Received 2 August 2023, accepted 8 September 2023, date of publication 14 September 2023,
date of current version 20 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3315659

RESEARCH ARTICLE

Application of Fountain Code to High-Rate Delay Tolerant Networks

NOAH P. DOUGLASS[1], JOHN LANGEL[1], WEILAND J. MOORE[1], LAWRENCE NG[1], RACHEL M. DUDUKOVICH[2], (Member, IEEE), AND SANCHITA MAL-SARKAR[1]
[1]Department of Electrical Engineering and Computer Science, Washkewicz College of Engineering, Cleveland State University, Cleveland, OH 44115, USA
[2]Glenn Research Center, Cleveland, OH 44115, USA

Corresponding author: Sanchita Mal-Sarkar (s.malsarkar@csuohio.edu)

ABSTRACT Space communication poses several unique challenges that are not always present in typical terrestrial communications. Currently, communication with satellites is based on point-to-point links, and development of an interplanetary internet is an active research area. Delay Tolerant Networking (DTN) has been proposed as a way to mitigate the long delays and disruptions found in deep space. A specialized version of DTN, called High-rate Delay Tolerant Networking (HDTN), has been developed by NASA to support a variety of missions requiring store-and-forward capability. However, there are still several features that are desired for HDTN including data fragmentation, multicast, and anycast. This project proposes the application of fountain code in HDTN as a means of fragmenting, distributing, and reassembling data (in the form of bundles) across multiple nodes (i.e. satellites) to any number of receivers (i.e. ground stations). Fountain code is shown to be a promising encoding method for use with the HDTN protocol suite due to its short runtimes, small encoded file sizes, and loss tolerance.

INDEX TERMS Delay tolerant networks, fountain code, data fragmentation, multicast, opportunistic routing.

I. INTRODUCTION

Space communication poses several unique challenges that do not typically present themselves in standard terrestrial networks. These challenges include disruptions due to weather or other extraneous sources, as well as extended transfer times of large data objects, such as high resolution image files stemming from long distances between sender and receiver. NASA, as well as multiple other government agencies and industry, has been developing delay tolerant networking (DTN) to provide an architecture and set of protocols to form the future interplanetary internet [1]. A specialized version of DTN, called High-Rate Delay Tolerant Networking, has been developed by NASA Glenn Research Center to provide increased functionality to DTN [2]. There are several capabilities that are currently being investigated for HDTN, including high-speed data compression, data fragmentation, streaming, and DTN multicast. This paper investigates fountain code as an efficient method of uplink and downlink data transfer available for use with HDTN, particularly in a distributed manner among multiple receivers.

The use-case for this work focuses on a Low-Earth orbit satellite that may perform a variety of high resolution imaging or sensing tasks, such as those needed for Earth science, weather, and climate-related research. Due to the size of the images and constraints of the satellite communciation system, the downlink may require multiple orbital passes around the earth or may be distributed among multiple ground stations in different locations. An example of this type of experiment is given in [3]. The authors used an early reference implementation of DTN, called DTN-2. They noted several drawbacks to the early DTN prototype including lack of

The associate editor coordinating the review of this manuscript and approving it for publication was Bijoy Chand Chatterjee.

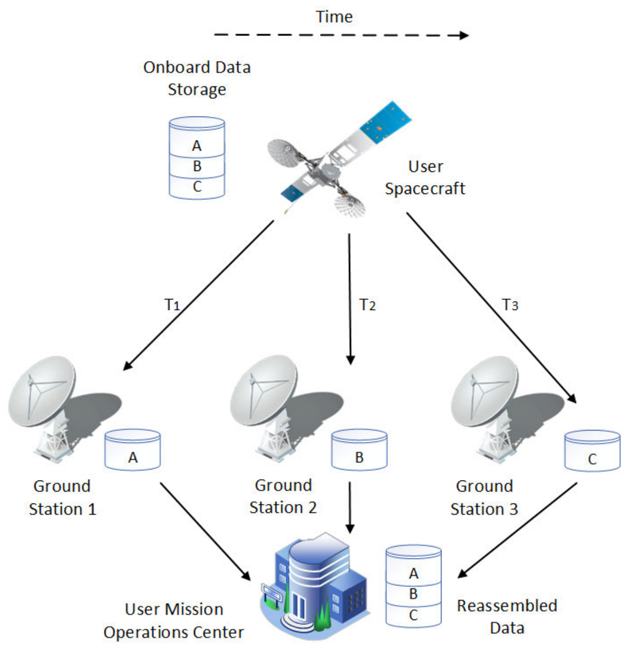


FIGURE 1. Example use-case of bundle fragmentation and reassembly.

reliability, error detection, and checksums. In addition, they commented on the complexity of the protocols, lack of standardization, and issues related to time synchronization. In our work, the use of fountain code inherently incorporates redundancy and reliability into file transfers and simplifies the bundle reassembly process. Fig. 1 shows a diagram of this process. The satellite will have access to ground station 1, 2, and 3 at different points in time. At each time step, a different fragment of the data will be received at the ground station, which is then transferred via the terrestrial network to the mission operations center.

In order to overcome the challenges presented in space communications – long round-trip times, disruptions to connectivity, and asymmetric link data rates – this work proposes the potential implementation of fountain code in High-rate Delay Tolerant Networking as a way to effectively distribute bundles across multiple receiving nodes. Application of fountain code in an HDTN environment would serve to mitigate the issues mentioned above, provided that the receiving nodes in the network are able to reliably reassemble the bundles they receive into one cohesive message.

The main objectives of this project are to investigate the manner in which fountain code can be used to fragment, distribute, and reassemble data bundles back into their original data format, develop a prototype implementation of the proposed fountain code design based on the open-source HDTN software provided, test the prototype software on various configurations of single-board computers, and provide software documentation detailing proper installation and operation of the software.

The rest of this paper is organized as follows. Section I introduces the overall content and contributions. Section II gives an overview of the HDTN implementation, and discusses the major design concepts for this work. Section III covers related work in fountain codes for DTN and the relation to opportunistic routing and multi-destination approaches. Section IV discusses the design of the fountain code encoder and decoder prototypes, as well as integration with HDTN. Section V discusses the software verification approach and performance analysis. Section VI concludes the main portion of the paper with recommended future work. Finally, the Appendix demonstrates the basic prototype usage for the encoder and decoder modules.

