

# Study of a Priority Paradigm for Deep-Space Applications \*

Manikantan Ramadas  
mramadas@irg.cs.ohiou.edu

Shawn Ostermann  
ostermann@eecs.ohiou.edu

Hans Kruse  
kruse@ohio.edu

Ohio University  
Athens, OH, USA

## ABSTRACT

Deep-space presents a challenging environment for communication by exhibiting constraints such as very high signal propagation delays, high channel error-rates, meager and expensive bandwidth availability, etc. Further, a Spacecraft participating in a deep-space mission typically carries a host of scientific instruments - each capable of generating data at different rates coupled with different reliability and timeliness needs for communication. In this paper we present a two-dimensional Priority Paradigm designed for applications operating in this scenario. The first dimension is Immediacy, a measure of how urgently data needs to be delivered; Orthogonal to that is Intrinsic Value, a measure of how reliable the data delivery needs to be. We then study the performance of two candidate mechanisms for implementing the Priority Paradigm policy sought with the two-dimensional priority-paradigm parameters requested: Adapting the FEC mechanism in use and Adaptively varying the packet size in use, for various link error rate characteristics.

## 1. INTRODUCTION

Deep-space exhibits various communication challenges such as long signal propagation delays - in the order of minutes and hours, high probability of signal corruption due to channel noise, meager and asymmetric bandwidth availability, short communication time-windows, etc. Spacecrafts communicating in this environment typically carry a host of science instruments performing various science endeavors, and each instrument is capable of generating data at its own calibrated maximum data rate; the net downlink bandwidth available from the spacecraft however, is typically less than the total calibrated maximum data rate of all instruments put together. The constraint of short communication-

\*This paper appears in SpaceOps 2006 - The Ninth International Conference on Space Operations, 19-23 June 2006, Rome, Italy.

**Paper Details: Last Name - Ramadas, Tracking Number 56182, Paper Number AIAA-2006-5663**

windows due to orbital dynamics and antenna scheduling constraints at Earth adds to the bandwidth demand problem even further. Although the presence of on-board storage helps mitigate this problem to a certain extent, under high bandwidth demand situations it could itself become a scarce resource prone to contention.

However, not all science instruments may be generating data of the same value to the mission; some classes of telemetry data could be of lower value compared to more interesting science mission data such as a rare picture of a moon around a planet or a critical status alert from an instrument. Certain applications may need both immediate and reliable delivery of data at the receiver or the data itself may become stale and lose its value; other applications might require reliable delivery of data with more relaxed requirements on timeliness. In this paper we study a priority paradigm approach that could be valuable in such circumstances.

## 2. PRIORITY PARADIGM

Our priority paradigm offers two dimensions (“knobs”) to the application :

- **Immediacy (imm)** A measure of how quickly a unit of application data (called a “job”) needs to be delivered to the receiver.
- **Intrinsic Value (iv)** A measure of how valuable the data is, i.e., a measure that specifies how important the reliable delivery of the data is.

Examples of high immediacy jobs are those application data units carrying critical instrument status messages on the downlink such as a High Temperature Alert, urgent commands sent on the uplink to a planetary rover such as “Stop! don’t go down that crater..” etc. Examples of high intrinsic value jobs are spectrograph results indicating the presence of water on a planet, valuable science observations made by the Huygens probe on its descent into Titan’s atmosphere, etc. A priority continuum of applications choosing various immediacy and intrinsic value requirements is illustrated in Figure 1.

The illustration uses opaque units for intrinsic value and immediacy. In reality these could be obtained via a mapping from more meaningful units for the application. For example, intrinsic value may be specified natively as the

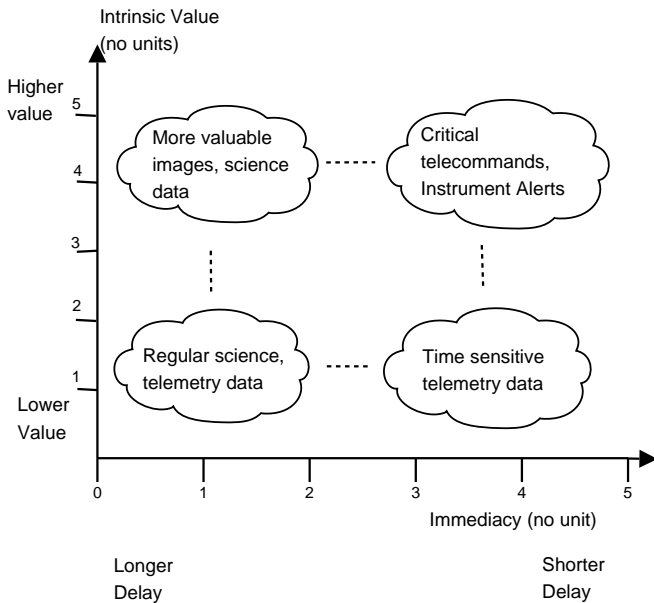


Figure 1: The Priority Continuum

probability of successful delivery sought by the application such as 95%, 99%, 99.99% etc., (Note that in the presence of channel errors, no communication mechanism can make an absolute 100% reliable delivery guarantee within any finite time bound) and immediacy may be specified in real time units such as delivery required within the next hour, within the active communication-window, etc. For the purposes of this paper, we assume that such a mapping is in place and an abstract immediacy and intrinsic value in the range [0-10] (with higher values meaning higher requirements of the parameter) is available for the Priority Paradigm to work with.

