Rising Above the Cloud- Toward High-Rate Delay-Tolerant Networking in Low-Earth Orbit

Alan Hylton, Daniel Raible, Gilbert Clark, Rachel Dudukovich, Brian Tomko, and Laura Burk

NASA Glenn Research Center, Cleveland, Ohio, 44135

Keywords: Delay Tolerant Networking, Optical Communication

Abstract

The High Data Rate Architecture (HiDRA) project is implementing a High-rate Delay Tolerant Networking (HDTN) capability that can support Low Earth Orbit (LEO) applications and environments. The present state of the effort, future work, and other elements of the work to date are described in this paper. This implementation is intended to support applications that run at 1+ Gbps, per the requirements of modern optical and high-frequency RF links. Uniquely, this implementation is also tuned to support relay and data trunking applications, which might require support for large numbers of small bundles per second. The design for this platform is based entirely on commercial-off-the-shelf (COTS) components, and possesses buffering capabilities in the 5 TB range.

This document takes results from previous individual tests and integrates them to demonstrate results in the presence of a coherent use-case: consider a network aboard the ISS which intends to utilize an upcoming optical communications capability. For this use-case, orbital analysis software is used to analyze orbital dynamics, from which a list of access times are generated that might take in to account weather, schedule competition, etc. A variant of Contact Graph Routing (CGR) is applied to these windows to determine an optimal schedule. This schedule is then loaded into the HDTN prototype and, in conjunction with various measurement tools, a complete end-to-end analysis of HDTN's performance is conducted. Various bottlenecks (including storage) are identified: these bottlenecks are expected to help us focus our future work on the elements of the system that are most likely to present issues moving forward. Finally, we discuss possible paths for evolution beyond the present rates supported by the system, including (but not limited to) hardware acceleration.

1 Introduction

For the past half-century of space exploration, radio frequency (RF) means of communication and navigation have served as the backbone for both human and robotic missions. While the current optical (laser) communication technology demonstrations promise to deliver an order-of-magnitude increase in bandwidth[1][2][3], the current space and ground infrastructure must adapt in order to scale concomitantly. One crucial aspect of the necessary evolution lies in the realization of capabilities that can support delay- and disruption-tolerance, both in the architectural[4] and the implementation[5] sense, at rates that will allow high-speed links to be used to their full capacity. This functionality is at the heart of delay tolerant networking (DTN).

Generally speaking, existing implementations of DTN have been designed to suit constrained end-nodes, which has (necessarily) limited their support for parallelism or pipelining capabilities present in modern systems. One reference implementation of DTN (Interplanetary Overlay Network, or ION), for example, relies on a shared database kept in shared memory. This database is protected by a series of mutual exclusion locks that prevent more than one process from ever obtaining access to its bundle database at once. Since each individual element in the ION pipeline requires access to that database, many elements in the pipeline are often stalled waiting for a different process to complete its task and free its lock. Further, heavy contention for the lock itself leads to a relatively high amount of inter-CPU synchronization overhead, especially when specific ION processes are freely allowed to migrate between cores.

While such designs are capable of achieving reasonable rates with sufficient tuning (e.g. maximizing the amount of data one sends in a single *bundle*, or *coherent unit of transport*), their architecture leads to a ceiling on their ability to scale outward. This forces one to spend a great deal of time bootstrapping existing implementations to support use-cases for which they were never intended. Along those lines, there has historically been a gap in implementations of DTN that are designed to achieve high-rate routing and transport. To fill that gap, NASA has begun evaluating and implementing new approaches to DTN that might better exploit the capabilities of modern systems. This paper describes one such implementation called High-rate DTN (HDTN), which is being developed at NASA's Glenn Research Center (GRC).

The major goal of this paper is to describe a fitness test of HDTN in a realistic setting, mixed with other DTN implementations. Success is defined by demonstrating three elements: HDTN can support up to 1Gbps rates even when sending "small" messages, HDTN maintains interoperability with DTN implementations that were designed for other purposes, and HDTN can scale each of its functions horizontally across a large number of cores (or even discrete systems). The first two criteria will be expanded upon in Section 2.

1.1 Network Model of Optical Communications

The necessity for DTN in a space network is driven by the manner in which such networks normally operate; connectivity between any two nodes is primarily ruled by orbital mechanics. Links are bidirectional in nature, but are highly asymmetric: the command and control messaging are a fraction of the bandwidth demanded by scientific instruments on-board the spacecraft. When optical communications are employed as an addition to RF systems, they do not fundamentally change this approach to mission design, though they strongly influence the manner in which the network operates. Whereas RF-based communications rely on longer links with a lower continuous rate, optical communications are envisioned as extremely high-rate bursts. This enables long-term buffering with quick flushes to the ground. Existing Tracking and Data Relay Satellite System (TDRSS) spacecraft, though well-proven for older communications, do not support the burst-oriented approach, and are anyways being phased out[6]. Thus future missions will need a different mode of operation than TDRSS.

