NASA/TM-20220011407

Developing High Performance Space Networking Capabilities for the International Space Station and Beyond

Daniel Raible, Rachel Dudukovich, Brian Tomko, Nadia Kortas, Blake LaFuente, and Dennis Iannicca
Glenn Research Center, Cleveland, Ohio

Thomas Basciano
Johnson Space Center, Houston, Texas

William Pohlchuck
The Boeing Company, Houston, Texas

Joshua Deaton
Huntsville Operations Support Center, Huntsville, Alabama

Alan Hylton
Goddard Space Flight Center, Greenbelt, Maryland

John Nowakowski
Glenn Research Center, Cleveland, Ohio

August 2022

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program plays a key part in helping NASA maintain this important role.

The NASA STI Program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI Program provides access to the NASA Technical Report Server—Registered (NTRS Reg) and NASA Technical Report Server—Public (NTRS) thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counter-part of peer-reviewed formal professional papers, but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., "quick-release" reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov

- E-mail your question to help@sti.nasa.gov

- Fax your question to the NASA STI Information Desk at 757-864-6500

- Telephone the NASA STI Information Desk at 757-864-9658

- Write to:
  NASA STI Program
  Mail Stop 148
  NASA Langley Research Center
  Hampton, VA 23681-2199

NASA/TM-20220011407

Developing High Performance Space Networking Capabilities for the International Space Station and Beyond

Daniel Raible, Rachel Dudukovich, Brian Tomko, Nadia Kortas, Blake LaFuente, and Dennis Iannicca
Glenn Research Center, Cleveland, Ohio

Thomas Basciano
Johnson Space Center, Houston, Texas

William Pohlchuck
The Boeing Company, Houston, Texas

Joshua Deaton
Huntsville Operations Support Center, Huntsville, Alabama

Alan Hylton
Goddard Space Flight Center, Greenbelt, Maryland

John Nowakowski
Glenn Research Center, Cleveland, Ohio

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

August 2022

Acknowledgments

Thank you to the NASA Space Communications and Navigation (SCaN) program for funding this work.

Level of Review: This material has been technically reviewed by technical management.

This report is available in electronic form at http://www.sti.nasa.gov/ and http://ntrs.nasa.gov/

Developing High Performance Space Networking Capabilities for the International Space Station and Beyond

Daniel Raible, Rachel Dudukovich, Brian Tomko, Nadia Kortas, Blake LaFuente, and Dennis Iannicca
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

Thomas Basciano
National Aeronautics and Space Administration
Johnson Space Center
Houston, Texas 77058

William Pohlchuck
The Boeing Company
Houston, Texas 77057

Joshua Deaton
Huntsville Operations Support Center
Huntsville, Alabama 35808

Alan Hylton
National Aeronautics and Space Administration
Goddard Space Flight Center
Greenbelt, Maryland 20771

John Nowakowski
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

***Abstract*—A performance optimized implementation of Delay Tolerant Networking (DTN) with the capacity of gigabit-per-second rates is developed for the International Space Station (ISS) and missions demanding large amounts of communications bandwidth. An overview of the High-rate Delay Tolerant Networking (HDTN) architecture and support for different convergence layers is provided. This paper then presents an overview of the testing and integration efforts to evaluate interoperability and capability in relevant environments. The first was interoperability testing with DTN Marshall Enterprise (DTNME) which resulted in near-gigabit per second data rates. This was followed by ISS emulation testing with the Software Development and Integration Laboratory (SDIL) at the Lyndon B. Johnson Space Center (JSC) and local testing based on the ISS DTN network topology. The local tests resulted in the discovery of potential sources of performance loss in the network and demonstrated near-gigabit rates between HDTN and DTNME.**

***Index Terms*—Delay tolerant networking, International Space Station, high-rate communications, interoperability**

I. Introduction

The High-rate Delay Tolerant Networking (HDTN) [1], [2], [3] project at NASA Glenn Research Center is developing a performance-optimized Delay Tolerant Networking (DTN) implementation which is able to provide reliable multi-gigabit per second automated network communications for near-Earth and deep space missions. To that end, this paper presents an overview of the testing and integration efforts leading toward future infusion of HDTN with the International Space Station (ISS).

The fundamental unit of data in a DTN is the *bundle*, which can be of essentially any size. Data rates can then be measured in bits per second, meaning "bits on the wire," but also in terms of bundles per second. HDTN is optimized for bundle per second, meaning that one does not need to carefully tune bundle sizes to reach gigabit per second rates. On top of size, bundle processing represents non-trivial overhead. This is due to the non-fixed width headers in bundles, which prohibit random access and constant time processing. Notably, there are two standards for the bundle protocol (BP), versions 6 and 7, which use two different schema for header data representation [4], [5].

The first in the series of tests was with a DTN implementation called DTN Marshall Enterprise (DTNME) [6], which was conducted using the DTN Experimental Network (DEN) [7]. This environment provided the opportunity to test the rate and connectivity of HDTN using different convergence layers as well as to determine the latency and interoperability of the software. These tests also provided a space to compare the performance of HDTN using both BP version 6 and BP version 7. HDTN was tested as both source and destination network nodes to assess the bidirectional connectivity and latency, as well as a gateway node. The expected use-case for HDTN onboard the ISS is a DTN gateway [8].

This was followed by tests using HDTN with the Software Development and Integration Laboratory (SDIL) at the Lyndon B. Johnson Space Center (JSC), which provides a realistic emulation of the network conditions on the ISS. Successful SDIL testing is a required step for any software to be deployed on the ISS. HDTN is evaluated in this environment to determine the data rate, connectivity, and interoperability between HDTN, DTNME, and the Interplanetary Overlay Network (ION) [9] implementations of the DTN protocol. Here HDTN is used as a gateway node between an ION node (space) and DTNME node (ground).

The results of these tests are analyzed, and the paper concludes with a discussion of future work for the ISS and extensions to other missions.

II. ISS as an Intra and Inter-Network

The International Space Station (ISS) exists as a very large intra-networking platform between all of its various avionics subsystems to keep the spacecraft operational (thermal, attitude control, power, etc.) As our national orbital laboratory,

the ISS is host to a multitude of onboard experiment payloads, connected together through a Joint Station LAN (JSL), presenting another intra-networking scenario of moving scientific data and command instructions between the payloads and the radios. Functioning as a system, the ISS can return data to the ground using several paths at different frequencies, including directly to Earth or primarily relaying through the Tracking and Data Relay Satellite (TDRS) constellation and down to one of two ground terminal locations. Finally, scientific data is routed across a multi-node terrestrial network, through the ISS Payload Operations and Integration Center (POIC) and ultimately delivered to a payload science team. The complete space communications system offers multiple paths to move data between the various client payloads and their corresponding science teams, through TDRS and across the JSL. In this way it is a large multiple-in-multiple-out (MIMO) system whose traffic must be carefully managed to successfully route data to the correct destination at given quality of service (QoS) levels.

The communications capabilities for the ISS have been upgraded several times, and most notably the Ku-band enhancements which enabled several hundred Mbps return-link services. The extension of the ISS to at least 2030 will see additional payload opportunities and evolutions to the infrastructure. The upcoming Integrated Laser Communications Relay Demonstration (LCRD) LEO User Modem and Amplifier Terminal (ILLUMA-T) will soon be installed on the ISS, where it will serve as a low-Earth orbit 'customer' terminal to operate in conjunction with the LCRD mission. This payload will demonstrate high-rate laser-based communications at several data rates up to 1 Gbps through the LCRD spacecraft in geostationary orbit and down to two new optical ground terminals. The ISS internal infrastructure was upgraded to operate at rates on the order of 1000 Mbps in order to improve internal communications and to better utilize the bandwidth which ILLUMA-T and LCRD will provide. This capability adds to the MIMO architecture of the ISS, which will require additional network storage and management to utilize its full capacity.

