Packet Layer Erasure Coding in Interplanetary Links: the LTP Erasure Coding Link Service Adapter

Nicola Alessi, Carlo Caini, *Member IEEE,* Tomaso De Cola, *Member IEEE,* and Marco Raminella

Abstract[1] — Interplanetary Networks are affected by long propagation delays, intermittent connectivity, possible packet losses due to residual errors, and other impairments. To cope with these challenges, the Delay-/Disruption-Tolerant Networking (DTN) architecture utilizes the Licklider Transmission Protocol (LTP) as Convergence Layer on space links. The LTP reliable service (red) relies on ARQ, but very long propagation delays make Packet Layer Forward Error Correcting (PL-FEC) codes very appealing to protect LTP segments from losses. The key advantage of FEC is that LTP retransmissions would be limited to the unlikely case of decoding failures. To this end, a new FEC based protocol, to be inserted immediately below LTP, the Erasure Coding Link Service Adapter (ECLSA), is presented here. ECLSA is completely transparent to LTP, relies on two alternative external libraries for coding/decoding, LibecDLR and OpenFEC, both using LDPC codes and it is fully integrated with the ION DTN suite of NASA JPL. The paper aims to provide a solid description of ELCSA, including features functional in a real deployment (such as the dynamic selection of codes). Performance is evaluated at the end of the paper, with nearly ideal results. ECLSA is released as free software and is already included in the "contrib" section of ION.

Index Terms— Interplanetary Networking (IPN), Delay-/Disruption-Tolerant Networking (DTN), LTP, upper-layer-FEC, Low Density Parity Check codes (LDPC).

I. INTRODUCTION

The idea of building a computer network in space dates back to 1963 when John Licklider wrote a document with the visionary title "Memorandum for Members and Affiliates of the Intergalactic Computer Network", six years before the first man on the Moon and the first successful experiment on ARPANET, both in 1969. In the early 2000's research on InterPlanetary Networking (IPN) at NASA JPL made clear that a new architecture and new protocols were necessary to cope with space link challenges, including long delays, link intermittency, non-negligible losses due to harsh propagation conditions and asymmetric bandwidth. Because many of these were common to terrestrial "challenged networks", the IPN research was generalized as DTN (Delay-/Disruption-Tolerant Networking), with the ambitious aim of providing a common solution [1]. DTN architecture is based on the introduction of the Bundle Layer, usually between Application and Transport [2], [3] and the corresponding Bundle Protocol (BP) is in charge of moving "Bundles" between DTN nodes [4]. A key novelty of DTN architecture is the ability to use different transport protocols on different DTN hops, which, on interplanetary links, permits the use of the Licklider Transmission Protocol (LTP), specifically designed to cope with the space challenges [5], [6]. The DTN architecture and related protocols are currently standardized in parallel by IETF [7] and, for space applications, by CCSDS [8], [9], [10], [11].

LTP encapsulates bundles into LTP blocks, which are then split in multiple LTP segments. These segments are passed to UDP, by means of interfaces that are called Link Service Adapter (LSA) in ION, the DTN suite by NASA-JPL. LTP offers BP both reliable and unreliable service, with red and green parts of an LTP block respectively. With the former, recovery of lost LTP segments is based on ARQ (Automatic Repeat reQuest) [12], as in TCP. In contrast to TCP, however, LTP tries to concentrate all retransmissions in one cycle, at the end of the LTP block, to minimize "chattiness". In spite of this smart feature, when the propagation delay is in the order of minutes as on Earth-Mars links, the delivery time penalization due to even a single retransmission cycle is huge. It is therefore clear that the use of Packet Layer Forward Error Correcting (PL-FEC) codes considering LTP segments as information symbols, as done here, becomes very appealing, despite its complexity and extra bandwidth requirement.

The use of PL-FEC codes on erasure channels (not necessarily in space) is not new and has been treated in abundant literature, including a few RFCs [13], [14], [15]. If we restrict the scope to space channels, the first studies date back to 2007 ([16], then extended in [17]). Later on, the study in [18] gives an overview of how (and where in the protocol stack) PL-FECs could be used in space, with the pros and cons of different options. The implementation of PL-FEC is in fact considered complementary to the use of physical layer channel coding as typically used in current data communications: the use of PL-FEC is aimed at recovering the datalink layer frame losses resulting from the excessive number of bit errors not

[1] Manuscript received XX, 2018.

Nicola Alessi, Carlo Caini and Marco Raminella are with the University of Bologna, Italy, and may be contacted at nicola.alessi4@unibo.it and carlo.caini@unibo.it marco.raminella@studio.unibo.it; Tomaso de Cola is with DLR- German Aerospace Center, Institute for Communications and Navigation, Weßling, Germany, and may be contacted at tomaso.decola@dlr.de .

recoverable by the corrective capacity of physical layer channel coding. Among the different possibilities presented in the CCSDS Orange Book [19], the authors' preferred solution consists in inserting the PL-FEC immediately below LTP, as analysed in this paper devoted to the Erasure Coding Link Service Adapter (ECLSA). This is not a mere internal interface from LTP to lower protocols as other LSA (e.g. UDPLSA), but a real intermediate layer, with FEC capabilities. The same approach, with different codes (Reed Solomon instead of LDPC) has been recently followed by other DTN researchers [20], [21], so confirming its validity.

ECLSA origin dates back to 2014 [22] and is the fruit of long established collaboration between DLR and the University of Bologna on DTN. The aim of this paper is to describe the second version (ECLSAv2), rewritten from scratch under new specifications. Among these we must cite the support to multiple FEC families (LDPC IRA and LDPC Staircase), the dynamic selection of both codeword length and code rate, and the use of a dynamic matrix buffer to reduce RAM occupancy, the completely new thread structure and the new header. Thanks to these improvements, ECLSA is now a real working protocol, fully integrated with the other protocols contained in ION [23]. In view of this, the paper aims to provide the reader not only with ECLSA rationale and general description, but also with the implementation details necessary to understand the complexity of transforming the original concept into a running code, plenty of options. To evaluate ECLSA performance the authors made use of a GNU/Linux virtual testbed running ION and the full protocol stack. Results, presented at the end of the paper, look very encouraging.

II. DTN Overview

This section summarizes the basics of DTN architecture and of LTP protocol for the unfamiliar reader.

A. *DTN architecture essentials*

DTN architecture relies on the introduction of the BP in selected nodes, between Application and lower layers (usually Transport). The interfaces with lower layers are called "Convergence Layer Adapters", and are specific for each protocol (TCPCLA, LTPCLA, UDPCLA, etc.). There are two critical differences with respect to Internet [2], [3]. First, by contrast to Internet packets, bundles can be stored for long periods at intermediate nodes, which is necessary to cope with link intermittency typical of space and other challenged environments. Second, as the Transport scope is no longer end-to-end, but confined within a DTN hop , it is possible the use of different transport protocols on different DTN hops, which is essential to tackle the different channel impairments present on each hop (see Transport A, B and C in Figure 1).

As shown in [24] the use of different transport protocols on different portion of the end-to-end path, somewhat extends the concept of TCP splitting PEPs, widely used in GEO satellite networks. Considering for example Earth to Mars communications, TCP could be still successfully used on both Earth and Mars, but it should be replaced by LTP on space links because of both link intermittency and propagation delay (with reference to Figure 1, we could have TCP in A and C hops and LTP in B).