Such support networks could exist as mix of government owned and operated assets and commercially supplied bandwidth. Any future-looking approach to modeling space networks must consider these possibilities. The corresponding mixture of data flows will push the consideration of quality of service (QoS) as commercial space, civilian science \& exploration (both robotic and human), and academic utilization of communication networks to extend small-satellite operations. Each user would need to develop a set of particular QoS metrics (bit error rates, throughput, security, etc.) so that their traffic could be managed effectively across the network.

Although particular implementations may vary, it is important that the system architecture serve as much more than a point solution by offering extensibility to lunar and planetary exploration, and across a heterogeneous array of deployed assets and technologies. This is quite the formidable challenge because space operations present a wide assortment of variables. Elements to consider include, for example, the particular radiation environment as well as dynamic link ranges limiting instantaneous data rates and imparting memory buffer requirements on the spacecraft. This yields a requirement that the network architecture be flexible enough to accommodate the variability of mission parameters. With that said, the network architecture must not be so general and abstract that the complexity would preclude the development of meaningful capabilities to accommodate an ever-increasing and -evolving portfolio of mission technologies and objectives.

1.2 Experiment Network

In order to test HDTN in a realistic environment, then, it was necessary to experiment with different approaches to a future space network. For the purposes of this experiment, the network was kept relatively simple: more details are offered in Section 3.

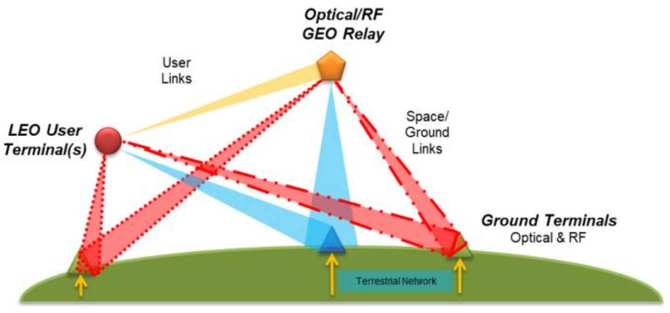

Figure 1 Basic architecture

Per Figure 1, which is explained further in Section 3, a user spacecraft in low Earth orbit (LEO) is considered. It may communicate to a second spacecraft in geosynchronous equatorial orbit (GEO), which houses an optical communications relay. Or it may opt to transmit optical ground stations directly – it may also use RF capability. Considered together, these assets and capabilities present a heterogeneous network which considers technology, range, weather and terminal masking, and multi-path options. As such, there are opportunities to test basic network policy as well as various approaches to switching and routing traffic.

2. Delay-Tolerant Networking

In order to cope with sporadic availability and long link propagation delay, a core component of the future NASA network is envisioned to be DTN. While this paper will not attempt to act as an introduction to DTN - for suitable material, the reader is encouraged to consult[4][5][7][8] - however the necessities are reviewed below.

From an architectural standpoint, delay-tolerance is achieved through the use of an overlay network. This overlay is a logical construct that can be thought of as a graph that exists on top of a number of existing assets and links. A vertex in this graph is a place at which data may exit the overlay: such a destination may map to either one physical asset or a collection of many (e.g. an entire constellation). An edge in this graph represents a logical link between two endpoints: such edges may be constructed upon any protocol, known as convergence layers (CL). These consist of small transport-later protocol adaptations which allow the atom of DTN data, the bundle, to pass through individual edges. The protocol by which such bundles are transported through the overlay itself is called the Bundle Protocol[5].

If a purpose of networking is to achieve scalability of communications in the number of nodes, it must not become the bottleneck. However, the staple implementations' limitations are well documented[8][9][10]: maximum rates in the low 100's of megabits/second are common regardless of computational horsepower. Consider two DTN nodes connected over TCP/IP. If a transmitted bundle is over several megabytes, one may correctly assume that the data rate would essentially be the line-rate. However, most bundles are fragmented into collections of much smaller bundles. This increases the overhead associated with processing the number of bundles. The aforementioned computational complexity begins to play a profound role, as the header fields are not of fixed width[5], and hence there is no random access within a given header. This and other factors imply that the software architecture will determine the maximum performance levels of a DTN when considering the metric *bundles per second*.

We recall that it is imperative to have several independent manifestations of a protocol for interoperability testing. One version can help unearth bugs in the code, but having two or more help find bugs in the specification. However, interoperability means more than "speaking the same language," and includes not speaking more quickly than one can listen. Hence the original goal for this paper implies that we need to maintain interoperability with DTN packages of all capabilities up to HDTN's.

