New Horizons for a Practical and Performance-Optimized Solar System Internet

Alan Hylton and Jacob Cleveland and Rachel Dudukovich and Dennis Iannicca and Nadia Kortas and Blake LaFuente and John Nowakowski and Daniel Raible and Robert Short and Brian Tomko and Adam Wroblewski
NASA Glenn Research Center

Abstract—**The High-rate Delay Tolerant Networking (HDTN) project at the NASA Glenn Research Center (GRC) is developing a performance-optimized Delay Tolerant Networking (DTN) implementation using the Bundle Protocol (BP) both for infusion in modern, distributed spacecraft systems and for foundational network research purposes.**

While one purpose of networking is to enable a returns-to-scale with communicating assets, it is recognized that neither the protocol nor its implementations may impose a bottleneck on data delivery if expected to be implemented. With this in mind, this paper explores the current status of the HDTN software, including its capabilities and also various performance metrics at the bundle layer, the convergence layers (e.g. performance-optimized Licklider Transmission Protocol, or LTP), and bundle storage and retrieval. This includes goals of software implementations that support multi-gigabit per second communications regardless of payload size.

A discussion on the current software architecture and internal scheduling and routing is included. Interoperability with such extant DTN implementations as Interplanetary Overlay Network (ION) and DTN Marshall Enterprise Implementation (DTNME) are included. The research goals of the project are also listed, including Software-Defined Networking (SDN) and the theory of DTNs. A table-based filtration system for the Bundle Protocol (BP), called BP Filter, is also explored: this addition enables table-based policies within the context of a DTN for management purposes.

Infusion begins with systems testing in the Johnson Space Center (JSC) Software Development and Integration Laboratory (SDIL), which supports International Space Station (ISS) Flight Software development, integration, and verification. Observations made from the results of testing in the SDIL and in flight testing are also discussed, pointing to a path for infusion into high data return low-earth orbit (LEO) missions. The paper concludes with areas for future work.

1. Introduction

The year 2020 was the busiest launch-year on record, with 1,283 satellites launched. By 2021, of the 6,542 satellites tracked, 3,372 were operational[1]. Many of these are communications satellites; on October 8th, 2021, it was estimated that there were 1,674 operational Starlink satellites[2]. This number is only expected to grow, pushing the need for *scalable* communications. However it is this scalability that is lacking; many of these systems do not communicate or interoperate with each other, that is, there is no networked approach. Rather the approach is manual; indeed NASA notes that scheduling ground stations can take up to five days[3] - and NASA does not manage communications for all satellites.

Scalability is a goal of networking, however giving the realities of space communications, networking is challenging; we offer an incomplete list of reasons:

1. End-to-end connectivity may never exist,
2. the latencies featured have a high variance, and
3. the system is highly mobile.

This list neglects security and several technical challenges, yet it is already daunting. Traditional networking depends on end-to-end connectivity and low enough latencies for real-time interaction. The standardized approach that NASA takes is Delay Tolerant Networking (DTN)[4], which is an overlay network that uses the *bundle protocol* to connect once disparate one-to-one links[5]. The aforementioned *bundles* are the primary unit of data in a DTN, and can be of essentially any size.

While DTN has made great strides and has begun to be adopted, it has been recognized that the implementations often have deficiencies in terms of performance; see e.g. [6][7][8][9]. An implementation developed at GRC, called High-rate Delay Tolerant Networking (HDTN)[1], has been created with performance optimization in mind. In this paper, we highlight the progress of the HDTN software along with a brief overview of the HDTN research efforts. We conclude with the future directions of the project.

Part of the HDTN project focus is to develop a DTN implementation that can help enable scalability of communications but also not be a bottleneck as higher-rate communications capabilities get implemented. For example, the Laser Communications Relay Demonstration (LCRD) will support a 1 Gbps channel from the International Space Station (ISS) to the Earth, which could be used in conjunction with the existing Ku-band radios[10].

In order to prepare HDTN for tests on the ISS, the project is testing and developing the software in relevant environments. The first is the Software Development and Integration Laboratory (SDIL) at the Sonny Carter Training Facility at the NASA Johnson Space Center (SDIL) - all ISS software is checked out in the SDIL prior to integration[11]. In addition to the SDIL, HDTN has also been tested over a freespace optical link from an aircraft to a ground terminal to ensure its reliability and performance. After a brief discussion of the software architecture, both the SDIL and flight tests are described in detail.