The HDTN project has identified several technological challenges to address the needed performance upgrades to support laser communications on the ISS while considering the existing RF infrastructure. The goal of HDTN is to provide a high speed path for moving networked data between spacecraft payloads and communication systems which operate across a range of disparate and asymmetric rates, so the enhanced ISS environment presents a relevant opportunity to evaluate HDTN. Data services span a wide range of priorities, rates and sizes including software updates, large file transfers, streaming multimedia, science data return, etc., and the networked nature of the overall system demands interoperability between each of the nodes within the system. In particular, this means that the network traffic (i.e., bundles) can be expected to have a wide range of characteristics, especially size. Network conditions are deleteriously affected by the orbit of the ISS, as well as TDRS coverage and inherent link latencies. Lastly the frequent disconnections of the communication links requires local storage at the nodes, to later forward when links become active. The particulars of this environment have been studied and modeled for the ISS use-case, and utilized as a test bed to develop and evaluate HDTN as described in the following sections.

III. HDTN Overview

Figure 1 shows HDTN modules and their interactions. The Ingress module intakes bundles generated by the BPGen tool and decodes the header fields to determine the source and destination of the bundles. It receives `linkUp` or `linkDown` events from the Scheduler module to determine if a given bundle should be forwarded immediately to Egress or should be sent to storage. To determine a given link availability, the scheduler reads a contact plan which is a JavaScript Object Notation (JSON) file that defines all the connections between all the nodes in the network. If the link is available, Ingress will send the bundles in a cut-through mode straight out to Egress and if the link is down or custody transfer is enabled then it will send the bundles to the storage module. The storage is always required when custody transfer is enabled even if an immediate forwarding opportunity exists because the bundle layer must be prepared to re-transmit the bundle if it does not receive acknowledgment within the time-to-acknowledge of the bundle that the subsequent custodian has received and accepted the bundle. The storage is a multi-threaded implementation distributed across multiple disks and custody transfer is also handled there. It receives messages from the Scheduler to determine when stored bundles can be released and forwarded to Egress.

The Contact Graph Routing (CGR) [10] client calculates the optimal route and the next hop for a given contact plan and final destination using PyCGR [11] which is a Python implementation of the Contact Graph Routing algorithm. The Router module communicates with the CGR client to get the next hop for the optimal route leading to the final destination, then sends a `RouteUpdate` event to Egress to update its outduct to the outduct of the next hop. The Egress module is responsible for forwarding the bundles received from Storage or Ingress to the right outduct and next hop determined based on the optimal route computed by CGR.

HDTN uses an event driven approach based on ZeroMQ pub-sub sockets [12] for sending updates for unexpected link and contact plan changes from Egress to Scheduler. When the connection is lost unexpectedly, Egress will send a `LinkStatus` change message to the Scheduler which will trigger the Scheduler to send `linkUp` or `linkDown` events to Ingress and Storage. In addition, Scheduler will recompute the contact plan and send the message `ContactsUpdate` to the Router for the PyCGR client to recompute the optimal route based on the new contact plan and send a `RouteUpdate` message to Egress.

BPSink tool validates the bundles received from Egress. The Web Interface currently displays data rates graph and statistics for network troubleshooting.

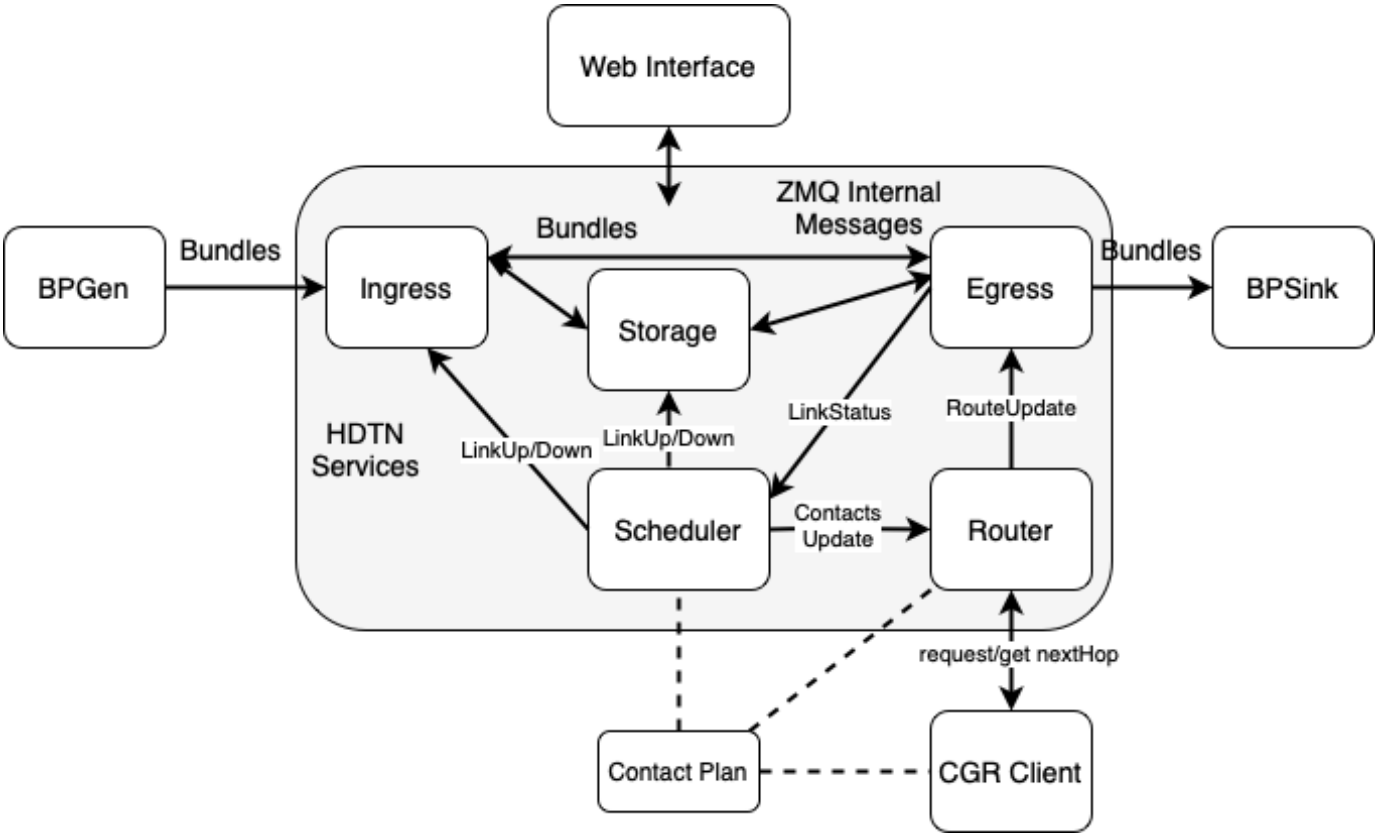

Fig. 1. HDTN modules interaction

IV. Implementation

A. Convergence Layers

HDTN supports several different convergence layers. All of these convergence layers are written in the `Boost.Asio` C++ library [13] using non-blocking, asynchronous patterns for sockets, resolvers, and timers. Using `Boost.Asio` ensures the convergence layers are cross-platform, and the demultiplexing automatically occurs on the selected platform; this means the operating system "select" function is used for Linux Kernel 2.4 and VxWorks, the "epoll" function is used for Linux Kernel 2.6, the "kqueue" function is used for Mac OS X and FreeBSD, and overlapped I/O and I/O completion ports (IOCP) is used for Windows NT. All HDTN convergence layers deliver received bundles to the HDTN Ingress with extra bytes of padding before and after the contiguous memory of the received bundle. This allows HDTN to avoid memory allocations/copies during bundle manipulation, particularly with BP version 7 (e.g. prepending a new previous node canonical block) [5]. There is an HDTN compile option for x86 hardware accelerated branchless Self-Delimiting Numeric Values (SDNV) [14] encoding and decoding. This is useful for Transmission Control Protocol Convergence Layer (TCPCL) version 3 [15] and Licklider Transmission Protocol (LTP) [16] which use SDNV's. These functions use special x86 instruction sets including SSE, SSE2, SSSE3, SSE4.1, BMI, BMI2, AVX, and AVX2.

