HSLTP - An LTP Variant for High Speed Links and Memory Constrained Nodes

Nicola Alessi, Carlo Caini, Azzurra Ciliberti, Tomaso de Cola,

Abstract — Delay-/Disruption-Tolerant Networking architecture relies on the use of Licklider Transmission Protocol (LTP) on interplanetary links. LTP loss recovery is based on ARQ retransmissions, which, when the propagation delay is very long, are costly. Alternatively, losses could be recovered by using Packet Layer Forward Error Correcting codes, as done by the authors in ECLSA (Erasure Coding Link Service Adapter), recently presented in a companion paper, where LTP segment retransmissions are limited to the unlikely case of decoding failures. However, on high bandwidth-delay-product links, the very possibility of segment retransmission requires that a huge number of Rx buffers be available. To resolve this problem, High-Speed LTP (HSLTP), presented here, has a more disruptive approach than ECLSA: it enforces almost one-to-one correspondence between LTP blocks and FEC codewords, and never requires LTP segment retransmissions. In the unlikely case of a FEC failure, the partially received block is discarded and its bundles are resent directly by the Bundle Protocol. The advantages and disadvantages of this approach are explored in the paper. On nodes limited in memory the results are significantly improved.

Index Terms— IPN, DTN, LTP, ARQ, PL-FEC, LDPC, ECLSA.

I. INTRODUCTION

SPACE networks are affected by long delays, link intermittency, link losses and asymmetric bandwidth. Some twenty years ago, researchers at NASA JPL, under Vinton Cerf's guide, realized that these challenges required the definition of a new architecture, the IPN (InterPlanetary Networking). IPN was soon renamed DTN (Delay-/Disruption-Tolerant Networking), when it appeared clear that many of the space challenges are actually common to terrestrial "challenged networks" (wireless sensor networks, underwater networks, emergency networks, and communications in remote or hostile environments) [1]. Space communications, however, are still the driver for DTN research and standardization. The latter started under the IRTF umbrella, but now is carried on in parallel by IETF [2] and for space applications by CCSDS [3].

DTN architecture [4], [5] is based on the insertion of a new layer, the Bundle Layer, between Application and other lower layers, usually Transport [6], [7]. This new layer acts as an overlay and splits the path between sender and destination into multiple DTN hops. DTN nodes can store "bundles" (packets at the homonymous layer) for long periods, to cope with link intermittency and network partitioning. Moreover, transmission reliability can be enhanced at bundle layer by means of the custody transfer option, which consists in bundle retransmission upon timer expiration. This option is particularly appealing when the underlying Transport protocol is unreliable (e.g., UDP) [8], [9] at least on one DTN hop. delays, losses, link intermittency and bandwidth asymmetry are be tackled inside each DTN hop by specialized protocols, such as LTP (Licklider Transmission Protocol) in interplanetary links [10], [11], [12]. In this respect, the scientific community has dedicated quite some attention to the performance modeling of LTP [13], [14], [15] and also elaborated significant theoretical models to getter deep insight into LTP dynamics as discussed in [16] and [17].

LTP can provide both a reliable and an unreliable service with red and green parts, respectively. Focusing on the former, loss recovery of LTP segments is based on an ARQ (Automatic Repeat Request) scheme [18]. Thanks to its ingenious design, loss recovery of an entire LTP session (the transmission of one LTP block) can often be completed in one RTT, if there are no further losses on retransmitted segments. However, on interplanetary links even a single RTT greatly penalizes delivery time, because the propagation delay is long (from 3 to 20 min, from Earth to Mars). This makes the use of Packet Layer FEC coding (PL-FEC) very appealing, as it could limit the need for LTP segment retransmissions to the rare case of decoding failure.

There is abundant literature and some RFCs [19], [20] on the use of erasure codes at packet layer for different purposes. PL-FEC must be considered complementary and not alternative to the almost ubiquitous physical layer channel coding: PL-FEC is used to recover the datalink frame losses resulting from residual bit errors after channel FEC decoding (a frame that does not pass the parity check is entirely discarded). On space channels, the first studies on PL-FEC date back to 2007 ([21], then extended in [22]). Their use in combination with LTP was first examined in [23], by comparing this choice to other solutions, then proposed in the CCSDS Orange Book [24] and then in [25] and [26].

Dealing with LTP and PL-FEC, two approaches are possible: segment oriented or block oriented. In both LTP segments are treated as information symbols of a codeword, in the former the PL-FEC encoder is unaware of LTP block boundaries and thus it can naturally treat all LTP segments, either data or signaling, in the same way. The disadvantage is

Manuscript received XX, 2019.
Nicola Alessi, Carlo Caini and Azzurra Ciliberti are with the University of Bologna, Italy, and may be contacted at nicola.alessi@studio.unibo.it, carlo.caini@unibo.it, azzurra.ciliberti@studio.unibo.it. Tomaso de Cola is with DLR and can be contacted at tomaso.decola@dlr.it.

that having no notion of LTP blocks, it needs the introduction of aggregation timers (with related delays and complex settings) to stop filling a codeword in the absence of new data. An extensive description of this approach was given by the present authors in a previous companion paper, devoted to ECLSA (Error Correcting Link Service Adapter) [27]. The block oriented approach, by contrast, preserves the boundaries of LTP blocks, coding each LTP block into a new codeword, as done in [25], [26]. One significant advantage is that aggregation timers are no longer necessary, but there are also two disadvantages. First, mapping one LTP block into one codeword requires the use of a FEC code that can fit the variable dimension of blocks. Second, the treatment of LTP signaling segments is an issue, as they need to be protected against losses even more than data segments do, but they are often transmitted apart (e.g. in case of retransmissions, or confirmations), or in the reverse direction.

The idea behind High-Speed LTP (HSLTP), presented in this paper, is that of going further with the link between LTP blocks and codewords pursued by the block-oriented approach, by interpreting the binary result of a codeword decoding (success or failure), as the result of the corresponding LTP block transfer (completed or canceled). In HSLTP retransmissions of single segments are no longer necessary, because in the unlikely event of a decoding failure, the partially received block is discarded and reliability is delegated to bundle retransmissions. As it will be shown in the paper, the biggest advantage is that by having eliminated the reception of incomplete blocks it possible to eliminate all LTP Rx buffers but one. This may be of paramount importance on high bandwidth delay product links, such as optical links in space, especially when the receiver node is memory-constrained, as often in space assets.

Despite its disruptive content (all LTP segment retransmissions are eliminated), HSLTP has been implemented in an evolutionary way, as an optional extension of the ECLSA code already present in ION (the NASA JPL implementation of DTN protocols [28]). Numerical results, obtained on a GNU/Linux testbed, are presented at the end of the paper to compare HSLTP performance with that achievable using ECLSA (below LTP) or LTP alone. They confirm the validity of the proposed approach.