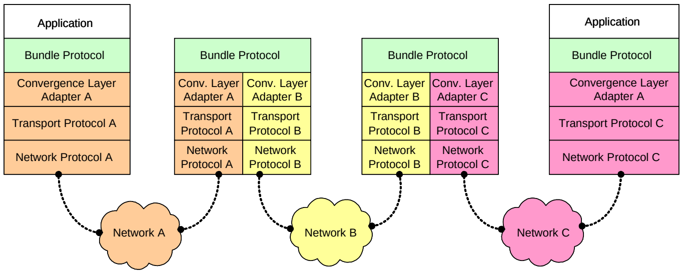


Figure 1: DTN architecture protocol stack; the end-to-end path is divided into three DTN hops, on each of which different transport (and lower) protocols can be used.

B. *Licklider Transmission Protocol*

LTP was specifically designed to cope with long delays and link intermittency typical of space links. To cope with long propagation delays LTP minimizes interaction between transmitting and receiving engines and replace the feedback-based TCP window based congestion control with a rate-based one. LTP can also successfully handle scheduled intermittent connectivity, taking advantage of "contact" information reported in contact plans.

As mentioned, LTP offers both reliable and unreliable services, with "red" and "green" parts of a LTP block, respectively. Let us focus on reliable service, by far the most important, and list the key features that differentiate LTP from TCP [6], [11].

- No connection establishment, such as TCP 3-way handshake.
- Unidirectional data flow (the reverse channel is used only for signaling) to cope with possible channel asymmetry.
- Multiple bundles can be aggregated by the LTPCLA in one LTP "block" [11]; blocks are transmitted by independent LTP "sessions"; multiple sessions in parallel are allowed.
- An LTP block is split into a number of LTP "segments", each passed to UDP.
- Segment acknowledgments are sent back in response to "check points", usually set only on the last segment of a block, to minimize retransmissions cycles.

A typical LTP red session with segment losses is presented in Figure 2. All LTP segments are transmitted in order, the last being flagged as End of Red Part (ERP), End of Block (EOB) and Check Point (CP). Segments 2, 5 and 6 are lost and thus not confirmed by the first Report Segment (RS) sent in response to the check point; this report is confirmed by a report-ack followed by retransmissions (R), the last flagged as check point. As there are no further losses on retransmitted segments, the second report segment is a "final" report, confirming the arrival of all segments. Delivery time and penalty time due to loss recovery are indicated by arrows on the right.

The performance investigation of LTP has received not little attention in recent years because of its suitability for both deep-space [25] and cislunar [26] scenarios, and of the implications of interfacing to the bundle protocol [27]. In this regard, the LTP performance dependence on propagation delay is worth stressing [28]. If the propagation delay is much longer than the time required to transmit a block, as is typical in deep space, the delivery time in the absence of losses can be reasonably approximated as the propagation delay (e.g. as 1/2 RTT), which is the theoretical minimum. Then, in case of losses on data segments only, and assuming no further loss on retransmitted segments, as in Figure 2, only one retransmission cycle (1 RTT) is sufficient to recover all data segment lost; this means that the delivery time is equal to 1.5 RTT, the theoretical minimum for ARQ based protocols, which is again an excellent result.

LTP, however, is more sensitive to loss of signalling segments, such as report segments or report-ack, or data segment flagged as check points, in which case the penalization can become of two RTTs or even more. Moreover, the original version of LTP was also particularly sensitive to the loss of the last report ack, confirming the final report segment. This has led to the development of an enhanced version of LTP [28], where signalling segments can be protected against losses by a variable amount of replication and the problems with the loss of the last report-ack fixed. This resulted in a much stronger resilience against losses, especially when frequent. The LTP “enhanced” version has thus become the standard in current ION releases and is also used here.

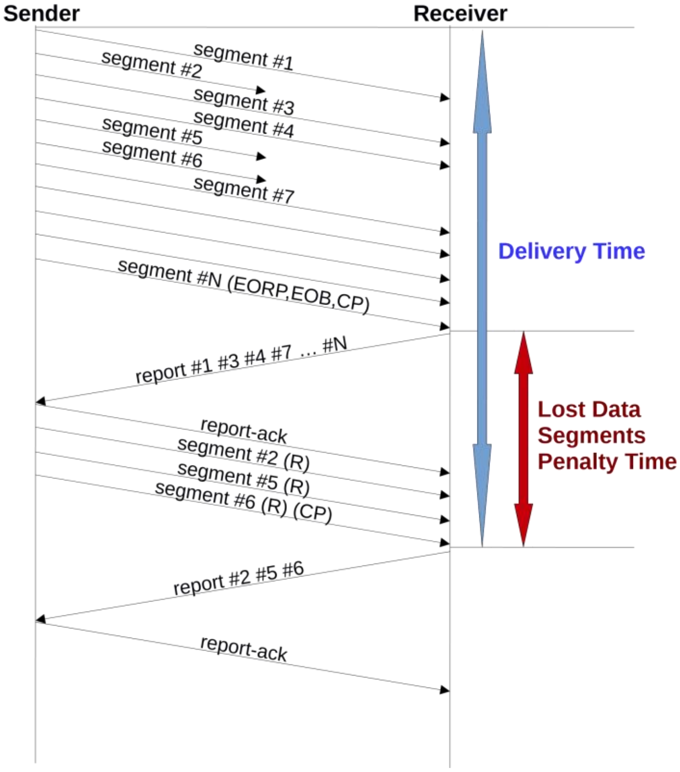


Figure 2: Example of LTP session (red) in the presence of losses on data segments.

III. Upper Layer Forward Erasure Coding

A. *Rationale of erasure codes at upper layers*

Let us recall that a FEC code is a code that transforms a message of K symbols into a codeword, i.e. a longer message with N (N=K+M) symbols; if the code is systematic the first K symbols of the codeword are the same as those of the message, so the final Ms are redundancy symbols. The code rate, R_c=K/N, represents the amount of information symbols per code symbol; the lower the code rate, the higher the redundancy introduced [29].

The FEC techniques that are usually applied to bits can be also applied to packets, in particular on erasure channels to recover from losses; in this case the terms “packet layer”, “application layer”, “upper layer” coding are alternatively used. At upper layers, packets can be lost either because of network congestion or because of residual error rate on bits (after coding on the physical channel) prevents the packet from passing the parity check. Lost packets are recovered in both Internet protocols and in LTP red, by means of ARQ. Although this is the best as well as the simplest solution in Internet, where worst case RTT is in the order of few hundred ms, it becomes extremely inconvenient in interplanetary links, where the propagation delays are much larger (from 3 to 23 minutes for Earth to Mars links). This is why erasure coding on packets is so appealing on space links, despite its disadvantages, such as complexity and the additional bandwidth required to transmit redundancy packets.

B. *LDPC coding in ECLSA*

ECLSA makes use of two alternative external libraries to perform erasure coding and decoding, LibecDLR and OpenFEC. Both of them use Low Density Parity Check codes [29], but of two different families: LibecDLR uses the Irregular-Repeat-Accumulate (IRA) family (see the CCSDS Orange book [19]), while OpenFEC relies on LDPC Staircase and LDPC Triangle codes (see RFC 5170 [13]). RFC 6816 [15] further specifies LDPC Staircase so that they can be used to protect media streams along the lines defined by FECFRAME, a particular framework for using FEC applications in IP networks, and makes an explicit reference to OpenFEC.