2. Architecture

The HDTN software consists of Ingress, Egress, Storage, Scheduler, and Router modules. Each module uses ZeroMQ [12] to share state information. This method of replicating state between processes using a message bus architecture is a defining concept of HDTN. Early on in development, care was taken to avoid shared memory, mutexes, semaphores or any other types of locking techniques. The shared memory

[1]HDTN is open source: `https://github.com/nasa/HDTN`

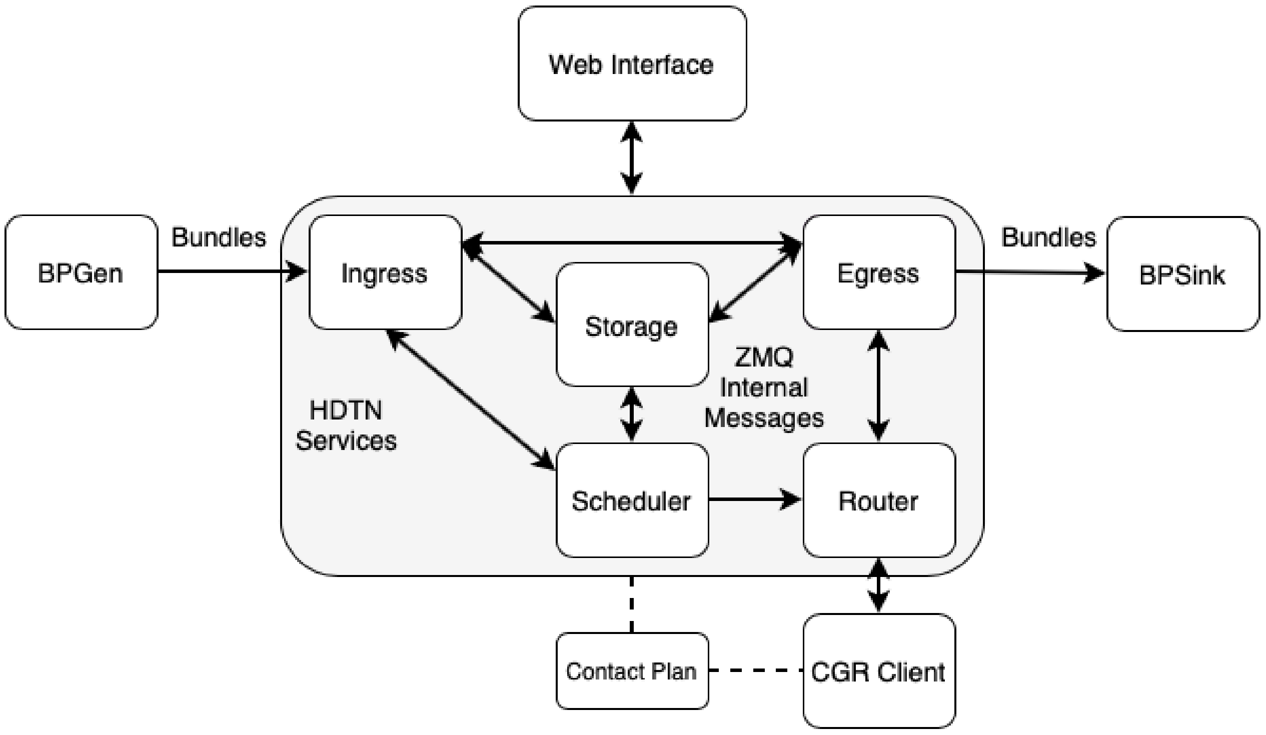


Figure 1. High Level HDTN Architecture

method of inter-process communication was found to create a number of bottlenecks in similar networking applications[7]. HDTN can operate in a distributed mode using the Transmission Control Protocol (TCP) for inter-process communication, or it can execute in a single process using ZeroMQ's local in-process transport.

Figure 1 shows a high level diagram of the HDTN architecture. The Ingress ingests bundles generated by BPGen tool and decodes the bundle header information. The Ingress receives messages from the Scheduler component to determine if a given bundle should be sent to long term storage or if it should be immediately forwarded to the Egress module. The basic Scheduler module reads a JavaScript Object Notation (JSON) contact plan to determine when a contact to a particular neighbor exists. The JSON contact plan is a custom format for HDTN, although it is based on the same fields as discussed in the Contact Graph Routing (CGR) [13] algorithm. The Scheduler generates `linkUp` or `linkDown` events to notify both the Ingress and Storage modules of a change in the contact availability. The Storage module receives messages from the Scheduler to determine when stored bundles can be released from storage and forwarded to the Egress. The CGR client calculates the optimal route and the next hop for a given contact plan and final destination using the PyCGR implementation. 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 that nextHop. If the link goes down unexpectedly or the contact plan gets updated, the router will be notified and will recalculate the next hop and send `RouteUpdate` events to egress accordingly. The Egress module is responsible for forwarding the bundles received from Storage or Ingress to the next hop based on the optimal route computed by CGR. BPSink is a tool to validate the bundles received from Egress. BpSink could be replaced by any application designed to receive bundles such as BpReceiveFile.