The UDP convergence layer [17] is the simplest of the convergence layers. However, its main drawback is that the bundle size must be limited to the max size of a UDP datagram, which is less than 1500 bytes for small Ethernet frames. Also, another drawback is that it is unreliable unless BP version 6 [4] custody transfer is enabled. The Simplified TCP (STCP) [18] convergence layer is also minimally complex like UDP. The STCP convergence layer specification is not documented in an RFC but rather is documented in the description section of the ION user manual [19] for STCPCLI which states, "Each bundle received on the connection is preceded by a 32-bit unsigned integer in network byte order indicating the length of the bundle." The benefits of STCP is that due to its minimal complexity, testing shows it is the fastest of the convergence layers, assuming TCP is using hardware acceleration. The drawback of STCP is that it is unidirectional, meaning that two TCP sockets (i.e. an induct and outduct pair) must be used for bidirectional communication such as in the case of having BP version 6 custody transfer enabled.

HDTN supports the TCP convergence layer (TCPCL) version 3 (RFC 7242) [15] and version 4 (RFC 9174) [20]. Both versions of TCPCL are bidirectional over one TCP socket, meaning that administrative records such as custody signals are receivable on the same TCPCL link. The TCPCL Request for Comments (RFC) essentially defines the TCP connection initiator as the "active entity" and defines the TCP connection

listener as the "passive entity." Unlike the other convergence layers, in which bundle communication is constrained to an active entity unidirectionally talking to a passive entity only, the HDTN TCPCL implementations also allow:

1) Two active entities talking to each other through HDTN on one TCP connection,
2) Two passive entities talking to each other through HDTN on one TCP connection,
3) A passive entity can send bundles to an active entity through HDTN on one TCP connection, and
4) An active entity can send pings and receive echoes on one TCP connection.

The HDTN TCPCL is implemented as a finite state machine with callback functions, so any number of bytes can be read and processed from the TCP stream as they arrive without the need to wait for an entire TCPCL data segment or an entire bundle. The HDTN TCPCL version 4 can optionally be compiled with or without Open Secure Sockets Layer (OpenSSL) [21] support. The implementation supports Transport Layer Security (TLS) versions 1.2 & 1.3 only. The OpenSSL support uses `Boost.Asio` TCP socket stream wrappers and still uses an asynchronous design. Testing shows that the maximum bundle rate is approximately half of Non-SSL, assuming OpenSSL is compiled with x86 hardware acceleration enabled.

HDTN supports the Licklider Transmission Protocol (LTP) convergence layer (RFC 5326). The HDTN LTP implementation is capable of supporting tens of thousands of simultaneous sessions in order to achieve gigabit rates for link delays on the order of several seconds. HDTN LTP packet read operations are implemented as a finite state machine with callback functions. The entire UDP packet can be fed in at once to the state machine, resulting in optimal performance, or a trickle of bytes can be fed into the state machine as they arrive, which is useful for other potential non-UDP protocols. The HDTN LTP implementation uses a hardware random number generator for LTP serial numbers and session numbers from the "exclusive or" of 3 sources of randomness:

1) the x86 RDSEED instruction, which is optionally enabled at compile time,
2) the `Boost::random_device` library which gets randomness from operating system sources such as `/dev/urandom`, and
3) the current time in microseconds.

An incremental part, or reserved section of bytes, of the generated number is used to prevent a "birthday paradox" or duplicate random number. The HDTN LTP implementation uses two threads in a link/connection. Thread 1 places received UDP datagrams on a circular buffer and alerts Thread 2 of data availability through `boost::asio::post`. Thread 2 handles the `boost::asio::io_service::run` thread for the LTP timers (implemented as `boost::asio::deadline_timer`) and the LTP packet processing of all sessions of this LTP Engine. Limited (pseudo) flow control is achieved by delaying the return of the `RedPartReceptionCallback` function, which is called when the bundle has been fully delivered over reliable LTP red data transfer, which delays the send of the final report segment needed to close the session, until the bundle is first either fully queued to the next hop or is fully stored on disk.

B. Applications

Since the HDTN software does not currently support a CCSDS File Delivery Protocol (CFDP) file transfer [22], a file transfer utility was implemented to break files into bundles (BPSendFile) and to reassemble them at the destination node (BPReceiveFile). These tools are discussed in further detail [23], which describes how they were developed and used in an aeronautical flight test.

The Bundle Protocol Ping (BPing) utility sends bundles between 2 nodes using DTN Bundle Ping. This tool tests for end-to-end connectivity and latency between DTN nodes. HDTN also supports an echo service that is used in conjunction with BPing.

V. Performance and Compatibility Testing

The HDTN team has been performing a series of interoperability tests to enable the integration of HDTN into the ISS DTN network. These tests can be broken into three categories: HDTN-DTNME interoperability over the DEN, testing within the SDIL's emulation of the ISS DTN network, and local stand alone testing within the HDTN lab at GRC. This section provides an overview of the testing within each scenario that has been done to date.

A. DTNME Interoperability Testing

Fundamental interoperability testing to establish connectivity and measure baseline data rates was completed between HDTN and Marshall Space Flight Center Delay Tolerant Network (MSFC) Delay Tolerant Networking Marshall Enterprise (DTNME) implementations. Testing was conducted virtually between Glenn Research Center (GRC) and MSFC over the DTN Experimental Network (DEN). Figure 2 shows the network topology.

The Internet Control Message Protocol (ICMP) Ping was used to measure the latency between the GRC and MSFC DEN servers. The latency was around 21 ms without introducing any delay. Next, bidirectional Bundle Protocol Ping was tested for different configurations and convergence layers using both BP version 6 and BP version 7. We also added a delay of 200 ms and 700 ms using the Linux kernel `netimpair` command line tool [24] to simulate poor network conditions and check how added delays impact the latency associated with the different convergence layers. The connectivity tests results are summarized in Table I for BP version 6 and in Table II for BP version 7.

The connectivity testing results show that the recorded round-trip latency was consistent with the standard ping time with and without delay for all convergence layers, as well as with an added hop when HDTN served as a gateway.