II. The Licklider Transmission Protocol (LTP)

A key element of DTN architecture is that the introduction of the Bundle layer confines the Transport layer scope to one DTN hop. This is essential in space and in other heterogeneous networks as it permits the use of different transport protocols on different DTN hops, in order to tackle different channel impairments (Figure 1). While on Earth and likely on other planets (e.g. Mars) TCP could still be used, on interplanetary links it must be replaced by a specialized protocol, such as LTP. The interfaces between BP and the lower layer (usually Transport) are called “Convergence Layer Adapters”, (CLAs) as the name “Convergence layer” is commonly used to define the protocol below the BP (e.g. LTP, TCP etc.).

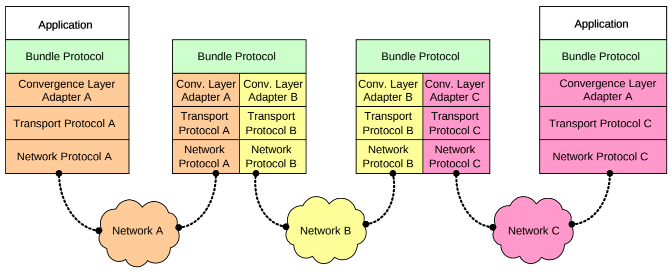

Figure 1: DTN architecture protocol stack.

A. *LTP main features*

LTP is the convergence layer of choice on interplanetary links [10], [11], [12], as it minimizes interaction (“chattiness”) between transmitting and receiving engines. LTP runs on top of UDP in test beds or on CCSDS space protocols in real deployments. In the following, we will assume UDP below LTP for the sake of exposition clarity, without loss of generality. LTP can offer both reliable and unreliable services, with “red” and “green” parts respectively. Let us focus on red parts and list the key features that differentiate LTP from TCP [29], most of which aim to minimize chattiness.

- No connection-establishment phase, such as TCP 3-way handshake.
- Rate based transmission speed.
- Unidirectional data flow (the reverse channel is used only for signaling) to cope with possible channel asymmetry.
- One or more bundles may be aggregated by LTPCL [7] in one LTP “block”; blocks are transmitted by independent LTP “sessions” running in parallel to fill the Bandwidth Delay Product (BDP).
- An LTP block is usually split into a number of smaller LTP “segments”, each passed to UDP, or other CCSDS space protocol;
- In contrast to TCP, segment acknowledgments (LTP “report segments”) are triggered only by data segments flagged as “checkpoints”, usually at the end of a block.

B. *LTP sessions without losses*

For a better description of LTP session mechanisms, it is necessary to introduce some examples. Let us begin with the simplest one, which considers a session without any loss (Figure 1). When the export session starts, all the segments of the LTP block are sent; the last data segment plays a double role, because it also carries signaling information, being flagged as EORP (End Of Red Part), EOB (End Of Block), and CP (Checkpoint) [11]. The arrival of the first segment opens the import session, while the arrival of the last, flagged as CP, triggers a Report Segment (RS), confirming data received, i.e. the full block in this case. As the block is completed, the bundles contained in it are delivered to BP (multiple small bundles can be aggregated into one LTP block by LTPCL to reduce overheads) and the Rx buffer is deallocated (in the latest ION versions, 3.6.x). RS reception confirms the CP and is in turn confirmed by a RA (Report-

ACK). As all segments are confirmed, the Export session is closed. When the RA arrives at receiver, the import session is closed.

If the RTT is long, as in space links, block transmission and processing times are dominated by the propagation delay. In this case, the delivery time is roughly equal to one-half RTT, and both the export and import session lifetimes equal to 1 RTT. This proves that LTP is ideal in the absence of losses, because both delivery time and the minimum time for reliable delivery (i.e. for receiving an acknowledgment of the transmitted data), coincide with their theoretical minima.

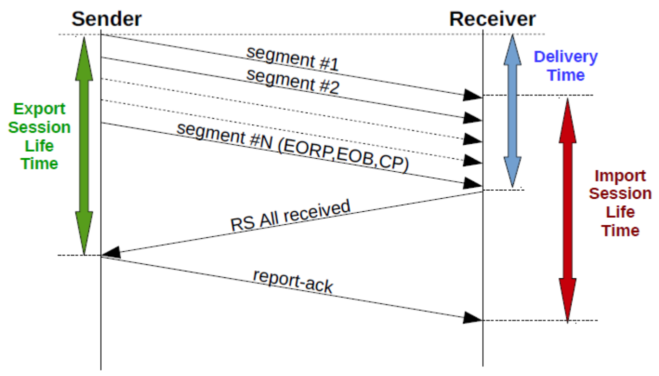

Figure 2: Example of LTP session (red-data only) in ideal conditions (absence of losses).

C. LTP sessions with losses

Now we are ready to examine a second example (Figure 3), which considers losses on both data and signaling segments to highlight their different impact on delivery time. The loss of the CP piggybacked on the last segment (#N) stalls the communication until the corresponding Retransmission Time Out (RTO) expires, forcing a new sending (CPs, RSs, and other signaling segments are always retransmitted unless confirmed in time). CP reception triggers a partial RS, confirming all data but segments #2 and #3, which were lost as well but the sender could not know. These two segments are retransmitted, #3 being flagged as CP. Their arrival completes the block, and everything goes on as in the ideal case.

A key difference from TCP is that pure data segments (i.e. those not flagged as CP) when lost (#2 and #3) are retransmitted together, in one retransmission cycle. As a result, the overall time penalty is one RTT, independently from the number of losses. One RTT for all losses is an excellent result for LTP, as it is the theoretical minimum for ARQ based recovery. Of course, if any retransmitted segment is lost, a further round of retransmission is necessary, with an additional RTT, and so on, but this should happen rarely, at least for low Packet Loss Rates (PLR).

Considering "pure" signaling segments, such as RSs or RAs, or mixed segments, i.e. CPs, the impact of losses is worse. This because their penalty time, one RTO (just a little longer than one RTT), adds to the delay due to retransmission of other segments, as in Figure 3. This has led to the development of an enhanced version of LTP [29], where the user can introduce a variable amount of replication to protect signaling segments against losses. This enhanced LTP version has now become the standard in current ION releases.

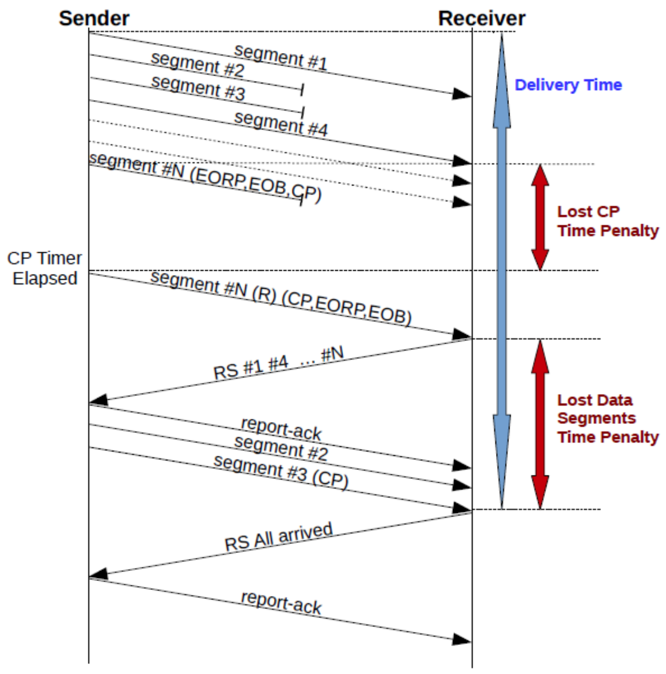

Figure 3: Example of LTP session (red-data only) in the presence of mixed losses (#2, #3 and #N). The loss of the CP piggybacked on #N stalls the communication until the corresponding RTO expires. The time penalty (one RTO) adds to the delay due to retransmission of other segments.

