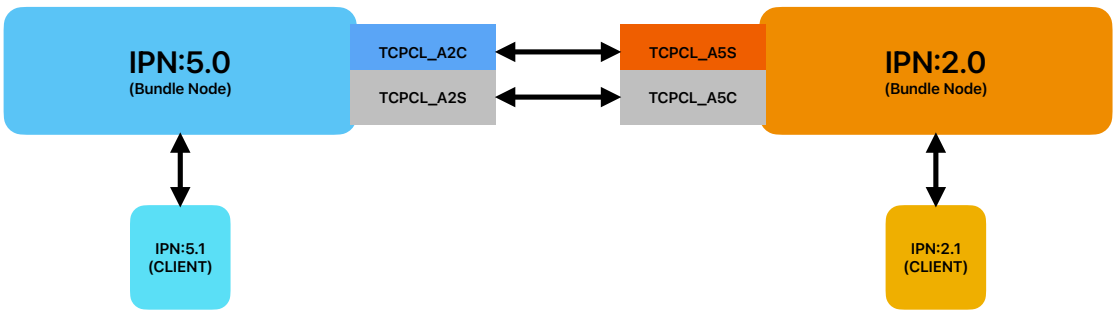

Figure 5.11 - Network Topology, TCPCL_A2S and TCPCL_A5C will not be active at the start of the bundle node

For the Host we have:

```
<mibNHTdyn>
    <endpoints>

        <endpoint endpointId="ipn:2.0">
            <adapters>
                <adapter adapterId="TCPCL_A2C" maxHopCount="10" />
                <adapter adapterId="TCPCL_A2S" maxHopCount="10" />
            </adapters>
        </endpoint>

    </endpoints>
</mibNHTdyn>
```

Figure 5.12- Host aac/resources/routing-mib/NHT.dyn

Analogously for VM2:

```
<mibNHTdyn>
    <endpoints>
        <endpoint endpointId="ipn:5.0">
            <adapters>
                <adapter adapterId="TCPCL_A5C" maxHopCount="10" />
                <adapter adapterId="TCPCL_A5S" maxHopCount="10" />
            </adapters>
        </endpoint>

    </endpoints>
</mibNHTdyn>
```

Figure 5.13 - VM2 aac/resources/routing-mib/NHT.dyn

5.2 ESA BP "CLIENT" CONFIGURATION

The "Client" is a CLI (Command Line Interface) provided by ESA BP to send or receive bundles, but also to control some aspects of the Bundle Node, i.e. of the BP Agent.

5.2.1 Config file

The "cli/resources/config/config.properties" file (cli stands for "esa.egos.bp.cli/") contains a miscellanea of parameters of very different origin and use. For example, it contains bundle settings, such as priority, BSR flags, custody option, etc., but also a lot of other parameters not referring to bundles, such as the address and port used by the CLI to communicate with the Bundle Node. Note that the former are only default values, as bundle settings should be set by the bundle source on a per bundle basis, i.e. the source should be free of setting different lifetimes or different priorities, etc, for subsequent bundles at its will.

The values we modified are: "**cli.sr.repEid**", i.e. the default reportTo EID, "**cli.sr.lifetime**", the default lifetime(in ms) and the "**zmq.address**", the IP address of the Bundle Node, which we set to localhost as we assumed in our layout the CLI and the Bundle Node run on the same machine.

For the Host CLI configuration we have:

```
cli.do.FRAGMENT = false
cli.do.ADMINISTRATIVE = false
cli.do.NON_FRAGMENTABLE = false
cli.do.CUSTODY_TRANSFER = false
cli.do.SINGLETON = true
cli.do.ACKNOWLEDGEMENT = false
cli.do.COS = expedited
cli.do.RECEPTION_REPO = false
cli.do.CUSTODY_REPO = false
cli.do.FORWARDING_REPO = false
cli.do.DELIVERY_REPO = false
cli.do.DELETION_REPO = false
cli.sr.repEid = ipn:5.1
cli.sr.lifetime = 60000
cli.sr.singleton = true
cli.do.STATUS_TIME = false

cli.bn.connection.timeout = 40

client.service.number = 1
client.registration.action = delivery

zmq.secure.message=false
zmq.certificate.folder=$(BP_CLI_RES)/daemon/.curve
zmq.server.certificate.name=server_cert.txt
zmq.client.certificate.name=client_cert.txt
zmq.connection=tcp://
zmq.address=localhost
zmq.port=5671
zmq.listener.rate=500
zmq.send.timeout=100

zmq.socket.identity.file=$(BP_CLI_RES)/daemon/identity.txt
```

Figure 5.14 - Host cli/resources/config/config.properties

Note that the lifetime value is deliberately set to only 60s, to avoid interference between consecutive tests. The user can set a larger value when necessary. Analogously for the VM2 CLI configuration:

```
cli.do.FRAGMENT = false
cli.do.ADMINISTRATIVE = false
cli.do.NON_FRAGMENTABLE = false
cli.do.CUSTODY_TRANSFER = false
cli.do.SINGLETON = true
cli.do.ACKNOWLEDGEMENT = false
cli.do.COS = expedited
cli.do.RECEPTION_REPO = false
cli.do.CUSTODY_REPO = false
cli.do.FORWARDING_REPO = false
cli.do.DELIVERY_REPO = false
cli.do.DELETION_REPO = false
cli.sr.repEid = ipn:2.1
cli.sr.lifetime = 60000
cli.sr.singleton = true
cli.do.STATUS_TIME = false

cli.bn.connection.timeout = 40

client.service.number = 1
client.registration.action = delivery

zmq.secure.message=false
zmq.certificate.folder=$(BP_CLI_RES)/daemon/.curve
zmq.server.certificate.name=server_cert.txt
zmq.client.certificate.name=client_cert.txt
zmq.connection=tcp://
zmq.address=localhost
zmq.port=5671
zmq.listener.rate=500
zmq.send.timeout=100

zmq.socket.identity.file=$(BP_CLI_RES)/daemon/identity.txt
```

Figure 5.15 - VM2 cli/resources/config/config.properties

5.3 ESA LTP CONFIGURATION (OPTIONAL EXTENSION)

ESA BP has an integrated LTP (Licklider Transmission Protocol) implementation. The configuration of LTP for ESA BP consists of a CLE, a CLA and other information contained in apposite configuration files. Here, we want to extend the previous setup of Host and VM2, by adding additional adapters based on LTP.

At present, this section has not yet a corresponding part in ESA BP official documentation.

Figure 5.16 summarizes the setup we are going to build here.

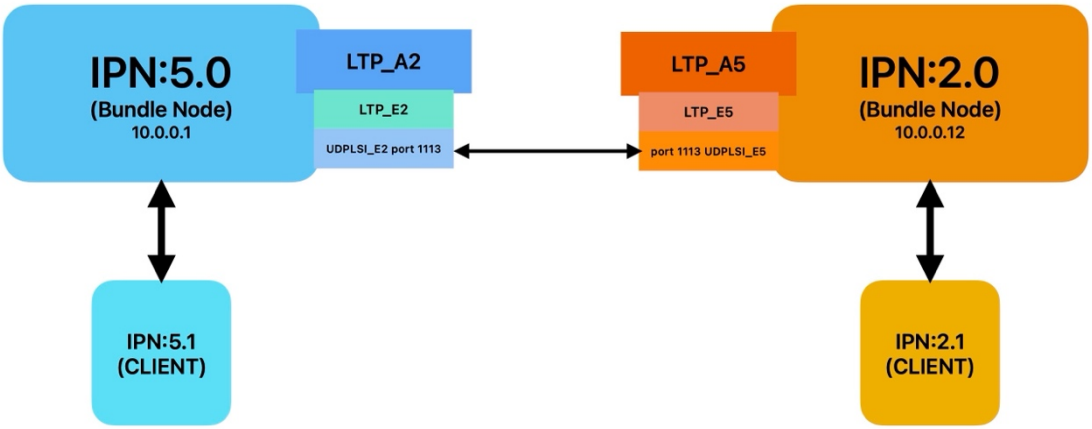

Figure 5.16 - LTP Engines Setup

Files are organized into a few directories (as usual, aac is a brief form for "esa.egos.bp.application.agent.core"):

- aac/resources/mib/LTP/
- aac/resources/mib
- aac/resources/routing-mib

5.3.1 MIB LTP

5.3.1.1 Static LTP

The "aac/resources/mib/LTP/staticLtpMIB.xml" file contains information about the local and remote LTP engines. This file can be divided in two sections: the first one contained inside the "localLTP" tag and the second one inside the "remoteLTPs" tag.

"localLTP" tag contains information such as the engine ID name, the segment size, the max number of retransmissions. We suggest the "ipn" node number as engineID, as in ION and Unibo-LTP [Unibo-LTP]. Pay attention that the "UCPReceivingPort" field is not used at all by the current implementation, thus the user should not modify it. In fact, the listening port and remote address are configured in the "CLE.dyn" file (see paragraph 5.3.2.1). In ION the aggregation properties are decided in the span section

and are different for each remote engine, In ESA LTP this is not possible as such properties are defined only once in the local engine configuration.

The "remoteLTPs" tag contains an entry for each remote LTP engine. The engineID tag specifies the id of the remote engine. The "maximumSegmentLength" specifies the maximum length of an LTP segment destined to that remote engine; if UDP is used as underlying layer (as link server adapter in ION terms), sufficiently smaller than 1500 byte is suggested to avoid IP fragmentation in the presence of usual MTUs (Maximum Transmission Unit), considering all overheads. Pay attention that the "UCPAddress" field is not used at all by the current implementation, thus the user should not modify it. The "outboundLightTime" and "inboundLightTime" tags, in ms, indicate the nominal delays between the two LTP engines (they may be different in the two directions), and are basically equivalent to ION “ranges”, which however are declared in the contact plan and are in seconds. They are used to calculate the LTP RTO (Retransmission Time Out), which by contrast to TCP RTO, is not dynamic. However, at present these ranges are not updateable, as in ION, which may be a problem when communicating with space assets moving away or towards the local node of more than 1s light time. By contrast, the ESA BP choice of ms-granularity (instead of s, as in DTNME and ION) may be advantageous when dealing with nodes relatively close to each other, such as in LEO sat communications, as shown in [Caini_2023].