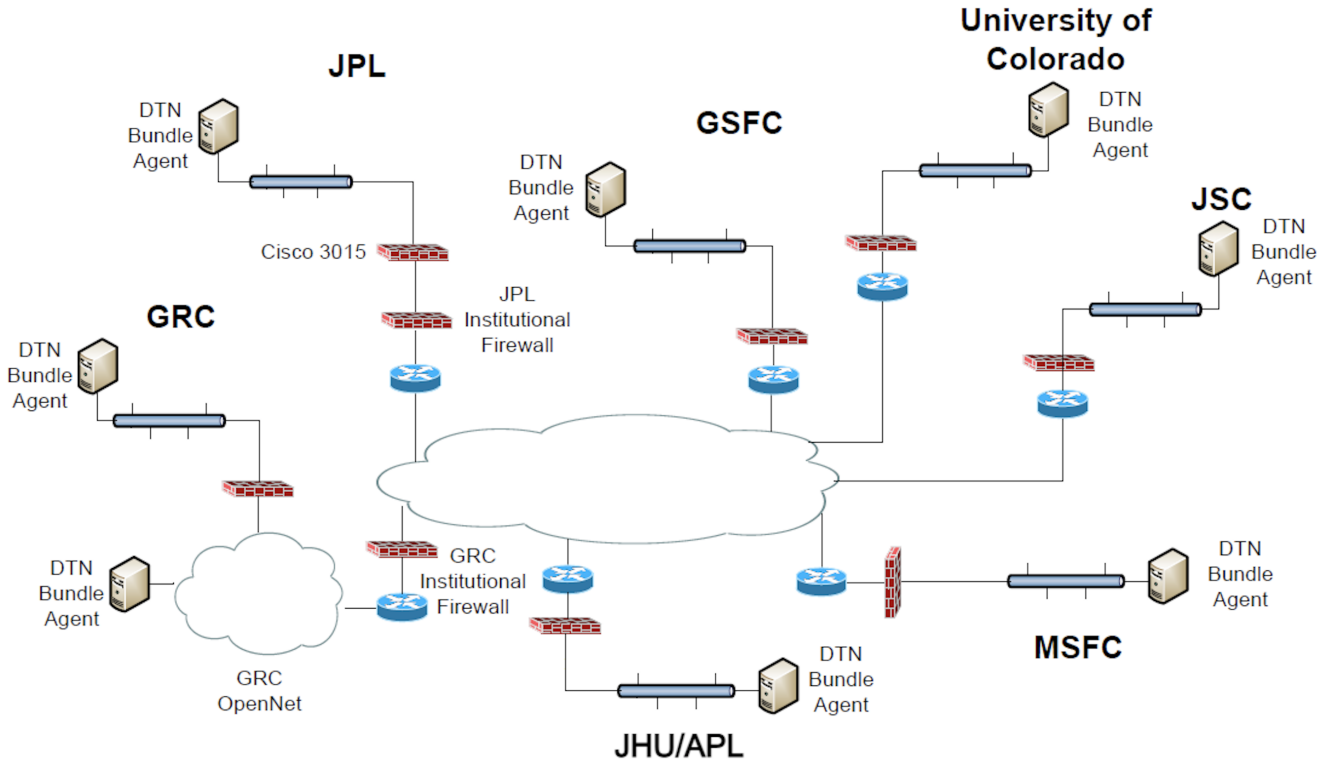

Fig. 2. Multi-Center DTN Testbed topology

Convergence Layer	Emulated Delay (ms)	Latency (ms) for DTNME to HDTN
LTP	0	22 to 31
	200	222 to 230
	700	723 to 732
TCPCLv3	0	21
	200	222
	700	1100 to 1800
STCP	0	21
	200	222
	700	722
UDP	0	21
	200	222
	700	722

TABLE I
RESULTS OF THE CONNECTIVITY TESTING USING BPV6

Convergence Layer	Emulated Delay (ms)	Latency (ms) for DTNME to HDTN
LTP	0	24 to 33
	200	223 to 231
	700	724 to 732
TCPCLv3	0	21
	200	222
	700	1277 to 1962
TCPCLv4	0	21 to 81
	200	221
	700	922 to 2124
TCPCLv4/TLS	0	21 to 81
	200	222
	700	896 to 2139
STCP	0	21
	200	221
	700	722
UDP	0	21
	200	221
	700	722

TABLE II
RESULTS OF THE CONNECTIVITY TESTING USING BPV7

In addition, we verified that after adding 25% loss, the re-transmission of lost data occurred correctly for all of the test cases. Finally, the performance of BP version 7 was comparable to BP version 6. This test showed there was no delay introduced from additional overhead or processing time due to the version 7 update.

The next test conducted was a data rate benchmark of the LTP convergence layer. Bundles were sent as fast as possible from HDTN to DTNME using a bundle size of 100 KB and LTP maximum transmission unit of 1360 bytes. The results are summarized in Table III. In this case, it was found that the DTNME server had only one core available, and that core

Emulated Delay (s)	Data rate (Mbps) with max 400 LTP sessions	Data rate (Mbps) with max 5000 LTP sessions
0	500 (+/- 100)	600
2	400 (+/- 100)	NA
4	400 (+/- 100)	NA

TABLE III
RESULTS OF THE DATA RATES TESTING FOR LTP CONVERGENCE LAYER USING BPV6

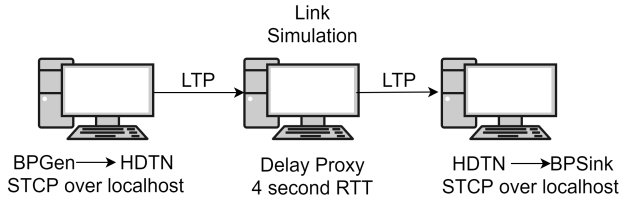

Fig. 3. LTP Long Delay Test Configuration

was fully utilized. This likely constrained the rates to lower than what could be achieved with additional cores.

Several LTP parameters were tuned to achieve the rates shown in Table III. The `ltpDataSegmentMtu` was set to 1,360 bytes to make sure UDP packets stayed under the Ethernet maximum transmission unit (MTU) of 1,500 bytes to prevent fragmentation. The `zmqMaxMessagesPerPath` corresponds to the maximum number of sessions and was set to either 400 or 5000 LTP sessions for this data rates test case. The `bundlePipelineLimit` was set to 6000 to make sure it can handle up to 5000 sessions. The `numRxCircularBufferElements` on the outduct corresponds to the number of report segments HDTN can receive and that was set to 10000 to handle burstiness of report segments.

The `oneWayLightTimeMs` on the outduct corresponds to signal propagation delay at the speed of light in the outbound direction and that was set to 2000 ms (2 seconds delay). We set `ltpMaxRetriesPerSerialNumber` to 500 retries to make sure the Egress will keep bundles for 30 minutes of disconnection time and prevent bundles from being dropped. This was calculated as in Formula 1, in milliseconds, where T_{md} is the maximum disconnection time, T_{OWLT} is the `oneWayLightTimeMs`, T_{OWM} is the `oneWayMarginTimeMs`, and LTP_{MR} is the `ltpMaxRetriesPerSerialNumber`.

$$T_{md} = 2 \times (T_{OWLT} + T_{OWM}) \times LTP_{MR} \quad (1)$$

This gave the maximum time of disconnection for a bundle before it is dropped as:

$$(4,000 \text{ ms roundtrip time}) \times (500 \text{ retries}) = 2,000 \text{ s}.$$

B. LTP Long Delay Testing

LTP is a space-related protocol specifically designed for long delays. Current estimates for delays from the ISS through NASA's Integrated LCRD Low-Earth Orbit User Modem and Amplifier Terminal (ILLUMA-T) and the Laser Communications Relay Demonstration (LCRD) are about 2 seconds one-way, or 4 seconds maximum round trip time. Initially the performance of LTP with these longer delays was poor. The HDTN LTP implementation was tested and tuned for longer delays to address this issue. Figure 3 shows the test configuration. The HDTN BPGen tool generates bundles which are transmitted to HDTN over localhost using STCP. STCP is used as the connection between local inducts and outducts due to its efficiency. The HDTN outduct transmits BP version 6 bundles over LTP to the delay proxy. The delay proxy emulates a 4 second round trip time (2 seconds each way). It resides on another computer so that its processing and resource consumption does not impact the tests results. On another computer, HDTN listens using an LTP induct to receive bundles. The bundles are then transmitted to the HDTN BPSink tool using STCP over localhost. The BPSink tool reports the final data rate results.

Several observations were noted in these longer delay tests. The first was that tools such as `netem` and Linux Traffic Control (TC) [25] were useful for emulating packet loss but did not perform well for delays greater than 1 second. This is possibly due to large amounts of data being queued. To solve this, HDTN now has a custom delay proxy tool that can be used to emulate longer delays. The tool was implemented using the `Boost.Asio` library. The CPU usage for the tool is low but it can require significant amounts of memory. The circular buffer memory usage is approximately the product of the data rate and the one-way delay.