D. LTP sessions and buffers

Each LTP block is independently transmitted during a single LTP session, and to fill the BDP more concurrent sessions are allowed and also necessary. For blocks sent as "red", it is necessary to have one Tx buffer and one Rx buffer for each on-going session. The former is necessary to resend missing segments, the latter to store received segments until the whole block is received. It is necessary to examine the ION implementation in detail, because there are important differences between Tx and Rx buffers, concerning the actual use of memory.

1) Tx buffers

At sender side the Tx buffer cannot be released until the associated session ends, i.e. when the final RS arrives, signaling the successful arrival of the whole block. It is worth noting that thanks to ION's "zero-copy" feature [28], which makes it possible to rebuild lost LTP data segments directly from the original bundle(s) still stored by BP, only one small subset of segments sent is actually saved in the Tx buffer, namely only those subjected to retransmissions caused by RTO timer expirations, i.e. signaling segments and CPs. This clever mechanism of ION greatly alleviates memory consumption at Tx side.

2) Rx buffers

This time all segments must be stored, as it is clearly impossible to make use of the zero-copy mechanism at Rx side, as by definition the bundles contained in the block are yet not available. The Rx buffer of a session can be released when all segments have arrived, i.e. before the end of the import session, which requires the final RS to be confirmed by its RA, i.e. an additional RTT (see Figure 2). Old versions of ION did not exploit this, and Rx buffers were released only at the end of a session, as at Tx side. In contrast, the latest ION versions (3.6.x) release the Rx buffer as soon as a block is completed to counteract the possible loss of the last RA [29]. With this enhancement, the Rx buffer can be immediately reused by following blocks if any (original blocks are sent sequentially, only retransmitted segments).

E. BDP and buffers

The number of parallel sessions should be set according to the bandwidth delay product (BDP) of the link (plus other parameters, such as loss probability). The larger the BDP, the higher the number of parallel sessions necessary to "fill the channel" (calculating the optimal number, however, is far from easy, especially with losses, see ION documentation [28]). On interplanetary links the delay is huge with available bandwidth continuously increasing thanks to technology, thus memory at Rx side (especially on space assets which are more easily memory constrained because of the higher cost of memory robust against space radiations) can become a limiting factor if not properly counteracted. To have an idea of the problem, consider that on an Earth-Mars link, with 46min of RTT (the worst case) and 500Mbit/s (optical link) we would have a BDP of 172 GB. ION alleviates this problem by not requiring Rx buffer to be entirely kept in RAM, but on the other hand disk access is slower. In spite of this optimization, it is clear that the Rx buffers still pose a serious challenge to future high-speed links when the receiver is memory-constrained.

III. ECLSA Essentials

This section aims to provide the reader with a concise description of ECLSA, from which HSLTP derives. For clarity of exposition, we will assume here and in the rest of the paper that ECLSA is put below LTP, although it could theoretically work also with other upper protocols. For a comprehensive treatment, see [27].

A. ECLSA as a Link Service Adapter

ECLSA, like all other link service adapters fits between LTP and lower protocols, such as UDP. However, ECLSA is not just a mere interface, but a true intermediate protocol that provides PL-FEC service to LTP. In Figure 4 the protocol stack of two DTN node using LTP with ECLSA is shown. On top we have the BP application that provides the message to be sent; this message is encapsulated in one bundle, then the bundle is possibly aggregated with other bundles into one LTP block. The block is then segmented and LTP segments are passed to ECLSA, which adds redundancy segments; all segments are eventually passed to UDP, which encapsulates each segment into one packet and so on towards lower layers; vice versa at receiving side.

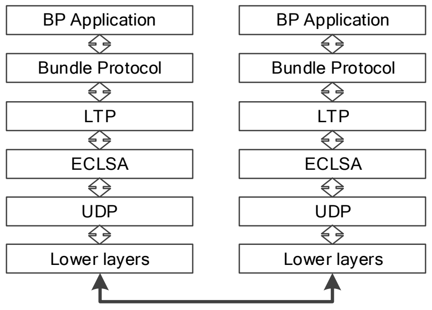

Figure 4: The protocol stack of two DTN nodes sing ECLSA on top of UDP (From [27]).

B. PL-FEC in ECLSA

The core of ECLSA is the PL-FEC encoding. A schematic description of the process is shown in Figure 5 (see [27] for a more rigorous treatment). We assume the use of a (N, K) code (N is the number of symbols of a codeword, i.e. is length, K the number of information symbols, $Rc= K/N$ the code rate). At Tx side, each LTP segment is essentially treated as an information symbol of a codeword. In details (not reported in the figure), after adding a 2B header, it is inserted into arrow of an N x T "coding matrix", where T-2 is the maximum length in B of an LTP segment (if the LTP segment is shorter, a few padding bytes are added). The K information symbols (matrix rows) are then coded, by means of a PL-FEC coder, which calculates the $M=N-K$ redundancy (or "repair") symbols. Each symbol is then passed to a lower layer, such as UDP. On the Rx side, because of losses on the channel, L symbols are missing (crossed packets); the PL-FEC decoder tries to recover all K information symbols from the $N-L$ arrived symbols. On ideal codes, i.e. Maximum Distance Separable (MDS) codes , the condition for a successful decoding is that losses are no more than redundancy symbols (i.e. it is enough that at least K segments arrive, whether of information or redundancy) [30], while on most real codes, e.g. LDPC, a small margin is required. If the decoding is successful, all K information symbols are recovered and passed to LTP (after removing the 2B header and padding bytes if present), otherwise only a subset. In this latter case, as ECLSA is transparent to LTP, the missing segments will be recovered by LTP usual mechanisms.

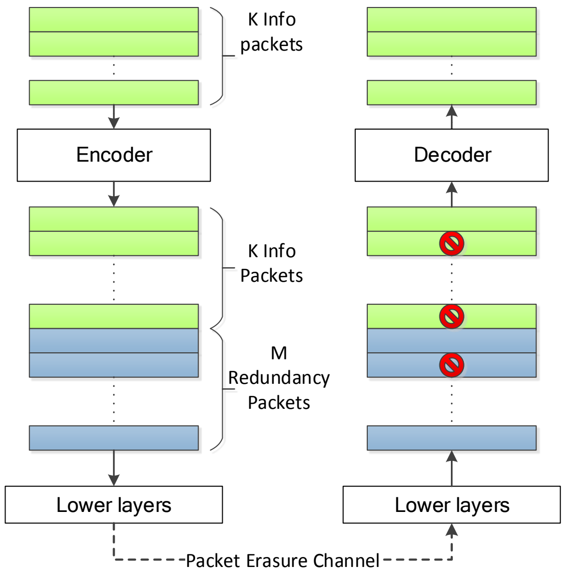

Figure 5: Simplified picture of PL-FEC logical process in ECLSA (From [27]). Tx side steps, on the left, are performed by ECLSO, those on Rx side, on the right, by ECLSI.