Below the file for the Host:

```
<LTPConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="staticLTPMIB.xsd">
  <localLTP>
    <engineID>5</engineID>
    <checkpointRetransmissionLimit>4</checkpointRetransmissionLimit>
    <reportSegmentRetransmissionLimit>4</reportSegmentRetransmissionLimit>
    <receptionProblemLimit>4</receptionProblemLimit>
    <cancellationSegmentRetransmissionLimit>4</cancellationSegmentRetransmissionLimit>
    <retransmissionCycleLimit>4</retransmissionCycleLimit>
    <queuingAndProcessingDelay>0</queuingAndProcessingDelay>
    <SDAEnabled>false</SDAEnabled>
    <SDAAggregationSize>50000</SDAAggregationSize>
    <SDAAggregationTime>5</SDAAggregationTime>
    <handleGreenPart>true</handleGreenPart>
    <UCPReceivingPort>1113</UCPReceivingPort>
    <timeToWaitBetweenSendings>10</timeToWaitBetweenSendings>
    <monitoringInterval>1000</monitoringInterval>
    <monitoringLogInterval>3000</monitoringLogInterval>
  </localLTP>

  <remoteLTPs>

    <remoteLTP>
        <engineID>2</engineID>
        <UCPAddress>10.0.0.12:1113</UCPAddress>
        <maximumSegmentLength>1024</maximumSegmentLength>
        <outboundLightTime>1000</outboundLightTime>
        <inboundLightTime>1000</inboundLightTime>
        <queuingAndProcessingDelay>0</queuingAndProcessingDelay>
        <TxDataRate>10000</TxDataRate>
        <RxDataRate>10000</RxDataRate>
    </remoteLTP>

  </remoteLTPs>
</LTPConfig>
```

Figure 5.17 - Host aac/resources/mib/LTP/staticLtpMIB.xml

Analogously for the VM2 configuration.

```
<LTPConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="staticLTPMIB.xsd">
  <localLTP>
    <engineID>2</engineID>
    <checkpointRetransmissionLimit>4</checkpointRetransmissionLimit>
    <reportSegmentRetransmissionLimit>4</reportSegmentRetransmissionLimit>
    <receptionProblemLimit>2</receptionProblemLimit>
    <cancellationSegmentRetransmissionLimit>3</cancellationSegmentRetransmissionLimit>
    <retransmissionCycleLimit>3</retransmissionCycleLimit>
    <queuingAndProcessingDelay>0</queuingAndProcessingDelay>
    <SDAEnabled>false</SDAEnabled>
    <SDAAggregationSize>50000</SDAAggregationSize>
    <SDAAggregationTime>5</SDAAggregationTime>
    <handleGreenPart>true</handleGreenPart>
    <UCPReceivingPort>1113</UCPReceivingPort>
    <timeToWaitBetweenSendings>10</timeToWaitBetweenSendings>
    <monitoringInterval>1000</monitoringInterval>
    <monitoringLogInterval>3000</monitoringLogInterval>
  </localLTP>

  <remoteLTPs>

    <remoteLTP>
        <engineID>5</engineID>
        <UCPAddress>10.0.0.1:1113</UCPAddress>
        <maximumSegmentLength>1024</maximumSegmentLength>
        <outboundLightTime>1000</outboundLightTime>
        <inboundLightTime>1000</inboundLightTime>
        <queuingAndProcessingDelay>0</queuingAndProcessingDelay>
        <TxDataRate>10000</TxDataRate>
        <RxDataRate>10000</RxDataRate>
    </remoteLTP>

  </remoteLTPs>
```

Figure 5.18 - VM2 acc/resources/mib/LTP/staticLtpMIB.xml

5.3.1.2 Dynamic LTP

"aac/resources/mib/LTP/dynamicLtpMIB.xml" contains an entry for each remote LTP engine and specifies if transmission and reception channels are currently open. Currently ESA LTP does not fully support a Contact Plan with contact and ranges as in ION. The link availability can be dynamically modified without the need of restarting the Bundle node, so currently contacts in ESA LTP can be only ON or OFF.

For the Host we have:

```
<?xml version="1.0" encoding="UTF-8"?>
<dynamicLTPInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="dynamicLTPMIB.xsd">
  <remoteLTPs>
    <remoteLTP>
      <engineID>2</engineID>
      <RxLinkAvailability>true</RxLinkAvailability>
      <TxLinkAvailability>true</TxLinkAvailability>
    </remoteLTP>
  </remoteLTPs>
</dynamicLTPInfo>
```

Figure 5.19 - Host aac/resources/mib/LTP/dynamicLtpMIB.xml

Analogously, for VM2:

```
<?xml version="1.0" encoding="UTF-8"?>
<dynamicLTPInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="dynamicLTPMIB.xsd">
  <remoteLTPs>
    <remoteLTP>
      <engineID>5</engineID>
      <RxLinkAvailability>true</RxLinkAvailability>
      <TxLinkAvailability>true</TxLinkAvailability>
    </remoteLTP>
  </remoteLTPs>
</dynamicLTPInfo>
```

Figure 5.20 - VM2 aac/resources/mib/LTP/dynamicLtpMIB.xml

5.3.2 MIB

In this subsection we will first set LTP convergence layer elements, and then add these CLEs to convergence layer adapters.

5.3.2.1 LTP Elements - CLE

Converge layers elements need to be added to CLE.dyn, as for TCPCL (paragraph 5.1.2.3). However, the LTP convergence element is not a singleton, as for TCP, but must be combined to an UDP element (the equivalent of ION “udplsi" or “udplso” “link service adapters”) to create an ESA BP adapter.

Starting from the bottom of the protocol stack, the first element to configure is the UDP CLE, the field to modify are "ip", which contains ip address and UPD port of the remote LTP engine, and "port" the UDP port the local engine is listening on. Note that in ION the former would have been configured in the udplso instruction (one for each span, i.e. one for each remote LTP engine), and the listening port in the udplsi instruction (one for all as usual for servers).

The second element is the LTP CLE, where the field to modify is “remoteLTPEngineId”. Both the UDP and the LTP elements are declared as non-singleton.

```
   ...
    <element id="UDPLSA_E2" status="B"
implClassName="esa.egos.bp.convergence.layers.elements.CLElementUDP" singleton="false" maxqs="20"
poll="1000">
            <elementConfig>
                <ip>10.0.0.12:1113</ip>
                <port>1113</port>
            </elementConfig>
    </element>

    <element id="LTP_E2" status="B" implClassName="esa.egos.bp.convergence.layers.elements.CLElementLTP"
singleton="false" maxqs="20" poll="1000">
        <elementConfig>
            <staticMibPath>staticLtpMIB.xml</staticMibPath>
            <dynamicMibPath>dynamicLtpMIB.xml</dynamicMibPath>
            <remoteLTPEngineId>2</remoteLTPEngineId>
            <maxSenderActiveSessionsNumber>20</maxSenderActiveSessionsNumber>
        </elementConfig>
    </element>

</mibCLEsta>
```

Figure 5.21 - Host aac/resources/mib/CLE.dyn

Analogously for VM2:

```
   ...
    <element id="UDPLSA_E5" status="B"
implClassName="esa.egos.bp.convergence.layers.elements.CLElementUDP" singleton="false" maxqs="20"
poll="1000">
            <elementConfig>
                <ip>10.0.0.1:1113</ip>
                <port>1113</port>
            </elementConfig>
    </element>

    <element id="LTP_E5" status="B" implClassName="esa.egos.bp.convergence.layers.elements.CLElementLTP"
singleton="false" maxqs="20" poll="1000">
        <elementConfig>
            <staticMibPath>staticLtpMIB.xml</staticMibPath>
            <dynamicMibPath>dynamicLtpMIB.xml</dynamicMibPath>
            <remoteLTPEngineId>5</remoteLTPEngineId>
            <maxSenderActiveSessionsNumber>20</maxSenderActiveSessionsNumber>
        </elementConfig>
    </element>

</mibCLEsta>
```

Figure 5.22 - VM2 aac/resources/mib/CLE.dyn

5.3.2.2 LTP Adapters - CLA

To have a working LTP adapter the last step is creating an entry inside CLA.sta (paragraph 5.1.2.4), each LTP adapter is composed of one LTP and one UDP element. ESA BP supports also TCPCL as underlying "link service adapter", but we strongly discourage the use of it for both theoretical and practical reasons.

```
    <adapter id="LTP_A2" endpoint="ipn:2.0" poll="1000" flush="1000" maxqs="20" maxbs="50000"
bibeEnabled="false"
        cbheEnabled="true" retransmissionLimit="5" active="true">
        <elements>
            <element>LTP_E2</element>
            <element>UDPLSA_E2</element>
        </elements>
    </adapter>
</mibCLAsta>
```

Figure 5.23 - Host aac/resources/mib/CLA.sta

```
    <adapter id="LTP_A5" endpoint="ipn:5.0" poll="1000" flush="1000" maxqs="20" maxbs="50000"
bibeEnabled="false"
        cbheEnabled="true" retransmissionLimit="5" active="true">
        <elements>
            <element>LTP_E5</element>
            <element>UDPLSA_E5</element>
        </elements>
    </adapter>
</mibCLAsta>
```

Figure 5.24 - VM2 aac/resources/mib/CLA.sta

5.3.3 Routing-mib

The LTP adapters can then be added to the respective routing tables inside the file "aac/resources/routing-mib/NHT.dyn" (paragraph 5.1.3) to complete the setup of ESA LTP.

```
<mibNHTdyn>
    <endpoints>
         <endpoint endpointId="ipn:2.0">
            <adapters>
                <adapter adapterId="TCPCL_A2C" maxHopCount="10" />
                <adapter adapterId="TCPCL_A2S" maxHopCount="10" />
                <adapter adapterId="LTP_A2" maxHopCount="10" />
            </adapters>
        </endpoint>
    </endpoints>
</mibNHTdyn>
```

Figure 5.25 - Host aac/resources/routing-mib/NHT.dyn

```
<mibNHTdyn>
    <endpoints>
        <endpoint endpointId="ipn:5.0">
            <adapters>
                <adapter adapterId="TCPCL_A5C" maxHopCount="10" />
                <adapter adapterId="TCPCL_A5S" maxHopCount="10" />
                <adapter adapterId="LTP_A5" maxHopCount="10" />
            </adapters>
        </endpoint>
    </endpoints>
</mibNHTdyn>
```

Figure 5.26 - VM2 aac/resources/routing-mib/NHT.dyn

5.4 USAGE EXAMPLE

5.4.1 Start Bundle Node and CLI

Once the Bundle Node and the Client (the Command line interface) are completely configured, to run them you need to execute StartBN.sh (aac/StartBN.sh) and StartCLI.sh (cli/StartCLI.sh) script files, on both machines.