The results of the LTP long delay tests are shown in Table IV. The table summarizes the overall data rate, report segment data rate, percentage of processor utilization for the sender and receiver node, the percentage of lost data segments, the number of bundles transmitted per second, the number of data segment UDP packets per second, and the number of simultaneous LTP sessions required to reach the these rates for several sizes of bundles. In addition to the LTP parameter tuning discussed in Section V-A, it was found that increasing the number of simultaneous LTP sessions was key to achieving gigabit rates in links with greater than one second delays.

C. SDIL Emulated Network

The GRC HDTN lab is connected to the SDIL lab through a secure firewall over the NASA VPN. The SDIL uses a Wide Area Network (WAN) emulator to introduce losses on both the uplink and downlink as well as a fixed transmission delay (600 ms) to recreate network conditions on the ISS.

Figure 4 shows the topology of the network between GRC and the SDIL at JSC. The SDIL emulates an ISS payload node that transmits bundles using the Telescience Resource Kit (TReK) [26] and/or ION to an onboard DTN gateway emulated in the GRC HDTN lab. The HDTN gateway forwards bundles to a ground DTNME gateway which represents the Huntsville Operations Support Center (HOSC). The ground gateway then delivers bundles to the ground user node, which is currently

Bundle Size (KB)	Raw Rate (Mbps)	RptSeg Rate (Mbps)	Sender CPU %	Receiver CPU %	Loss %	Bundles/sec	DS UDP Pack/sec	LTP Sessions
100	998	0.39	100	70	0	1250	92000	5000
50	999	0.85	105	92	0.047	2500	103000	10000
10	994	3.68	150	140	0.267	12000	112000	48000
5	824	12	200	200	70	21000	143000	84000

TABLE IV
DTN TO HDTN OVER LTP SENDING VERSION 6 BUNDLES, 4 SECOND ROUND TRIP TIME, 1360 BYTE LTP DATA SEGMENT SIZE.

anticipated to use TReK and ION. The payload to onboard gateway connection uses STCP, the ISS to ground connection uses LTP, and the ground gateway to ground user node uses TCPCL. Currently, only BP version 6 has been tested in this configuration.

HDTN has completed two initial tests with the SDIL. The first test established one-way communication from the ISS payload node to the ground user node. The next test, conducted several months later, established full bi-directional communication. This was done using BPing from the ION payload node to the DTNME ground gateway via the HDTN gateway, demonstrating interoperability between ION, HDTN, and DTNME. In addition, large file transfer from the ION payload (Space) to the Ground (Downlink) was tested and we were able to successfully send a 1GB file with a data rate of up to 10 MBits/s. This data rate was much lower than expected and several factors were suspected as contributing to the poor performance. The VPN connection between GRC and JSC was one possible issue. For the uplink, we were able to only send a small file from Space to Ground. We are suspecting that the root cause is that DTNME version on the ground is an older version and does not include the capability to use LTP to connect to multiple remote clients. That capability is in DTNME 1.0.1 and later but not in DTNME 0.1.1 or earlier. In addition, the SDIL and GRC labs use virtual machines to host the DTN nodes.

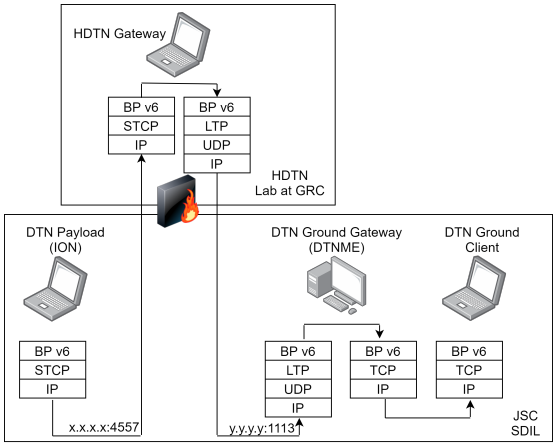

Fig. 4. NASA GRC to JSC SDIL Network

D. HDTN Local Testing

In order to determine the cause of the low data rates seen in the initial SDIL testing, a series of tests were conducted locally at GRC. This would eliminate the VPN connection between JSC and GRC as a possible bottleneck in the network. The tests were conducted using a ”4-box” configuration that is representative of a single ISS payload, connected to the ISS onboard gateway, the HOSC DTN ground gateway, and a ground node. The onboard link between the payload and ISS gateway uses STCP, the link from the ISS to ground uses LTP, and the terrestrial link from the ground gateway to the ground node uses TCPCL. This set of convergence layers was used throughout the local testing. Several environments were used for the tests: fully virtualized, native hardware-based using computers in the HDTN lab, and a combination of both. The hardware-based test and the combination test both use an HP ZBook laptop that is an equivalent model to what is used on the ISS for the onboard gateway. In addition, the test using both virtual and hardware-based environments used a virtual machine image that is the same as what is used on the ISS. The next sections will discuss each test in detail.

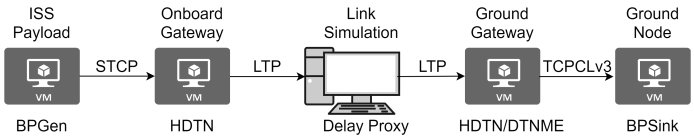

Fig. 5. Virtualized Configuration for Local Testing

1) Virtualized Test: Figure 5 shows the configuration for the fully virtualized test. This test was conducted using a Powerworks-174026 server. The server has 256 x AMD EPYC 7H12 64-Core Processors (2 Sockets) and 1.48 TiB RAM. It hosts several virtual machines using Kernel-based Virtual Machines (KVM) as well as Linux containers (LXC). The virtual machines used for the initial local testing were based on Ubuntu 20.04.4 for the onboard gateway and ground gateway and Oracle Linux 8 for the ISS payload and ground node. Each virtual machine has access to 16 cores based on the host architecture and 32 GB RAM. The baseline network rates between two of the virtual machines were measured using iPerf3 [27]. This showed the system reaching 1.5 Gbps using UDP and 16.4 Gbps using TCP.

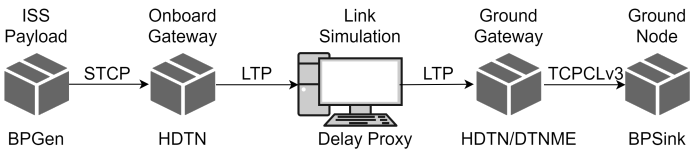

Fig. 6. Linux Container Configuration for Local Testing

2) Linux Containers Test: The next testing configuration was done using Linux containers (LXC) as shown in Figure 6.

The containers were hosted on the same virtualization server as described in section V-D1. Each container has access to 8 cores and 32 GB of RAM. The containers share the same operating system as the host, which is Debian 11 (Bullseye). The Linux containers are virtualized at the operating system level and share the same kernel. The KVM virtual machines are virtualized at the hardware level and each have their own kernel. The Linux containers are lightweight, have faster start up times, and are generally more efficient. In our use-case the main advantage of the KVM virtual machines is that they are able to run operating systems that are compatible with the different software components, for example TReK recommends Red Hat Enterprise Linux 7.x, whereas most the HDTN laboratory commonly uses Debian and Ubuntu.