The LibecDLR library is proprietary and is not included in the open source version of ECLSA in ION, while OpenFEC is open source [30]. Although they use different LDPC codes, both of them can support the N and K values specified in the CCSDS Orange book [19], shown in Table I.

TABLE 1: N AND K OF CODES DEFINED IN THE CCSDS ORANGE BOOK [19]

	R_{c1}=8/9	R_{c2}=4/5	R_{c3}=2/3
K_1=512	N_{11}=576	N_{12}=640	N_{13}=768
K_2=2048	N_{21}=2304	N_{22}=2560	N_{23}=3072
K_3=16384	N_{31}=18432	N_{32}=20480	N_{33}=24576

C. *LDPC Decoding*

LDPC decoding consists of solving a system of N-K linear equations whose unknown variables are the L missing symbols of the N-length code word. A first decoding attempt can thus be done with a simple iterative (IT) algorithm that tries to solve the equations where only one unknown variable remains, one at a time. IT is fast but its decoding performance is suboptimal. Maximum Likelihood (ML) techniques are also possible; they offer better decoding performance, but also require more computations. Hybrid solutions [31], [32] used by both LibecDLR and OpenFEC, start by using the IT algorithm and complete the decoding, if necessary, with the ML one, thus obtaining the best of both worlds: moderate complexity and excellent performance.

IV. ECLSA Overview

To describe ECLSA it is necessary to recall first the role played by LTP LSAs (Link Service Adapters). In the DTN architecture with LTP as convergence layer, after LTP block segmentation, LTP segments are transferred to the corresponding pair via a lower layer protocol, e.g. UDP. LSAs are the LTP interfaces towards these lower protocols. By contrast to UDPLSA (and other possible LSAs), ECLSA is not just an interface internal to the LTP code, but a new layer with FEC capabilities intermediate to LTP and lower layers and totally transparent to both (

Figure 3). ECLSA is segment oriented, i.e. it does not consider LTP blocks boundaries at all, which has the advantage of offering a natural equal protection of all segments, either original data segments, or retransmissions, or signalling (whose protection is of particular importance as highlighted in the LTP section). On the other hand, this requires the introduction of aggregation timers, as it will be shown below.

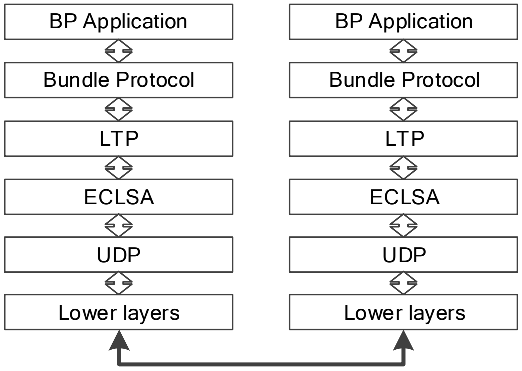


Figure 3: The protocol stack of two DTN nodes using ECLSA on top of UDP.

ECLSA consists of two processes, ECLSO and ECLSI, respectively in charge of segment transmission (Outduct) and reception (Induct).

ECLSO processing involves three logical phases (left hand of Figure 4):

- K LTP segments are passed to ECLSA; each segment is to be treated as an information symbol (Info packets in the figure)
- The LDPC encoder adds M=N-K redundancy symbols (Redundancy packets in the figure).
- The N (=K+M) symbols of the codeword are passed one-by-one to UDP; each symbol is to be encapsulated into one UDP datagram.

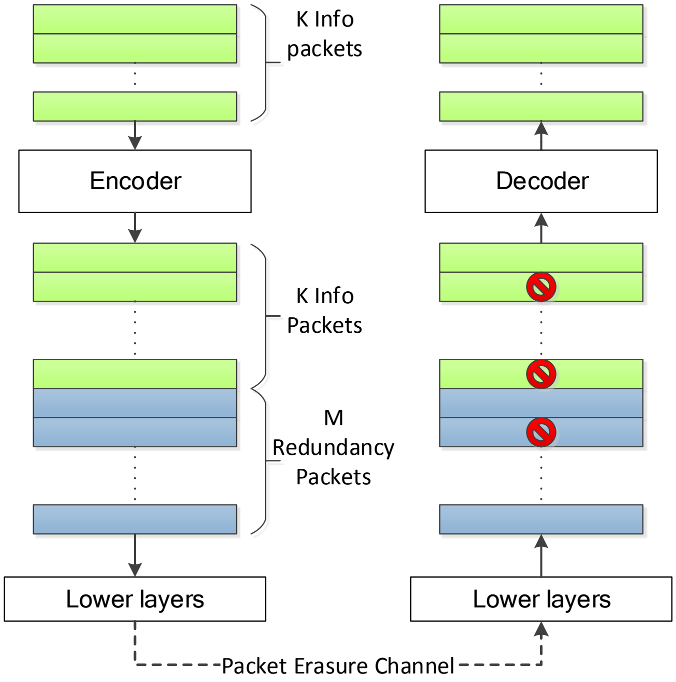


Figure 4: Simplified picture of PL-FEC logical process in ECLSA.

On the receiver side, both information and redundancy packets can be lost because of link impairments. The task of ECLSI is to mask these losses as much as possible to LTP. It comprises the following three phases, which are the dual of those just described but in reverse order (now from UDP to LTP):

- Let L be the number of UDP packets lost, each containing one symbol of the codeword (3 crossed packets on the right side of Figure 4); N-L symbols have arrived.
- The decoder try to extract the K information symbols from the N-L ones received. If the decoding is successful, (as in Figure 4) all the K information symbol are recovered, thanks to the redundancy symbols. This decoding phase is skipped if there are not any losses in the first K symbols, i.e. if all information symbols have arrived.
- All known information symbols (either received or recovered) are passed to LTP.

A successful decoding needs reception of at least K of the N symbols transmitted (i.e. L≤M) for ideal codes, whatever the received symbols are (information or redundancy), but is usually not sufficient for LDPC codes, where a small margin is requested (L<M). In the unlikely event of unsuccessful decoding, some information symbols will be still missing. They will be recovered by LTP (red) by means of the usual ARQ mechanisms, being ECLSA transparent to it. Note that for the same reason retransmitted segments (and related signalling segments) will be protected by ECLSA as original segments were, making thus extremely unlikely the event of consecutive losses of the same segments (two decoding failures should happen).

V. ECLSO Detailed Description

Here we re-examine the logical phases performed by ECLSO in detail, with a look to our implementation, where they correspond to three threads, which run concurrently for better efficiency [33].

A. *First thread (T1): Matrix filling (from LTP)*

LTP segments are passed by LTP to ECLSO, each segment destined to become one information symbol of the N-symbol codeword. All symbols must have the same size, thus if we imagine a symbol as a row vector of length T (in bytes), the N-symbol codeword becomes a matrix of N rows and T columns, as shown in Figure 5.

Although LTP data segments are usually of the same size (set in LTP configuration) some segments may be shorter than others, as usually happens for the last segment of an LTP block (also flagged as CP). To deal with this length diversity, the first two bytes of a symbol must be reserved for indicating the actual size of the LTP segment contained in the symbol. If the LTP segment is shorter than T-2 (maximum possible length), the tail bytes must be filled with zeros ("row padding", in white). Another kind of padding is necessary if the number of LTP segments received by LTP, I, is lower than K. In this case the K-I empty rows are filled with "information" padding (row in white). After encoding, the remaining M rows are filled with redundancy symbols (last rows).