5.4.2 Send Bundle

These example uses the same setup described above, except the Host ipn node number is 104 and not 5. Before starting the transmission is opportune to control what adapters are active by using the CLI command **show adapters**; the result should be like the following:

```
ipn:104.1> show adapters
ipn:104.1>
================================================================================
 [Adapter Status]
--------------------------------------------------------------------------------
 Adapter              Connection           Activation
--------------------------------------------------------------------------------
LTP_A2               CONNECTED            ACTIVE
TCPCL_A101C          NOT_CONNECTED        INACTIVE
TCPCL_A101S          NOT_CONNECTED        INACTIVE
TCPCL_A2C            NOT_CONNECTED        INACTIVE
TCPCL_A2S            NOT_CONNECTED        INACTIVE
UDPCL_A101           NOT_CONNECTED        INACTIVE
--------------------------------------------------------------------------------
```

Figure 5.27 -ESA BP CLI Host, show adapters command output.

The adapters can then be activated or deactivated by using the command: **activate/deactivate adapter-name**. With the command **activate TCPCL_A2S,** we start the TCPCL adapter designed to listen for incoming connections from node 2 (the server devoted to 2). The output of this command his shown in the logs of the Bundle Node (Figure 5.28): after a few seconds, the Adapter confirms that a TCPCL connection is going to be established with 10.0.0.12:4556; this connection is actually started by the previously activated TCPCL_A104C, on VM2, which can only succeed after the server is activated.

```
[04:02:26 PM] INFO: TCPCL_A2S: TCPCL: Waiting for incoming connection.
[04:02:33 PM] INFO: TCPCL_A2S: TCPCL: Establishing TCPCL connection with: 10.0.0.12:4556
         Acknowledge Segments:    true
         Reactive Fragmentation:  true
         Bundle Refusal Capable:  true
         Use Keep Alive Messages: true
         Keep Alive Interval:     10
         Use Length Messages:     true
```

Figure 5.28 - ESA BP Bundle Node logs Host, TCPCL_A2S activation

To send a bundle from the CLI interface of node 104 to the corresponding of node 2 we then use the command **send** with the **-d** option to specify the destination EID (ipn:2.1) and **-adu** to specify the payload, as shown below.

```
ipn:104.1> send -d=ipn:2.1 -adu="Hello Mars!"
ipn:104.1> Apr 18, 2024 4:03:30 PM esa.egos.bp.cli.CLICallback localBundleId
FINE: Bundle ipn:104.1:766764210.1 Created - BundleCreationTime[Thu Apr 18 16:03:30 CEST 2024 1]
Bundle ipn:104.1:766764210.1 Created - BundleCreationTime[Thu Apr 18 16:03:30 CEST 2024 1]
```

Figure 5.29 - ESA BP CLI Host, Sending a Bundle

The output on the receiving machine (see below), confirms reception.

```
ipn:2.1> Received bundle BundleCreationTime[Thu Apr 18 16:03:30 CEST 2024 1] source ipn:104.1 - Deliver
y Time: Thu Apr 18 16:03:31 CEST 2024 Remaining LifeTime: 59 - adu - Hello Mars!
```

Figure 5.30 - ESA BP CLI VM2, Bundle Reception Output

For more examples, including interoperability tests, see chapter 6.

6 TESTS

6.1 NETWORK TOPOLOGY AND TESTING INSTRUMENTS

To setup the testbed we used Virtualbricks [Virtualbricks], a network emulator and VM manager developed by UniBo. We chose a multi-node layout (Figure 6.1), with each node running on a different Debian 11 machine, either virtual (VM1, VM2) or physical (Host).

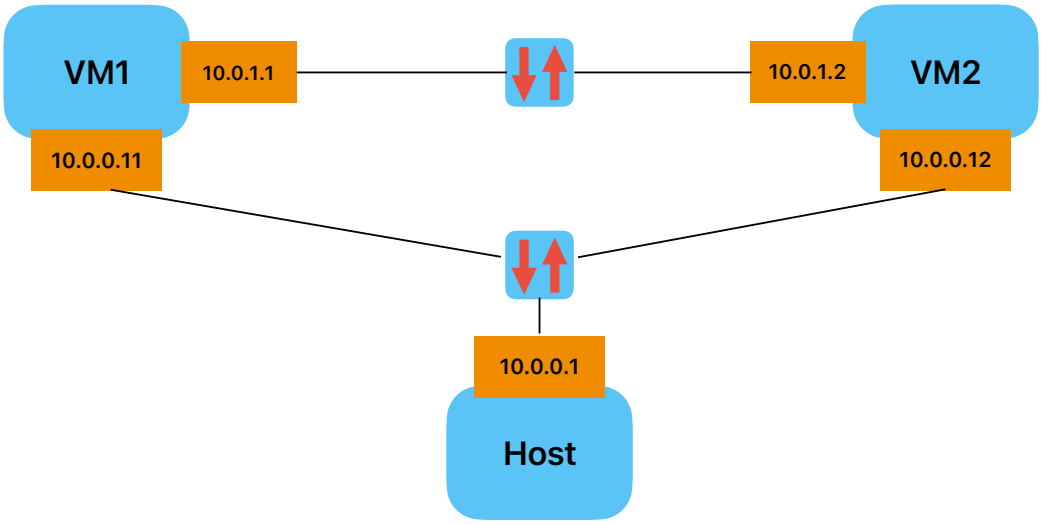

Figure 6.1 - Network topology used in tests.

The testbed is composed of two subnets. The former, **10.0.0.0/24**, connects the host machine to all virtual machines, and enables the control of the virtual machines by SSH (Secure Shell) without interfering with test traffic. The latter, **10.0.1.0/24,** connects VM1 and VM2 and is used to transfer bundles. All the subnets are equipped with a network emulator inside to introduce challenges such as (propagation delay, high packet loss rate, etc.).

The ipn Node IDs associated to VMs are respectively: **ipn:1.0**, **ipn:2.0**. DTN nodes communicate bidirectionally by means of either TCPCLv3 or LTP over UDP.

6.1.1 Traffic Capture and Analysis

Frames between nodes were captured with tcpdump (version 4.99.0, libpcap version 1.10.0 with TPACKET_V3) [tcpdump]. The .dmp files produced were then analyzed with Wireshark (version 4.2.0) [Wireshark]

6.1.2 LTPDrop

LTPDrop [LTPDrop] is a Python command-line tool developed during this thesis to introduce deterministic losses of LTP segments. It can be descrived as a basic Application Level Gateway for LTP, with rules to filter certain LTP segments. This feature's utility emerged soon after the a few preliminary tests. Virtualbrick's channel emulator can randomly drop frames, which results in corresponding LTP segment random losses. Due to their randomness, a variable number of attempts are required to simulate corner cases, such as the loss of the first data segment (see below) of a session. Loss randomness produces a large amount of redundant data which then must be manually inspected and filtered, with a significant waste of time. By contrast, LTPDrop allows the user to introduce LTP segment selective losses (they are filtered on the basis of their LTP "type" field and other parameters), thus making possible to deterministically generate even the most unlikely corner cases. Although primarily designed to this aim, LTPDrop can also introduce random losses on request. LTPDrop produce very informative logs (.csv files) for non-real time inspection or to be used as an input to the "ltp_performance_analyzer", [Zappacosta_202x], a Python tool for macroscopic LTP session analysis. This tool was designed to analyze either Unibo-LTP logs, or Wireshark files, the former option being preferable as Wireshark files are much larger. LTPDrop has been designed to produce the same kind of logs of Unibo-LTP, thus extending the former option to all LTP implementations, as logs are produced by traffic inspection and not by the LTP implementation.

LTPDrop functioning is conceptually simple, as it essentially works as a "man-in-the-middle" between two LTP engines: it exposes a UDP socket, from which it expects to receive UDP packets encapsulating LTP segments; then, for each incoming LTP segment it inspects the type field and evaluates whether the segment must be

forwarded or dropped, based on the arguments the user gave when lunching the program. The insertion of the LTPDrop cannot be transparent to both LTP engines, as IP addresses and UDP ports must be carefully set, and various layouts are possible, with or without a dedicated machine for LTPDrop. Figure 6.2 shows a possible setup with LTPDrop on the sender machine where LTPDrop and LTP Engine 1 communicate via the IP loopback interface.