### 3. BACKGROUND

We present the following background material briefly to set up the discussion on Priority Paradigm mechanisms and our experimental study on a firmer footing.

#### 3.1 LTP

Licklider Transmission Protocol aka Long-haul Transmission Protocol (LTP) is designed as a reliable datalink protocol operating over deep-space datalinks; it operates over the datalink layer in the protocol stack and offers a reliable datalink abstraction to applications by performing ARQ (Automatic Repeat Request) of corrupted/lost datalink frames using the Selective Repeat mechanism. It is specified as an open Internet standard in three Internet Drafts : Motivation [2], Specification [7], and Extensions [3] under the auspices of the Delay Tolerant Networking Research Group (DTNRG)[1], a research group chartered with the Internet Research Task Force (IRTF). The Internet drafts are in their final stages of evolution and are close to getting published as experimental RFCs.

LTP is designed to operate over single-hop long-haul datalinks; therefore it makes the assumption that the LTP protocol

data units (“segments”) are received in their transmission order unless they are lost due to channel error. LTP sessions are assumed to be uni-directional with data flowing in one direction and acknowledgments flowing in the other; bidirectional data transfer therefore entails opening two separate LTP sessions. It is designed to be as less “chatty” as possible; session parameters are chosen unilaterally to avoid time-expensive connection setup negotiations. Several protocol header elements are variable length encoded as Self-Delimiting Numeric Values (SDNV) [7] to be more conservative in header size while also remaining scalable. For each block (application data unit, same as a “job”) of data transferred, LTP offers a notion of red and green part: when a designated prefix of the block is marked red and the remainder of the block is marked green, LTP transmits the red-part reliably (retransmitting upon loss) and the green-part on best efforts. This feature lets applications send their meta-data (such as application headers) reliably and send the application data on best-efforts, since in typical cases application meta-data is imperative to interpreting the fragments of data received in best-efforts. It is link-state aware by supporting suspension and restart of various timers needed for reliable delivery when the datalink is known to go down based on operating schedules and orbital dynamics.

We used LTP to provide us with a reliable datalink abstraction in our experiments discussed below.

#### 3.2 LT Codes

We needed a flexible and tunable FEC (Forward Error Correction) code for our experiments; Luby Transform (LT) Codes [5] seemed like a good candidate, and we used them as a priority-paradigm mechanism in our study.

LT codes are rateless digital-fountain codes designed to operate over erasure channels. They are rateless - meaning any  $k$  segments of data can be encoded into an arbitrarily many  $K$  encoded segments of data; they are fountain codes in the sense that all the encoded segments are equivalent during decoding: i.e., decoding operation can be done by collecting any  $K$  of the encoded symbols gathered from an infinite (in theory) digital-fountain. LT Codes only make probabilistic guarantees of decoding success: the  $k$  original symbols can be recovered from any of the  $k + O(\sqrt{k} * \ln^2(\frac{k}{\delta}))$  encoded symbols with a  $1 - \delta$  probability [5]. LT-Codes also have simpler encoding and decoding complexities compared to many other FEC schemes, and are easier to implement needing only XOR operations on segments of data (as opposed to the more complex Galois Field arithmetic used by Reed-Solomon codes, for example). More importantly, since they are rateless, they offer us a way to adaptively increase or decrease the FEC applied on a job by simply generating more or less encoded symbols on the channel respectively.

Note that LT-Codes are patent protected.

### 4. PERFORMANCE STUDY

Here, we present our performance study of the Priority-Paradigm mechanisms: Adapting the FEC in use, and Adapting the Packet Size for different emulated channel bit error rates.