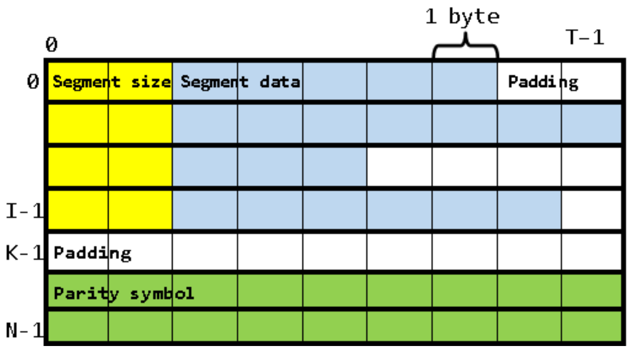


Figure 5: An example of coding matrix. Row and column index from 0. N= codeword dimension; T= symbol size (B); K= number of Information symbols; I=actual number of LTP segments added. The first two bytes represents the LTP segment size, the other bytes the LTP segment data, last rows the redundancy symbols. White spaces are padding.

The matrix-filling algorithm is summarized in Figure 6. When the first LTP segment is received, the aggregation timer starts and the segment is added in the first row; the process goes on either until K segments arrive, i.e. the information part of matrix is full, or until the aggregation timer expires. In both cases the matrix is passed to the encoder (Matrix encoding phase) and then to UDP (Matrix passing phase).

In order to avoid both premature expiring and excessive delays, it is crucial to set the aggregation timer properly. A conservative rule is to set it to the maximum time necessary for LTP to pass K information symbols at the nominal Tx rate (declared in the contact plan configured in ION and expressed in B/s), resulting in (K T)/Tx_Rate. Note that the aggregation timer expires only in the absence of persistent traffic, i.e. when the last LTP block inserted in the matrix is not promptly followed by others, meaning that there are no more bundles to be passed to LTP. As ECLSA is transparent to LTP, both retransmitted and signalling segments are naturally multiplexed to data segments of new LTP blocks, thus reducing the probability of a timer expiring.

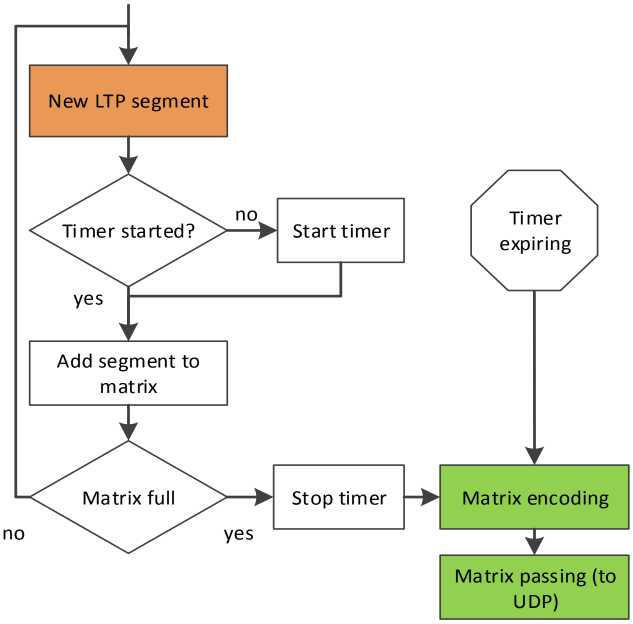


Figure 6: ECLSO process: matrix filling from LTP (T1, expanded) followed by matrix encoding (T2) and matrix passing to UDP (T3).

B. *Second thread (T2): Matrix Encoding*

The coding adapter, LibecDLR or OpenFEC is external to ECLSA and must be chosen at compilation time. Code selection can be either static (default) or dynamic (with 3 adaptive options), and the latter is one of the most powerful characteristics of ECLSA. In both cases it is necessary to specify a reference code (N_{span}, K_{span}) in ECLSO settings (see Table I); the chosen pair indirectly sets also the reference code rate, R_c.

1) *Selection of K*

Let us start by considering a static code rate R_c (e.g. R_c=8/9) and examine the pros and cons of different K values. A large K performs closer to the ideal but there are three cons: greater amount of memory in use, longer coding delay, and more probable excess of redundancy. In fact, whenever the aggregation timer expires, because of lack of new LTP segments, there are only I (I<K) info rows filled; although the K-I padding rows are not sent (they are empty), all the M redundancy rows are. Thus, if I<<K the result is that the actual code rate (R_{c_actual}=I/N) can be much lower than the nominal code rate (R_c=K/N), with a corresponding redundancy excess. For example, if the code selected is (24576, 18432), there are M=6144 redundancy rows; if the aggregation timer expires when only a small fraction of the information matrix rows are filled (e.g. I=500), the redundancy will be many times larger than requested by the nominal rate (about 37 times). To keep the best of both long and short codes, a dynamic selection is in order.

If the "Adaptive selection of K" option is set, the K reference value (K_{span}) will represent the maximum K value, to be selected on the basis of the two other drawbacks, memory and delay, so allowing ECLSI to reduce the actual K if the matrix is not full (I<Kspan). In the example considered, K=512 would be the best choice, as 500<512, with an actual code rate very close to the nominal one.

As LDPC Staircase codes used by OpenFEC allow for on the spot building of a code for a given K, it is even possible to tailor the code on the actual number of rows filled when the aggregation timer expires, i.e. we can set K=I. In this case, we can also have a perfect match of the actual code rate, i.e. no redundancy rows in excess. This feature can be set in ECLSA (OpenFEC only), by selecting the alternative “K continuous” option.

2) *Selection of R_c*

Moving on to the code rate, we recall that the smaller the R_c the higher the amount of redundancy introduced per information symbol, and thus the more powerful the code. For example, assuming an ideal code, with R_c=8/9 we can on average recover one loss over 9 packets, with R_c=4/5, one over 5, with R_c=2/3, one over 3. R_c choice should therefore dictated by an estimate of the channel Packet Loss Rate (PLR). If the channel is stationary and loss probability known, it is easy to make the right choice. Otherwise, if the channel is time-varying, or is stationary but unknown, a sensible choice is to conservatively select R_c (high redundancy). This, however, would result in excess of protection whenever the channel is better than the worst case. To cope with this second cause of potential bandwidth waste, in ECLSA it possible to dynamically select the code R_c, by enabling the “Feedback Adaptive R_c” option. In this case, the variable target R_c rate is calculated on feedbacks (decoding success/failure plus additional information) sent back by the ECLSI process running on the receiver node. It is worth stressing that this feature is useful only if the coherence time of the channel is longer than the RTT, otherwise it can become ineffective or possibly harmful, given the impossibility of tracking the channel. For a comprehensive description of the adaptive options, see [33].

C. *Third thread (T3): Matrix passing (to UDP)*

After encoding, the N symbols of the codeword (rows of the matrix) are read from the matrix and inserted one-by-one into UDP datagrams, skipping padding to save bandwidth.

The encapsulation performed by ECLSO (and vice versa by ECLSI) is shown in Figure 7. Note that the figure holds true only for the K information symbols (the first two layers are missing for the M redundancy symbols, entirely generated by the FEC coding process).