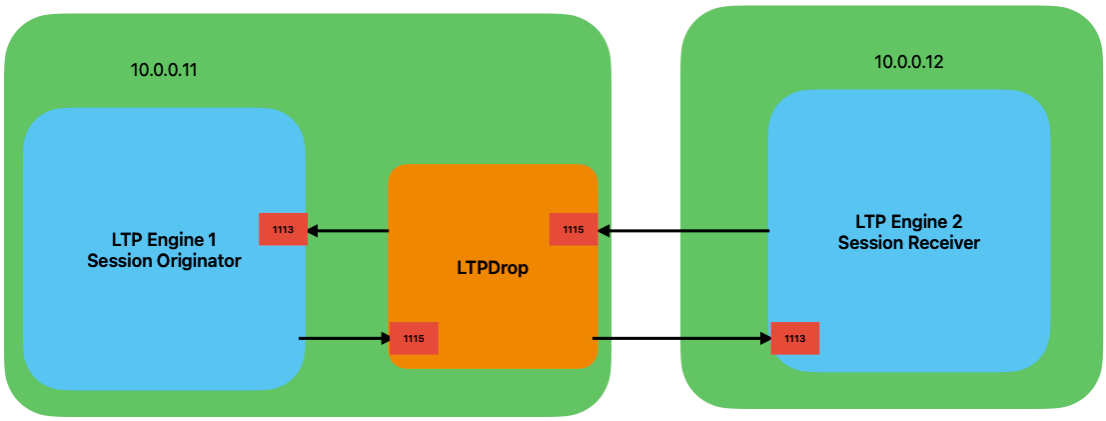

Figure 6.2 - LTPDrop Originator Side Setup

6.2 BP Interoperability Tests

To test ESA BP interoperability with other BP implementations and its compliance with the standard we choose ION as ESA BP counterpart, as ION can be considered as a sort of reference implementation for the bundle protocol having being developed by Scott Burleigh, who is also the main author of the BPv7 specifications [RFC9171]. In the two following tests ION is executed on VM1 and ESA BP on VM2, connected via TCPCLv3. The communication channel is ideal without any delay or losses, unless otherwise specified.

6.2.1 ION (source) - ESA BP (destination)

We send a simple bundle from an ION to an ESA BP node, ipn:1.0 and ipn:2.0, respectively. In this test ESA BP uses the TCPCL_A1S adapter, a TCPCL responder (for more information see 4.2 and 5.1.2.3).

Command on ION node: ***bptrace ipn:1.1 ipn:2.1 ipn:1.0 60 0.0 "Ground Control to Voyager 2"*** (Figure 6.3 - VM1, bptrace command and ION debug letters.

```
student@vm1:~/ion$ bptrace ipn:1.1 ipn:2.1 ipn:1.0 60 0.0 "Ground Control to Voyager 2"
abcstudent@vm1:~/ion$
```

Figure 6.3 - VM1, bptrace command and ION debug letters.

The **a,b,c** letters are dubug characters inserted on request by ION, respectively meaning that the bundle has been queued for forwarding, queued for transmission and popped from the transmission queue, which means that the bundle transmission is terminated from ION side. However, the logs on VM2(Figure 6.4) reveal an error in the bundle reception.

```
[09:40:51 AM] SEVERE: [esa.egos.bp.processing.BundleProcessorManager] Bundle cannot be decoded by BPv6 or BPv7 decoders.
esa.egos.bp.processing.exceptions.BundleBlockException: CRC_16_X25 Check failed for 'EXTENSION BLOCK' block: Received '0' Calculated '30959'
```

Figure 6.4 -ESA BP logs on VM2 after the reception of a bundle from ION.

This error is generated by discrepancies in the CRC found in the Extension block and the one calculated by ESA BP. After analyzing the traffic with Wireshark, we noted that ESA BP bundles had the same CRC type, CRC_16, set on all blocks, while ION uses this CRC only for the primary block, while all other blocks have the CRC flag set to NO_CRC. We assumed that this was the cause of bundle rejection by ESA BP, and to test this bug hypothesis, we modified the ION code (with Burleigh's help) to add CRC_16 to every block. In file "*ion-open-source-4.1.2/bpv7/library/ext/bpextensions.c*" we modified each instance of NoCRC with X25CRC16 in the lines ranging from 241 to 252, and in the file "*ion-open-source-4.1.2/bpv7/library/libbpP.c*" we changed at line 63 the value of the macro PAYLOAD_BLOCK_CRC_TYPE from 0 to 1 (see paragraph 2.3 for more information). We then repeated the previous experiment by launching the command below from the ION node:

bptrace ipn:1.1 ipn:2.1 ipn:1.0 60 0.0 "Can you hear me voyager 2" (Figure 6.5)

```
student@vm1:~/ion$ bptrace ipn:1.1 ipn:2.1 ipn:1.0 60 0.0 "Can you hear me voyager 2"
abcstudent@vm1:~/ion$
```

Figure 6.5 - VM1, ION modified version bptrace command and ION debug letters.

As we can see from the ESA BP logs (Figure 6.6), this time the bundle is correctly delivered to the CLI (Figure 6.7), thus confirming our thesis.

```
[10:20:46 AM] INFO: TCPCL_A1S: TCPCL: Establishing TCPCL connection with: 10.0.1.1:4556
         Acknowledge Segments:    true
         Reactive Fragmentation:  false
         Bundle Refusal Capable:  false
         Use Keep Alive Messages: true
         Keep Alive Interval:     10
         Use Length Messages:     false
[10:22:07 AM] INFO: Received bundle 1:
----- Source: ipn:1.1
----- BundleCreationTime[Sat May 04 10:22:07 CEST 2024 0]

[10:22:08 AM] WARNING: Deleting bundle [1] with reason: NO_RETENTIONS
```

Figure 6.6 - ESA BP logs VM2, TCPCL_A1S connection established and bundle reception.

```
ipn:2.1> Received bundle BundleCreationTime[Sat May 04 10:22:07 CEST 2024 0] source ipn:1.1 - Delivery Time: Sat
 May 04 10:22:07 CEST 2024 Remaining LifeTime: 60 - adu - Can you hear me voyager 2
```

Figure 6.7 - ESA BP CLI VM2, bundle delivery.

However, to guarantee interoperability we need to check whether ION modified version is still compatible with himself. To test this, we used ION to send a bundle to itself, particularly to its bprecv instance (for more information see 6.2.2). We used the following command:

bptrace ipn:1.1 ipn:1.2 ipn:1.0 60 0.0 "JPL to Houston"

```
student@vm1:~/ion$ bptrace ipn:1.1 ipn:1.2 ipn:1.0 60 0.0 "JPL to Houston"
abcdstudent@vm1:~/ion$ efgstgshgscat ion.log
```

Figure 6.8 -VM1, ION modified sending a bundle to itself and ION debug letters

From the above figure we can see that the transmission does not terminate correctly, in fact no letter "z" appears in the debug letters, meaning that the bundle was never delivered to an application.

```
[2024/05/23-09:18:51] at line 4895 of ltp/library/libltpP.c, Opened import session. (3)
[2024/05/23-09:18:51] [i] Span to engine 1 (max BER 0.000100, max xmit segment size 1024
size 102): xmit segment loss rate 0.559294, recv segment loss rate 0.078376, max timeout
[2024/05/23-09:18:51] [i] Max report segments = 2 for red part length 102, max segment s
ss rate 0.078376.
[2024/05/23-09:18:51] at line 4439 of ltp/library/libltpP.c, Sending RS: 0 to 102, ckpt
[2024/05/23-09:18:51] at line 4487 of ltp/library/libltpP.c, Reporting all data receivec
[2024/05/23-09:18:51] [?] CRC check failed for extension block.
```

Figure 6.9 - VM1, ion.log of a loopback modified ION transmission

In the file ion.log (Figure 6.9) we can see that a LTP session was opened to send the bundle, which was correctly delivered to the bundle node. However, the bundle was discarded due to a CRC inconsistency as we can see in the last line of the figure above. We then checked the bundle with Wireshark (Figure 6.10) looking for any peculiarities or errors in the CRCs, but we found none.

```
DTN Bundle Protocol Version 7, Src: ipn:1.1, Dst: ipn:1.2, Time: 769763930983,
  Indefinite Array: 9f
> Primary Block, CRC Type: CRC-16
  [Bundle Identity: Source: ipn:1.1, DTN Time: 769763930983, Seq: 0]
  [Destination Service: 2]
> Canonical Block: Previous Node, Block Num: 2, CRC Type: CRC-16
> Canonical Block: Type 193, Block Num: 3, CRC Type: CRC-16
> Canonical Block: Bundle Age, Block Num: 4, CRC Type: CRC-16
v Canonical Block: Payload, Block Num: 1, CRC Type: CRC-16
    Type Code: Payload (1)
    Block Number: 1
  > Block Flags: 0x0000000000000001, Replicate block in fragment
    CRC Type: CRC-16 (1)
    [Block Type-Specific Data Length: 15 octets]
    Block Type-Specific Data: 4a504c20746f20486f7573746f6e00
    CRC Field Integer: 0x9860 [correct]
    [CRC Status: Good]
  Indefinite Break: ff
```

Figure 6.10 -Wireshark dissection of the bundle, all the CRC, although not shown, are correctly formatted

We interacted with ION developers (Github_2024) that came up with a temporary fix by modifying the block processing control flag in file "*ion-open-source-4.1.2/bpv7/library/bpP.h*" from 475239 to 23. This change was tested and solved the issue we encountered.

6.2.2 ESA BP (source) - ION (destination)

This test is the dual of the previous one. We send a simple bundle from ipn:2.1 containing a short string. In this test ESA BP uses TCPCL_A1C adapter, a TCPCL initiator (for more information see 4.2 and 5.1.2.3). A bprecv instance registered at ipn:1.2 is running on the ION node.

ESA BP Command: ***send -d=ipn:1.2 -adu="Ground Control to Voyager 1"*** (Figure 6.11, for more information about send command see paragraph 5.4.2).

```
ipn:2.1> send -d=ipn:1.2 -adu="Ground Control to Voyager 1"
ipn:2.1> Bundle ipn:2.1:767974451.1 Created - BundleCreationTime[Thu May 02 16:14:11 CEST 2024 1]
```

Figure 6.11 - ESA BP CLI VM2, send bundle to ION on VM1 output

In the Bundle node logs, we see the TCPCL connection-established messages, and the send command outputs:

```
[04:13:18 PM] INFO: TCPCL_A1C: TCPCL: Activating TCPCL Sender
[04:13:19 PM] INFO: TCPCL_A1C: TCPCL: Establishing TCPCL connection with: 10.0.1.1:4556
         Acknowledge Segments:    true
         Reactive Fragmentation:  false
         Bundle Refusal Capable:  false
         Use Keep Alive Messages: true
         Keep Alive Interval:     10
         Use Length Messages:     false
[04:14:11 PM] INFO: Successful Send bundle 1:
----- Source: ipn:2.1
----- BundleCreationTime[Thu May 02 16:14:11 CEST 2024 1]

[04:14:11 PM] WARNING: Deleting bundle [1] with reason: NO_RETENTIONS
```

Figure 6.12 - ESA BP Bundle Node logs VM2, TCPCLv3 Initiator Connection Established output and send command output

The output on the terminal on ION side is just “yz” (Figure 6.13): the letter **y** means the bundle was accepted and **z** that the bundle was queued to be delivered to the wanted application, in this case bprecv. This application creates a new file (testfile#), each time a bundle is received, which contains the bundle payload. We can see this in ion.log (Figure 6.14).

```
student@vm1:~/ion$ yz
```

Figure 6.13 - VM1, ION debug letters

```
[2024/05/02-16:14:11] [i] bprecvfile is creating 'testfile1', size 27.
[2024/05/02-16:14:11] [i] bprecvfile has created 'testfile1', size 27.
```

Figure 6.14 - ion.log VM1, bprecv output on ion.log.