Existing DTN versions are held to other metrics than data-rate, and should be considered different tools for different applications; they may all have a place in a single given network. We note that naming is consistent between the implementations (in this case); we use InterPlanetary Network (IPN) naming. Thus a name is of the form `ipn:x.y`, where the number `x` is the name of the node and the number `y` refers to the service number (see [11]). The high-rate DTN bplib[12] uses the flow-concept to determine where data should be sent and received. A flow is defined as a source node and service number followed by a destination node and service number, e.g. `a.b_m.n`, where `a` is the source node number, `b` is the source service number, `m` is the destination node number and `n` is the destination service number. These service numbers correspond to application threads on the local and remote nodes which will call bplib functions to store and accept data.

For the tests conducted, we used the following DTN implementations:

- Interplanetary Overlay Network (ION)[13]
  - Developed by the Jet Propulsion Laboratory (JPL), this is the NASA reference implementation
  - Designed with deep-space in mind
  - Performance ~100Mbps for small bundles[8][9]
- Bplib (Bundle Protocol Library)[12]
  - Developed by the NASA Goddard Space Flight Center
  - Lightweight framework on which more complete implementations of the bundle protocol may be based
  - Bplib may be used to support the development of code that operates at high rates'
- High-Rate Delay Tolerant Networking (HDTN)[8]
  - Developed by the NASA Glenn Research Center
  - Distributed implementation of the bundle protocol
  - Focuses on support for horizontal scaling, pipelining, and parallelism

The test setup for this paper includes three ION nodes: `ipn:1`, `ipn:2`, and `ipn:3`, one HDTN node: `ipn:4`, and three bplib nodes: `ipn:5`, `ipn:6`, and `ipn:7`. They are connected as in Figure 2. The intuition is that the ION nodes would serve as sensor nodes aboard the spacecraft, which then connect to HDTN directly, which would function as a store, carry, and forward hub between this internal satellite network and ground stations running bplib. Time-varying connectivity is described in further detail in Section 3.

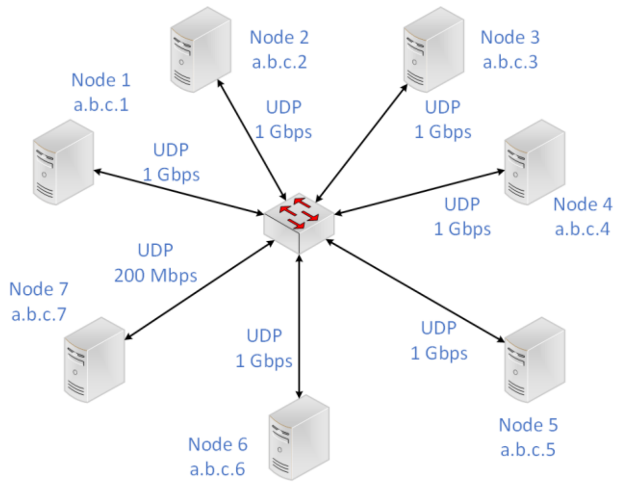

Figure 2 Network configuration

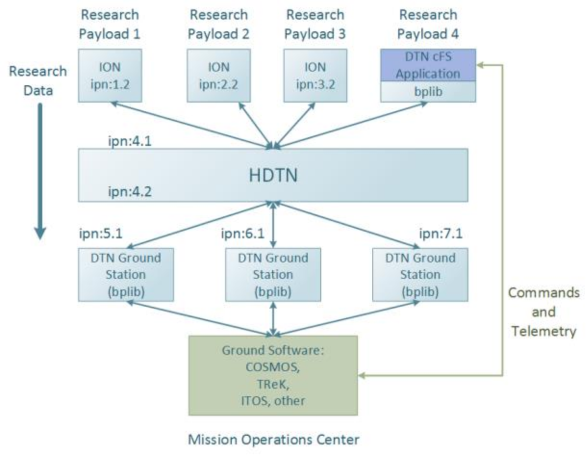

Figure 3 Software configuration

2.1 Interoperability of Software Components

The three DTN software packages (ION, HDTN, and bplib) perform the data storage and bundle encoding & decoding functions, but it is anticipated that there will be additional software components to command the experimental payloads and send health and status telemetry to a mission operations center. In particular, NASA core Flight System (cFS)[14] has been used extensively among space flight missions for experiment control, commands and telemetry. Several ground software packages are commonly used in conjunction with cFS to provide a graphical operator interface, such as ITOS (Integrated Test and Operations System)[15], COSMOS[16], and TReK (Telescience Resource Kit)[17] as well as custom developed solutions, as shown in Figure 3.