#### 4.1 Experimental Setup

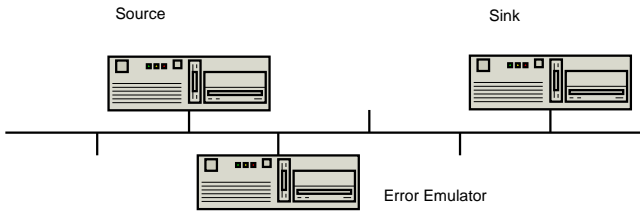


Figure 2: Testbed Setting

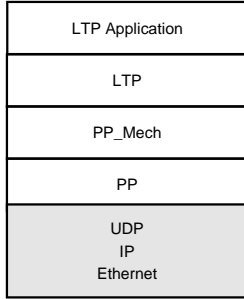


Figure 3: Protocol Stack

Our experiments were performed on Intel Celeron 431 MHz or 565 MHz workstations with 256 MB of Memory running GNU/Linux 2.6.14.3. The traffic control module `tc / netem` was used to emulate various network characteristics such as link delay and bandwidth capacity. We used the `tun` device interface available on the GNU/Linux platform to emulate link error on packets in user-space with our own error emulator module `errorem` written in C. Host specific routes were added on the source and sink machines directing the traffic to their peer to go via the `errorem` machine where the configured bit error rate was emulated. Figure 2 illustrates this experiment setup.

The protocol stack in use during the emulations on the source and sink machines is illustrated in Figure 3.

LTP and the LTP Sender and Receiver applications were implemented in Java 1.5; communication between the LTP daemon and the LTP applications were established via Java RMI (Remote Method Invocation). LTP segments are encapsulated by a `PP_Mech` (Priority Paradigm Mechanism) specific header which together is encapsulated in a generic `PP` header. The generic `PP` header is quite simple: it carries information on the `PP_Mech` type in use, and an integer `jobid` field to uniquely identify the set of received segments that are part of the same job. The supported `PP_Mech` types are: `LT` to identify Luby Transform based FEC encoding, `PSZ` to indicate adaptive packet sizing, `REPEAT` to indicate repeated redundant retransmissions, or `NULL` to indicate a best-efforts transmission carrying no `PP_Mech` header. The `PP_Mech` header carries the `PP_Mech` specific information necessary; for `LT` codes this includes the Number of Input Symbols, Number of Encoded Symbols, Encoded Symbol Sequence #, the Pseudo-random sequence number generator seed for deriving neighbor indices as suggested in [5], and other `LT` parameters. The `PP`- header-encapsulated data

is sent in UDP datagrams on the UDP port reserved with IANA for LTP 1113 with UDP checksums enabled (which is the default behavior of the GNU/Linux UDP stack). We discovered that UDP checksums alone were not sufficient to detect all packet errors; therefore we prepended a 32-bit CRC to the beginning of each transmitted segment ahead of the generic `PP` header. When erasure channel emulation is imperative (as with `LT` codes), the CRC is checked for each incoming segment, and segments failing CRC are silently discarded.

`LT` encoding parameters were chosen as follows. We use the same parameter nomenclature used in the `LT Codes` paper [5] for easy reference. The symbol size was chosen to be 256 bytes, and the maximum input job size  $k$  was chosen to be 100 symbols (application jobs bigger than 25600 bytes were chunked up into 25600-byte sized jobs for encoding) We chose the `c` parameter value used in the  $\tau()$  function part of the Robust Soliton Distribution so that Ripple size  $R$  is at least  $\frac{k}{3}$  in size. This causes the Robust Soliton distribution to bias the degree distribution until degree value 3. The delta parameter specifying the decoding failure probability was chosen to be 0.001. All the `LT Code` parameters cited above were chosen from “good” values found with empirical tests; our goal here is primarily to study the potential of this Coding scheme in our environment. Although we eliminated really poor parameter choices with our empirical study, we do not make any claims for these parameter settings being optimal; performance of `LT Codes` could perhaps be improved even more by further tuning of these parameter values. Finally, we defined the tunable parameter `mult` to control the amount of encoded symbols generated from the set of input symbols such that, the number of encoded symbols  $K$  is given by :  $K = k + mult * (\sqrt{k} * \ln^2(\frac{k}{\delta}))$

We implemented the `LT` encoding process to be independent and stateless with respect to the `LTP` layer above. The `LTP` symbol size of 256 bytes was presented as the `MTU` to `LTP`; thus even if `LT` decoding failed to recover all the encoded symbols, the symbols that were recovered would still be valid `LTP` segments and could be passed up to `LTP`. Further, the overhead of `LT` codes seemed wasteful for trivial single segment jobs created for `LTP` control segments such as Report Segments and Report Acknowledgment Segments, we invoked redundant retransmissions of such control segments instead of `LT` encoding them. If `mult=n` for example, the original transmission of a trivial single segment job is immediately followed by  $n$  redundant retransmissions in the `PP_Mech REPEAT`.