Last, as a further proof that the ESA BP bundle is fully compliant with BPv7 specification, we can see that Wireshark correctly dissects it (Figure 6.15).

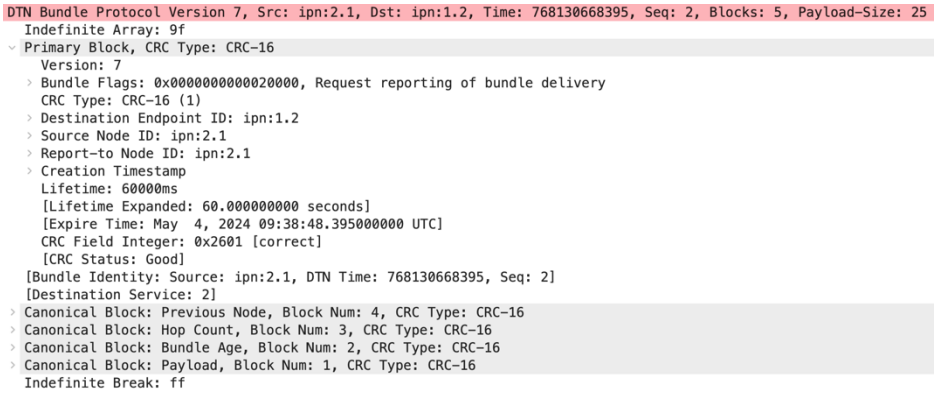

Figure 6.15 - Wireshark, Bundle dissection from ESA BP to ION

6.2.3 Bundle Status Report

Bundle Status Reports (BSRs) is a particular bundle generated by the administrative element for status reporting purposes. The four status that can be reported are:

- Deletion of the bundle
- Reception of the bundle by a node
- Delivery of the bundle to an application
- Forwarding of the bundle to another node

Whenever one of this condition occurs and the corresponding flags are active, an administrative record will be generated, incapsulated inside a bundle and sent to the "report to" EID.

To test the BSR generated by ESA BP we decided to send a bundle from ESA (VM2) to unmodified ION (VM1) with the forwarding flag activated, to generate a BSR directed to ION. We used the following command:

ESA BP command: ***send -d=ipn:1.2 -adu="A small step..." -r=ipn:1.0 -fr -str***

"r" option is used to indicate the report to EID, "fr" option is used to indicate we want to generate a forwarded BSR, "str" option to indicate we want the timestamp the forwarding occurred. As we can see from ESA BP logs (**Errore. L'origine riferimento non è stata trovata.**) two bundle are generated and sent, respectively the data bundle generated by the CLI (source: ipn:2.1) and the BSR generated by the bundle node (source: ipn:2.0).

```
[09:08:32 AM] INFO: Successful Send bundle 1:
----- Source: ipn:2.1
----- BundleCreationTime[Thu May 23 09:08:31 CEST 2024 1]

[09:08:32 AM] WARNING: Deleting bundle [1] with reason: NO_RETENTIONS
[09:08:32 AM] INFO: Successful Send bundle 2:
----- Source: ipn:2.0
----- BundleCreationTime[Thu May 23 09:08:32 CEST 2024 2]

[09:08:32 AM] WARNING: Deleting bundle [2] with reason: NO_RETENTIONS
```

Figure 6.16 - VM2 ESA BP logs, bundle 1 is the bundle we sent, bundle 2 is the forwarding BSR

The two bundle arrive to the destination, but the BSR is incorrectly formatted: this is reported both by ION in the file ion.log (**Errore. L'origine riferimento non è stata trovata.**) and in the Wireshark dissection of the bundle (**Errore. L'origine riferimento non è stata trovata.**).

```
[2024/05/23-09:08:32] [i] bprecvfile is creating 'testfile1', size 15.
[2024/05/23-09:08:32] [i] bprecvfile has created 'testfile1', size 15.
[2024/05/23-09:08:32] [?] CBOR array size is wrong: 5
[2024/05/23-09:08:32] [?] Can't decode admin record array open.
```

Figure 6.17 - VM1, ion.log line 3-4 show the reaction to the malformed bundle

```
BPv7 Administrative Record
 [Expert Info (Warning/Malformed): Array has 5 items, should be within [1, 2]]
    [Array has 5 items, should be within [1, 2]]
    [Severity level: Warning]
    [Group: Malformed]
```

Figure 6.18 - Wireshark dissection of malformed BSR

Decoding the CBOR (Figure 6.19) we discovered the following format errors:

- Instead of creating an array of 2 elements, with the second element being an array of 4 elements, ESA BP generates an array of 5 elements.
- The DTN time in the Bundle Status Item is reported only in seconds, instead of the expected milliseconds.
- The BSR Reason Code is 7, which indicates that there was no timely contact with next node on route, is incorrect. It is not true in this case and does not make sense in the general case of a forwarded BSR.
- The Creation Timestamp, which is needed to identify the bundle that generated the BSR, is missing a piece. It should consist of an array of 2 elements containing the DTN Time, this time reported in milliseconds, and the sequence number, which is missing.

```
[   1(Administrative Record Type),
    [   [false](Bundle Status Item Received),
        [true, 770224284 (Bundle Status Item Forwarded, DTN Time)],
        [false] (Bundle Status Item Delivered),
        [false] (Bundle Status Item Deleted)
    ],
    7 (BSR Reason Code),
    [   2 (EID Scheme Name, 2 is IPN),
        [   2 (Node Number),
            1 (Service Number)
        ]
    ],
    770224284434 (Creation timestamp)
]
```

Figure 6.19 - ESA BP Administrative Record CBOR Decoding, this record is incorrectly formatted

6.2.4 Comments on BP test results

From the test displayed above we can conclude that ESA BP can correctly format application bundles and its TCPCLv3 implementation is interoperable with ION's one. However, on the receiver side ESA BP does not fully comply with the standard, as it does not accept a bundle unless all its blocks carry the same CRC. The co-supervisor, Camillo Malnati, author of ESA BP, was informed of this problem, whose fix should be simple.

6.3 ESA LTP TESTS

This section describes the tests that were carried out to verify ESA LTP compliance to the LTP standard; it comprises both ESA LTP tests and interoperability tests with ION. In these latter tests, we used the ION version with CRC modified to bypass the BP bug, and we also introduced some modification to LTP files:

- in ion-open-source-4.1.2/ltp/library/libltpP.c line 34 we changed the macro LTPDEBUG value to 1, to have more extensive logs.
- in ion-open-source-4.1.2/ltp/library/ltpP.h line 63 we changed the macro CLOSED_EXPORTS_ENABLED value to 1, (see 6.3.3 for more information).
- in ion-open-source-4.1.2/ltp/library/ltpP.h line 96 we changed the macro DEFAULT_MAX_BER from (.000001) to (.0001). In ION the number of retransmissions is calculated using ranges in the contact plan and the BER value, unlike in ESA, where it is set in the configuration. The aim of this change was to make ION perform at least 3 or 4 LTP segment retransmissions before cancelling the session.

All tests were carried out with LTP Red.

6.3.1 Test 1, ideal case (ESA BP-> ESA BP)

We send from ipn:1.1 to ipn:2.1 a file of 10 kB, by requesting the generation of a Bundle Status Report “delivered” (report to: ipn:1.1, the sender).

Command (on VM1, ipn:1.0): ***sendf -d=ipn:2.1 -adu="file10K.txt***

Channel: ideal, no losses, no delay

The transmission terminates correctly with the delivery of the bundle to ipn:2.1, confirmed by a RS, acknowledge by a RA.

```
LTP          1055 Session 1/63202000, Red data, range 0-1001 [Reassembled in #11]
LTP          1055 Session 1/63202000, Red data, range 1002-2002 [Reassembled in #11]
LTP          1055 Session 1/63202000, Red data, range 2003-3003 [Reassembled in #11]
LTP          1055 Session 1/63202000, Red data, range 3004-4004 [Reassembled in #11]
LTP          1055 Session 1/63202000, Red data, range 4005-5005 [Reassembled in #11]
LTP          1055 Session 1/63202000, Red data, range 5006-6006 [Reassembled in #11]
LTP          1055 Session 1/63202000, Red data, range 6007-7007 [Reassembled in #11]
LTP          1055 Session 1/63202000, Red data, range 7008-8008 [Reassembled in #11]
LTP          1055 Session 1/63202000, Red data, range 8009-9009 [Reassembled in #11]
LTP          1055 Session 1/63202000, Red data, range 9010-10010 [Reassembled in #11]
BPv7          377 Payload-Size: 10240
LTP            60 Session 1/63202000, Report segment, range 0-10330, gaps: 0, gap total: 0
LTP            51 Session 1/63202000, Report ack segment
```

Figure 6.20 - Wireshark dissection of a LTP session and Bundle Delivery without any losses.

6.3.2 Test 2, losses (ESA BP-> ESA BP)

The same as before, but with a file of 20 kB (twice the previous dimension) and a channel with 10% losses on the forward direction (it is ideal only in the return direction, the losses were generated with the channel emulator).