Bplib is an open source library actively under development by Goddard Space Flight Center and implements a subset of the RFC 5050 Bundle Protocol [5]. The library provides bundle encode, decode and storage capabilities and is intended for use in embedded applications. It was developed as a library meant to be integrated into a board support package for core Flight System projects as well as other applications requiring basic Bundle Protocol compatibility. For this reason, bplib requires that its calling application implements the actual threads of execution to perform the bundle reception, processing and transmission tasks, and uses thread safe synchronous blocking I/O. HDTN is interested in compatibility with bplib not only for its use as a research payload DTN interface but also for the possibility that the same library maybe used as the basis of a DTN ground software implementation.

A basic example of a bplib application is given in [12] that can be developed with 4 threads: a bundle reader, data writer, bundle writer and data reader. The bundle reader listens to a DTN CL such as a UDP socket. When data is received, the bundle payload is stored. The data writer thread accepts the payload data and may pipe the application data to services used by the calling application. Data can be sent from the calling application using a data reader thread which will pass application data to bplib for bundle encoding. The bundles are then queued until they can be sent using a bundle writer thread to send the bundles to the DTN CL.

3 The Scenario: Generating the Connectivity Model

The example scenario used was simulated using the Satellite Orbit Analysis Program (SOAP) across a two year mission profile to generate statistical data on the availability to complete links between the communication terminals. The idea is to create a system that includes optical links, RF links, features multiple hops and multiple paths, and finally, enjoys some semblance of reality. The scenario used is a blend of real assets, upcoming assets, and some imagination. ION has been used about the International Space Station (ISS) since 2016[18], which will also host upcoming optical payloads[19] which are targeting GEO-relay and direct-to-Earth (DTE) optical communications. Therefore we model our scenario on a potential use case of these assets: we envision ION nodes connected to HDTN, all aboard the ISS, which then communicates to both optical ground stations and also an RF ground station as described in Section 2.

Figure 4 shows a screenshot of the SOAP model which captures Figure 1 and demonstrates the scope of the system. The Earth has two optical ground stations (OGS) and an RF ground stations, OGS-1, OGS-2, and White Sands, respectively. The ISS can communicate using optical and RF, and finally there is STPSat-6, which will host the optical relay[20].

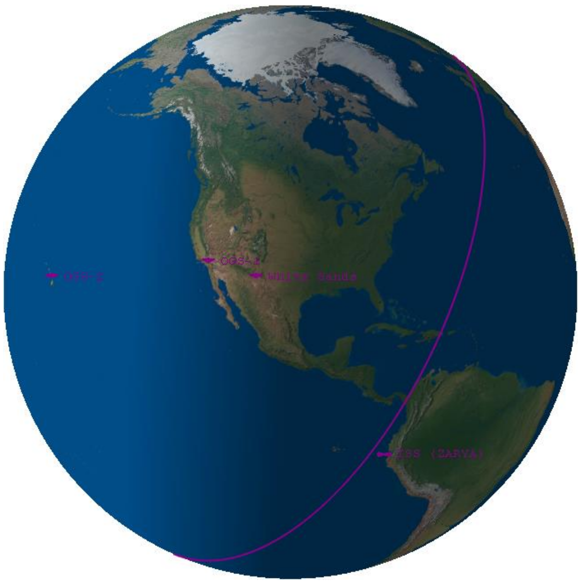

Figure 4 SOAP scenario

SOAP calculates the line-of-site availability between orbital assets. To simplify matters, and because we are more interested in higher-layer protocols than Layer 1 (particularly encoding), we assume cloud-free line of sight (CFLOS). For our scenario, we are interested in:

- Optical contact from the ISS to the relay,
- Optical contact from the ISS direct-to-ground, and
- Direct RF contact from the ISS to White Sands.

A 24-hour window was chosen from which to create the network-test bed. The only assumption was that no two optical links would operate simultaneously. Moreover, we wanted to switch between the relay and direct to Earth (DTE) communications when possible in order to stress the flexibility of the network. The RF link was used sparsely, to represent the contention for using an already constrained link. When making switch-overs, we blocked out a minute to simulate beam steering and link establishment. Ultimately, DTN should accommodate any variations.

DTN typically uses schedules, known as *contact graphs*[7], to determine routing. The data used to generate the contact schedule over the aforementioned 24-hour period is shown, loosely, in Figure 5. While generated from the data used, this figure is not meant to be rigorous, but rather used for intuition; hence units are purposefully excluded. Over one day, the red blocks represent contacts from the ISS to the relay (and hence either ground station), which are interrupted, as discussed, in favor of direct contact with either OGS-1 (green) or OGS-2 (blue). The RF link, in yellow, is always utilized when available.