ECLSA makes use of alternative external libraries to perform erasure coding and decoding, LibecDLR and OpenFEC [32]. LibecDLR uses the LDPC Irregular-Repeat-Accumulate (IRA) family (see the CCSDS Orange book [24]), while OpenFEC relies on LDPC Staircase and LDPC Triangle codes (see RFC 5170 [19]). Although they use different LDPC codes, both of them can support the *N* and *K* values specified in the CCSDS Orange book.

C. ECLSA implementation

Our ECLSA implementation in ION consists of two processes, ECLSO and ECLSI, corresponding to Outduct and Induct channels (left and right sides of Figure 5). Each consists of three threads, as shown in Figure 6: ECLSA implementation: ECLSO (left) and ECLSI (right) threads.

a) ECLSO threads

The first thread of ECLSO (left) is “matrix filling from LTP”. Its task is to put each segment passed by LTP into a new row of the coding matrix. This filling process stops (i.e. the matrix is “closed”) when either the matrix is full (*K* information segments have arrived), or when an aggregation time expires. In the latter case the number of segments arrived, I, i.e. of rows filled, is less than the nominal *K*. An option (“*K* continuous”) allows a code with *K*=*I* to be created on the spot. In ECLSA the filling process is transparent to LTP and block boundaries are not considered at all. Once the matrix is “closed”, the “Matrix Encoding” thread adds the redundancy symbols in the last *N*-*K* rows, thus forming the (*N*,*K*) codeword. Finally, the “Matrix Passing to UDP” adds an ECLSA header to each symbol then passes it to UDP.

b) ECLSI threads

On the receiving side (right) ECLSI performs dual operations. From the bottom, the “matrix filing from UDP” thread for each ECLSA packet received from UDP removes the ECLSA header and fills it in the correct coding matrix. Once “closed” the matrix is passed to the “Decoding Matrix process”, which tries to fill in any gaps in the *K* information rows; finally, the “Matrix passing to LTP” thread delivers all information symbols available (either received or recovered) to LTP.

Many other aspects ECLSA, although significant, are beyond our scope and, again, the interested reader is referred to [27] and [31].

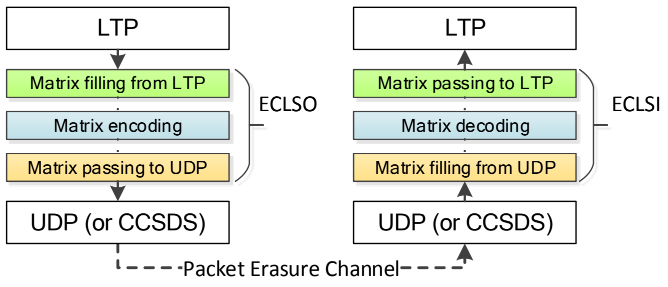

Figure 6: ECLSA implementation: ECLSO (left) and ECLSI (right) threads.

D. ECLSA: advantages and limits

In brief, the ECLSA link service adapter protects LTP segments using a systematic FEC code: if decoding succeeds, all losses on the *K* information symbols are recovered; otherwise, residual losses are left to LTP to recover.

ECLSA is segment-oriented, thus it is conscious of segments but has no concept of LTP block boundaries (it could in fact be used with other protocols as well). An obvious advantage of the segment-oriented approach is that signaling and retransmitted segments are naturally inserted into an ECLSA matrix together with data segments from successive blocks, so they are naturally protected by FEC. As detailed in section II.C and in [29], losses on signaling segments are worse than on ordinary data segments, thus FEC protection is essential. The same holds true for retransmitted segments.

To summarize, on an erasure channel with significant Packet Loss rate (PLR), ECLSA offers these advantages:

a) It greatly reduces delivery time by making retransmissions unlikely.

b) It eliminates delivery time dependency on LTP block dimensions (for a given PLR, the greater the block, the longer the expected delivery time, due to the higher probability of multiple retransmission cycles).

c) It almost eliminates block delivery time variability caused by random losses. In LTP, “lucky” blocks with no losses can be delivered in 0.5 RTT, “unlucky” ones require one additional RTT for each retransmission round. When RTT is in the order of minutes, this sort of “jitter” is a significant disadvantage, as it cannot be easily compensated by buffering, as on Earth.

The disadvantages are:

a) First, on high-speed interplanetary links the Rx buffer problem remains unresolved.

b) ECLSA aggregation timers are rather complex to set, as proper settings depend on matrix dimensions and link characteristics [33]. They also contribute to FEC processing time.

IV. HIGH-SPEED LTP (HSLTP)

A. Motivation

The principal aim of HSLTP is to reduce LTP buffering needs, and so allow full exploitation of bandwidth available on high-speed interplanetary links (this justifies the “high-speed” prefix).

HSLTP abandons the segment-oriented approach of ECLSA in favor of a block-oriented one. It enforces almost one-to-one correspondence between LTP blocks and ECLSA coding matrixes: each LTP block is now fully contained in one matrix and each matrix cannot contain more than one LTP block (but signaling segments can still be aggregated to a block and also matrixes of signaling segments only can exist). This has two key advantages:

1) Decoding state interpreted as state of the session

The decoding state (success or failure) of a matrix containing a block can now be interpreted as the state of the session (successful closing or cancellation). If successful, as likely, the whole block is passed to LTP with no need for retransmission; otherwise, HSLTP immediately cancels the session. In this case, to ensure reliability, HSLTP uses a mechanism already present in ION, by which once BP is informed of a convergence layer failure, it immediately retransmits all bundles involved. In our case, the session cancellation triggered by the decoding failure will force BP to retransmit all bundles contained in the canceled-session block from scratch (i.e. by building a new block a starting a new session). The first key advantage of HSLTP is that independently of decoding state it never retransmits single LTP segments, thus one Rx buffer is sufficient, which solves the problem of Rx buffers. The disadvantage is that in case of decoding failure, an entire (new) block has to be transmitted (in a new LTP session) instead of a few segments (in the still on-going LTP session). FEC failures, however, should be rare, and as HSLTP is designed for high-speed links, where bandwidth should be relatively abundant, the price could be well worth paying, especially when the receiver is memory-limited.

2) Processing time reduction, simplification of settings

The second significant advantage of HSLTP is the simplification of ECLSO timer settings and the related reduction of processing time, as it is no longer necessary either to wait for a (potentially large) fixed-dimension matrix to be completely filled, or for the ECLSA aggregation timer to expire. In HSLTP the coding matrix is immediately closed as soon as the last segment of the block is passed by LTP and a codeword of the right dimension is built on the spot (see “*K* Continuous Mode”, in [31][31]).