3) Native Hardware Test: In order to determine any potential performance loss caused by virtualization, the same topology and convergence layer configuration was tested with Linux running on bare metal. The ISS onboard payload, ground gateway, and ground node were running on lab servers with Debian 10 operating system, 8 x Intel Core i7-7700K CPU @ 4.2 GHZ, and 32 GB RAM. The ISS DTN onboard gateway was running on an HP ZBook 15 G4 with 8 x Intel Core i7-7700HQ CPU @ 2.8 GHZ and 32 GB RAM, also using Debian 10. This model of laptop is the same that is currently used to host the operational DTN gateway on ISS at the time of this paper. Figure 7 shows the network for this test.

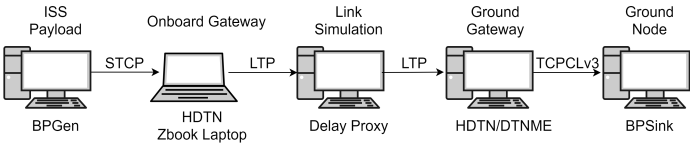

Fig. 7. Native Hardware Configuration for Local Testing

4) ISS Onboard Gateway Image Test: Finally, the same hardware from section V-D3 was used but the HP ZBook hosted a KVM image which is used for the onboard DTN gateway on ISS. This combination of the ZBook laptop and VM image is representative of the actual operating environment expected for HDTN on ISS. The KVM image is configured to have 2 cores based on the host architecture and 1600 MB RAM. This is a much more constrained environment than the other tests conducted, however it is the most realistic case. The ISS must conserve resources on each machine as much as possible and hosts several KVM guests on a single ZBook laptop. Figure 8 shows this test configuration.

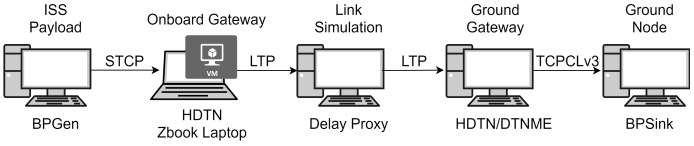

Fig. 8. Configuration with ISS DTN Gateway Image for Local Testing, now with ZBook KVM image

Application	KVM	LXC	Lab Servers
Iperf3 UDP	1.5 Gbps	1.5 Gbps	945 Mbps
Iperf3 TCP	16.4 Gbps	18.4 Gbps	661 Mbps
HDTN	260 Mbps	~900 Mbps	930 Mbps
DTNME	~240 Mbps	~430 Mbps	930 Mbps

TABLE V
RESULTS OF THE HDTN/DTNME GROUND GATEWAY INITIAL TEST

5) Local Testing Results: Table V summarizes the results from sections V-D1 to V-D4. Each test was conducted with HDTN as the ground gateway as well as DTNME. This was done since we are less familiar with DTNME configuration, so HDTN was first configured as a sanity check on the test setup. The configurations for HDTN as a gateway node and HDTN as a ground node are shown in Figures 9 and 10 respectively in the appendix. In the SDIL and the real ISS network, it is expected that DTNME will be the ground gateway. While iperf3 shows that each configuration should be able to reach 900 Mbps or greater using UDP there were significant performance losses for both HDTN and DTNME using the KVM virtual machines. We believe the UDP benchmark is representative of the LTP rates HDTN should be able to achieve since the current LTP implementation uses UDP as the transport layer. The Linux containers were more efficient and were closer to the performance achieved using native hardware. The end-to-end rates measured by BPSink show that both HDTN and DTNME are able to achieve line rates when using the native hardware.

While the native hardware test was quite successful, we saw a significant performance loss when the ISS gateway image was incorporated on the ZBook laptop. Intuitively, the rates were expected to be lower since the resources allotted to the virtual machine are much less than what is available on the laptop (2 cores and 1600 MB RAM versus 8 cores and 32 GB RAM). The virtual machine image was reconfigured several times with more cores and memory allocated to it, but it did not seem to make a difference in the rates. There are several possible explanations for this. One possibility is that there was an issue changing the configuration using the virt-manager tool [28]. Another potential issue could be the type of network interface virtualization that is being used, however the iperf3 results do not seem to indicate this. Finally, it is possible that there is some issue inherent in the KVM virtualization itself. We have begun exploring the possibility of changing to a different hypervisor such as VMware vSphere Hypervisor.

The tests in this section were completed with a zero second delay configured in the UDP delay proxy. This was done to determine the baseline rates for each configuration. Since the native hardware configuration was successful at achieving over 900 Mbps for both HDTN and DTNME, we incorporated a 4 second round trip time (2 seconds each way) into the native hardware test set up using HDTN as the ISS onboard gateway and DTNME as the ground gateway. This test was able to reach an average of 940 Mbps with similar LTP parameter tuning as discussed in section V-B. This demonstrates inter-

operability of HDTN and DTNME in a similar topology to the ISS DTN network with a realistic delay for ILLUMA-T and LCRD.

VI. Future Work

The initial deployment of HDTN on the ISS is targeting the existing computing capabilities offered by the existing in-cabin laptop systems. In the future, should upgrades facilitate higher performing systems such as server-class computing with extended storage for buffering, then the HDTN team could potentially target those platforms to extend the capability on orbit.

There are additional local tests which will be conducted to more closely match the configuration of the SDIL. The next step for interoperability will be to incorporate TReK and ION on the ISS payload node and ground node. The ISS frequently uses TReK and ION to send files using CFDP. Our current tests did not use CFDP but rather the custom HDTN file transfer tools. We anticipate that this will cause a change in performance. Another local test will be conducted to determine the cause of the poor performance using KVM. The hypervisor on the ZBook laptop will be changed from KVM to another tool, such as those offered by VMware. The local virtualization testing was significant since it uncovered an issue which may have also impacted our SDIL testing. A KVM hypervisor was also used for the HDTN gateway during the last round of SDIL testing. We believed at that time that the low rates might have been due to the VPN between GRC and JSC, or possibly the version of DNTME used on the ground gateway. Our local testing has revealed KVM itself may impact performance more significantly than originally anticipated.

In addition to pursuing further testing with the SDIL, HDTN is currently performing a series of convergence layer benchmark tests using the Goddard Space Flight Center Mission Cloud Platform (MCP). MCP provides access to Amazon Web Services (AWS) in a managed environment for NASA. Many aspects of the Near Earth Network are expected to utilize cloud and container-based services due to the low cost, flexible computing resources, distributed locations, ease of deployment, high levels of redundancy, and resource availability. For these reasons, HDTN has been evaluating the network and compute performance of AWS Elastic Compute Cloud. The results will be published in a future paper.

Finally, the HDTN team has been experimenting with implementations on ARM architectures, both considering multiple-core processors as well as distributed HDTN implementations across multiple processors. The results of these parametric ARM tests will be released in a forthcoming publication, and utilize similar baseline testing techniques as presented in this paper. The evaluation of HDTN across a range of processing platforms will create a portfolio of implementation solutions spanning different mission classes and ground infrastructure needs.

VII. Special Thanks

The authors would like to thank the NASA Space Communications and Navigation (SCaN) program, and in particular Philip Baldwin (NASA/HQ) for supporting this work and Penny Roberts (NASA/JSC) for her guidance and integration opportunity.