We assume that onboard the satellite, all nodes are constantly connected, and that the optical relay can always communicate with either ground station. Given an opportunity for DTE optical communications, this is always taken. If no DTE path exists, but a relay path does, the ground stations were scheduled in an alternating pattern. We note that relays are transparent at the network layer. We also assume that data rates do not depend on whether or not the relay is utilized. Finally, an unreasonable assumption is made for the ground stations: there is sufficient connectivity between the ground stations and some MOC such that terrestrial data transport is guaranteed. This is definitely not the case, especially when the best sites for optical ground stations tend to be remote. However, addressing infrastructural concerns is beyond the scope.

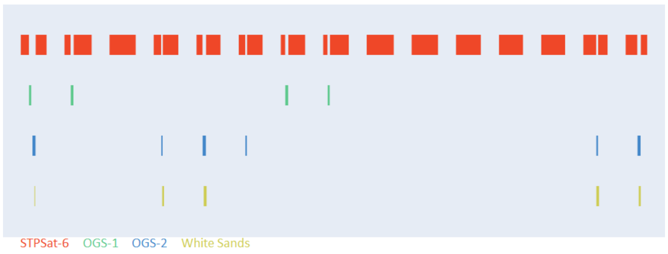

Figure 5 Schedule data visualization

The output of SOAP gives start and stop times of each contact in the simulation time. The data was filtered and then used to generate schedules. ION uses a globally distributed table of connectivity, the contact graph. This includes one-way light time, however in this case the round trip time (RTT) was low enough to be inconsequential even for TCP, and so was omitted. HDTN uses Multiprotocol Label Switching (MPLS)[21], which is described further below. Finally bplib does not yet have a scheduler, and so Bash scripts were created to schedule transmissions.

Brief statistics are shown in Table 1 for the 2-year period. Given the chosen orbits, the DTE durations are not surprising. However, it was unexpected to see roughly hour-long contacts when the relay was utilized. This is something that would be pruned by adding fidelity to the SOAP model, however this unnecessarily over specifies the problem; we recall that the network should react gracefully to the schedule changes.

	STPSat-6	OGS-1	OGS-2	White Sands
Duty Cycle %	58.06	1.74	1.29	1.34
Average Contact(min)	57.61	5.62	5.69	5.15
Variance	7.17	2.77	2.62	2.25

Table 1 Basic statistics over 2-year period

4 High-Rate Delay-Tolerant Networking

4.1 Introduction

High-rate DTN (HDTN) is an implementation of DTN that demonstrates various concepts and experiments that have been performed within the scope of the High Data Rate Architecture (HiDRA) project over the past few years. HDTN has a hard requirement to support sustained bundle flows at rates of 1+ Gbps, with support for expanding that number as requirements change. As such, HDTN has placed an emphasis on demonstrating how a DTN engine can support horizontal scaling, incremental hardware acceleration, and generally achieve high data rates (both in terms of throughput and in terms of bundles per second) without necessitating large investments in hardware that has been developed specifically to support the Bundle Protocol.

HDTN is a work in progress, and as such may see some changes to design or implementation. With that said, lab experiments have been encouraging: the software has been able to consistently process 5 - 6 Gbps of traffic (given 1K bundle sizes) that can be forwarded without needing to first be stored. When disk-based storage is required, that number falls somewhat: the specific impact depends on the storage medium selected. Our implementation of HDTN currently relies on a collection of three 1 TB SSD persistent storage devices, each of which is running over a SATA 3 (6 Gbps) interface. These disks are sufficient to support the necessary 1 Gbps rate, with results to date demonstrating performance in the 2 Gbps range. RAM-based storage has proven to be much faster, but is necessarily limited in what it can support: most systems support far more disk space than they do RAM. As such, approaches to tiered caching and retrieval are an active area of study for the project: HDTN's design does allow for multiple instances of storage (each with different rates), but how best to utilize available storage remains an open problem in this area.

4.2 Components

Previous sections of this document have explained the need for support of parallel and pipelined approaches to network processing. In order to support such approaches, it is necessary to decompose DTN into a series of components that describe its core function. For the purposes of this paper, these components are:

- Ingress - CL adapter that accepts traffic in bundle format
- Egress - manages a collection of CL adapters that forward bundle traffic
- Switch - evaluates and forwards traffic based on various header fields
- Controller - manages schedules and indicates when specific events should happen
- Storage - manages storage and release of bundles for which a forward link is not immediately available

- Applications - applications can bind to the fabric directly and register themselves for bundle traffic

4.3 Interconnect

These components are realized as individual processes, but they do not use the bundle protocol when communicating internally. Instead, they rely on a custom lightweight message fabric built on top of Ethernet. Netmap's software VALE switch[22] is used as a local interconnect when multiple HDTN processes are running on a single host. When clustered operation (e.g. operation across multiple hosts) is desired, physical interfaces can be bound to the HDTN processes directly. When hybrid modes of operation are desired (e.g. two ingress processes on one system communicating with a switch operating on another), physical interfaces may be bound to virtual interfaces on a specific instance of VALE.