II. BACKGROUND

This section introduces the concepts from delay tolerant networking that have served as the basis of our work.

A. HDTN OVERVIEW

The High-rate Delay Tolerant Networking project [4] has been developing a performance-optimized DTN implementation focused on supporting high-rate communications systems (greater than 1 Gbps), such as optical terminals and relays. The HDTN software is publicly available at [5]. The software provides multiple convergence layers, applications, store-and-forward capabilites, and routing. Currently, the HDTN software supports existing Delay Tolerant Network (DTN) protocols like Licklider Transmission Protocol (LTP) [6] and Bundle Protocol (version 6 [7] or version 7 [8]). Future planned capabilities include DTN multicast, neighbor discovery, distributed storage, streaming, and opportunistic routing. The work in this article studies fountain code as a possible technique to distribute packets in a reliable manner, which relates to many of the new planned features. Figure 2 shows the high-level architecture of the HDTN bundle agent.

B. ENUMERATING THE DESIGN CONCEPTS

DTN, or delay-tolerant networking, is an architecture and set of protocols designed for networks in which an end-to-end path through the network does not always exist and delays between nodes make traditional terrestrial protocols infeasible. Environments like deep space introduce delays of lengths that are normally intolerable, hence the need for "delay-tolerant" space communications. DTN encompasses a set of protocols with which it can employ automatic retransmission, interoperability, efficiency, security, and other vital operations with respect to its data units, called bundles. DTN's resilience in suboptimal conditions make it an ideal candidate for the basis of networking in a space environment. HDTN seeks to build upon DTN in the effort of increasing its transfer speeds and delay tolerance.

Fountain code is a technique by which a data transmission can be decoded using a potentially unordered subset of the original transmission. When a message is transmitted, the fountain code splits it into blocks and applies the Exclusive-Or operation (XOR) on blocks at random [9].

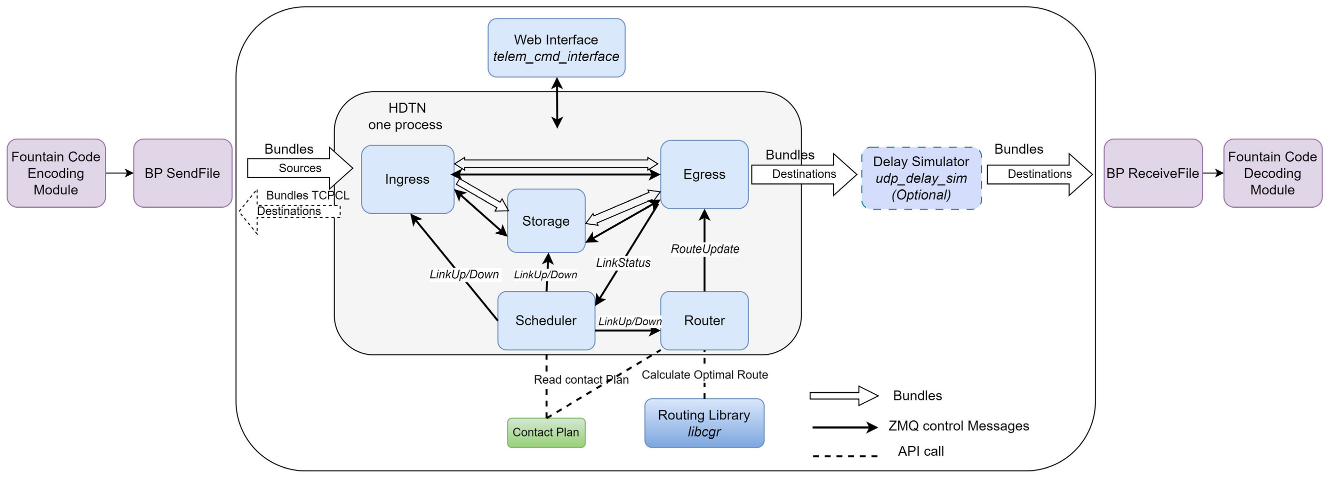


FIGURE 2. High level HDTN architecture.

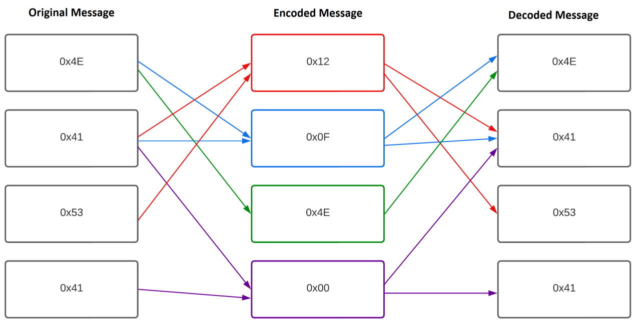


FIGURE 3. An example of message encoding and decoding using fountain code.

The transmitter sends the encoded message along with information about how the message was encoded. The receiving fountain code decodes the message by applying the XOR operation once more according to the encoding information. This process makes it possible for a large bundle of data to be fragmented, transmitted, and reconstructed. This technique is highly scalable in comparison to the traditional TCP suite and is thus ideal for use in the environment of downlink communication with large files. Figure 3 shows the basic fountain code concept.

Fountain code is a promising way to distribute data and improve reliability for DTN's since it does not require a feedback packet for acknowledgement. The original file can be reassembled from any subset of encoded data from a set equal or slightly larger than the original data [10]. Many DTN protocols such as LTP [6] have been developed to attempt to reduce the number of acknowledgement packets required to reliably transmit data since long round-trip times greatly impact performance.

Anycast is a method by which a network can handle addressing and routing between transmitting and receiving nodes. An incoming transmission is routed to the nearest receiver node that is capable of handling the transmission with appropriate buffer capacity and general efficiency. In this case, "nearest" refers to physical distance, thus motivating the network router to select a destination node that is nearest to the original transmitter node.