Appendix

```
{
    "hdtnConfigName": "my hdtn config",
    "userInterfaceOn": "true",
    "mySchemeName": "unused_scheme_name",
    "myNodeId": 10,
    "myBpEchoServiceId": 2047,
    "myCustodialSsp": "unused_custodial_ssp",
    "myCustodialServiceId": 0,
    "isAcsAware": true,
    "acsMaxFillsPerAcsPacket": 100,
    "acsSendPeriodMilliseconds": 1000,
    "retransmitBundleAfterNoCustodySignalMilliseconds": 10000,
    "maxBundleSizeBytes": 10000000,
    "maxIngressBundleWaitOnEgressMilliseconds": 2000,
    "maxLtpReceiveUdpPacketSizeBytes": 1500,
    "zmqIngressAddress": "localhost",
    "zmqEgressAddress": "localhost",
    "zmqStorageAddress": "localhost",
    "zmqRegistrationServerAddress": "localhost",
    "zmqSchedulerAddress": "localhost",
    "zmqRouterAddress": "localhost",
    "zmqContactGraphRoutingAddress": "tcp://localhost:5556",
    "zmqBoundIngressToConnectingEgressPortPath": 10100,
    "zmqConnectingEgressToBoundIngressPortPath": 10160,
    "zmqConnectingEgressBundlesOnlyToBoundIngressPortPath": 10161,
    "zmqBoundIngressToConnectingStoragePortPath": 10110,
    "zmqConnectingStorageToBoundIngressPortPath": 10150,
    "zmqConnectingStorageToBoundEgressPortPath": 10120,
    "zmqBoundEgressToConnectingStoragePortPath": 10130,
    "zmqRegistrationServerPortPath": 10140,
    "zmqBoundSchedulerPubSubPortPath": 10200,
    "zmqBoundRouterPubSubPortPath": 10210,
    "zmqMaxMessagesPerPath": 5000,
    "zmqMaxMessageSizeBytes": 100000000,
```

```
"inductsConfig": {
    "inductConfigName": "myconfig",
    "inductVector": [
        {
            "name" : "stcp_ingress",
            "convergenceLayer": "stcp",
            "myEndpointId": "ipn:10.1",
            "boundPort":4556,
            "numRxCircularBufferElements":200,
            "keepAliveIntervalSeconds":15
        }
    ]
},
"outductsConfig": {
    "outductConfigName": "myconfig",
    "outductVector": [
        {
            "name": "for egress DTNME",
            "convergenceLayer": "ltp_over_udp",
            "nextHopEndpointId": "ipn:32.0",
            "remoteHostname": "Delay Proxy Node",
            "remotePort": 1113,
            "bundlePipelineLimit": 6000,
            "finalDestinationEidUris": [
                "ipn:2.1",
                "ipn:32.2047",
                "ipn:32.0"
            ],
            "thisLtpEngineId": 10,
            "remoteLtpEngineId": 32,
            "ltpDataSegmentMtu": 1360,
            "oneWayLightTimeMs": 2000,
            "oneWayMarginTimeMs": 200,
            "clientServiceId": 1,
            "numRxCircularBufferElements": 10000,
            "ltpMaxRetriesPerSerialNumber": 500,
            "ltpCheckpointEveryNthDataSegment": 0,
            "ltpRandomNumberSizeBits": 32,
            "ltpSenderBoundPort": 1113,
            "ltpMaxSendRateBitsPerSecOrZeroToDisable": 0
        }
```


Fig. 9. HDTN Gateway node configuration

```
{
    "hdtnConfigName": "my hdtn config",
    "userInterfaceOn": "true",
    "mySchemeName": "unused_scheme_name",
    "myNodeId": 32,
    "myBpEchoServiceId": 2047,
    "myCustodialSsp": "unused_custodial_ssp",
    "myCustodialServiceId": 0,
    "isAcsAware": true,
    "acsMaxFillsPerAcsPacket": 100,
    "acsSendPeriodMilliseconds": 1000,
    "retransmitBundleAfterNoCustodySignalMilliseconds": 10000,
    "maxBundleSizeBytes": 10000000,
    "maxIngressBundleWaitOnEgressMilliseconds": 2000,
    "maxLtpReceiveUdpPacketSizeBytes": 1500,
    "zmqIngressAddress": "localhost",
    "zmqEgressAddress": "localhost",
    "zmqStorageAddress": "localhost",
    "zmqRegistrationServerAddress": "localhost",
    "zmqSchedulerAddress": "localhost",
    "zmqRouterAddress": "localhost",
    "zmqContactGraphRoutingAddress": "tcp://localhost:5556",
    "zmqBoundIngressToConnectingEgressPortPath": 10100,
    "zmqConnectingEgressToBoundIngressPortPath": 10160,
    "zmqConnectingEgressBundlesOnlyToBoundIngressPortPath": 10161,
    "zmqBoundIngressToConnectingStoragePortPath": 10110,
    "zmqConnectingStorageToBoundIngressPortPath": 10150,
    "zmqConnectingStorageToBoundEgressPortPath": 10120,
    "zmqBoundEgressToConnectingStoragePortPath": 10130,
    "zmqRegistrationServerPortPath": 10140,
    "zmqBoundSchedulerPubSubPortPath": 10200,
    "zmqBoundRouterPubSubPortPath": 10210,
    "zmqMaxMessagesPerPath": 5000,
    "zmqMaxMessageSizeBytes": 100000000,
```

```
"inductsConfig": {
    "inductConfigName": "myconfig",
    "inductVector": [
        {
            "name" : "i1 from DTNME",
            "convergenceLayer": "ltp_over_udp",
            "myEndpointId": "ipn:32.0",
            "boundPort": 4556,
            "numRxCircularBufferElements": 10000,
            "thisLtpEngineId": 32,
            "remoteLtpEngineId": 10,
            "ltpReportSegmentMtu": 1360,
            "oneWayLightTimeMs": 200,
            "oneWayMarginTimeMs": 200,
            "clientServiceId": 1,
            "preallocatedRedDataBytes": 200000,
            "ltpMaxRetriesPerSerialNumber": 50,
            "ltpRandomNumberSizeBits": 32,
            "ltpRemoteUdpHostname": "Delay Proxy Node",
            "ltpRemoteUdpPort": 4556,
            "ltpRxDataSegmentSessionNumberRecreationPreventerHistorySize": 0,
            "ltpMaxExpectedSimultaneousSessions": 6000
        }
    ]
},
"outductsConfig": {
    "outductConfigName": "myconfig",
    "outductVector": [
        {
            "name": "bpsink",
            "convergenceLayer": "tcpcl_v3",
            "nextHopEndpointId": "ipn:2.1",
            "remoteHostname": "BPSink Node",
            "remotePort": 4558,
            "bundlePipelineLimit": 5,
            "finalDestinationEidUris": [
                "ipn:2.1"
            ],
            "keepAliveIntervalSeconds": 17,
            "tcpclV3MyMaxTxSegmentSizeBytes":2000000,
            "tcpclAllowOpportunisticReceiveBundles":true
        }
```


Fig. 10. HDTN Ground node configuration

REFERENCES

[1] NASA Glenn Research Center, "High Rate Delay Tolerant Networking (HDTN)," 2022. [Online]. Available: https://www1.grc.nasa.gov/space/scan/acs/tech-studies/dtn/

[2] A. Hylton and D. Raible, "High Data Rate Architecture (HiDRA)," in *Ka and Broadband Communications Conference 2016*, 2016.

[3] A. Hylton, D. Raible, G. Clark, R. Dudukovich, B. Tomko, and L. Burke, "Rising above the cloud: Toward high-rate delay-tolerant networking in low earth orbit," in *Advances in Communications Satellite Systems. Proceedings of the 37th International Communications Satellite Systems Conference (ICSSC-2019)*, 2019, pp. 1–17.

[4] The Consultative Committee for Space Data Systems, "CCSDS Bundle Protocol Specification," *CCSDS Blue Book*, 2015. [Online]. Available: https://public.ccsds.org/Pubs/734x2b1.pdf

[5] S. Burleigh, K. Fall, and E. Birrane, "RFC 9171, Bundle Protocol Version 7," *IETF Network Working Group*, 2022. [Online]. Available: https://datatracker.ietf.org/doc/rfc9171/

[6] NASA Marshall Space Flight Center, "DTNME," 2020. [Online]. Available: https://github.com/nasa/DTNME

[7] E. Birrane, K. Collins, and K. Scott, "The delay-tolerant networking experimental network constructing a cross-agency supported internetworking testbed," 06 2012.