A lightweight discovery protocol runs between all connected components, allowing multiple instances of the same component to spin up / spin down as needed to address load. This allows HDTN to be deployed and extended incrementally as load begins to catch up to the capabilities of the system. Further, individual components may be replaced by specific hardware components (e.g. ASICs and / or FPGAs) in order to achieve SWaP numbers that are realistic for modern spacecraft to support: this effort largely remains future work at present.

Netmap's performance has been acceptable to date: initial testing has shown performance in the range of roughly 70 Gbps (at roughly 8 million messages per second) when pushing messages between processes connected to one another through a single instance of the VALE software switch. Netmap's own packet generator can achieve throughput that is substantially higher than this on identical hardware, so further tuning is expected to improve performance to a degree. The system is heavily memory constrained, however, as data is copied between processes in the current configuration.

Note that no data is shared directly between any two (or more) processes. Instead, necessary data is replicated between processes as needed - this configuration requires substantial care with respect to how configuration is propagated to the nodes and updated, but yields benefits to the speed and scalability that can be achieved: since each element operates completely independently of the rest, many bottlenecks can be addressed by e.g. spinning up additional instances of elements and load-balancing across all of them. Tuning the number of instances of each element is a manual process, but the discovery and automatic registration alleviates the associated configuration burden associated with this.

4.4 System Flow

Bundles arrive at the HDTN system through an ingress process. The ingress process examines the headers of a bundle and transforms it into an intermediate format (IF). To facilitate switching and routing within a specific HDTN, the system relies on label-switching. Upon reception of a bundle, the ingress element examines aspects of the bundle and translates it to a specific forwarding equivalence class (FEC) by assigning it a numeric label. This label is then used to guide the bundle's movement through other elements of the system. Once this label has been assigned, the bundle is sent to the switch element of the system.

The switch keeps a schedule internally that describes when specific links will be available and what the rates are. Schedules are managed on a per-label basis: any traffic that is labeled in a specific way will share a rate allocation and logical path to a specific destination. As such, the switch's purpose is to examine the label of the incoming traffic and evaluate, based on its label, what to do with it. This determination occurs through the application of a series of rules that are contained in a table - the process is similar to what occurs in an SDN-enabled switch, for example. In general, the switch can choose between four discrete actions:

- Deliver: the bundle is forwarded to a specific local endpoint when an application has registered for such
- Drop: the bundle is dropped
- Forward: the bundle is immediately forwarded to the next hop in its assigned path
- Store: the bundle is sent to a locally attached storage device for later retrieval

In the delivery case, the bundle is sent to an application. Deliver can either provide metadata only (e.g. data related to all bundle headers that the switch knows how to process), or the entirety of the bundle. Metadata-only is useful for cases where applications are being developed to enforce network-level policy (e.g. basic firewalls): the bundle payload can be held in storage while the processing application makes a decision based on the metadata provided. Complete delivery, on the other hand, is used to support more traditional applications that need to operate on bundle data directly.

In the forward case, the bundle is sent to an egress component for further processing. Upon arrival, this component evaluates the IF headers and determines how best to forward the bundle. In the case that direct interaction with a remote DTN endpoint is desired, the bundle protocol is used to directly send the bundle to its destination over, for example, traditional CCSDS protocols or the Internet Protocol (IP). Alternatively, the internal label can be translated into a conventional Multi-Protocol Label Switching stack for direct transport between instances of HDTN: MPLS has the benefit of being supported directly in hardware on many existing switching / routing chipsets, and generally has excellent support for making guarantees related to latency and rate allocations per-FEC.

For the drop case, the bundle is silently discarded. For space applications, this is an option only in limited cases: scientific data is extremely valuable, and thus extreme care should be taken to ensure that it is preserved and delivered as appropriate. Still, in certain situations, data may need to be removed from the network. Note that the switch supports modes of operation where bundle lifetimes are either enforced or ignored, and the bundle age block is directly supported as well.

Finally, the store action is one of the most complex. In general, the switching element keeps a local buffer (e.g. in system RAM) that it can use to store data for a specific label: this is intended for short-term buffering in the event of a link disruption. Once bundle traffic in a specific FEC has exceeded a quota (or the available buffer in the switch has been exhausted), both existing and future traffic for the FEC are forwarded to a storage element. This element acts as a kind of network-attached storage device, and possesses the capability to, when commanded, release data to the egress element at a specified rate. This element is also responsible for enforcing quotas, which are assigned and managed on a per-FEC basis.