Command: ***sendf -d=ipn:2.1 -adu="file20K.txt"***

Channel: PER 10% (VM1->VM2), PER 0% (VM2->VM1), no delay

```
20 … … … LTP          1055 Session 1/63202003, Red data, range 19018-20017 [Reassembled in #21]
21 … … … BPv7          609 Payload-Size: 20480
22 … … … LTP            74 Session 1/63202003, Report segment, range 0-20568, gaps: 3, gap total: 3002
23 … … … LTP            51 Session 1/63202003, Report ack segment
24 … … … LTP          1055 Session 1/63202003, Red data, range 4005-5005 [Retransmission]
25 … … … LTP          1055 Session 1/63202003, Red data, range 13014-14014 [Retransmission]
26 … … … LTP          1059 Session 1/63202003, Red data, range 19018-20017 [Retransmission]
27 … … … LTP            62 Session 1/63202003, Report segment, range 0-20568, gaps: 0, gap total: 0
28 … … … LTP            51 Session 1/63202003, Report ack segment
```

Figure 6.21 -Wireshark dissection of the arrival of a CP (segment n. 21) and the retransmission of lost segment (segment n. 24-26)

Three LTP data segments are discarded by the channel emulator, they are claimed in the Report Segment and are correctly retransmitted. The session concludes as expected with a RA after the 1 RTT penalty due to losses.

6.3.3 Test 3, bundle fragmentation and losses (ESA BP -> ESA BP)

As in test 2, but with a file of 50 kB. As before, the channel has 10% losses on the forward direction, is ideal in the return direction.

Command***: sendf -d=ipn:2.1 -adu="file50K.txt"***

Channel: PER 10% (Channel Emulator Losses) (VM1->VM2), PER 0% (VM2->VM1), no delay

The 50 kB file is split into two bundle fragments by the BP Agent, which are then sent in two separate LTP blocks (sessions 2008 and 2009); as a result of fragmentation, the second block is much smaller (1359 Bytes of payload) and its session (2009) finishes first. From Wireshark trace we discovered that when there are two parallel sessions using the same UDP socket, as in this case, ESA BP interlaces data segments of both sessions (see , while ION would send first all segments of the first session and then those of the second one, thus interlacing only control segments and retransmitted segments. Both policies are compliant with RFC5326, but it was worth mentioning the difference. To make the analysis clearer, we will consider the two sessions separately, starting by session 2009, the simplest one (Figure 6.22, obtained by filtering out 2009 segments from the original dump).

Session 2009 does not incur in any data losses (only two segments were transmitted, corresponding to frames 3 and 5) and concludes on the sender side with an RA (frame 10), which is however presumably lost, as the receiver sends three more RS (frames 67, 68, 73). These RSs do not trigger any RA, thus not complying with the standard [RFC_5326]. After these three RSs, the RTO on the receiver presumably fires, and a first CR segment is sent (frame 74). This CR is not acked by a CAR, as it should, which result after one RTO into the retransmissions of a new CR (75) and again for the last CR (76). As there are no more segments, we assume that the Rx session has closed unilaterally as the max number of CR retransmissions has been reached. In this case, there is no harm (all data segments have been received at first attempt) but the Rx session has lasted a lot more than necessary and a lot of redundant segments have been sent. This problem is due to the fact that signaling segments of a closed sessions are not confirmed at reception, as they should [RFC5326], but silently discarded. The problem

is well known, as it was found also in ION. To fix it in ION it is enough to set the CLOSED_EXPORTS_ENABLED macro, which is disabled by default. The problem, its impact on performance and the fix are described in full details in [Alessi_2019].

```
 3 … … … LTP          1055 Session 1/63202009, Red data, range 0-1001 [Reassembled in #5]
 5 … … … BPv7          414 Payload-Size: 1264 (fragment)
 7 … … … LTP            60 Session 1/63202009, Report segment, range 0-1358, gaps: 0, gap total: 0
10 … … … LTP            51 Session 1/63202009, Report ack segment
67 … … … LTP            60 Session 1/63202009, Report segment, range 0-1358, gaps: 0, gap total: 0
68 … … … LTP            60 Session 1/63202009, Report segment, range 0-1358, gaps: 0, gap total: 0
73 … … … LTP            60 Session 1/63202009, Report segment, range 0-1358, gaps: 0, gap total: 0
74 … … … LTP            60 Session 1/63202009, Cancel segment
75 … … … LTP            60 Session 1/63202009, Cancel segment
76 … … … LTP            60 Session 1/63202009, Cancel segment
```

Figure 6.22 - Wireshark dissection of session 2009, the RA (frame 10) is still visible as the dump was captured on VM1.

Now we can move to session 2008 (Figure 6.23, obtained by filtering out session 2008 segments), which is composed of 55 LTP data segments and incurs in losses in the first transmission; reTx segments are sent and one is lost again (frame 60); this segment is thus transmitted for the third time (frame 64). This behavior is correct, but the section is canceled by the receiver CR, which states that the retransmission limit was exceeded (RELXC). The CR is correctly acknowledged by a CAR (by contrast with session 2009, Tx session 2008 is still open at CR arrival).

```
55 … … … BPv7           81 Payload-Size: 49936 (reassembled)
56 … … … LTP            80 Session 1/63202008, Report segment, range 0-50028, gaps: 4, gap total: 4002
57 … … … LTP            51 Session 1/63202008, Report ack segment
58 … … … LTP          1055 Session 1/63202008, Red data, range 1002-2002 [Retransmission]
59 … … … LTP          1055 Session 1/63202008, Red data, range 6007-7007 [Retransmission]
60 … … … LTP          1055 Session 1/63202008, Red data, range 32018-33017 [Retransmission]
61 … … … LTP          1059 Session 1/63202008, Red data, range 43018-44017 [Retransmission]
62 … … … LTP            68 Session 1/63202008, Report segment, range 0-50028, gaps: 1, gap total: 1000
63 … … … LTP            51 Session 1/63202008, Report ack segment
64 … … … LTP          1059 Session 1/63202008, Red data, range 32018-33017 [Retransmission]
65 … … … LTP            60 Session 1/63202008, Cancel segment
66 … … … LTP            49 Session 1/63202008, Cancel ack segment
```

Figure 6.23 -Wireshark dissection of the last LTP segments from session 2008.

6.3.4 Test 4, ESA BP interoperability with an ION receiver, without and with losses (ESA BP VM1-> ION VM2)

This is the first LTP test focused on ESA LTP interoperability with ION implementation of LTP. To speed up the analysis, this time many bundles of different dimensions were consecutively sent (and on different channels conditions), not just one as in previous tests.

ESA BP command (attempts n: 1, 3): ***sendf -d= ipn:2.1 -adu="file40K.txt"***

ESA BP command (attempts n: 2, 4-7): ***sendf -d= ipn:2.1 -adu="file50K.txt"***

Channel (attempts n: 1-2): ideal, no losses, no delay

Channel (attempts n:3-7): PER 10%(Channel Emulator Losses)(VM1->VM2), PER 0%(VM2->VM1), no delay

All the test were successful, and no unforeseen circumstances were encountered.

```
41 … … … LTP          1055 Session 1/63202020, Red data, range 40018-41017 [Reassembled in #42]
42 … … … BPv7           86 Payload-Size: 40960
43 … … … LTP            62 Session 1/63202020, Report segment, range 0-41046, gaps: 0, gap total: 0
44 … … … LTP            51 Session 1/63202020, Report ack segment
```

Figure 6.24 - Wireshark dissection of the last LTP segments exchanged in attempt n.1

```
420 … … … BPv7           81 Payload-Size: 49936 (reassembled)
421 … … … LTP            89 Session 1/63202030, Report segment, range 0-50028, gaps: 6, gap total: 5989
422 … … … LTP            51 Session 1/63202030, Report ack segment
423 … … … LTP          1055 Session 1/63202030, Red data, range 4005-5005 [Retransmission]
424 … … … LTP          1055 Session 1/63202030, Red data, range 6007-7007 [Retransmission]
425 … … … LTP          1055 Session 1/63202030, Red data, range 25018-26017 [Retransmission]
426 … … … LTP          1055 Session 1/63202030, Red data, range 31018-32017 [Retransmission]
427 … … … LTP          1055 Session 1/63202030, Red data, range 37018-38017 [Retransmission]
428 … … … LTP          1046 Session 1/63202030, Red data, range 49018-50004 [Retransmission]
429 … … … LTP            66 Session 1/63202030, Report segment, range 0-50004, gaps: 1, gap total: 1001
430 … … … LTP            51 Session 1/63202030, Report ack segment
431 … … … LTP          1059 Session 1/63202030, Red data, range 6007-7007 [Retransmission]
432 … … … LTP          1059 Session 1/63202030, Red data, range 6007-7007 [Retransmission]
433 … … … LTP            62 Session 1/63202030, Report segment, range 0-50028, gaps: 0, gap total: 0
434 … … … LTP            51 Session 1/63202030, Report ack segment
```

Figure 6.25 - Wireshark dissection of the last LTP segments exchanged in attempt n. 7. Multiple segments are correctly retransmitted.

6.3.5 Test 5, interoperability with an ION sender, without and with losses (ION VM1-> ESA BP VM2)

This time the sender ran ION and the receiver ESA BP. As before, many bundles were consecutively sent, not just one as in previous tests. Traffic was captured on both sides.

ION command (attempt n: 1):***bptrace ipn:1.1 ipn:2.1 ipn:1.0 60 0.0 "@file40K.txt"***

ION command (attempts n: 2-8):***bptrace ipn:1.1 ipn:2.1 ipn:1.0 60 0.0 "@file50K.txt"***

Channel: PER 10%(VM1->VM2), PER 0%(VM2->VM1), no delay

All tests were successful, except the first, where an important anomaly occurred. In attempt n.1 (session 2), the first and twelfth data segments were lost, as they did not appear in the dump captured on VM2. In Figure 6.26 (an excerpt of the full dump) we can notice that the first data segment received starts at 1015, instead of 0.

```
1 … … … LTP          1065 Session 1/2, Red data, range 1015-2028 [Unfinished LTP Block]
2 … … … LTP          1065 Session 1/2, Red data, range 2029-3042 [Unfinished LTP Block]
3 … … … LTP          1065 Session 1/2, Red data, range 3043-4056 [Unfinished LTP Block]
```

Figure 6.26 -Wireshark dissection of the LTP traffic received by VM2, the first LTP segment was discarded by the channel emulator as the data range starts from 1015, instead of 0.