B. An example of HSLTP flow

A successful block transfer is presented at the top of Figure 7. LTP block 1 (containing bundles a and b) is sent and the corresponding codeword is correctly decoded at Rx side in spite of the possible loss of a few segments; all *K* information segments of the block are immediately passed to LTP. The newly opened import session, is immediately closed after a final RS is sent, confirming successful arrival of the full block, without waiting for the corresponding RACK, which immediately allows HSLTP to reuse the Rx buffer in all ION versions. At Tx side, the export session is closed on arrival of the final RS and BP is notified by the LTPCL of the successful transfer of bundles in block 1 (i.e. bundles a and b). This is what usually happens. Then a case of failure is shown, with bundle retransmission by BP, using a new LTP block and a new session. In detail, LTP block 2 (containing bundles c and d) is sent but decoding fails because there are too many losses. The new import session is immediately canceled and a Cancel Segment (CS) is sent back. The Rx buffer is immediately released in this case too. On Tx side CS reception triggers the cancellation of the export session and LTPCL notifies the BP of the transmission failure of all bundles that were aggregated in block 2 (bundles c and d). In response, BP immediately resends these bundles to LTPCL: they are aggregated in block 3 and a new export session is opened. This time decoding is successful and all continues as for block 1.

Note that in HSLTP the RS triggered by the CP at the end of a block is always “final”, i.e. it always acknowledges all segments, as it is sent only in case of successful decoding of a full block.

In case of decoding failure and consequent bundle retransmission, the bundle delivery time increases by one RTT (Figure 7), as with ECLSA. As said, the main advantage of HSLTP is that, success or failure, the Rx buffer is immediately available for subsequent sessions. Finally, note that in both HSLTP and ECLSA an optional feedback mechanism temporarily increases redundancy in response to a decoding failure. This considerably reduces the chances of consecutive failures on the same blocks.

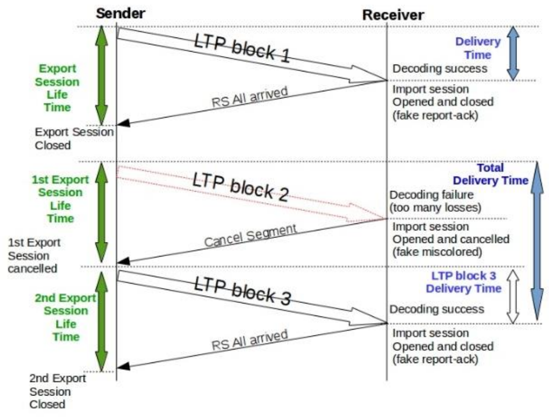
Sender
Receiver
LTP block 1
Delivery Time
Decoding success
Export Session Life Time
RS All arrived
Import session Opened and closed (fake report-ack)
Export Session Closed
Total Delivery Time
1st Export Session Life Time
LTP block 2
Decoding failure (too many losses)
Cancel Segment
Import session Opened and cancelled (fake miscolored)
1st Export Session cancelled
LTP block 3 Delivery Time
2nd Export Session Life Time
LTP block 3
Decoding success
RS All arrived
Import session Opened and closed (fake report-ack)
2nd Export Session Closed

Figure 7: An example of HSLTP flow. LTP block 1 decoding is successful; the new import session opens and closes immediately, while the export session closes after the arrival of RS confirming reception of all segments. Block 2 decoding fails, however, due to too many losses causing closure of new import session. Export session canceled on arrival of CS. Block 2’s bundles are resent by BP after aggregation in block 3, a new session starts and terminates successfully. Bundle retransmission increases total delivery time by just one RTT.

V. An Evolutionary Approach to HSLTP Implementation

HSLTP can be seen in two ways: either as a new protocol, derived from LTP but no more compliant with LTP specs (e.g. segment retransmissions are eliminated), or as a derivative of ECLSA. As the former, it could be implemented as a brand-new integrated module; as the latter, it could be implemented incrementally, to exploit the existing code of both LTP and ECLSA as much as possible. As HSLTP is an experimental protocol, we preferred this evolutionary approach and managed to implement HSLTP by modifying only two threads of ECLSA (the interfaces from and to LTP). In this way we left LTP code totally unaltered, which facilitates both code maintenance and protocol configurations. In our implementation, HSLTP can be activated simply by setting the "HSLTP enabling interface" option of ECLSA. More generally, the user can easily switch between LTP alone, or LTP with ECLSA, or HSLTP simply by modifying very few essential settings.

The modifications introduced to ECLSA interface to LTP are shown below. As will be evident, they required an in-depth study of both LTP standard and ION implementation.

A. Block-aware modifications

On the transmitting side, the main difference between HSLTP and ECLSA is that HSLTP is block-aware and needs an almost one-to-one correspondence between LTP blocks and FEC codewords. To this end, is necessary to modify the "matrix filling from LTP" thread of ECLSO, to stop the matrix filling of a coding matrix at the end of a block. This however, is not sufficient, because LTP signaling segments (RS, RA, CS, CAS etc. [11]) belong to a session but not to a block; they too need to be inserted into a coding matrix, and possibly sent alone, i.e. in a codeword that does not contain any block, if necessary.

Our solution consists in reading the "type" [11], of the LTP segment passed by LTP: to infer if the segment is the last of a block (flagged as EOB); to infer if it is a normal data segment, or is a pure signaling segment. The segment tagged as EOB always stops the filling process. The other types are used to distinguish whether a coding matrix contains either only signaling segments, or only data, or signaling segments followed by data. In the first case, a "signal only aggregation" timer is necessary to avoid waiting for a block (and its EOB) that may never arrive. This timer setting must be short, to avoid excessive delay on signaling segments. Setting is easy, at it is completely independent of code or channel characteristics.

The old conditions for matrix closure are preserved (matrix full and data aggregation timer elapsed), for safety reasons only, as they should normally never be enforced in HSLTP. An option can be activated to enable proactive fragmentation of bundles, to prevent the block from being so large as not to fit in the maximum K allowed by ECLSA settings.

B. Decoding failure: cancellation of import session

On the receiver side, HSLTP needs the success/failure of FEC decoding of a codeword to correspond with the successful closure/cancellation of the corresponding LTP session. For convenience, we will start from unsuccessful decoding, assuming for the sake of generality that there are some signaling segments followed by a full block in the codeword.

A decoding failure means that there will be residual gaps in the K information symbols of the received codeword. As mentioned, in HSLTP there is no attempt to recover these residual losses using the usual LTP mechanisms. What HSLTP does is as follows: first, all signaling segments (always from previous sessions) must be passed to LTP. Then, it is necessary to make LTP cancel the newly opened import session, instrumental so as not to keep engaged the Rx buffer. To trigger immediate cancellation without modifying the LTP code, we decided to disguise LTP by inserting a locally generated "miscolored" segment. Details below for the benefit of readers expert in LTP.

a) Implementation details

On receiving the first data segment of a codeword when decoding has failed, the "passing matrix to LTP" thread of ECLSI (when the "HSLTP enabling interface" is on) triggers session cancellation by passing the received segment to LTP as usual, but followed by a locally generated clone with the LTP "color" inverted ("green" instead of "red"). This second segment is recognized by LTP as "miscolored", which causes immediate cancellation of the newly opened import session. This cancellation is notified to the sender LTP engine by the usual LTP mechanisms, i.e. by means of a CS that triggers the export session cancellation at its arrival (see Figure 3).