To compare results with a more standard FEC scheme, we studied the performance of  $K=7$  rate  $\frac{1}{2}$  convolutional codes. We ported the public-domain Convolutional code encoder / decoder written by Phil Karn [4] to Java. Convolutional codes require different channel emulation requirements; most importantly, they should not be emulated in an erasure channel. We accommodated this by prepending an `LTP` segment with a 32-bit CRC covering it in a buffer, and then convolutionally encoded this buffer for transmission. At the receiver end, the received buffer is decoded with the Viterbi (Maximum Likelihood) decoder and then the CRC is verified on the decoded buffer to make sure that no undetected errors were left behind. Therefore, while experimenting with

Convolutional codes, the generic PP and PP\_Mech specific layers were by-passed in the protocol stack. Convolutional codes were also tested on blocks of size 256 bytes to give us a fair comparison with LT codes.

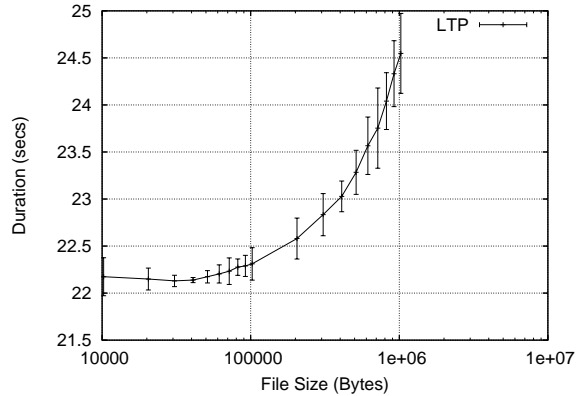
We noticed with empirical study that our errorem module (which operates in user-space via the `tun` virtual interface) could only be driven at a maximum data-rate of approximately 11 Mbps (even though the network interfaces are in a 100 Mbps Ethernet). Giving ourselves a 1:2 engineering slack, we chose to emulate a 5 Mbps channel bandwidth with the `tc` Token Bucket Filter implementation. We emulate a 10 second one-way signal propagation delay; this time was chosen so that the segment transmission delay i.e., the time required to completely transmit all the bits of the segment given the emulated bandwidth, would be well within the signal propagation round-trip time. Thus our results could in effect be independent of the emulated signal propagation delay i.e., if we could somehow present LTP performance results in units of number of round-trips seen instead of absolute time, our results could be directly extended to any signal propagation delay emulation required (4 minutes for Earth-Mars, and 1 hour 15 minutes for Earth-Saturn, for example). Other emulation details (for the benefit of anyone interested in repeating and verifying our study): since the emulation eventually happens in an Ethernet with UDP/IP, we made sure that the arp tables of the source, sink, and errorem machines were populated with static entries before the emulation starts (to prevent 20 second ARP lookups from misleading our analysis); we also made sure that UDP socket buffers were large enough to hold the maximum generated data possible for a single job so that no data was lost due to socket buffer overflow.

The following sections describe the results obtained for the No FEC (regular LTP transmissions with no FEC), LT Fountain Codes run with various rates, and Convolutional codes ( $K=7$ ,  $\text{rate}=\frac{1}{2}$ ) under uniform emulated bit error rates  $10^{-5}$  and  $10^{-4}$ . In all the experiments, file sizes in the range: (10 KB, 20 KB, ... , 100 KB), (200 KB, 300 KB, ... 1 MB) were transferred for 25 trials each. Since priority paradigm processing (except for convolutional coding) is done on a per job basis, the priority paradigm receiver has to wait until all segments of a job are obtained before processing can begin. Since there is no in-band signaling of job completion, the receiver side waits until a segment with a new jobid is obtained (indicating thereby that the earlier job finished) or a Timeout interval of 1 second elapses from the time the last job segment is received. Note this configured Timeout period adds up in our connection lifetime calculations below. In the convolutional code emulation however, there is no notion of a job, and encoding and decoding operations are performed on a segment-by-segment basis.

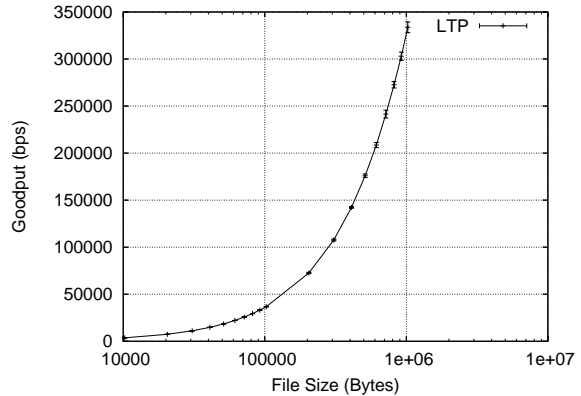
## 4.2 LTP Performance

We first present the performance results for the case with no errors emulated on the channel to serve as a baseline.