The final element of HDTN discussed here is the egress element. This element accepts traffic from other HDTN components, evaluates the FEC and determines an appropriate destination, repackages the data into its original bundle format (if needed), and finally forwards the traffic as appropriate. The egress element includes support for various approaches to rate control, including support for specific inter-frame gap times when operating over certain protocols (e.g. UDP and LTP). Support for UDP/IP is stable, and support for TCP/IP and LTP/IP are far more experimental: they have been demonstrated, but should be considered works in progress.

There are other elements of HDTN (e.g. command and control interfaces) that are not discussed here. Additional detail is expected to be published in a future paper on the subject.

5 Test Discussion

The HDTN test bed consists of seven physical nodes running on Debian 10 for x86-64 architectures (amd64). UDP is used for the DTN CL since it is common among all three implementations, although it is planned to support additional CLs in the future. Considering the IPN naming (`ipn:x.y`), the service number `y` is either a 1 to indicate a data ingress service or a 2 to indicates a data egress service.

The desired data rates as shown in Figure 3 are controlled with Linux traffic control (tc). Bash scripts are used to automate and schedule the flow of data through the network. Each ION node (nodes 1-3) continuously listens for bundles using bprecvfile, which receives bundles and writes the payload file to disk. The ION nodes then repeatedly attempt to send 1 kB files using bpsendfile to the bplib nodes (nodes 5-7) through HDTN (node 4), as shown in Table 2. Each sending node will attempt to send the percentage of traffic shown in Table 2, however bundles will only actually be able to be forwarded when there are contacts scheduled between the sending node and node 4 according to ION's contact plan. HDTN's switch process will determine the appropriate node to deliver the bundle to based on the destination node and service numbers in the bundle's primary block.

	Receiving Nodes % of Traffic		
Sending Node	5	6	7
1 (1 Gbps)	90	0	10
2 (1 Gbps)	0	90	10
3 (1 Gbps)	40	40	20

Table 2 ION source traffic summary

	Receiving Nodes % of Traffic		
Sending Node	1	2	3
5 (1 Gbps)	100	0	0
6 (1 Gbps)	0	100	0
3 (1 Gbps)	0	0	100

Table 3 bplib source traffic summary

Since bplib does not follow a contact schedule in the same manner that ION does, the results of the orbital analysis were used to generate a series of transmission start times and durations. These contact times were then used in a Lua script which will create a new bundle writer thread for bplib, and attempt to continuously send a 1 kB file to the corresponding ION node, with a 0.03 second delay between iterations to approximately control the rate of traffic to the ION nodes. Table 3 shows the percentage of traffic sent from each bplib node (nodes 5-7) to ION (nodes 1-3) through HDTN (node 4). Each node also has a bundle reader thread which is always listening for bundles. An MD5 checksum is calculated for each file received and the total number of bundles received, correct bundles, and incorrect bundles are logged.

5.1 Networking Test Results

- Bplib does not support extension block 0x05 (Previous Hop Block) or 0x14 (Bundle Age Extension Block) used by ION. The blocks do not cause any problems, and are simply skipped and a warning message is printed.
- In order to send larger volumes of data at rates of up to approximately 480 Mbps, two constants needed to be increased in bplib. In bplib.c `BP_DEFAULT_ACTIVE_TABLE_SIZE` and in bplib_store\file.c, `FILE_DATA_CACHE_SIZE` were both increased from 16384 to 1000000. In addition, bundles were set to expire after one second so that storage queues would be cleared out faster.
- There were several queuing and storage issues experienced in bplib and ION when attempting to transmit and receive data at higher rates. This was an initial round of preliminary testing, and additional work must be done to understand the appropriate configuration

settings within ION and bplib, as well as the test bed automation scripts.

- The ION working memory (`wmSize`) and heap sizes (`heapWords`) were increased from the default to 1000000000 in order to allow for higher data rate testing.
- Additional network statistics collection could be added to the HDTN test set up to facilitate troubleshooting.

In general, HDTN did not have any issues handling the rates and number of bundles sent from the ION and bplib nodes. A previous paper[8] showed rates of 10,000+bundles/s were possible even for bundles less than 512 bytes large. Here, data transmission rates were intentionally constrained for this test so that initial troubleshooting of all of the software components could take place. A small sleep period of a few milliseconds was placed in the loop of each script for automating the transmission of bundles. We show that HDTN is capable of processing hundreds of bundles per second and additional high speed testing is underway.