C. Decoding success: immediate closure of the import session

Let us now consider successful decoding. Signaling segments, if any, are passed to LTP, then the first data segment causes the opening of a new import session as before, but this time the session is not canceled and all data segments are stored one-by-one in a new Rx buffer. As successful decoding means no gaps in the received block, the bundles contained can be passed to BP and the Rx session closed immediately, after sending the final RS, confirming the successful arrival of the block to the sending engine, without waiting for the final RA. This is instrumental to immediate release of Rx buffer, even in old ION versions. To trigger the immediate closure of the import session, without modifying the LTP code, we decided to disguise LTP by inserting a locally generated fake RA. Details below for expert readers.

a) Implementation details

When the HSLTP enabling interface is set, the "matrix filling from LTP thread" of ECLSO intercepts the final RS, which acknowledges successful reception of the full block. This RS is still sent back to confirm the full block arrival, but also triggers the local insertion of a fake RA. For LTP this appears to be a genuine RA and consequently it closes the session.

As RAs are no longer necessary in HSLTP, their dispatch can be disabled by a debug feature introduced in LTP enhanced.

Finally, to protect the final RS against possible losses (its arrival is necessary to confirm successful delivery of the block and consequent export session closure, see Figure 3), it can be adequately protected by replication, as shown in [29]. In the theoretical case that all copies are lost, HSLTP triggers cancellation of the export session when the RTO of the unconfirmed CP expires. The same remarks apply to the CS sent to the Tx engine in case of FEC failure, considered above.

VI. Performance Evaluation

We evaluated performance of HSLTP and other benchmark techniques (ECLSA [27] and LTP enhanced [29]) using a GNU/Linux virtual testbed, built and managed by Virtualbricks [33]. ION 3.6.2 [28] was installed on all virtual machines and DTNperf was [34] used to generate series of bundles of desired dimensions (100 kB). The codes used here are the LDPC Scalar provided by OpenFEC [32]. The LTP maximum segment size is of 1022, which leads to a coding matrix row length of *T*=1024, by allowing 2B to insert the actual LTP segment length [27]). As pointed out in the previous sections, LTP segments are in turn encapsulated into UDP/IP datagrams. As to physical layer processing operations, it is assumed that LDPC channel coding is applied to a stream of sync-marked transfer frames according to the CCSDS TM [36] specifications, similarly to what also taken as reference in the companion paper [27].

Finally, a data rate of 10 Mbit/s is considered and packet error rates (PER) of 10% and 15% have been taken into account in the performance analysis for the sake of the exemplification. A more detailed investigation about ECLSA performance with different codes (LDPC IRA provided by LibecDLR) and against different values of PER is provided in [27], which the interested is referred to.

A. *Immediate closing of LTP import session*

This scenario aims to evaluate the HSLTP performance improvement that may derive from immediately releasing the Rx buffer. This is particularly important on nodes with memory constraints, like most nodes in space.

As an extreme example, we assumed that there is only one buffer at Rx side. In LTP (and in ECLSA) the maximum number of export sessions is obliged to be the same as the number of Rx buffers available on the receiver, thus we can have only one export session. This means that we can send bundles only one by one, after the previous import session has closed, which ideally, with no losses, takes one RTT. By contrast, in HSLTP this symmetry constraint is released. The import session is always opened and closed immediately (what is more, independently of decoding success or failure), which allows the unconditional reuse of the sole Rx buffer, and thus the transmission of multiple blocks in sequence.

With no symmetry constraint, we set a maximum of 15 export sessions for HSLTP, instead of only one for LTP and ECLSA.

Of the many experiments done, only the two most significant are reported here.

1) *One Rx-buffer, PLR=10%*

The first experiment (Figure 8), considers the transfer of ten bundles generated simultaneously. The PLR is high (10%), but below the recovery capabilities of the FEC code used (about 11% with *Rc*=8/9 assuming ideal performance). The elapsed time is normalized to RTT for analysis convenience (here RTT=6s). We have four series: the first, on the y-axis, refers to bundle generation, the other three to the delivery times achieved by HSLTP, ECLSA and LTP. The superiority of HSLTP is immediately evident; its delivery time is only marginally longer than one-half RTT, the theoretical minimum without retransmissions. This proves that HSLTP processing time is already negligible for a propagation delay of only 3 s and that, more importantly, that all LTP blocks have been successfully recovered by the FEC mechanism, as expected. The same perfect recovery is also present in the ECLSA series, as shown by the regular intervals between consecutive markers, here almost equal to one RTT, i.e. the minimum duration of a Tx/Rx session. Last, in the LTP case, delivery time is further penalized by retransmission cycles of lost LTP segments (at least one RTT penalty for each additional retransmission cycle). In conclusion, in the presence of significant memory constraints, which could severely limit the number of Rx buffers, the advantage of HSLTP is outstanding.

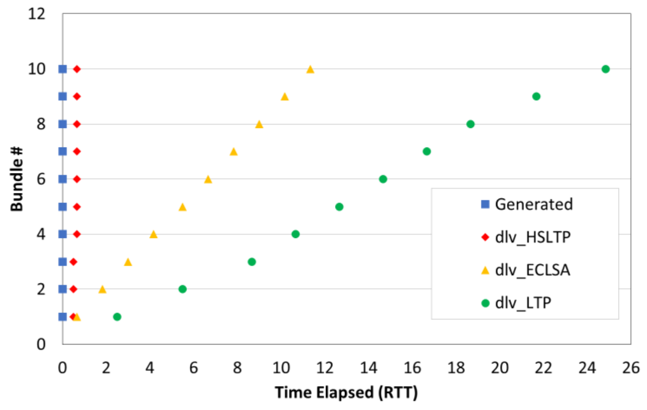

Figure 8: Performance comparison between HSLTP, ECLSA and LTP enhanced (with signal replication burst=3). Delivery series as a function of time elapsed. PLR=10% (on the forward link only); *Rc*=8/9; #Rx buffers=1; #Tx buffers=15 for HSLTP, 1 for others]. In HSLTP all bundles are delivered after about one-half RTT (no bundle retransmissions).

2) *One RX buffer, PLR=15%*

The second experiment (Figure 9) is the same, but with PLR=15%, which is deliberately a little higher (+2.5%) than code recovery capabilities. As a result, decoding may occasionally fail, as happens in HSLTP for blocks containing bundles 4 and 9. Each failure triggers session cancellation and bundle retransmission in a new successful session. The delivery time becomes only marginally longer than 1.5 RTTs, the theoretical minimum in case of retransmissions. By contrast, for ECLSA decoding fails for bundles 7 and 9, but in this case single LTP segments are retransmitted (those not previously recovered by FEC). The retransmission cycle requires about one RTT as before. Last, for enhanced LTP, segments lost are no longer protected by FEC so must all be retransmitted. The PLR is high, so the average number of retransmission cycles, is greater than with ECLSA, and this accounts for much longer bundle deliveries (Figure 9).