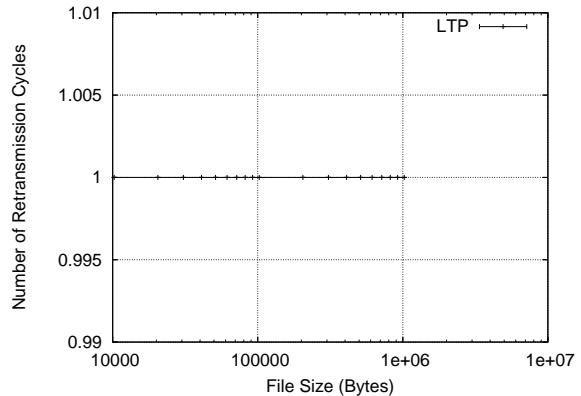
Figure 4 plots Session Duration - defined as the time elapsed at the Sender LTP daemon between the time the LTP sender session starts, and the time the final LTP Report Acknowledgment is sent (right after the reception of the final LTP Report Segment indicating complete block reception); good-



(a) Duration



(b) Goodput



(c) Retransmission Cycles

**Figure 4: LTP Performance: No Channel Errors, 5 Mbps, 10 sec one-way delay**

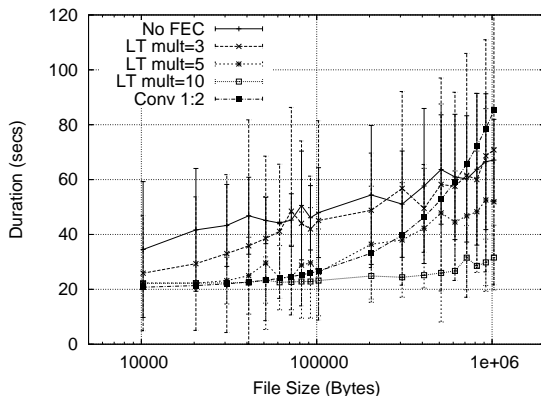
put - defined as the amount of application data transmitted per second; number of retransmission cycles seen. The graphs (like all the other graphs in this paper) depict averages of 25 trials, with the error-bars indicating a 95% confidence interval (assuming Normal distribution). The theoretical minimum session duration for a 1 MB file would be the sum of signal propagation delay (20 sec), segment transmission time 1.64 sec (at 5 Mbps), and our implementation imposed job-timeout of 1 sec, and is therefore 22.64 sec; the theoretical maximum throughput therefore is approximately 362 Kbps. The goodput graph in Figure 4 indicates that LTP application goodput comes close to this on average, at approximately 333 Kbps. Since there were no channel errors, all data sent for each session are successfully received by the receiver from original transmission and a successful Reception Report is sent back right away. Thus all transmissions complete in a single retransmission cycle.

### 4.3 Adapting the FEC

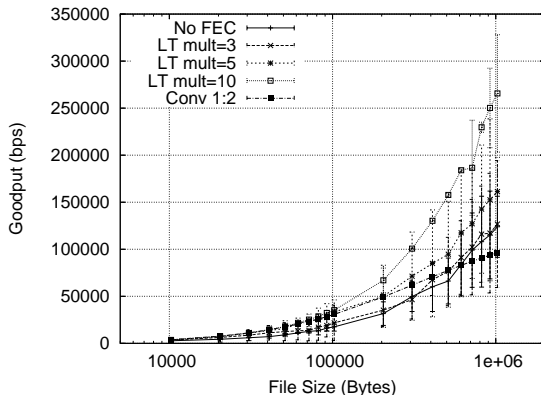
Figure 5 illustrates the performance of the fountain codes operating at different rates (different values of the mult parameter described in Section 4.1), Convolutional codes, and the No-FEC case for channel bit error probability  $10^{-5}$ .

The results illustrated in Figure 5 could be what one might expect. Generating more and more encoded symbols from the fountain-code makes a session more and more robust to the need for retransmission-based recovery - this can be easily observed from the session duration for the mult parameter values 3, 5, and 10 respectively. The No-FEC case where no FEC is applied and pure retransmission-based recovery is employed exhibits the poorest performance in terms of session-duration and goodput. We also notice that while Convolutional codes perform close to the mult=10 case for smaller file-sizes, duration starts climbing linearly with file-size, primarily due to the fact that convolutional codes are more computationally intensive than their fountain code counterparts. The retransmission-cycles graph verifies this fact; note that all of the convolutionally encoded LTP sessions completed in a single retransmission cycle confirming that all the delay in the sessions must be due to the decoding time taken. We would note here that we gave up on many code-optimizations of the original Convolutional code implementation [4] in C when we ported it to Java such as liberal usage of macros and crafty pointer arithmetic as these are not available in Java; a partial cause for the poor decoding performance could thus be attributed to the lack of such optimization techniques in our implementation.