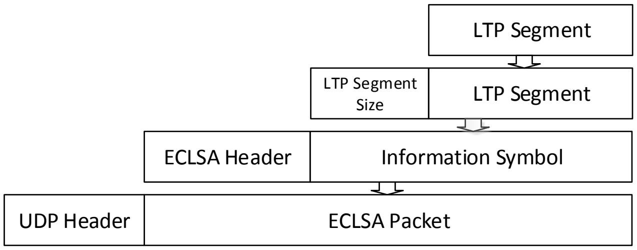


Figure 7: LTP segment encapsulation in a UDP datagram via ECLSA.

The ECLSA header contains the information necessary to ECLSI running on the receiver to correctly fill the coding matrix at reception with the received symbols and to use the same FEC code used for encoding in the decoding phase. The current format (the protocol is in an experimental phase, thus modifications are likely) is shown in Figure 8.

Ver.	Ext.	Flags	Engine	Matrix	Symbol	I	K	N	T
(1B)	(1B)	(1B)	ID (2B)	ID (2B)	ID (2B)	(2B)	(2B)	(2B)	(2B)

Figure 8: ECLSA header.

The header contains the following fields:

- Version (1B): this value is at present set at 0.
- ExtCount (Extension Count, 1B): reserved for future purposes.
- Flags (1B): if b0=1 the decoding feedback is requested; if b1=1 the k continuous mode is enabled. Others bits are reserved for future purposes.
- EngineID (2B): the node identifier, as given by the upper protocol. For LTP, it is the same as the LTP Engine ID.
- MatrixID (2B): it is the unique identifier of the coding matrix (i.e. of the codeword) for the given EngineID. Thus, the pair [EngineID,MatrixID] is the unique identifier for ECLSI (ECLSI can receive by multiple senders). This pair allows ECLSI to know to which coding matrix (codeword) an incoming packet belongs.
- SymbolID (2B): the codeword symbol index; This is the only field that changes for ECLSA packets referring to the same coding matrix.
- I (SegAdded, 2B): the actual number of LTP segments added to the coding matrix (see Figure 5). Used by ECLSI to know the amount of matrix padding (unfilled rows).
- K (2B): The FEC parameter K of the code actually used for encoding;
- N (2B): The FEC parameter N of the code actually used for encoding; K and N fields are necessary to let ECLSI know the code used by ECLSO (for the specific codeword, if the FEC code selection is dynamic).
- T (2B): The symbol size. It is necessary to ECLSI to know the length of a coding matrix row.

VI. ECLSI Detailed Description

The three logical phases of ECLSI are implemented as independent threads [33], as those of ECLSO.

A. *First thread (T1): Matrix filling (from UDP)*

The first phase of ECLSI is the dual of the last of ECLSO and is described in Figure 9. ECLSA packets are extracted from incoming UDP datagrams, then the ECLSA header of each packet is read to derive the parameters of the coding matrix to which the ECLSA packet belong. Each coding matrix is assigned a unique identifier, which also identifies the source node (the same as engineID), to cope with possible multiple sources, and is temporarily stored by ECLSI in a buffer of matrixes (dynamic to save RAM). From the ECLSA header, the field “I” tells ECLSI how many information symbols are actually present in the coding matrix (thus we have K-I information padding rows), while N, K are the parameters of the FEC code used to encode the received codeword and T the symbol dimension in bytes. These four parameters are associated to one coding matrix, thus are the same for all the corresponding ECLSA packets. They are deliberately replicated in all packets because in the presence of losses it is not possible to know which symbols arrive and which are lost, thus it would be unsafe to send this information with only one specific symbol.

For the very same reason, it is paramount to define when the matrix filling process can reasonably be considered completed, a much less trivial task than might be imagined. In ELCSA the process is considered completed when one of these four conditions occurs, whatever the first:

- All information symbols have been correctly received (this is why the ECLSA header must contain the "I" value). In this case there is no need to wait for redundancy segments nor to decode the matrix. This happens only when the loss probability is very low.
- Some information symbols are missing, but the last redundancy symbol has arrived. This usually happens in the presence of moderate losses.
- A symbol generated by the same source and belonging to a coding matrix with higher sequential ID than the current one has arrived. The probability of this happening is equal to the loss probability, as it requires the loss of a specific symbol, the last redundancy symbol.
- A timeout occurs. On reception of each new symbol of a specific codeword, a closing timer is set (first symbol) or reset (other symbols); if no other symbols arrive the timer expires and the coding matrix is considered complete. This aims to prevent deadlocks, should the last redundancy symbol of the last matrix sent by the same source be lost. The probability is very low, as it requires not only the loss of the last redundancy symbol, but also that the codeword is the last of the flow (no other symbols are coming). In addition to that, the closing timer is usually very small, as it should be set to the expected inter-arrival time between two consecutive ECLSA packets, which is usually very short, plus a small safety margin to tackle possible losses. In brief, its impact on delivery delay is virtually always negligible.

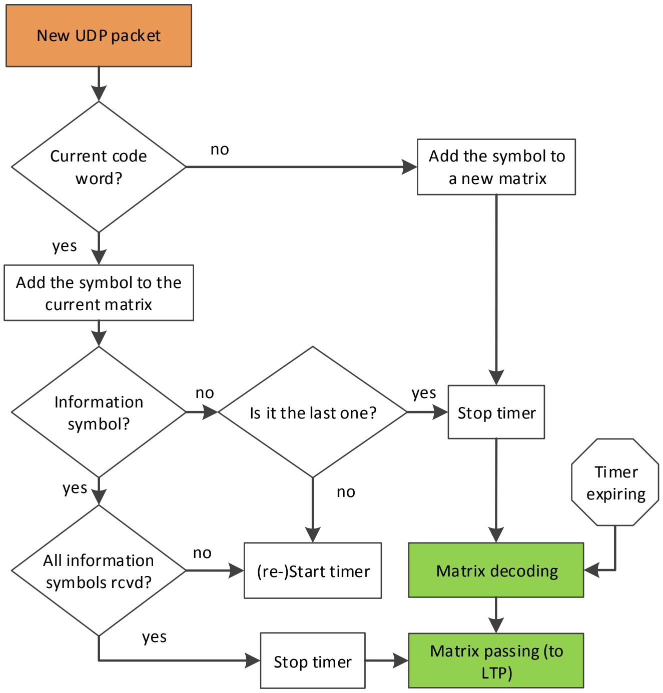


Figure 9: ECLSI process: matrix filling from UDP (T1, expanded) followed by matrix decoding (T2) and passing to LTP (T3).

B. T2: Matrix decoding

Decoding is skipped either if all information symbols are received or if the matrix was not encoded (because the segments added before a timeout, "I", were less than the "Encoding Threshold" parameter). In all other cases the matrix must be decoded. To this end, ECLSI must know the code used, which is specified by the parameters N, K, and T as above.

C. Third thread (T3): Matrix passing (to LTP)

After decoding, the LTP segments contained in the information symbols are passed sequentially to LTP (the 2B header and row padding are removed). If the "Request Feedback" option is set, feedback is sent to the ECLSO engine on the source reporting the decoding status (success or failure, plus other information). The feedback packet is sent directly by ECLSI to ECLSO through UDP, without applying any FEC. To protect it from losses, it can however be replicated by setting the "Feedback burst" parameter in ECLSI configuration.