In conclusion, the comparative advantage of HSLTP over ECLSA and LTP is outstanding in this case too.

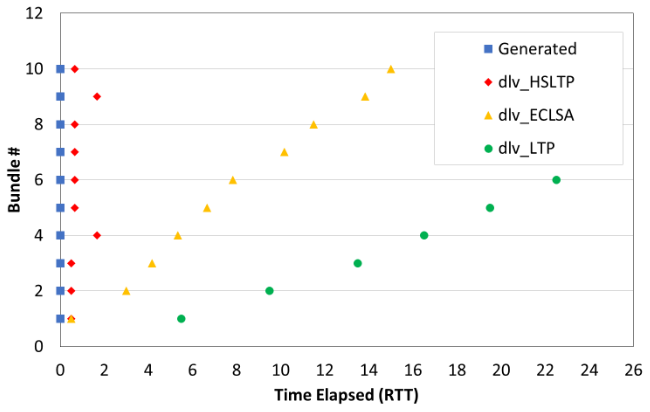

Figure 9: The same as Figure 8, but with PLR=15% (deliberately beyond the margin for *Rc*=8/9). For HSLTP note that the delivery time of #4 and #9 is now of about 1.5 half RTTs, due to FEC failure and consequent automatic bundle retransmission. For the other techniques, the higher PLR results in a much longer delivery time penalty.

B. *Reduction of ECLSA waiting time*

In Figure 10 the time necessary to transmit an ECLSA matrix (a codeword) is shown as a function of the number of its information symbols, *K*, for *Rc*=8/9 and a Tx rate of 10 Mbit/s. This time also represents the minimum value of the ECLSO "aggregation timer", after which a partially filled matrix is closed by ECLSO and then coded and sent. This timer has been eliminated in HSLTP as when LTP passes a segment flagged as EOB the matrix is immediately closed. It is worth noting that in ECLSA the corresponding delay can occur twice in the worst case. To illustrate this, let us consider the case of a small LTP block inserted into a new matrix, which is then almost filled up with subsequent blocks. The small block has to wait first for the aggregation timer to expire, then because decoding cannot start at Rx side until the coded matrix is completely received, which takes roughly the same time again.

These waiting times are almost eliminated in HSLTP as each block is contained in one matrix, which is decoded and delivered independently. This also leads to a simplification of HSLTP settings, now easier than in ECLSA. On the other hand, the immediate closure of the matrix at the end of a block can lead to inferior FEC performance with very small blocks, because of the reduced length of the codeword. It is however easily possible to reduce the likelihood of very small blocks by proper setting of LTP aggregation size and time. In addition, in HSLTP, as in ECLSA with the "*K* continuous" option, we set a threshold on the minimum number of redundancy symbols in a codeword, to cope with very small blocks.

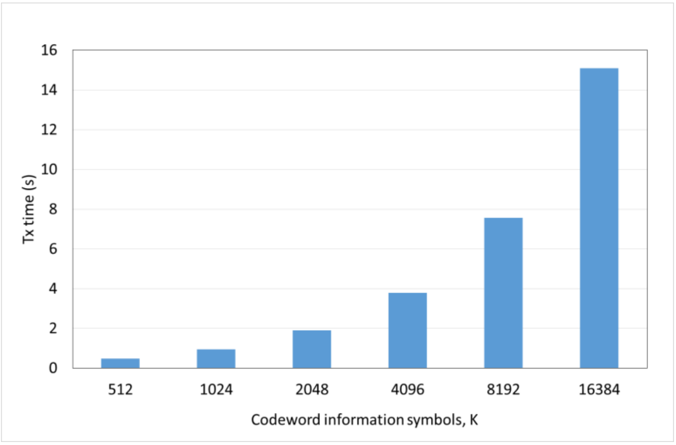

Figure 10: Transmission time of an ECLSA matrix as a function of its dimension. *Rc*=8/9 and *T*=1024B, Tx speed=10Mbit/s.

VII. Conclusions

HSLTP can be considered a variant of LTP designed for high-speed links, where the bandwidth is high and buffering requirements may become the performance-limiting factor.

HSLTP is based on upper layer coding FEC, like ECLSA, with two main differences. First, while ECLSA is designed to be transparent to LTP, and thus is unaware of LTP block boundaries, HSLTP is not, as it imposes an almost one-to-one correspondence between LTP blocks and codewords. Second, by contrast to ECLSA, in HSLTP LTP blocks are either entirely received (FEC success) or entirely dropped (FEC failure). This binary result eliminates by root the need of LTP data segment retransmissions in HSLTP. In fact, in HSLTP it is BP that takes responsibility of loss recovery by immediately retransmitting bundles contained in the discarded LTP block.

The most important advantage of this more radical approach is the possibility of using only one Rx buffer, even in the presence of high PLR, which makes the use of HSLTP very appealing in very high bandwidth-delay- product links, such as optical links in space, especially when the receiver has significant memory constraints.

The numerical results presented in the paper, achieved on a GNU/Linux testbed running the latest version of ION, confirm the validity of the approach. HSLTP is released as free software and is already included in ION as an optional extension of the ECLSA code.

Acknowledgements

The authors would like to thank maintainers of OpenFEC for their open source codecs. A special thanks to Scott Burleigh for ECLSA and HSLTP inclusion in ION as "contrib" code and for his valuable and continuous support to our research.

References

[1] S. Burleigh, A. Hooke,, L. Torgerson, K. Fall, V. Cerf, R. Durst, K. Scott H. Weiss, "Delay-tolerant networking: An approach to inter-planetary Internet," IEEE Communications Magazine, vol. 41, No. 6, Jun. 2003, pp. 128–136.

[2] Internet Engineering Task Force DTN Working Group (DTNWG) web site: https://datatracker.ietf.org/group/dtn/documents/ Accessed on: June 26, 2019.

[3] CCSDS DTN Working Group web site: http://cwe.ccsds.org/sis/ Accessed on: June 26, 2019.

[4] V. Cerf , A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, H. Weiss "Delay-Tolerant Networking Architecture," Internet RFC 4838, Apr. 2007.http://www.rfc-editor.org/rfc/rfc4838.txt Accessed on: June 26, 2019.

[5] CCSDS 734.0-G-1, "Rationale, Scenarios, and Requirements for DTN in Space," 2010, https://public.ccsds.org/Pubs/734x0g1e1.pdf Accessed on: June 26, 2019

[6] K. Scott, S. Burleigh, "Bundle Protocol Specification," Internet RFC 5050, Nov. 2007, http://www.rfc-editor.org/rfc/rfc5050.txt Accessed on: June 26, 2019.