The problem is that the loss of the first segment causes the generation of a poorly formatted RS, which declares 1015 bytes as lower bound instead of 0, (Figure 6.27).

```
Licklider Transmission Protocol, Session: 1/2
 › LTP Header
 ⌄ Report Segment
     Report serial number: 12645
   › Checkpoint serial number: 15480
     Upper bound: 41037 bytes
     Lower bound: 1015 bytes
     [Report bound length: 40022 bytes]
     Reception claim count: 2
   › [Reception gap: 1015-2029 (1015 bytes)]
   › Reception claim: 2030-12169 (10140 bytes)
   › [Reception gap: 12170-13183 (1014 bytes)]
   › Reception claim: 13184-42051 (28868 bytes)
     [Reception gap: 42052-41036 (18446744073709550601 bytes)]
     [Total gap length: 1014 bytes]
```

Figure 6.27 - Wireshark dissection of the malformed RS, the Lower Bound should be 0 instead of 1015, the last gap is clearly wrong, as it starts with a value greater than its supposed end.

As a result of this error, all “claims” (ranges of consecutively received bytes) are interpreted by Wireshark as shifted by the content of the first segment, with the evident anomaly of having the last claim totally wrong.

ION is smarter than Wireshark dissector as it recognizes the anomaly and silently discards the RS at his arrival. The problem here is maybe that ION does not provide any information about that in the standard log (ion.log). To realize what was actually going on, we had to enable LTP logs (off by default), where it was reported that the RS was actually received but also discarded. Continuing with the analysis, the drop of the

malformed RS leads to the retransmissions of other malformed RSs (Figure 6.28).

```
 39 0.033082  … … LTP          539 Session 1/2, Red data, range 40553–41036 [Unfinished LTP Block]
 40 0.210771  … … LTP           65 Session 1/2, Report segment, range 1015–41036, gaps: 3, gap total: 1014
 41 2.230697  … … LTP           65 Session 1/2, Report segment, range 1015–41036, gaps: 3, gap total: 1014
 42 2.729413  … … LTP          539 Session 1/2, Red data, range 40553–41036 [Retransmission]
 43 2.744194  … … LTP           65 Session 1/2, Report segment, range 1015–41036, gaps: 3, gap total: 1014
 44 4.242555  … … LTP           65 Session 1/2, Report segment, range 1015–41036, gaps: 3, gap total: 1014
 45 4.757351  … … LTP           65 Session 1/2, Report segment, range 1015–41036, gaps: 3, gap total: 1014
 50 5.729820  … … LTP          539 Session 1/2, Red data, range 40553–41036 [Retransmission]
 51 5.746702  … … LTP           47 Session 1/2, Cancel segment
 52 5.752241  … … LTP           60 34100 → 1113 Len=4
```

Figure 6.28 -Wireshark dissection of segment exchanged between VM1 and VM2; the last segment, although not correctly dissected, is correctly formatted, this is likely due to the port 34100

ION ignores all of them and after the RTO runs out, it resends the unconfirmed CP (segment 42), which generates other three malformed RSs (frames 43, 44 and 45) followed by another retransmitted CP (50). This session is then closed by a CR (segment 51) acknowledged by a CAR (52), poorly recognized by Wireshark (after manual inspection it resulted OK).

6.3.6 Test 6, first data segment lost (ESA BP vm2-> ION vm1)

This sixth test aimed at verifying the behavior of the ION receiver in the same condition as the last test, when the first data segment was lost; in particular, we were interested in checking the value set as lower bound in the ION RS. In this test we used LTPDrop in a very preliminary version (v. 0.0.1), in series with the Virtualbricks channel emulator (switched off, as we were not interested in inserting other losses, or delays, etc.). LTPDrop was instructed to eliminate only the first data segment.

ESA BP Command: ***sendf -d=ipn:1.2 -adu="file40K.txt"***

Channel: First data segment dropped (VM2->VM1), PER 0% on the channel emulator (VM1->VM2), no delay.

LTPDrop was placed on the sender side thus segment 1 is not recorded in the dump.

```
 1 0.000000  … … LTP          1055 Session 2/456577363, Red data, range 1001–2000 [Reassembled in #44]
 2 0.011586  … … LTP          1055 Session 2/456577363, Red data, range 2001–3000 [Reassembled in #44]
 3 0.023121  … … LTP          1055 Session 2/456577363, Red data, range 3001–4000 [Reassembled in #44]
```

Figure 6.29 - Wireshark dissection of segment received by VM2, the first segment received is the second data segment as its offset is not 0

The arrival of CP, generates an RS (Figure 6.30) by ION which is correctly formatted, reporting 0 as lower bound.

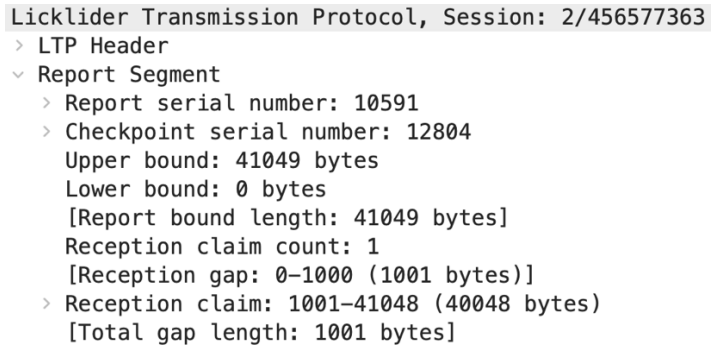

Figure 6.30 -Wireshark dissection of the RS generated by ION

The loss is then correctly recovered by ESA BP.

6.3.7 Test 7, first segment loss(ESA BP VM1 -> ESA BP VM2)

This test was aimed at verifying the behavior of an ESA LTP sender after the reception of an incorrectly formatted RS due to the loss of the first segment. The first segment loss was induced using LTPDrop.

Command: ***sendf -d=ipn:2.1 -adu="file40K.txt"***

Channel: First data segment dropped (VM2->VM1), PER 0% on the channel emulator (VM1->VM2), no delay.

```
41 … … … LTP            88 Session 1/63202035, Red data, range 41018–41048 [Reassembled in #44]
42 … … … LTP            64 Session 1/63202035, Report segment, range 1002–41048, gaps: 2, gap total: 0
43 … … … LTP            51 Session 1/63202035, Report ack segment
44 … … … BPv7         1059 Payload-Size: 40960
45 … … … LTP            62 Session 1/63202035, Report segment, range 0–41048, gaps: 0, gap total: 0
46 … … … LTP            51 Session 1/63202035, Report ack segment
```

Figure 6.31 - Wireshark dissection. Segment 42 is a malformed RS, however ESA LTP still sends the correct data segment.

Although the RS is malformed, ESA LTP manages to send the correct segment, which is acknowledged by an RS containing 0 gaps, thus the session concludes correctly.

6.3.8 Comments on Esa LTP tests

From the tests above we can conclude that the LTP implementation developed by ESA proved compliant with the standard but in two cases (non acks after Tx-session is closed and malformed RS in case of first data segment lost). To fix the code, however, should be relatively easy. The second bug proves once again the fundamental

importance of carrying out interoperability tests, as this bug would not have been detected otherwise. It was ION that triggered the Wireshark analysis by discarding the malformed RS.

.

7 CONCLUSIONS

The aim of this thesis was to study Bundle Protocol version 7 and Licklider Transmission Protocol implementations developed by the European Space Agency and to carry out some interoperability tests with other major BP implementations, such as ION, by NASA JPL.

ESA software often follows an original approach and in order to set up a running testbed it was essential the support of its author, Dr. Camillo Malnati, the co-supervisor of this thesis. As a result, we decided to complement the original documentation with a practical configuration guide (chapter 5). This guide enables the quick setup of two ESABP nodes and is tailored to users already familiar with ION, the most widely adopted BP implementation. It was crafted by integrating information from ESA documentation with co-supervisor suggestions and insights gained from testing.

The tests performed during the thesis were largely positive, but also revealed some bugs, not only in the brand new ESA software, but also in ION and in other BP implementations including Unibo-BP, in particular referring to BP status report formatting and CRC handling in different bundle blocks. All problems were reported to relevant developers. The tests showed that ESABP is compliant with RFC 9171 and RFC 5326, except for the few implementation bugs described in the thesis.

To support testing activities, LTPDrop was developed, the first basic Application Level Gateway for LTP. The software has been released as free software and added to other DTN software by Unibo, already available on GitLab under the Unibo-DTN umbrella project.

BIBLIOGRAPHY

[Alessi_2019] N. Alessi, S. Burleigh, C. Caini, T. De Cola, "*Design and Performance Evaluation of LTP Enhancements for Lossy Space Channels*", Wiley, International Journal of Satellite Communications and Networking, Vol.37, No.1, pp.3-14, Jan-Feb 2019, DOI: 10.1002/sat.1238.

[Araniti_2015] G. Araniti, N. Bezirgiannidis, E. Birrane, I. Bisio, S. Burleigh, C. Caini, M. Feldmann, M. Marchese, J. Segui, K. Suzuki. "*Contact Graph Routing in DTN Space Networks: Overview, Enhancements and Performance*", IEEE Commun. Mag., Vol.53, No.3, pp.38-46, March 2015, DOI: 10.1109/MCOM.2015.7060480.

[ASN1] "*ASN.1 encoding rules. Part 1: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER)*", June 2021, ISO/IEC: 8825-1:2021

[Bash] Web site: https://www.gnu.org/software/bash/.

[Bisacchi_2022A] A. Bisacchi, C. Caini and T. de Cola, "*Multicolor Licklider Transmission Protocol: An LTP Version for Future Interplanetary Links*", in IEEE Transactions on Aerospace and Electronic Systems, vol. 58, no. 5, pp. 3859-3869, Oct. 2022, doi: 10.1109/TAES.2022.3176847. (Open Source).