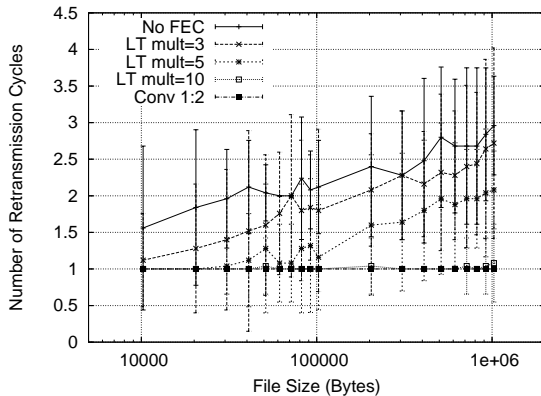
Figure 6 illustrates the encoding cost of various FEC schemes. FEC Overhead is defined as the ratio of net FEC encoded data to all the LTP data (including segment headers and application data) encoded with the FEC scheme. The maximum overhead is exhibited by LT Codes with mult=10, and the minimum overhead (besides the No FEC case) is exhibited by Convolutional codes which have a fixed 1:2 encoding overhead. Thus convolutional codes seem like the best choice for the emulated error-rate if decoding time were not a priority. For extremely high immediacy jobs that are also of larger size, one could probably choose the mult=10 fountain encoding instead, if the job could justify the extra channel bandwidth (twice as that consumed by convolutional codes) as well.



(a) Duration



(b) Goodput



(c) Retransmission Cycles

Figure 5: FEC Performance:  $BER = 10^{-5}$ , 5 Mbps, 10 sec one-way delay

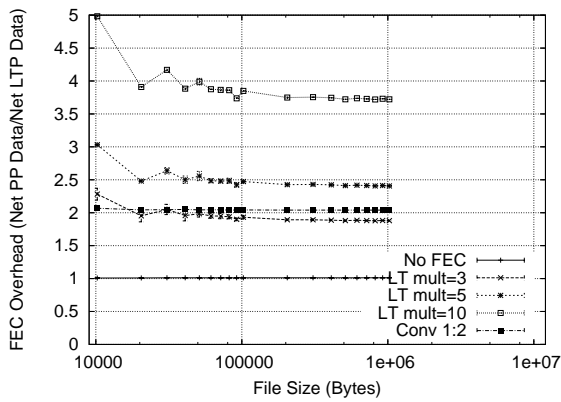


Figure 6: FEC Overhead:  $BER = 10^{-5}$

Figure 4.3 illustrates experimental results with an emulated bit error rate of  $10^{-4}$ . While the general relative performance of various encoding schemes are similar to the  $10^{-5}$  BER case, the mere fact that we have much more errors on the channel makes the results more interesting and the FEC schemes more valuable. For 1 MB file transfers, while the No-FEC case exhibited an average number of retransmission cycles of 3 (approximately) in the  $10^{-5}$  BER case, it now has the value of 5.6. Therefore the goodput performance falls as well. Even the mult=10 case exhibits more variability in performance for larger file-sizes compared to our previous results for BER  $10^{-5}$ . Convolutional codes on the other hand perform much better with little variability in performance except for their costly decoding time.

The FEC Overhead plot for the  $10^{-4}$  BER case is quite similar to the  $10^{-5}$  case shown in Figure 6.

#### 4.4 Adapting the Packet Size

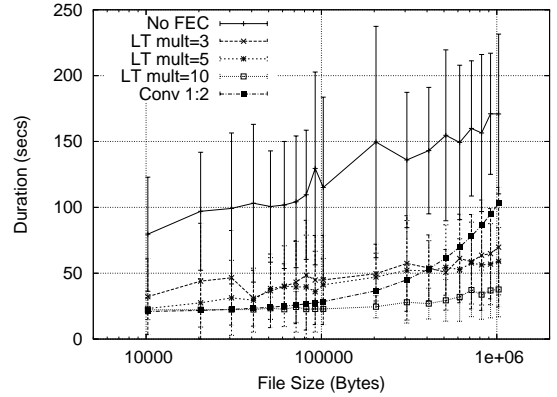
We varied the packet size down from our default value 256 bytes to lesser values for the channel bit error probability  $10^{-5}$ . In a Selective Repeat protocol when the channel bit error rate is known, optimal packet size, i.e., optimal for maximum throughput is given by [6] to be:

$$k_{opt} = \frac{-h \ln(1-p) - \sqrt{(-4h \ln(1-p) + h^2 \ln(1-p^2))}}{2 \ln(1-p)}$$

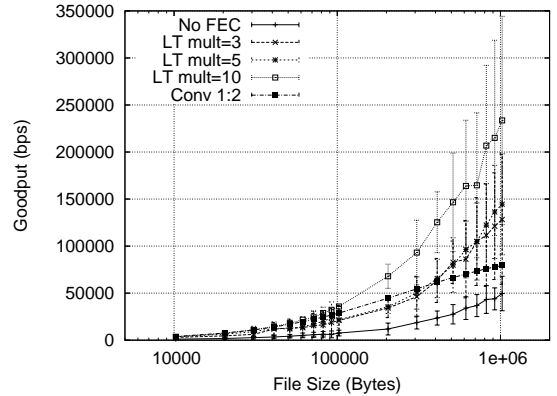
where  $k_{opt}$  is the optimal packet size,  $p$  is the channel bit error rate, and  $h$  is the number of overhead bits per packet. For the bit error rate of  $10^{-5}$  and header size of 20 bytes (approximate LTP header overhead), we find the optimal packet size to be 175 bytes.