[7] CCSDS 734.2-B-1, "Bundle Protocol Specification," 2015. https://public.ccsds.org/Pubs/734x2b1.pdf Accessed on: June 26, 2019

[8] A. Sabbagh, R. Wang, S. Burleigh, and K. Zhao, "Analytical Framework for Effect of Link Disruption on Bundle Protocol in Deep-Space Communications," IEEE Journal on Selected Areas in Communications special issue on Advances in Satellite Communications, vol. 36, No. 5, May 2018, pp. 1086-1096.

[9] G. Yang, R. Wang, A. Sabbagh, K. Zhao, and X. Zhang, "Modeling Optimal Retransmission Timeout Interval for Bundle Protocol," IEEE Transactions on Aerospace and Electronic Systems, vol. 54, No. 5, October 2018, pp. 2493-2508.

[10] M. Ramadas, S. Burleigh and S. Farrell, "Licklider Transmission Protocol – Motivation," Internet RFC 5326, Sept. 2008. http://www.rfc-editor.org/rfc/rfc5325.txt Accessed on: June 26, 2019.

[11] M. Ramadas, S. Burleigh and S. Farrell, "Licklider Transmission Protocol – Specification," Internet RFC 5326, Sept. 2008, http:/www.rfc-editor.org/rfc/rfc5326.txt Accessed on: June 26, 2019.

[12] CCSDS 734.1-B-1, "Licklider Transmission Protocol (LTP) for CCSDS". 2015. https://public.ccsds.org/Pubs/734x1b1.pdf Accessed on: June 26, 2019.

[13] Q. Yu, S. C. Burleigh, R. Wang and K. Zhao, "Performance modeling of Licklider transmission protocol (LTP) in deep-space communication," in IEEE Transactions on Aerospace and Electronic Systems, vol. 51, no. 3, pp. 1609-1620, July 2015.

[14] Z. Yang et al., "Analytical characterization of Licklider transmission protocol (LTP) in cislunar communications," in IEEE Transactions on Aerospace and Electronic Systems, vol. 50, no. 3, pp. 2019-2031, July 2014.

[15] R. Wang, Z. Wei, Q. Zhang and J. Hou, "LTP Aggregation of DTN Bundles in Space Communications," in IEEE Transactions on Aerospace and Electronic Systems, vol. 49, no. 3, pp. 1677-1691, July 2013.

[16] R. Lent, "Analysis of the Block Delivery Time of the Licklider Transmission Protocol," in IEEE Transactions on Communications, vol. 67, no. 1, pp. 518-526, Jan. 2019.

[17] G. Yang, R. Wang, S. C. Burleigh and K. Zhao, "Analysis of Licklider Transmission Protocol for Reliable File Delivery in Space Vehicle Communications With Random Link Interruptions," in IEEE Transactions on Vehicular Technology, vol. 68, no. 4, pp. 3919-3932, April 2019.

[18] S. Lin, D. Costello, M. Miller, "Automatic-Repeat-Request Error-Control Schemes," IEEE Commun. Mag., Vol.22, No.12, p5-17, Dec. 1984.

[19] V. Roca, C. Neumann and D. Furodet, "Low Density Parity Check (LDPC) Staircase and Triangle Forward Error Correction (FEC) Schemes," Internet RFC 5170, June 2008. http://www.rfc-editor.org/rfc/rfc5170.txt Accessed on: June 26, 2019.

[20] V. Roca, M. Cunche and J. Lacan, "Simple Low-Density Parity Check (LDPC) Staircase Forward Error Correction (FEC) Scheme for FECFRAME," Internet RFC 6816, Dec. 2012. http://www.rfc-editor.org/rfc/rfc6816.txt Accessed on: June 26, 2019

[21] T. de Cola, H. Ernst, and M. Marchese, "Performance analysis of CCSDS File Delivery Protocol and erasure coding techniques in deep space environments", Comput. Netw. Vol.51, no.14, pp. 4032-4049, Oct. 2007.

[22] T. de Cola and M. Marchese, "Reliable data delivery over deep space networks: Benefits of long erasure codes over ARQ strategies," in IEEE Wireless Communications, vol. 17, no. 2, pp. 57-65, April 2010.

[23] T. d. Cola, E. Paolini, G. Liva and G. P. Calzolari, "Reliability Options for Data Communications in the Future Deep-Space Missions," Proceedings of the IEEE, vol. 99, no. 11, p. 2069, 2011.

[24] CCSDS, "131.5-O-1, "Erasure Correcting Codes for Near Earth and Deep Space communications," 2014. https://public.ccsds.org/Pubs/131x5o1.pdf Accessed on: June 26, 2018.

[25] L. Shi et al., "Integration of Reed-Solomon codes to Licklider transmission protocol (LTP) for space DTN," in IEEE Aerospace and Electronic Systems Magazine, vol. 32, no. 4, pp. 48-55, April 2017.

[26] J. Jiao et al., "Reliable Deep-Space File Transfers: How Data Transfer Can Be Ensured Within a Single Round-Trip Interval," in IEEE Vehicular Technology Magazine, vol. 12, no. 4, pp. 86-94, Dec. 2017.

[27] N. Alessi, C. Caini, T. de Cola, M. Raminella, "Packet Layer Erasure Coding in Interplanetary Links: the LTP Erasure Coding Link Service Adapter", accepted for publication in IEEE Transactions on Aerospace and Electronic Systems, pre-print DOI: 10.1109/TAES.2019.2916271

[28] Sourceforge, "ION-DTN Delay-Tolerant Networking suitable for use in spacecraft", [Online]. Available: https://sourceforge.net/projects/ion-dtn/. Accessed on: June 26, 2018..

[29] N. Alessi, S. Burleigh, C. Caini, T. De Cola, "Design and Performance Evaluation of LTP Enhancements for Lossy Space Channels," Wiley, International J. of Sat. Commun. and Networking, pp.1-12 March 2018.

[30] G. Liva, E. Paolini and M. Chiani, "Bounds on the Error Probability of Block Codes over the q-Ary Erasure Channel," in IEEE Transactions on Communications, vol. 61, no. 6, pp. 2156-2165, June 2013.

[31] M.Raminella,"ECLSA enhancements to support the OpenFEC codec library and to take advantage of it characteristic features," undergraduate thesis, University of Bologna, Italy, Feb. 2016. An excerpt is included in the /contrib/ECLSAv2 directory in ION.

[32] INRIA, ISAE, OpenFEC web site: http://openfec.org/. Accessed on: June 26, 2019.

[33] P. Apollonio, C. Caini, M. Giusti and D. Lacamera, "Virtualbricks for DTN satellite communications research and education", in Proc. of PSATS 2014, Genoa, Italy, July 2014, pp. 1-14.

[34] C. Caini, A. d'Amico and M. Rodolfi, "DTNperf_3: A further enhanced tool for Delay-/Disruption- Tolerant Networking Performance evaluation," in Proc. GLOBECOM 2013, Atlanta, GA, US, 2013, pp. 3009-3015.

[35] CCSDS 132.0-B-2, TM Space Data Link Protocol, CCSDS Blue Book, Issue 2, September 2015.

[36] CCSDS 131.0-B-3, TM Synchronization and Channel Coding, CCSDS Blue Book, Issue 3, September 2017.