6 Conclusion

In this paper, we described recent progress on a High-rate DTN (HDTN). We described the design and flow of bundles through our system, highlighting key aspects of its design and implementation to date. Further, we demonstrated the use of our implementation in a realistic scenario. We implemented this scenario in a lab setting, and sent many gigabytes of data over the course of a day of continuous testing. In the course of this testing, we demonstrated that our system was able to operate as intended, and that all implementations of the Bundle Protocol (ours included) used for this test were able to successfully inter-operate with one another.

To truly stress HDTN and bplib, much larger scale tests (more ION nodes) must be used. This will be the focus of an upcoming publication, however given bplib-HDTN performance, it is very promising.

7 Acknowledgements

The authors wish to acknowledge the contributions of Dr. Don Cornwell and Gary Pease for their continuing support, and Tom Basciano, Scott Burleigh, Marian Cronin, Dave Israel, Penny Roberts, and Jonathan Wilmot for everything from technical support to programmatic help to unwavering encouragement and patience.

8 References

[1] Murphy, D. V., Kansky, J. E., Grein, M. E., Schulein, et al., "LLCD operations using the Lunar Lasercom Ground Terminal," *Free-Space Laser Communication and Atmospheric Propagation XXVI*, Vol. 8971, 10. International Society for Optics and Photonics, SPIE, 2014, pp. 250 – 256.
[2] Israel, D. J., Edwards, B. L., and Staren, J. W., "Laser Communications Relay Demonstration (LCRD) update and the path towards optical relay operations," *2017 IEEE Aerospace Conference*, 2017, pp. 1–6.
[3] Biswas, A., Srinivasan, M., Rogalin, R., Piazzolla, S., et al., "Status of NASA's deep space optical communication technology demonstration," *2017 IEEEInternational Conference on Space Optical Systems and Applications (ICSOS)*, 2017, pp. 23–27.
[4] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., et al., "RFC 4838, Delay-Tolerant Networking Architecture," *IETF Network Working Group*, 2007.
[5] Scott, K., and Burleigh, S., "RFC 5050, Bundle Protocol Specification," *IETF Network Working Group*, 2007.
[6] Foust, J., and Foust, J., "TDRS launch marks end of an era," Aug 2017.
[7] Araniti, G., Bezirgiannidis, N., Birrane, E., et al., "Contact graph routing in DTN space networks: overview, enhancements and performance," *IEEE Communications Magazine*, Vol. 53, No. 3, 2015, pp. 38–46.
[8] Hylton, A., Raible, D., and Clark, G., "A Delay Tolerant Networking-Based Approach to a High Data Rate Architecture for Spacecraft," *2019 IEEE Aerospace Conference*, 2019.
[9] Muri, P., and McNair, J., "A performance comparison of DTN protocols for high delay optical channels," 2013, pp. 183–188.
[10] Schildt, S., Morgenroth, J., Pöttner, et al., "IBR-DTN: A lightweight, modular and highly portable Bundle Protocol implementation," *Electronic Communications of the EASST*, Vol. 37, 2011..
[11] Clare, L., Burleigh, S., and Scott, K., "Endpoint naming for space delay / Disruption Tolerant Networking," *2010 IEEE Aerospace Conference*, 2010, pp. 1–10.
[12] Joe-Paul Swinski, e. a., "bplib," https://github.com/nasa/bplib, 2018.
[13] Burleigh, S., "Interplanetary Overlay Network: An Implementation of the DTN Bundle Protocol," *2007 4th IEEE Consumer Communications and Networking Conference*, 2007, pp. 222–226.
[14] Goddard Space Flight Center, "core Flight System," https://cfs.gsfc.nasa.gov/, accessed September 9, 2019.
[15] Goddard Space Flight Center, "Integrated Test and Operations System (ITOS)," https://itos.gsfc.nasa.gov/index.php, accessed September 9, 2019.
[16] Ball Aerospace, "COSMOS," https://www.ball.com/aerospace/programs/cosmos, accessed September 9, 2019.
[17] Marshall Space Flight Center, "Telescience Resource Kit," https://trek.msfc.nasa.gov/, accessed September 9, 2019.
[18] Willman, B., Davidson, S., Pohlchuck, B., Pitts, L., and Schlesinger, A., "DTN Leads the International Space Station Payload Operation in Advanced Exploration," *NASA STI*, 2016.
[19] Seas, A., Gonnsen, Z., and Yarnall, T., "ILLUMA-T (Integrated LCRD LEO User Modem and Amplifier Terminal) Payload," 2018. URL https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20180002846.pdf.
[20] Edwards, B. L., and Israel, D. J., *Update on NASA's Laser Communications Relay Demonstration Project*, 2018.
[21] 1897, L. D. G. C. N., *MPLS Fundamentals*, Cisco Press, 2007.
[22] Rizzo, L., "Netmap," https://github.com/luigirizzo/netmap, 2012.