Multicast is another network addressing and routing method. An incoming transmission may have multiple destinations, and the network routes it to only these destination nodes. It is often optimal in comparison to broadcast, which routes traffic to all receiver nodes, since a node may be transmitting sensitive data that it does not want just any node to receive.

III. RELATED WORK

Table 1 summarizes a literature review of fountain code in DTN, replication-based techniques and opportunistic routing in DTN. This section will further discuss the related papers on fountain code for DTN, as well as opportunistic routing.

A. FOUNTAIN CODES IN DTN

Packet replication is often used as a way to decrease delivery delay and improve the probability of delivery. Epidemic routing and flooding are the two most common naive approaches. Flooding based approaches replicate all packets, whereas epidemic routing replicates packets for any node that has not already seen the packet [11]. Forward error correction and fountain codes are proposed as a way to store partial frames of data among multiple nodes in [12]. The work is based on a simulation of both fixed erasure codes (Reed Solomon type codes) and rateless fountain codes. The authors note that typically in fountain codes, the coding is done at the source. There would need to be additional work to develop an approach that handles coding and storage at a relay node.

Fountain codes are applied to file transfer for vehicular delay tolerant networks in [10]. The authors propose the use of fountain coding in the application layer along with UDP

TABLE 1. Literature review summary.

Author	Topic	Objective	Reference
Langari et al. (2013)	Vehicle DTN and fountain code	Developed a new routing algorithm, AODV-DTN, in the network layer based on AODV and Store-Carry and Forward policy. They compared their architecture with FOUNTAIN and classic FTP scenarios in terms of file delivery ratio and byte throughput.	[10]
Spyropoulos et al. (2008)	Fountain code	Introduced a routing scheme and showed that spray routing performs significantly fewer transmissions per message. Their scheme has lower average delivery delays than existing schemes and highly scalable and retains good performance under a large range of scenarios.	[11]
Altman and Pellegrini (2009)	Fountain code	Studied a class of replication mechanisms to decrease delivery delay and quantified tradeoffs between resources and performance measures (energy and delay).	[12]
Das et al. (2003)	Ad hoc on-demand Distance vector routing (AODV)	Used as the basis for the routing method in [10]. Enables dynamic, self-starting, multi-hop routing.	[13]
Tournoux et al. (2010)	DTN and fountain code	Investigated the streaming-like applications over DTN. They identified how DTN characteristics impact on the overall performances of these applications. Also presented a transport layer mechanism (Tetrys) which enables robust streaming over DTN.	[14]
Cao and Sun (2013)	DTN	Reviewed the existing multicasting and anycasting issues in DTN and surveyed several routing algorithms in DTN.	[15]
Ding et al. (2018)	Opportunistic routing	Proposed a pre-decoding recovery mechanism (PDRM) that removes residual copies after the destination receives the original packet by generating re-decoding element (acknowledgement) and maintaining the immune-lists.	[16]
Walter and Feldman (2016)	Opportunistic routing	Proposed a mechanism for LEO satellites, with low hardware resources, to dynamically discover ground stations and predict future contacts to them.	[17]
Kirkpatrick et al. (2023)	Cluster-based routing and messaging	Proposed cooperative and non-cooperative clustering techniques for network partitioning and controller placement.	[18]

as the transport layer and develop a DTN routing algorithm based on Ad hoc On-Demand Distance Vector (AODV) [13]. This approach allows packets to be received in any order, does not require acknowledgements, and allows the application layer to provide a large amount of storage buffer to prevent packets from being dropped.

Forward error correction, uncoded data, and the Tetrys transport protocol are compared in [14]. The authors developed Tetrys, which is a transport layer that enables robust streaming over DTN. Tetrys is based on an ‘‘on the fly’’ coding mechanism that is able to ensure reliability without retransmission and fast in-order bundle delivery.

B. OPPORTUNISTIC ROUTING

The type of replication mechanisms that fountain code seeks to improve upon are frequently used in opportunistic routing algorithms. In this type of routing, a deterministic contact plan may not be known, so methods to detect available nodes and replication of packets to multiple destinations are often used. An overview of DTN routing is given in [15]. It compares approaches for deterministic versus stochastic cases for unicast routing, multicast and anycast methods, as well as the differences between mobile ad hoc networks and DTNs. Network Coding Opportunistic Routing (NCOR) is proposed in [16] as a way to improve upon packet replication techniques often used in opportunistic routing for DTN’s where node storage and transmission capabilities are limited. Cluster-based routing is proposed in [18] and [19]. In particular, [18] uses multicast techniques for cluster messaging. Trusted routing schemes are used in both [19] and [20]. Ring Road Routing is discussed in [17]. While it is a deterministic style of routing, the approach uses discovery of nodes via beacons to detect multiple ground stations and make contact predictions.

IV. DETAILED DESIGN

Our fountain code algorithm has been implemented in the style of a Luby transform (LT) code [21]. LT codes are a fairly simple form of error correcting algorithm developed by computer scientist Michael Luby. This style of fountain code was chosen because it utilizes the exclusive-or (XOR) operation, which is fairly straight-forward and simple to implement. The software implementation consists mainly of encoder and decoder modules written in Python. Data bundles generated by a transmitter using HDTN are fed into the encoder, and the encoded data is transmitted to the receiver(s). Each receiver accepts the encoded data and passes it through the decoder module to recover the original data.

A. ENCODER

The encoding process begins by reading a file as binary data. This allows the data to be separated cleanly in multiples of bytes and manipulated with the XOR operation later. The exact size of each segment of the separated data is determined by a parameter called `BUNDLE_BYTES`, which specifies the number of bytes per bundle to be created. It is likely that the data does not contain a number of bytes that is an exact multiple of `BUNDLE_BYTES`, so the final bundle will need to be padded with zeros, if necessary. Zero padding is used because the result of XORing any number with 0 is the first number. Encoding is then initiated with the original data, its number of segments, and the desired encoded data size, defined as the original data segment number multiplied by a `REDUNDANCY` constant parameter.