The optimal packet size mentioned above is for optimizing throughput; since we are interested in optimizing other connection metrics besides throughput, we studied performance for different packet sizes: 256, 175, and 128 bytes. This packet size was given as the LTP MTU as well, and tests were performed with no FEC applied. Figure 8 illustrates our results.

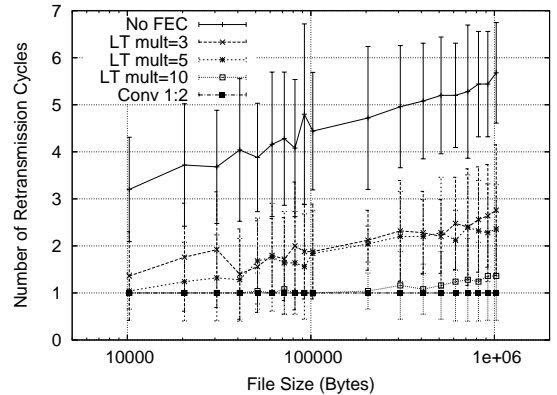
We notice that session duration and goodput performance improve although by modest amounts, as we reduce the packet size. This could be attributed to the fact that when bit-errors occur, they take fewer segments down with them as the packet sizes become smaller, thereby needing fewer



(a) Duration

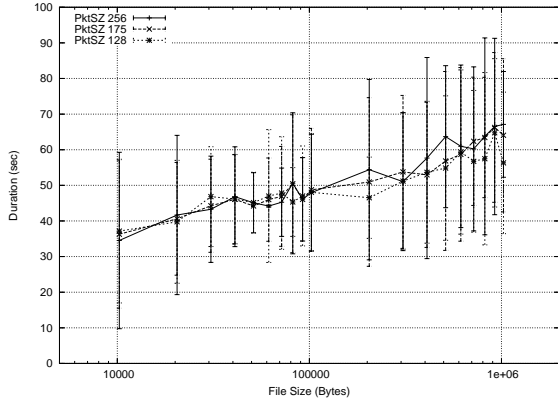


(b) Goodput

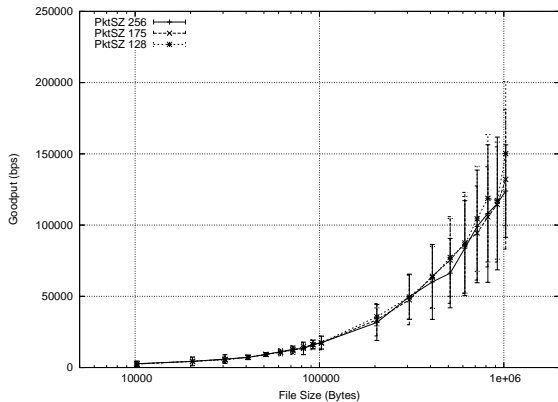


(c) Retransmission Cycles

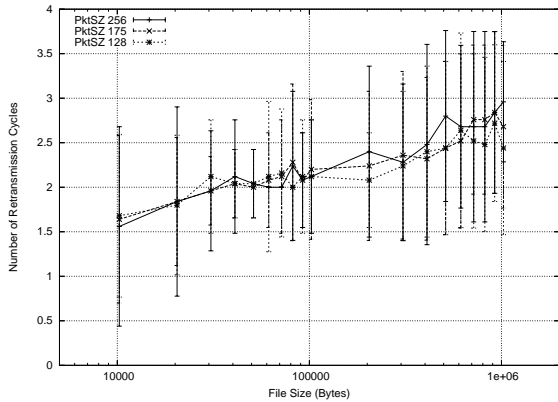
Figure 7: FEC Performance :  $BER = 10^{-4}$ , 5 Mbps, 10 sec one-way delay