As said, if decoding fails, LTP will require retransmission of missing segments (red part), which should theoretically be completely transparent to ECLSA. In practice, however, there is a small difference, as LTP RTO timers must now include the ECLSA processing delay.

VII. Performance Evaluation

ECLSA performance evaluation is complex, as results depend on a variety of factors, including the codes used and the characteristics of the erasure channel. Having assumed uncorrelated losses[2], the first step is to assess the performance of available erasure codes. To this end, we used a virtual GNU/Linux testbed with Virtualbricks [34], running the full protocol stack, with ION on all VMs. As illustrated in Figure 7, LTP/ECLSA runs on top of UDP, whose generated packets are in turn encapsulated into IP datagrams and eventually transferred over Ethernet. Packet losses and the other characteristics of the scenarios studied (for more details, see the next two subsections) are reproduced by means of a channel emulator, positioned on the link connecting the two nodes involved, working at link layer (in particular, packet losses are injected on frames). In real environment there are some important differences with respect to the protocol stack imposed by the emulation environment. In CCSDS-based space missions, the IP datagrams to be transmitted over the space link cannot be encapsulated into Ethernet frames, but according to CCSDS standards will be properly transported by the Telemetry (TM) transfer frame on downlink (i.e., from a spacecraft to a ground station) [35] and Telecommand (TC) transfer frame in uplink [36]. In this paper, it is assumed that

[2] The assumption of uncorrelated packet losses stems from the fact that bit errors even correlated within the same frame will result in the discarding of that frame upon CRC validation. In the case of fading events extending to multiple frames, it is common to resort to the application of interleavers, helping to spread errored bits over several frames and eventually restoring the pattern of uncorrelated losses. The occurrence of correlated frame losses is typical in free-space optical communications. The present paper, however, does not focus on a specific transmission technology (i.e., RF or laser) but provides a general study of ECLSA performance.

the physical layer of the downlink implements LDPC 4/5 channel coding (as one of the possible recommendations available in the CCSDS TM standard) applied to a stream of sync-marked transfer frames, allowing up to 2048 information bytes to be accommodated in the corresponding information blocks subject to LDPC encoding. In this respect, LTP segment length was set in our experiments to 1024 bytes (T=1026), which means that the loss of a TM transfer frame corresponds to the loss of an LTP segment. Concerning the uplink, it is assumed that the physical layer implements the LDPC 1/2 (128,64), as recommended by the TC CCSDS standard, which allows a very low undetected codeword error rate, in the order of 10^{-5} for operational values of E_b/N_0. This allows us to assume the uplink to be error-free.

A. *PL-FEC LDPC performance comparison*

We commence the analysis by comparing performance achievable by the packet layer LDPC codes (PL-FEC) provided in LibecDLR (IRA) and OpenFEC (Staircase), with ideal performance, achievable by Maximum Distance Separable (MDS) codes, which can be expressed through the Singleton bound of idealised MDS codes [37]:

$$P_{err} = \sum_{i=N-K+1}^{N} \binom{N}{i} \varepsilon^{i} (1-\varepsilon)^{N-i},$$

where ε denotes the packet loss rate), (N,K) denotes the reference MDS code and P_{err} denotes the error decoding probability. Hence, the successful decoding probability (rate in the rest of the paper) can be computed as 1- P_{err}.

Let us start from the (576,512) code (R_c=8/9), which is likely the most appealing from an operational point of view, at least for PLRs <8%. LDPC code results shown as markers in Figure 10 were obtained by sending in our testbed isolated bundles of 500kB as green LTP blocks (one bundle per block, and one block per matrix), using the DTNperf tool [38]. Each block consists of 494 information segments (i.e. I=494, very close to K=512). Ideal performance is analytically computed and plotted with a continuous line; for the sake of comparison, it refers to the actual code rate R_c=I/(I+M), just marginally lower than the nominal one. By examining the figure, we can see that the two LDPC codec offer performance very close to the ideal one). This proves that both IRA and Staircase families are very effective.

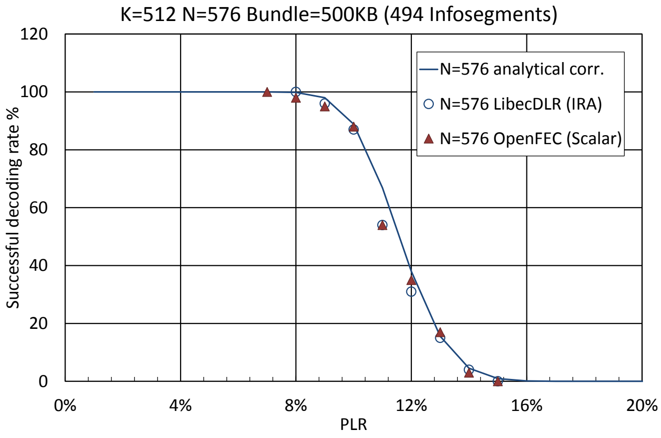


Figure 10: Successful decoding probability vs packet loss rate for (576,512) codes.

Then, by keeping the same K we rose N to 640, to increase redundancy, obtaining results presented in Figure 11 (Rc=4/5, one redundancy symbol every four information ones). To keep centred the knee (now around 20% in accordance with the R_c value) the chart shows the values starting from 10% on the x-axis. LDPC codes performance is still very close to the ideal one. Finally, we further increased N to 768, obtaining Figure 12 (Rc=2/3, one redundancy symbol every two information ones). The knee is now around 33%, again in accordance to R_c.

In conclusions, in all cases considered here, both IRA and Staircase LDPC code families offer performance very close to the ideal one, with only minor differences between them.

These results can also be interpreted in LTP terms, by stating that ECLSA, when coupled with LTP green, offers a sort of "almost reliable service" to the bundle protocol. Chances of losses inside one LTP green block are negligible until the PLR is lower than the recovery capabilities of the codes used, i.e. before the knee (see Figure 10, Figure 11 and Figure 12).

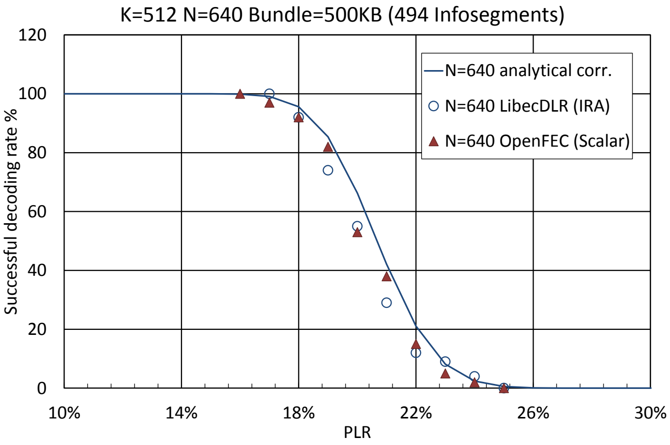


Figure 11: Successful decoding probability vs packet loss rate for (640,512) codes.