[Bisacchi_2022B] A. Bisacchi, C. Caini, and S. Lanzoni, "*Design and Implementation of Bundle* Protocol Unified API", doi: <10.1109/ASMS/SPSC55670.2022.9914734>

[Burleigh_2007] S. Burleigh, "*Interplanetary Overlay Network: An Implementation of the DTN Bundle Protocol*", 2007 4th IEEE Consumer Communications and Networking Conference, 2007, pp. 222-226, doi: 10.1109/CCNC.2007.51.

[Caini_2021] C. Caini, G. M. De Cola, L. Persampieri, "Schedule-Aware Bundle Routing: Analysis and Enhancements", International Journal of Satellite Communications and Networking, vol. 39, no.3, pp. 237-243, May/June 2021. DOI: 10.1002/sat.1384

[Caini_2023] C. Caini, T. de Cola, A. Shrestha and A. Zappacosta, "LTP Performance on Near Earth Optical Links," in IEEE Transactions on Aerospace and Electronic Systems, vol. 59, no. 6, pp. 9501-9511, Dec. 2023, doi: 10.1109/TAES.2023.3322392 (Open Source).

[Caini_2024] C. Caini and L. Persampieri, "Design and Features of Unibo-BP, the Unibo Implementation of the DTN Bundle Protocol," in IEEE Journal of Radio Frequency Identification, early access 2024, doi: 10.1109/JRFID.2024.3358012 (Open Source)

[CCSDS_AOS] CCSDS 732.0-B-4, "*AOS Space Data Link Protocol*", recommended standard, Blue Book, October 2021, <https://public.ccsds.org/Pubs/732x0b4.pdf>

[CCSDS_BPV6] CCSDS 734.2-B-1, "*CCSDS Bundle Protocol Specification*", recommended standard, Blue Book, September 2015, <https://public.ccsds.org/Pubs/734x2b1.pdf>.

[CCSDS_LTP] CCSDS 734.1-B-1, "*Licklider Transmission Protocol (LTP) for CCSDS*", recommended standard, Blue Book, May 2015, <https://public.ccsds.org/Pubs/734x1b1.pdf>.

[CCSDS_SABR] CCSDS 734.3-B-1, "*Schedule-Aware Bundle Routing*", recommended standard, Blue Book, July 2019, <https://public.ccsds.org/Pubs/734x3b1.pdf>.

[CCSDS_SLE] CCSDS 913.1-B-2, "*Space Link Extension - Internet Protocol for Transfer Services"*, recommended standard, September 2015, <https://public.ccsds.org/Pubs/913x1b2.pdf>

[CCSDS_SPP] CCSDS 133.0-B-2, "*Space Packet Protocol*", recommended standard, Blue Book, June 2020, <https://public.ccsds.org/Pubs/133x0b2e1.pdf>

[CCSDS_TC] CCSDS 232.0-B-4, "*TC Space Data Link Protocol*", recommended standard, Blue Book, October 2021, <https://public.ccsds.org/Pubs/232x0b4c1.pdf>

[CCSDS_TM] CCSDS 132.0-B-3, "*TM Space Data Link Protocol",* recommended standard, Blue Book, October 2021, <https://public.ccsds.org/Pubs/132x0b3.pdf>

[CRC16] ITU-T, "*X.25: Interface between Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DCE) for terminals operating in the packet mode and connected to public data networks by dedicated circuit*", p. 9, Section 2.2.7.4, ITU-T Recommendation X.25, October 1996, <https://www.itu.int/rec/T-REC-X.25-199610-I/>.

[DTN2] Web site: https://sourceforge.net/projects/dtn/.

[DTNME] Web site: https://github.com/nasa/DTNME.

[DTNsuite] Web site: https://gitlab.com/dtnsuite

[ECLSA] Web site: https://gitlab.com/unibo-dtn/ECLSA

[ESA BP_CIG] C. Malnati, "*Bundle Protocol Implementation Configuration Installation Guide*", ref: EGOS-GEN-DTN-CIG-001, Issue/Revision: 2.0, March 2023.

[ESA BP_ICD] C. Malnati, "*Bundle Protocol Implementation Interface Control Document"*, ref: EGOS-GEN-DTN-ICD-001, Issue/Revision: 2.0, March 2023

[ESA BP_SUM] C. Malnati, "*Bundle Protocol Implementation Software User Manual"*, ref: EGOS-GEN-DTN-SUM-001, Issue/Revision: 2.0, March 2023

[Github_2024] Issue: https://github.com/nasa-jpl/ION-DTN/issues/33

[Gori_2020] G. Gori, "*Inserimento dell'algoritmo di routing CGR-SABR in DTN2*", Tesi di Laurea in Ingegneria Informatica, Università di Bologna, 2020.

[ION] Web site: https://sourceforge.net/projects/ion-dtn/.

[Java] Web site: https://www.java.com/en/

[JeroMQ] Web site: https://github.com/zeromq/jeromq/.

[Licklider_1963] Licklider, J.C.R, "*Memorandum for Members and Affiliates of the Intergalactic Computer Network*", April 1963, <http://www.chick.net/wizards/memo.html>

[LTPDrop] Web site: https://gitlab.com/unibo-dtn/ltpdrop

[MTCP] S. Burleigh, "*Minimal TCP Convergence Layer*", draft v.00, September 2019, <https://datatracker.ietf.org/doc/html/draft-ietf-dtn-mtcpcl-00>

[Python] Web site: https://www.python.org/

[RFC_3448] Handley, M., Floyd, S., Padhye, J., Widmer, J., "*TCP Friendly Rate Control (TFRC), Protocol Specification*", RFC 3448, DOI 10.17487/RFC3448, January 2003, <https://www.rfc-editor.org/info/rfc3448>.

[RFC_3986] Berners-Lee, T., Fielding, R., Masinter, L., "*Uniform Resource Identifier (URI): Generic Syntax*", RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC_4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "*Delay-Tolerant Networking Architecture*", RFC 4838, DOI 10.17487/RFC4838, April 2007, <https://www.rfc-editor.org/info/rfc4838>.

[RFC_5050] Scott, K. and S. Burleigh, "*Bundle Protocol Specification*", RFC 5050, DOI 10.17487/RFC5050, November 2007, <https://www.rfc-editor.org/info/rfc5050>.

[RFC_5325] Burleigh, S., Ramadas, M., and S. Farrell, "*Licklider Transmission Protocol - Motivation*", RFC 5325, DOI 10.17487/RFC5325, September 2008, <https://www.rfc-editor.org/info/rfc5325>.

[RFC_5326] Ramadas, M., Burleigh, S., and S. Farrell, "*Licklider Transmission Protocol - Specification*", RFC 5326, DOI 10.17487/RFC5326, September 2008, <https://www.rfc-editor.org/info/rfc5326>.

[RFC_7122] Kruse, H., Jero, S., Ostermann, S. "*Datagram Convergence Layers for the Delay- and Disruption Tolerant Networking (DTN) Bundle Protocol and Licklider Transmission Protocol (LTP)*", RFC 7122, DOI 10.17487/RFC7122, March 2014, <https://www.rfc-editor.org/info/rfc7122>.

[RFC_7242] Demmer, M., Ott, J., and S. Perreault, "*Delay-Tolerant Networking TCP Convergence-Layer Protocol*", RFC 7242, DOI 10.17487/RFC7242, June 2014, <https://www.rfc-editor.org/info/rfc7242>.

[RFC_8949] Bormann, C. and P. Hoffman, "*Concise Binary Object Representation (CBOR)*", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[RFC_9171] Burleigh, S., Fall, K., Birrane, E. III, "*Bundle Protocol Version 7*", RFC 9171, DOI 10.17487/RFC9171, January 2022, <https://www.rfc-editor.org/info/rfc9171>.

[RFC_9172] Birrane, E. III, McKeever, K., "*Bundle Protocol Security BPSec"*, RFC 9172, RFC 9172, DOI 10.17487/RFC9172, January 2022, <https://www.rfc-editor.org/info/rfc9172>.

[RFC_9174] Sipos, B., Demmer, M., Ott, J., and S. Perreault, "*Delay-Tolerant Networking TCP Convergence-Layer Protocol Version 4*", RFC 9174, DOI 10.17487/RFC9174, January 2022, <https://www.rfc-editor.org/info/rfc9174>.

[tcpdump] Web site: https://www.tcpdump.org/

[Unibo-LTP] Web site: https://gitlab.com/unibo-dtn/ltp

[Unibo-CGR] Web site: https://gitlab.com/unibo-dtn/unibo-cgr

[Unibo-DTNME] Web site: https://gitlab.com/ccaini/unibo-dtnme

[Unified-API] Web site: https://gitlab.com/dtnsuite/unified_api

[Virtualbricks] P. Apollonio, C. Caini, M. Giusti, D. Lacamera, "*Virtualbricks for DTN satellite communications research and education*", in Proc. of PSATS 2014, Genoa, Italy, July 2014, pp. 1-14. DOI: 10.1007/978-3-319-47081-8_7.

[Wireshark] Web site: https://www.wireshark.org/.

[Zappacosta_2023] A. Zappacosta, "*Performance evaluation of Licklider Transmission Protocol over Free Space Optical communication testbeds*", Master's thesis, University of Bologna, March 2023. Available: http://amslaurea.unibo.it/view/cds/CDS0937/

[ZeroMQ] Web site: https://zeromq.org/.

ACKNOWLEDGMENTS

Ringrazio i miei genitori che mi hanno supportato e fornito gli strumenti per arrivare fino a qui. Ringrazio i miei fratelli Tommaso, Filippo e Francesco compagni di gioia e di liti, che mi ascoltano ancora.

Ringrazio i miei nonni Maria Rita e Franco, Anna Maria e Pasquale per il loro supporto instancabile e infinito.

Ringrazio Jordan, Jembe, Toby, Chicco, Gian e Samu: compagni di studio inarrestabili fra i tavoli della mensa, i banchi delle aule e le librerie della biblioteca.

Ringrazio tutti i compagni di barca e il gruppo del liceo, mi avete conosciuto quando ancora ero indeciso su che via accademica intraprendere e mi avete sempre spinto a continuare.

Ringrazio il Centro Velico Caprera, uno dei fari nel mare della mia vita.