(a) Duration

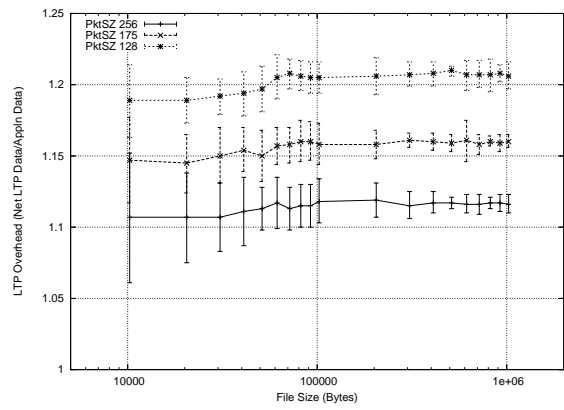


(b) Goodput



(c) Retransmission Cycles

**Figure 8: Adapting Packet Size:  $BER = 10^{-5}$ , 5 Mbps, 10 sec one-way delay**



**Figure 9: Adapting Packet Size: LTP Overhead**

bytes to recover by retransmissions. This fact is corroborated by the retransmission cycles graph in Figure 8 that shows that the average number of retransmission cycles required drops down to approximately 2.4 for packet size 128, instead of 2.9 for packet size 256 bytes (for 1 MB file transfer). This could be a valuable improvement on deep-space links such as those to Saturn, where a retransmission-cycle is typically  $2\frac{1}{2}$  hours.

We also note that adaptive packet sizing could probably be more valuable for a channel with higher error-rates, such as  $10^{-3}$  or  $10^{-4}$ . We have not yet thoroughly explored those cases.

The LTP Overhead graph in Figure 9 shows the cost for this mechanism in terms of total LTP data (LTP headers + LTP application data) to LTP application data. For smaller packet sizes of course, we end up spending a larger proportion of bandwidth for meta-data. Further, smaller MTU sizes also means that LTP reception reports would get sliced into multiple LTP Report Segments (selective acknowledgment segments), and also add to the overhead.

## 5. CONCLUSIONS

We have presented a priority-paradigm for use by applications operating in the deep-space communication environment. The paradigm provides two dimensions of priority: Intrinsic Value - a measure of how valuable the data unit is from a semantic perspective, and Immediacy - a measure of how valuable the timeliness of delivery of the data unit is. We have then explored a few candidate mechanisms for implementing the priority-paradigm policy parameters Intrinsic Value and Immediacy specified by the applications. The mechanisms include classes of FEC, and adaptive packet sizing, which were studied for various uniform channel bit error rate conditions.

We observe from our study that there probably is no “one-size-fits-all” mapping from Priority Paradigm policy to mechanism choices. This mapping would instead be a function of channel error-rate characteristics, the size of the job being transferred, etc. However, for the channel error-rate characteristics we have studied here, the following could be a reasonable mapping (for larger file-sizes):

- Jobs with higher intrinsic value, but lower immediacy could be convolutionally encoded or sent with LT Codes mult parameter 10 (if the extra bandwidth usage consumption could be justified for quicker decoding).
- Jobs with medium intrinsic values and lower immediacy could be fountain encoded with lower LT Code mult parameter values, such as 3.
- Bulk telemetry jobs that happen to have low intrinsic value and immediacy requirement could be transmitted utilizing the red/green part notion of LTP sending their meta-data as red-part and their data as green-part with no FEC invoked.

## Future Work

A natural follow-up to our work here would be to study performance under burst-error conditions. It would be important to also test Reed-Solomon codes under these conditions. Under certain burst-error conditions such as those that are infrequent but destroy a sequence of segments when they hit, the LT Fountain Codes could give us the best performance. However, it is only a conjecture at this point, and we intend to continue this work for burst error conditions, as well as with higher uniform bit-error-rate conditions to explore this.

## 6. ACKNOWLEDGMENTS

We thank Peter Bentley of DigitalFountain for providing us with useful document pointers on the LT and Raptor DigitalFountain FEC codes for our study.

## 7. REFERENCES

- [1] Delay Tolerant Networking Research Group.  
<http://www.dtnrg.org>.
- [2] S. Burleigh, M. Ramadas, and S. Farrell. Licklider Transmission Protocol - Motivation, March 2006. Internet Draft (work in progress)  
draft-irtf-dtnrg-ltp-motivation-02.txt.
- [3] S. Farrell, M. Ramadas, and S. Burleigh. Licklider Transmission Protocol - Extensions, March 2006. Internet Draft (work in progress)  
draft-irtf-dtnrg-ltp-extensions-03.txt.
- [4] Phil Karn. Convolutional Code - Viterbi Decoder Implementation in C.  
<http://www.ka9q.net/code/fec/viterbi-3.0.2.tar.gz>.
- [5] Michael Luby. LT Codes. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 271–282, 2002.
- [6] Eytan Modiano. An adaptive algorithm for optimizing the packet size used in wireless arq protocols. In *Wireless Networks 5*, pages 279–286, 1999.
- [7] M. Ramadas, S. Burleigh, and S. Farrell. Licklider Transmission Protocol - Specification, March 2006. Internet Draft (work in progress)  
draft-irtf-dtnrg-ltp-04.txt.