One of the main goals of LT code is to introduce redundancy in a more intelligent way than simply sending multiple copies of the same data. The creator of LT code, Michael Luby, designed a probability distribution for this purpose. The ideal soliton probability distribution is used to determine exactly how likely it is for a given number of data segments to be XORed in the creation of one encoded data bundle. It is the optimal solution to the delicate balance of maximizing redundancy while still guaranteeing that the encoded data can be decoded. The ideal distribution function is defined as

$$ideal\_soliton(1) = \frac{1}{K} \tag{1}$$

and

$$ideal\_soliton(i) = \frac{1}{i(i-1)}, \tag{2}$$

where $2 \leq i \leq K$ and K is the maximum number of XOR neighbors possible [21]. For the purposes of this encoder, K is defined as the number of data segments created from the original data.

After obtaining the ideal soliton distribution, the encoding process begins. Algorithm 1 shows the encoding algorithm pseudocode.

Randomly choosing a number of XOR neighbors, even from the ideal soliton probability distribution, may lead to an unsolvable encoding. To help ensure that this does not happen, the encoder must create at least one encoded bundle that is created from a single original data segment. This makes the distribution of XOR neighbors less than ideal, but having a solvable encoding is more important.

The created encoded data is fairly large, so it is then compressed using the GZIP utility [22], creating a zipped file. After the encoding algorithm is complete, the resulting set of encoded data is ready to be sent to the receiver via HDTN.

Algorithm 1 Encoder

```
1: Obtain an ideal soliton distribution of size original_data_bundle_count
2: Define an empty list to which encoded data will be added

3: for $i = 0$ to original_data_bundle_count do
4:   if i = 0 then
5:     xor_neighbors $\leq 1$
6:   else
7:     xor_neighbors $\leq$ a random number of neighbors according to ideal soliton
8:   end if
9:   Take xor_neighbors number of bundles to be used as components
10:  Set the current encoded bundle value to the first component value
11:  for $j = 1$ to xor_neighbors do
12:    XOR the current encoded bundle value with the value of component $j$
13:  end for
14:  Add the final encoded bundle value and list components to the encoded data list
15: end for
```

B. DECODER

Before the decoder does anything, it decompresses the encoded file. Then, similarly to the encoding process, the decoding process begins with reading the encoded data as a set of binary data in segments. This is done by finding text data between curly braces in the encoded data and transforming it into dictionaries. The decoder also initializes a list to which the decoded data will be appended as it is decoded. Rather than being empty, this list is initialized as an array of -1's. This gives the decoder a way to distinguish between solved and unsolved portions of the decoded data, as the XOR operation will never output a negative number in this configuration. Algorithm 2 shows the decoding algorithm pseudocode.

The decoder iterates over the encoded data, making a list of occurrences for each component. For example, in the list of component occurrences, index 0 contains a list of indices representing encoded bundles that used component 0 in the encoding process. Then, the decoder begins checking for bundles that are ready to be solved. Such a bundle would be recognizable by having only one unsolved component left in its list of components. Upon finding a solvable bundle, the decoder checks to see if the remaining component is already solved. If not, it sets the value of that component to be the same as the value of the encoded bundle. Then, it checks the list of bundles in which the solved component appears. For each of these other bundles, it XORs the component value with the other bundle value and removes the component from the component list of that bundle. This process repeats until no bundles have been solved during the current decoding

Algorithm 2 Decoder

```
1:  for all bundle in encoded_data do
2:      for all component in bundle do
3:          Record that component appears in bundle in list
            component_occurrences
4:      end for
5:  end for
6:  while more than one component has been solved during
    the current iteration do
7:      for all bundle in encoded_data do
8:          if there is only one component left unsolved in
            bundle then
9:              Remove the encoded bundle from the list of
                encoded data
10:             if the component has not been solved through
                another bundle then
11:                 Set the decoded data entry at the same
                    component index to the encoded bundle value
12:                 for all other_bundle_containing_component
                    in encoded_data do
13:                     Set the value of other_bundle to the XOR of
                        itself and the value of the component
14:                     Remove the newly solved component from
                        the list of components for this bundle
15:                 end for
16:             end if
17:         end if
18:     end for
19: end while
```

iteration, indicating that no more decoding is possible. The resulting output is identical to the original data.

C. HDTN INTEGRATION

In the current HDTN framework, there are two methods by which data can be acquired to pass through the network protocol. By default, HDTN creates its own data with the BPGen module, generating bundles to be passed through the HDTN protocol. HDTN also contains the functionality to pass an input file through the protocol.

If using the input file functionality in HDTN, the integration of fountain code with HDTN is rather simple. An input file can be passed to the encoder, which outputs the encoded data file. The encoded data can then be transmitted through HDTN. Once it is received, it can be passed to the decoder, which outputs the original data. In essence, the functionality of HDTN is completely contained between the encoder and decoder scripts.

If utilizing the BPGen module within HDTN to create bundles, the optimal solution is to use the encoder and decoder as modules within the existing HDTN framework. For the encoder, bundles would be redirected from the current BPGen encoding process to a buffer. This buffer would then be passed to the proposed encoder script to be processed.

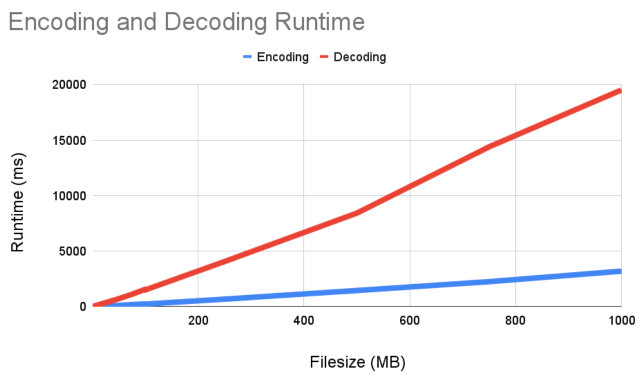


FIGURE 4. Encoding and decoding runtime.

Once all of the bundles are processed, they would be placed back into the buffer and returned to the Ingress module to be used within the rest of the HDTN software. A similar process would be used to integrate the proposed decoder script with the pre-existing BPSink and Egress modules. Due to time constraints, this integration of the encoder and decoder scripts with the HDTN framework has been left as future work.