[8] A. Schlesinger, B. M. Willman, L. Pitts, S. R. Davidson, and W. A. Pohlchuck, "Delay/disruption tolerant networking for the international space station (iss)," in *2017 IEEE Aerospace Conference*, 2017, pp. 1–14.

[9] NASA Jet Propulsion Laboratory, "Interplanetary Overlay Network (ION)." [Online]. Available: https://sourceforge.net/projects/ion-dtn/

[10] S. Burleigh, "Contact Graph Routing," https://tools.ietf.org/html/draft-burleigh-dtnrg-cgr-00, 2009.

[11] J. Fraire, "pyCGR," https://bitbucket.org/juanfraire/pycgr/src/master/, 2020.

[12] P. Hintjens, "ZeroMQ," 2020. [Online]. Available: https://zeromq.org/

[13] Boost, "Boost C++ Libraries," http://www.boost.org/, 2022.

[14] W. Eddy and E. Davies, "RFC 6256, Using Self-Delimiting Numeric Values in Protocols," *Internet Research Task Force (IRTF)*, 2011. [Online]. Available: https://tools.ietf.org/html/rfc6256

[15] M. Demmer, J. Ott, and S. Perreault, "RFC 7242, Delay-Tolerant Networking TCP Convergence-Layer Protocol," *Internet Research Task Force (IRTF)*, 2014. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7242

[16] The Consultative Committee for Space Data Systems, "Licklider Transmission Protocol (LTP) for CCSDS," *CCSDS Blue Book*, 2015. [Online]. Available: https://public.ccsds.org/Pubs/734x1b1.pdf

[17] B. Sipos, "Delay-Tolerant Networking UDP Convergence Layer Protocol," *IETF Delay-Tolerant Networking*, 2021. [Online]. Available: https://www.ietf.org/archive/id/draft-sipos-dtn-udpcl-01.html

[18] S. Burleigh, "Simple TCP Convergence-Layer Protocol," *IETF Delay-Tolerant Networking*, 2018. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-burleigh-dtn-stcp-00

[19] ——, *Interplanetary Overlay Network (ION) Design and Operation*, NASA Jet Propulsion Laboratory, 03 2016.

[20] B. Sipos, M. Demmer, J. Ott, and S. Perreault, "Delay-tolerant networking tcp convergence layer protocol version 4," *IETF Delay-Tolerant Networking*, 2021. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-dtn-tcpclv4

[21] The OpenSSL Project, "OpenSSL: Cryptography and SSL/TLS Toolkit," April 2003, www.openssl.org.

[22] The Consultative Committee for Space Data Systems, "CCSDS File Delivery Protocol (CFDP) ," *CCSDS Blue Book*, 2020. [Online]. Available: https://public.ccsds.org/Pubs/727x0b5.pdf

[23] A. Hylton, J. Cleveland, R. Dudukovich, D. Iannicca, N. Kortas, B. LaFuente, J. Nowakowski, D. Raible, B. Short, B. Tomko, and A. Wroblewski, "New horizons for a practical and performance-optimized solar system internet," in *IEEE Aerospace Conference 2022*, 2022.

[24] B. Xiao, "Linux Kernel Network Impairment Tool." [Online]. Available: https://github.com/urbenlegend/netimpair

[25] S. Hemminger, "tc-netem Linux Manual Page," 2011. [Online]. Available: https://man7.org/linux/man-pages/man8/tc-netem.8.html

[26] NASA Marshall Space Flight Center, "Telescience Resource Kit (TReK)." [Online]. Available: https://trek.msfc.nasa.gov/

[27] ESnet and Lawrence Berkeley National Laboratory, "iPerf3." [Online]. Available: https://iperf.fr/

[28] Red Hat Software, "Virtual Machine Manager." [Online]. Available: https://virt-manager.org/

BIOGRAPHY

Daniel Raible is an Electronics Engineer within the Optics and Photonics Branch at NASA's John H. Glenn Research Center (GRC) at Lewis Field in Cleveland, OH. He serves as GRC's optical communications subject matter expert, and the majority of Daniel's work supports NASA's Space Communications and Navigation (SCaN) program. His areas of research include developing and characterizing devices to support high bandwidth communications and navigation systems, as well as studying architectures for enabling new satellite capabilities and the migration towards higher performance and autonomy.

Rachel Dudukovich is currently a member of the Cognitive Signal Processing Branch at NASA Glenn Research Center and has worked at NASA since 2010. She earned her PhD in Computer Engineering and M.S. in Electrical Engineering from Case Western Reserve University. Her interests include cognitive networking techniques, delay tolerant networking, network emulation, artificial intelligence, machine learning, and algorithm development.

Brian Tomko earned his Bachelor of Science in Computer Engineering from Ohio Northern University in 2009. He has been working as a computer programmer at the NASA Glenn Research Center since 2009. His primary computer languages are C and C++ on both desktop and embedded platforms. Currently, his focus is on contributing code to the High-rate Delay Tolerant Networking (HDTN) software project.

Nadia Kortas is a research computer engineer working on the development of software to support embedded systems, space communication systems and other Space flight and Science projects at NASA. She currently works on High-rate Delay Tolerant Networking implementation with focus on routing and security. She holds a Master Of Science Degree in Engineering and Computer Science from Mines ParisTech, France.

Blake LaFuente is a computer engineer in the Cognitive Signal Processing Branch at NASA Glenn Research Center in Cleveland Ohio. His work focuses on the advancement of delay tolerant networking and aerospace communications. He holds a Bachelor of Arts in Public Policy and a Master of Science in Computer and Information Science from the University of Michigan.

Dennis Iannicca received his B.S. and M.S. in Computer and Information Science from Cleveland State University in 2007 and 2013, respectively. He has been working in the areas of Delay Tolerant Networking and Aeronautical Communications at NASA Glenn Research Center with a focus on network architecture and security.

Thomas Basciano is an engineer at the Johnson Space Center (JSC) in Houston, TX. He currently leads the International Space Station (ISS) Joint Station LAN (JSL) Team and has spent the last 15 years working various projects to integrate Commercial Off the Shelf (COTS) networking devices on ISS. He holds a Bachelor and Masters of Science in Aerospace Engineering from the University of Cincinnati.

William Pohlchuck is a software engineer for the Joint Station LAN (JSL) team. He is responsible for development of the on-orbit DTN gateways and integration of the DTN system on ISS. Mr. Pohlchuck holds a Bachelor of Science in Electrical Engineering from Ohio Northern University and a Master of Science in Electrical Engineering from the University of Utah.

Joshua Deaton is a software engineer for the Huntsville Operations Support Center (HOSC) at Marshal Space Flight Center (MSFC) where he is a part of the PDSS Software Development Team which supports high performance telemetry processing applications. As part of this support he is the primary developer for DTNME. He holds a Bachelor of Science in Business Administration specialized as a Systems Analyst from the University of Alabama Huntsville.

Alan Hylton should probably be designing tube audio circuits, but is instead the Network Studies Manager for the Near Space Network (NSN) at the NASA Goddard Space Flight Center, where he is humbled to work with his powerful and multidisciplinary team. His formal education is in mathematics from Cleveland State University and Lehigh University, and he considers it his mission to advocate for students. Where possible, he creates venues for mathematicians to work on applied problems, who add an essential diversity to the group.

John Nowakowski joined the NASA Glenn Research Center in 2020 as a Project Manager in the Space Communications and Spectrum Management Office. He earned degrees in Physics-Computer Science and Mechanical Engineering from the University of Dayton and a Masters of Fluid-Thermal Engineering Science from Case Western Reserve University. He has spent his career in the research and development of applied combustion systems and materials laser diagnostics, as well as specialized building infrastructure engineering.