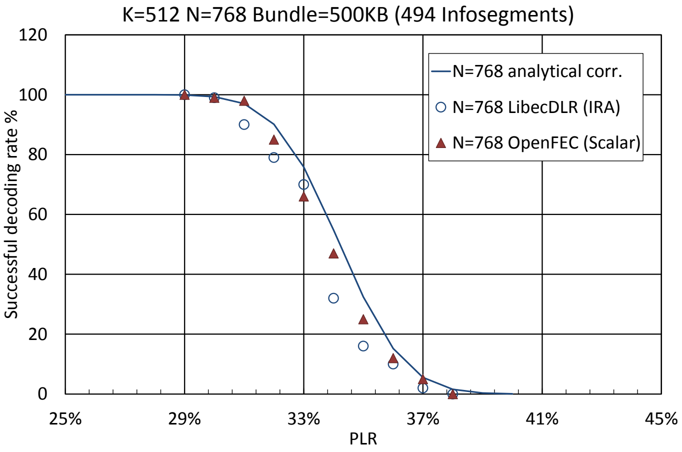


Figure 12: Successful decoding probability vs packet loss rate for (768,512) codes.

B. *ECLSA and UDPLSA performance comparison*

The second and most significant step of our analysis consists in directly comparing ECLSA and UDPLSA. To this end we have considered the same experiment as before, but with LTP red. A new performance metric, delivery time, is in order here, because, thanks to retransmissions, LTP red blocks are always completely received (and the encapsulated bundles delivered). The question here is thus *when* and not *if* blocks are completely received.

Delivery time is strongly influenced by propagation delay, and to a lesser extent by bandwidth. Results presented in Figure 13 refer to RTT=2s (short by interplanetary standards, roughly corresponding to Earth to Moon RTT), and to a Tx rate of 10 Mbit/s. Delivery time is expressed in half RTT units for convenience, as half RTT is the theoretical absolute minimum achievable in ideal conditions, i.e. without losses and if the block Tx time is negligible with respect to propagation delay. Moreover, one RTT is the minimum penalty time in case of segment retransmissions, which also facilitates the interpretation of results in terms of retransmission cycles.

Let us start first consider "LTP enhanced" results (dashed line taken from [28]). When PLR=1%, delivery time is about 3.5s (here half RTT=1s). This because, on average, we will have about 5 losses over the first round of 494 segments sent; these 5 losses will almost always require one retransmission cycle, which adds one RTT (2s) to time necessary to deliver the first round. By increasing PLR, there is a greater chance of losses on the first retransmission cycle, which would require a second cycle and would imply an additional RTT. When this happens, delivery time becomes 5s instead of 3s, etc. In brief, delivery time increases rapidly with PLR, because the need of extra retransmission cycles.

Now we can examine ECLSA results, all referring to K=512. The ECLSO aggregation timer is conservatively set to 500ms (a little more than the 0.4s expected filling time for K=512), and the ECLSI closing timer to 100ms. Delivery time is constant at about 3s when decoding failures are negligible, i.e. with PLRs below 9%, 17% and 30% respectively, for the three codes considered (OpenFEC LDPC Staircase, K=512, Rc=8/9, 3/4, 2/3). Comparing these results with those of LTP enhanced, we can see a trade off at PLR<1%, with fast increasing superiority of ECLSA for higher PLRs.

For a thorough evaluation, however, it is necessary to examine the nature of ECLSA delivery time components. Of the 3s, 1s is due to the propagation delay, 0.4s to the Tx time of 500 kB at 10 Mbit/s, the rest, variable, can be ascribed to ECLSA overall processing times (not merely the coding and encoding times, but also aggregation timers and other delays), which are independent of RTT.

If we now consider interplanetary links, propagation delay becomes dominant, and we can drop the processing delay and the Tx time from the previous results, so obtaining the curves in Figure 14 valid for every RTT>>2s. They clearly show the advantage of ECLSA in terms of delivery time for PLR≥1%. By considering that 1 RTT means from 6 to 40 min for Earth to Mars links, the practical relevance of the ECLSA improvement is evident. On the other hand, for fairness, we must bear in mind the price in terms of bandwidth and complexity (memory and processing power). For the former, however, the increase factor can be very modest ($1/R_c$=1.125), at least for PLR<9%, while the latter is destined to decrease with technological advances.

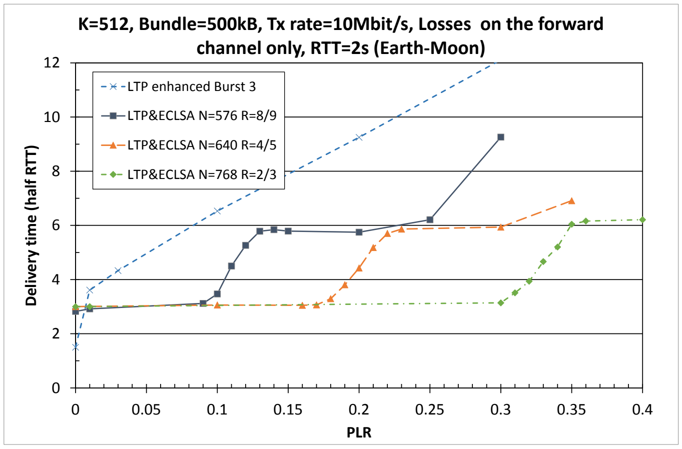


Figure 13: Delivery time vs PLR: comparison between LTP enhanced (burst 3) and ECLSA (LTP+ECLSA). Losses on the direct direction only. RTT=2s.

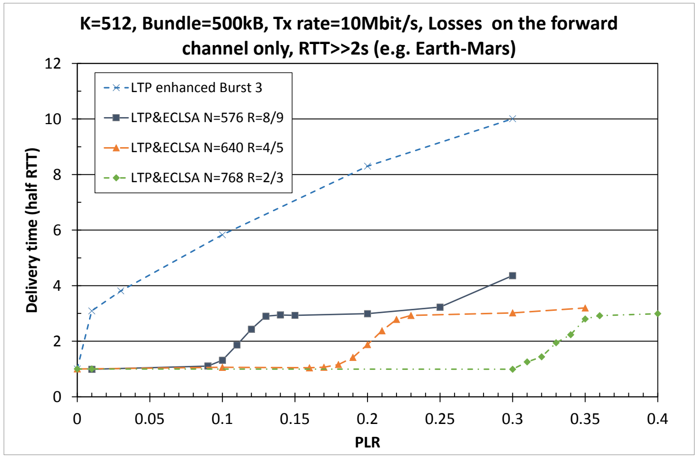


Figure 14: The same as Figure 13, but for RTT>>2, (ECLSA processing time dominated by propagation delay).

VIII. Conclusions

ECLSA was developed in ION as an LTP Link Service Adapter, but unlike other LSAs, it does not consist of a simple interface towards lower protocols, but is a real intermediate layer based on an upper layer FEC encoding. ECLSA protects LTP segments with Packet Layer FEC before encapsulating them into a lower layer protocol, such as UDP. As it is transparent to LTP, it naturally protects all data, retransmitted and signalling segments, the same way, and preserves the ARQ mechanism of LTP, now to be used only in the unlikely event of a decoding failure. ECLSA design shown in detail in the paper is based on multiple threads, to minimize the processing time and to allow reception and decoding of concurrent flows from different sources.

Results show that the performance of LDPC codes provided by LibecDLR (LDPC IRA) and OpenFEC (LDPC Staircase), the two external libraries that can be used by ECLSA, is always very close to that of ideal codes (MDS). When applied to LTP red, ECLSA significantly reduce the delivery time. On short RTTs (2s, less than Earth to Moon RTT), results shows a significant reduction for PLR≥3%; however, on the interplanetary links (RTT>>2s), where the processing time is dominated by propagation time, the advantage of ECLSA becomes outstanding for all PLR≥1%.