V. SIMULATION AND RESULTS

Verification of the fountain code software was conducted by examining the time elapsed in encoding and decoding data, as well as the impacts of the bytes and redundancy parameters on file sizes and loss tolerance.

A. ENCODED DATA SIZE AND RUNTIME

The encoder and decoder modules calculate the runtime of the encoding and decoding processes on every execution. A study of this property can be completed by testing the encoder and decoder scripts with input files of various sizes. On Ubuntu 20.04.4 LTS, the "yes" command was used to create various dummy files for testing purposes. For example, a 15 MB dummy file was created using the command "yes this is a 15mb file | head -c 15000000 > 15mb". The result is a file called "15mb" consisting of the repeated phrase "this is a 15mb file" that is 15 MB in size. Twenty-six such files were created, with sizes ranging from 1 MB to 1100 MB. Each file was put through the encoder and decoder ten times, with both modules set to the settings of 64-bit data type, no transmission loss, 65536-byte bundle size, and a redundancy coefficient of 2.0. The results are visualized in Figure 4, comparing input file size with encoding and decoding runtimes.

The nature of the relationship between runtime and file size appears to be linear in nature, with the decoder runtime increasing much more rapidly than the encoder runtime. Decoding takes significantly longer than encoding across all tested file sizes, which is the expected behavior. Future work relating to runtime will focus on improving the efficiency of the encoder and decoder modules. In addition, runtime of each module increases as bundle size decreases, so future work may also include finding intelligent methods

TABLE 2. Reliability approach and results comparison.

Approach	Purpose	Metric	Disruption	Result
LTP Aggregation [23]	Cope with channel rate asymmetry	Goodput (Bytes/sec)	BER=0, 10^{-6}, 10^{-5}	Goodput increases with increasing number of bundles aggregated per block
Global Selective ACKnowledgement [24]	Provide global information about the receipt of packets at any destination, supports unicast and multicast	Round Trip Delay and Success Probability	Multi-hop scenario with 100 relays	Achieves up to 0.75 success probability
LTP [25]	Compare LTP, UDP, and TCP Convergence Layers	Goodput (Bytes/sec)	Delays from 1000 ms - 5500 ms	LTP is recommended for delays longer than 4000 ms with BER of 10^{-6} and greater
Fountain Code (described here)	Encode packets such that a subset can be lost without affecting final data	Loss Tolerance , Encoded Size	Loss	Encoded file provides data compression and loss tolerance

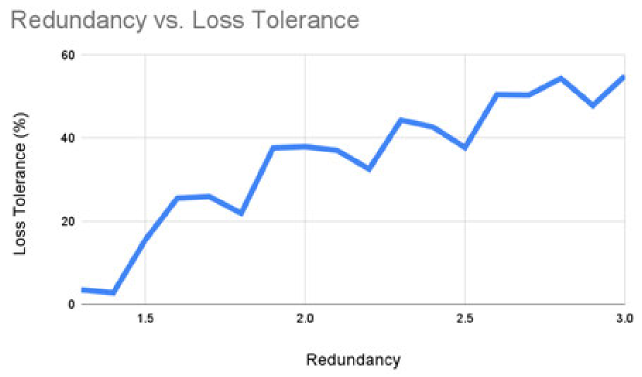


FIGURE 5. **Redundancy versus loss tolerance.**

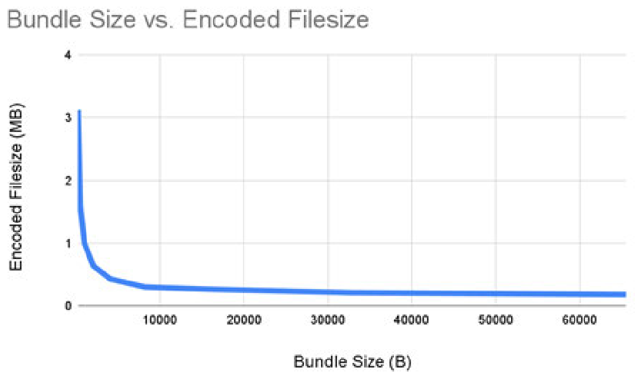


FIGURE 7. **Bundle size versus encoded filesize.**

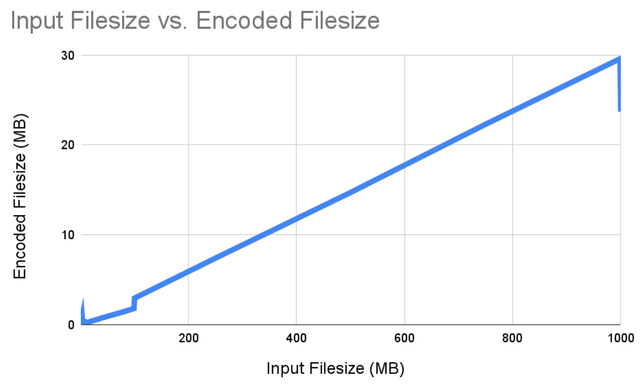


FIGURE 6. **Input filesize versus encoded filesize.**

of determining appropriate bundle sizes based on input file size.

B. REDUNDANCY AND LOSS TOLERANCE

The redundancy parameter for the encoder module allows the user to control the size of the encoded data file. When the encoded data is created, its size is determined by the size of the original data multiplied by the redundancy scalar. Ideally, increased redundancy should result in higher loss tolerance for the encoded data. The loss tolerance of the encoded data can be tested using the transmission loss percentage parameter, which simulates data loss after creating the data. This property was examined by passing a 10 MB dummy file through the encoder and decoder modules with redundancy scalars ranging from 1.3 to 3.0 and examining how much of the resulting encoded data could be lost before it was rendered unsalvageable. Figure 5 shows the relationship between redundancy and loss tolerance.

As expected, the loss tolerance of the encoded data increases as the redundancy increases. Though the general positive correlation is clear, the data shows that certain redundancy values result in lower loss tolerance. Future work in this area would include intelligent determination of where these "pitfalls" would exist and, thus, how to avoid them.

C. INPUT DATA SIZE AND ENCODED DATA SIZE

An important consideration when performing any type of data encoding is the size of the encoded version of the data. This is especially important in the case of a network protocol such as HDTN, where transmission of as small a file as possible would be preferable. To examine the relationship between input and encoded data sizes in this fountain code implementation, a set of dummy files was created. On Ubuntu 20.04.4 LTS, the "yes" command was used to create various dummy files for testing purposes. For example, a 1 MB dummy file was created using the command "yes this is a 1mb file | head -c 1000000 > 1mb". The result is a file called "1mb" consisting of the repeated phrase "this is a 1mb file" that is 1 MB in size. Twenty-six such files were created, with sizes ranging from 1 MB to 1100 MB. Each file was put through the encoder, and the size of the resulting encoded data

TABLE 3. Testing data for encoding runtime, decoding runtime, and encoded filesize based on input filesize.

Input File (MB)	Encoding (ms)	Decoding (ms)	Output File (MB)
1	1.17	13.56	0.319
2	2.35	26.77	0.607
3	3.65	40.50	0.895
4	5.60	53.42	0.119
5	7.10	70.98	0.150
6	9.06	81.13	0.180
7	10.76	96.24	0.205
8	12.41	109.07	0.236
9	15.25	124.54	0.263
10	16.25	136.14	0.179
15	27.15	209.61	0.261
20	33.90	288.04	0.365
25	48.11	346.51	0.456
30	61.09	412.50	0.546
35	69.76	493.18	0.640
40	83.55	565.36	0.737
45	93.32	629.92	0.817
50	104.85	710.24	0.908
75	177.05	1106.51	1.330
99	240.91	1539.90	1.780
100	220.88	1497.88	2.930
250	668.58	4057.76	7.390
500	1446.11	8420.67	14.700
750	2242.98	14394.25	22.300
999	3186.81	19496.57	29.600
1000	3311.92	19554.45	23.700

TABLE 4. Testing data for comparing bundle size and output filesize.

Bundle Size (B)	Encoded Filesize (MB)
256	3.120
512	1.580
1024	0.990
2048	0.638
4096	0.428
8192	0.301
16384	0.263
32768	0.211
65536	0.181

TABLE 5. Testing data for comparing redundancy scalar and resulting loss tolerance.

Redundancy Scalar	Transmission Loss tolerance (%)
1.3	3.5
1.4	2.8
1.5	15.5
1.6	25.5
1.7	25.9
1.8	21.9
1.9	37.6
2.0	37.9
2.1	37.0
2.2	32.5
2.3	44.3
2.4	42.6
2.5	37.7
2.6	50.4
2.7	50.3
2.8	54.3
2.9	47.8
3.0	54.9

was recorded. The relationship between input and encoded file sizes is shown in Figure 6.

The relationship between these factors appears to be linear. The steep declines in encoded file size at each power of ten (10 MB, 100 MB, and 1000 MB) is of note, and is likely attributed to the way in which GZIP file compression is achieved. More importantly, however, is the comparison between input and encoded file sizes. The encoded file size is always less than half of the input file size, showing the efficiency of this encoding method.

D. BUNDLE SIZE AND ENCODED DATA SIZE

As discussed earlier, given a constant input file, the runtime of the fountain code software decreases as bundle size increases. Thus, it was hypothesized that this was due to larger bundle sizes resulting in smaller encoded data file sizes. To test this hypothesis, a 5 MB file was passed through the encoder with various bundle sizes, ranging from 256 B to 65536 B. Figure 7 shows the relationship between bundle size and encoded data file size.

The encoded data file size decreases as the bundle size increases. The graph of this relationship resembles that of the familiar function $y = 1/x$, showing that extremely small bundle sizes will result in inefficient encoded file sizes. However, since increasing the bundle size has rapidly diminishing returns, it would be best to locate an optimal bundle size for a given input file size. This concern is reserved for future developments of this software module.

E. RESULTS COMPARISON

We compared our novel fountain code approach to several related works studying reliable transport for unicast and multicast in delay tolerant networks. LTP is frequently recommended for use in long delay, error prone links. The approach described in [25] compared TCP, LTP, and UDP convergence layers for scenarios with delays ranging from 1000 ms- 5500 ms with bit error rates (BER) up to 10^{-5}. LTP was further optimized in [23] by aggregating multiple bundles in a single LTP block to reduce the number of required acknowledgements. The metric used in [23] is Goodput (Bytes/sec), which is a measurement of error-free, completed data received, as opposed to throughput, which does not consider lost or corrupted data. The results show that aggregating multiple bundles per block provides a significant advantage. Global Selective Acknowledgements are used in [24] to provide global information about the receipt of packets throughout the network and is suitable for both unicast and multicast. The metrics used for evaluation are round trip delay and success probability. The system was tested in several multi-hop scenarios and shows the ability to reach up to 0.75 success probability.

Our method is evaluated based on compressed file size, encoding/decoding run-time, and loss tolerance. The fountain code method is able to reduce file size, while also providing redundancy and loss tolerance as described in the sections above. Future tests will focus on assessing delivery probability and Goodput as relevant metrics. Table 2 shows a summary of the evaluation comparisons.

VI. CONCLUSION

Fountain code is suggested as a promising encoding method for HDTN, as it can fragment, distribute, and reassemble bundles across multiple nodes and receivers, while also providing loss tolerance. This makes it well-suited for use in space networking environments, where communication paths are often disrupted and data transmission can be delayed or lost. The application of fountain code is expected to improve the scalability and efficiency of HDTN in space networking environments, where delays and disruptions can occur at magnitudes not present in typical communications on the ground, thus making it possible to transmit large amounts of data over long distances and under hostile conditions. This is an important development for space exploration and satellite communication, as it will allow NASA scientists and engineers to confidently gather and transmit more data, enabling the engagement of more advanced scientific research and space networking endeavors in the future.

Future work will focus on testing the fountain code application with two HDTN nodes over an emulated link. The HDTN laboratory at GRC has acquired a Netropy 10 G4 network emulator, which will be used for simulating long delays (up to 10 seconds), packet loss, and reordering. The HDTN LTP implementation will be tested alone and then enhanced with the fountain code application. In addition to this testing, the fountain code prototype will be integrated and tested with HDTN multicast capabilities.