Future extensions of this work may include the analysis of the proposed ECLSA implementation over more realistic channels, i.e. introducing bursty packet erasures and in dependence of different datalink layer frame lengths.

Acknowledgements

The authors would like to thank Gianluigi Liva and Balazs Matuz for their support with LibecDLR code, the authors of OpenFEC for their codecs, and Pietrofrancesco Apollonio who authored the first version of ECLSA. A special thanks to Scott Burleigh for his support and for ECLSA inclusion in ION.

References

[1] S. Burleigh *et al.*, "Delay-tolerant networking: An approach to inter-planetary Internet," *IEEE Comm. Mag.*, vol. 41, No. 6, Jun. 2003, pp. 128–136.

[2] V. Cerf *et al.* "Delay-Tolerant Networking Architecture," Internet RFC 4838, Apr. 2007.http://www.rfc-editor.org/rfc/rfc4838.txt Accessed on: June 26, 2018.

[3] F. Warthmann, "Delay-and Disruption-Tolerant Networks (DTNs), A tutorial" v.3.2, Sept.2015, [Online], Available: http://ipnsig.org/wp-content/uploads/2015/09/DTN_Tutorial_v3.2.pdf Accessed on: June 26, 2018.

[4] K. Scott and S. Burleigh, "Bundle Protocol Specification," Internet RFC 5050, Nov. 2007, http://www.rfc-editor.org/rfc/rfc5050.txt Accessed on: June 26, 2018..

[5] 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, 2018.

[6] 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, 2018.

[7] Internet Engineering Task Force DTN Working Group (DTNWG) web site: https://datatracker.ietf.org/group/dtn/documents/ Accessed on: June 26, 2018.

[8] CCSDS DTN Working Group web site: http://cwe.ccsds.org/sis/ Accessed on: June 26, 2018..

[9] 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, 2018

[10] CCSDS 734.2-B-1, "Bundle Protocol Specification," 2015. https://public.ccsds.org/Pubs/734x2b1.pdf Accessed on: June 26, 2018

[11] CCSDS 734.1-B-1, "Licklider Transmission Protocol (LTP) for CCSDS". 2015. https://public.ccsds.org/Pubs/734x1b1.pdf Accessed on: June 26, 2018.

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

[13] 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, 2018.

[14] J. Lacan, V. Roca, J. Peltotalo and S.Peltotalo, "Reed-Solomon Forward Error Correction (FEC) Schemes," Apr. 2009, Internet RFC 5510 http://www.rfc-editor.org/rfc/rfc5510.txt. Accessed on: June 26, 2018.

[15] 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, 2018

[16] 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.

[17] 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.

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

[19] 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.

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

[21] 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 Mag.*, vol. 12, no. 4, pp. 86-94, Dec. 2017.

[22] P. Apollonio, "Erasure Error Correcting Codes Applied to DTN Communications," M.S. thesis, University of Bologna, Italy, Feb.2014,http://amslaurea.unibo.it/6852/4/apollonio_erasure_error_correcting_codes_applied_to_dtn_communications.pdf Accessed on: June 26, 2018

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

[24] C. Caini *et al.*, "Delay- and Disruption-Tolerant Networking (DTN): An Alternative Solution for Future Satellite Networking Applications," *Proc. of the IEEE*, vol. 99, pp. 1980-1997, Nov 2011.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] J. K. Wolf, "An introduction to error correcting codes," 2010.[Online]: http://circuit.ucsd.edu/~yhk/ece154c-spr15/ErrorCorrectionIII.pdf.

[30] INRIA, ISAE, OpenFEC web site: http://openfec.org/. Accessed on: June 26, 2018..

[31] E. Paolini, M. Varrella, M. Chiani, B. Matuz and G. Liva, "Low-Complexity LDPC Codes with Near-Optimum Performance over the BEC," in Proc. ASMS 2008, Bologna, Italy, 2008, pp. 274-282.

[32] M. Cunche and V. Roca, "Optimizing the error recovery capabilities of LDPC-staircase codes featuring a Gaussian elimination decoding scheme," in Proc. SPSC 2008, Rhodes Island, US, 2008, pp. 1-7.

[33] 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.

[34] 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.

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

[36] CCSDS 231.0-B-3, TC Synchronization and Channel Coding, CCSDS Blue Book, Issue 3, September 2017.

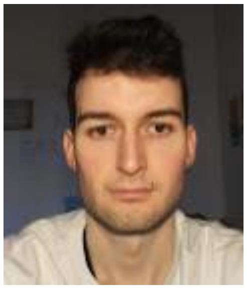
[37] G. Liva, E. Paolini and M. Chiani, "Bounds on the Error Probability of Block Codes over the q-Ary Erasure Channel," in IEEE Transactions on Communications, vol. 61, no. 6, pp. 2156-2165, June 2013.

[38] 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.

Nicola Alessi received the Bachelor Degree in Computer Science Engineering from the University of Bologna, in March 2016, with a thesis based on the analysis of LTP performance. After graduating he implemented ECLSA (Error Correction Link Service Adapter) for ION. In March 2019 he achieved the Master’s Degree at the same university with a thesis on the Hierarchical Inter-Regional Routing Algorithm for Interplanetary Networks, carried out at NASA-JPL under the supervision of Scott Burleigh. His main research interests concern Delay-/Disruption- Tolerant Networking and related protocols.

Carlo Caini received the Master degree in Electrical Engineering “summa cum laude” from the University of Bologna, in 1986, with a thesis on Convolutional Codes and Viterbi decoding. Since 1990 he has been with the Department of Electronics Computer Science and Systems of the same University, where now is working as Associate Professor of Telecommunications. At present, his main research interests are focused on Delay-/Disruption-Tolerant Networking, Transport Protocols for space environments and Satellite Networks. He has supervised the development of many free software components for these environments.

Tomaso de Cola was born in Manosque, France, on April 28, 1977. He received the ``Laurea" degree (with honors) in telecommunication engineering, in 2001, the Qualification degree as Professional Engineer in 2002 and the Ph. D. degree in Electronic and Computer Engineering, Robotics and Telecommunications in 2010 from the University of Genoa, Italy. From 2002 until 2007, he has worked with the Italian Consortium of Telecommunications (CNIT), University of Genoa Research Unit, as scientist researcher. Since 2008, he has been with the German Aerospace Centre (DLR), where he is involved in different European Projects focusing on different aspects of DVB standards, CCSDS protocols and testbed design. He is co-author of more than 50 papers, including international conferences and journals. His main research activity concerns: TCP/IP protocols, satellite networks, transport protocols for wireless links, interplanetary networks as well as delay tolerant networks. Dr. de Cola has served on the technical program committee at several IEEE International Conferences. He is member of the IEEE Communications Society. He is serving as chair of the Satellite and Space Communications (SSC) technical committee within ComSoc.



Marco Raminella received the Degree in Computer Science Engineering from the University of Bologna in March 2017, with a thesis on the insertion of the open source FEC library OpenFEC, developed by INRIA, in ECLSA (Error Correction Link Service Adapter). At present, he is continuing his studies at the same university as a Master student. His main research interests concern DTN and related protocols, Cloud Computing, Big Data and Machine Learning.