APPENDIX

A. FOUNTAIN CODE PROTOTYPE

The fountain code prototype described in this article is in the process of release to the public via the NASA Open Source Agreement. Once released, the software repository will be linked to the main HDTN repository information page located at https://github.com/nasa/HDTN/wiki.

B. ENCODER PROTOTYPE USAGE

The encoder module takes several command line arguments. The "-b" or "–bytes" flag sets the size of each encoded bundle in bytes, with a default value of 65536. The "-r" or "–redundancy" flag sets a redundancy scalar that correlates with error tolerance and encoded file size and has a default value of 2.0. The "-tlp" or "–transmission-loss-percentage" flag allows for simulation of data loss during encoding to test the error tolerance of the encoded data. Since it is intended for testing purposes only, its default value is 0.0. For 32-bit systems, the "–x86" flag can be used to make sure that the encoded data is created using a 32-bit datatype. The usage information is detailed in the figure below, and can be accessed by running the program with the "-h" or "–help" flag. Table 6 shows basic usage of the encoder prototype and Table 7 shows optional arguments to use with said prototype.

TABLE 6. Encoder usage information.

Fountain code encoder for use with NASA's HDTN
Usage: encode.py [-h] [-b BYTES] [-r REDUNDANCY] [-tlp TRANSMISSION_LOSS_PERCENTAGE] [–x86] filename
Positional Arguments: filename Input file path

TABLE 7. Encoder usage information (Optional arguments).

Optional Arguments	
-h, –help	Show this help message and exit
-b BYTES, –bytes BYTES	number of bytes per bundle >= 0
-r REDUNDANCY, –redundancy REDUNDANCY	Scalar for the encoded data's size >= 1.3; higher values will increase redundancy as well as file size
-tlp TRANSMISSION_LOSS_PERCENTAGE, –transmission-loss-percentage TRANSMISSION_LOSS_PERCENTAGE	Simulate transmission loss; percentage from 0 to 100
–x86	Use 32-bit unsigned int datatype for the encoded data buffer

TABLE 8. Decoder usage information.

Fountain code decoder for use with NASA's HDTN
Usage: decode.py [-h] [–x86] filename
Positional Arguments: filename Input file path

TABLE 9. Decoder usage information (Optional arguments).

Optional Arguments	
-h, –help	Show this help message and exit
–x86	Use 32-bit unsigned int datatype for the encoded data buffer

C. DECODER PROTOTYPE USAGE

Like the encoder, the decoder module takes command line arguments. The "–x86" flag can be used to make sure that the encoded data is created using a 32-bit datatype, which is necessary when using systems running on 32-bit operating systems. The usage information is detailed in Table 8 below, and can be accessed by running the program with the "-h" or "–help" flag. Table 9 shows the optional arguments that can be used with the decoder prototype.

ACKNOWLEDGMENT

This work was completed at Cleveland State University as part of the 2022–2023 University Senior Capstone Projects at NASA Glenn, Space Project 4—Application of Fountain Code to Delay Tolerant Networks. The senior capstone team was sponsored by the High-rate Delay Tolerant Networking (HDTN) project funded by the NASA Space Communications and Navigation (SCaN) Program. The authors would like to thank Dr. Daniel Raible, John Nowakowski, Jay Wyatt, Philip Baldwin, and the HDTN software development team.

REFERENCES

[1] L. Torgerson, S. C. Burleigh, H. Weiss, A. J. Hooke, K. Fall, D. V. G. Cerf, K. Scott, and R. C. Durst, *Delay-Tolerant Networking Architecture*, document RFC 4838, Apr. 2007. [Online]. Available: https://www.rfc-editor.org/info/rfc4838

[2] D. Raible, R. Dudukovich, B. Tomko, N. Kortas, B. LaFuente, D. Iannicca, T. Basciano, W. Pohlchuck, J. Deaton, A. Hylton, and J. Nowakowski, "Developing high performance space networking capabilities for the International Space Station and beyond," NASA, Washington, DC, USA, Tech. Memorandum NASA/TM-20220011407, 2022. [Online]. Available: https://ntrs.nasa.gov/api/citations/20220011407/downloads/TM-20220011407.pdf

[3] L. Wood, W. Ivancic, W. Eddy, D. Stewart, J. Northam, C. Jackson, A. Da, and A. Da Silva Curiel, "Use of the delay-tolerant networking bundle protocol from space," in *Proc. Int. Astron. Congr.*, vol. 5, Sep. 2008, pp. 3123–3133.

[4] NASA Glenn Research Center. *High-Rate Delay Tolerant Networking (HDTN) Project*. Accessed: May 8, 2023. [Online]. Available: https://www1.grc.nasa.gov/space/scan/acs/tech-studies/dtn/

[5] *High-Rate Delay Tolerant Network*. Accessed: May 8, 2023. [Online]. Available: https://github.com/nasa/HDTN

[6] M. Ramadas, S. Burleigh, and S. Farrell, *Licklider Transmission Protocol—Specification*, document RFC 5326, IETF Network Working Group, 2008. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5326

[7] K. Scott and S. Burleigh, *Bundle Protocol Specification*, document RFC 5050, IETF Network Working Group, 2007. [Online]. Available: https://tools.ietf.org/html/rfc5050

[8] S. Burleigh, K. Fall, and E. J. Birrane, *Bundle Protocol Version 7*, document RFC 9171, Jan. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9171

[9] A. Shokrollahi, "Fountain codes," *IEE Proc. Commun.*, vol. 152, pp. 1290–1297, Jan. 2005.

[10] S. M. M. Langari, S. Jabbehdari, S. Yousefi, and K. Zayer, "File transfer in vehicular delay tolerant networks using fountain coding," in *Proc. ICCKE*, Oct. 2013, pp. 121–128.

[11] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, "Efficient routing in intermittently connected mobile networks: The multiple-copy case," *IEEE/ACM Trans. Netw.*, vol. 16, no. 1, pp. 77–90, Feb. 2008.

[12] E. Altman and F. De Pellegrini, "Forward correction and fountain codes in delay tolerant networks," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 1899–1907.

[13] S. R. Das, C. E. Perkins, and E. M. Belding-Royer, *Ad Hoc On-Demand Distance Vector (AODV) Routing*, document RFC 3561, Jul. 2003. [Online]. Available: https://www.rfc-editor.org/info/rfc3561

[14] P. U. Tournoux, E. Lochin, J. Leguay, and J. Lacan, "Robust streaming in delay tolerant networks," in *Proc. IEEE Int. Conf. Commun.*, May 2010, pp. 1–5.

[15] Y. Cao and Z. Sun, "Routing in delay/disruption tolerant networks: A taxonomy, survey and challenges," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 2, pp. 654–677, 2nd Quart., 2013.

[16] S. Ding, X. He, J. Wang, and J. Liu, "Pre-decoding recovery mechanism for network coding opportunistic routing in delay tolerant networks," *IEEE Access*, vol. 6, pp. 14130–14140, 2018.

[17] F. Walter and M. Feldmann, "Dynamic discovery of ground stations in ring road networks," in *Proc. IEEE Int. Conf. Wireless Space Extreme Environ. (WiSEE)*, Sep. 2016, pp. 93–98.

[18] Y. Kirkpatrick, R. Dudukovich, P. Choksi, and D. Ta, "Cooperative clustering techniques for space network scalability," in *Proc. IEEE Cognit. Commun. Aerosp. Appl. Workshop (CCAAW)*, Jun. 2023, pp. 1–8.

[19] V. Juyal, R. Saggar, and N. Pandey, "An optimized trusted-cluster–based routing in disruption-tolerant network using experiential learning model," *Int. J. Commun. Syst.*, vol. 33, no. 1, p. e4196, Jan. 2020.

[20] V. Juyal, R. Saggar, and N. Pandey, "On exploiting dynamic trusted routing scheme in delay tolerant networks," *Wireless Pers. Commun.*, vol. 112, pp. 1705–1718, Jun. 2020.

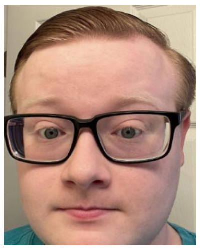
[21] M. Luby, "LT codes," in *Proc. 43rd Annu. IEEE Symp. Found. Comput. Sci.*, Nov. 2002, pp. 271–280.

[22] P. Deutsch. (1996). *GZIP File Format Specification Version 4.3*. Network Working Group. [Online]. Available: https://www.rfc-editor.org/rfc/rfc1952

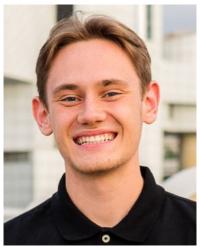
[23] R. Wang, Z. Wei, Q. Zhang, and J. Hou, "LTP aggregation of DTN bundles in space communications," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 49, no. 3, pp. 1677–1691, Jul. 2013.

[24] A. Ali, T. Chahed, E. Altman, M. Panda, and L. Sassatelli, "A new proposal for reliable unicast and multicast transport in delay tolerant networks," in *Proc. IEEE 22nd Int. Symp. Pers., Indoor Mobile Radio Commun.*, Sep. 2011, pp. 1129–1134.

[25] R. Wang, S. C. Burleigh, P. Parikh, C.-J. Lin, and B. Sun, "Licklider Transmission Protocol (LTP)-based DTN for cislunar communications," *IEEE/ACM Trans. Netw.*, vol. 19, no. 2, pp. 359–368, Apr. 2011.



NOAH P. DOUGLASS received the B.Sc. degree in computer science from Cleveland State University, in 2023. He was an Application Developer with Applied Industrial Technologies, since 2023. His interests include artificial intelligence, machine learning, software development, and wearable devices.



JOHN LANGEL received the B.Sc. degree in computer science from Cleveland State University, in 2023. His interests include artificial intelligence, cryptology, and computer security.

WEILAND J. MOORE received the B.Sc. degree in computer science from Cleveland State University, in 2023. His interests include artificial intelligence, machine learning, and automation.

LAWRENCE NG received the B.Sc. degree in computer science from Cleveland State University, in 2023. His interests include artificial intelligence and software development.

RACHEL M. DUDUKOVICH (Member, IEEE) received the M.S. degree in electrical engineering and the Ph.D. degree in computer engineering from Case Western Reserve University. She is currently a member of the Cognitive Signal Processing Branch at NASA Glenn Research Center and has been with NASA, since 2010. She serves in a dual role as the Cognitive Networking Lead Researcher for the Cognitive Communications Project and a Software Development Lead for the HDTN Project. Her interests include cognitive networking techniques, delay-tolerant networking, network emulation, artificial intelligence, machine learning, and algorithm development.

SANCHITA MAL-SARKAR received the M.S. degree in computer science from the University of Windsor, Winsor, Canada, the M.Sc. degree in physics (with specialization in electronics) from Benaras Hindu University, Benaras, India, the Ph.D. degree from the Department of Electrical Engineering and Computer Science (EECS), Cleveland State University. She is currently an Associate Lecturer with EECS Department, Cleveland State University. Her research interests include hardware and software security and trust, the IoT security, fault-tolerant networks, soft/granular computing and risk analysis, and wireless sensor networks. She has authored numerous journal articles and presented papers at several national and international conferences. She has served as a peer-reviewer for several conferences and in the Technical Program Committee and for journals, such as IEEE TRANSACTIONS ON MULTI-SCALE COMPUTING SYSTEMS, IEEE TRANSACTIONS ON FUZZY SYSTEMS, IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUIT AND SYSTEMS, *ACM Journal of Engineering Technologies in Computing* (JETC), *Expert Systems with Applications*, *Data and Knowledge Engineering* (DKE), and *International Journal of Sensor Networks* (IJSNET). Her research was supported by the U.S. National Science Foundation (NSF) and Cleveland State University. She is a three-time Merit Recognition Award recipient at CSU for her record of academic achievement.