3. SDIL TESTING

The ISS has been implementing a Consultative Committee for Space Data Systems (CCSDS)-compliant DTN to provide reliable service to a variety of client payloads [14]. The CCSDS DTN protocols were selected to mitigate the unique challenges of the space environment as experienced by the ISS. These include highly asymmetric and constrained uplink and downlink rates, intermittent connections between the NASA Marshall Space Flight Center's Huntsville Operations Support Center (HOSC) and the ISS, as well as intermittent connections from the HOSC to payload users. The ISS uses the geosynchronous Tracking and Data Relay Satellite System (TDRSS) Ku-band system. The service is predictable but not continuous. Network conditions are affected by the orbit of the ISS, as well as TDRSS coverage and data latency. Latency can range from approximately 0.6 seconds to 1 second. The Joint Station LAN (JSL) uses BP over such convergence layers (CLs) as the Transmission Control Protocol CL (TCPCL)[15] or Simple TCP (STCP)[16]. Regions of the network that feature disruptive or high latency links utilize BP over the Licklider Transmission Protocol (LTP)[17] over the User Datagram Protocol (UDP). The ground nodes connect to the HOSC using BP over TCP. Figure 2 shows the ISS DTN topology as described in [14].

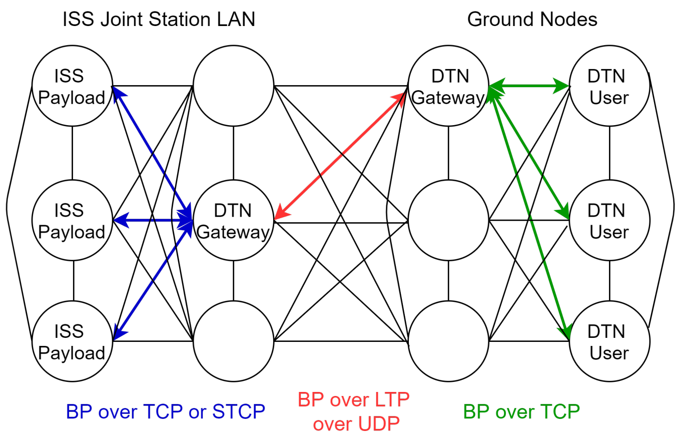


Figure 2. ISS DTN Network [14]

HDTN has been developing compatible protocols, local emulation capabilities, and evolving towards remote integration with the ISS DTN services. In early stages of testing, HDTN created a local test environment using an IBM (Lenovo) T61P laptop, Kernel-based Virtual Machine (KVM), and a Debian 10 guest operating system. This specification is noted in [14] as the platform of choice for a DTN gateway on the ISS. Through ongoing collaboration with the SDIL, HDTN has conducted preliminary integration tests within a realistic emulation of the ISS network. The current network configuration from the HDTN lab at NASA GRC to the SDIL is shown in Figure 3.

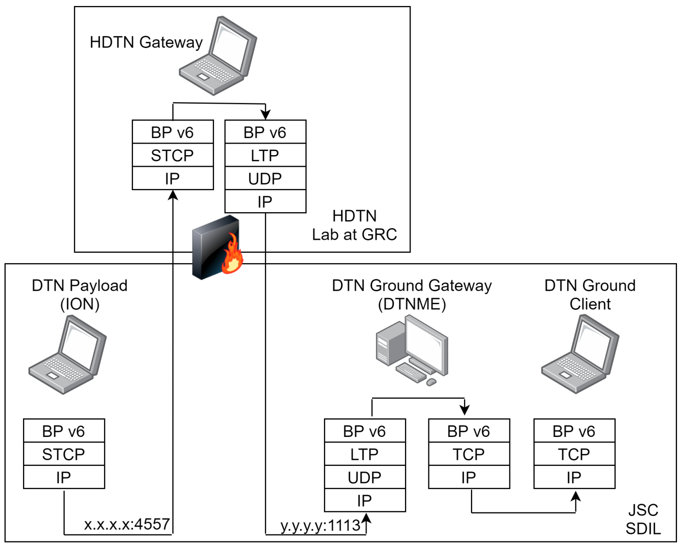


Figure 3. NASA GRC to JSC SDIL Network

The GRC HDTN lab is connected to the JSC SDIL 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. ION payload nodes and Marshall Space Flight Center's DTN Marshall Edition (DTNME)[2] gateway are configured to create a protocol stack which matches the real ISS network. The capabilities of the JSC SDIL combined with the Lenovo laptop and Debian virtual environment in the HDTN lab together create a high fidelity software test environment.

Preliminary tests with the SDIL verified that HDTN was capable of receiving bundles from an ION payload node using BP over STCP and forwarding this data to a DTNME gateway representative of the HOSC using BP over LTP over UDP. The DTNME gateway then forwards the bundle to a user node. This unidirectional test was successful, although additional work is needed to complete bidirectional testing of the LTP convergence layer. The next step of testing will be to complete a DTN ping (bping with bpecho) from the ION payload to the ground node via HDTN. Once this fundamental testing is completed, larger file transfers from the ION payloads to the ground will be completed to establish a baseline data rate for HDTN in the ISS environment.

The complexity of the LTP protocol, along with the challenges of developing an implementation compatible with both DTNME and ION will likely require multiple iterations of testing. In order to facilitate compatibility with both ION and DTNME, HDTN has taken care to develop a highly flexible configuration capability based on JSON. The human-readable format provided by JSON is essential for proper configuration of the LTP parameters in particular.

After the testing was conducted, software revisions and preparations were made to try HDTN over an unpredictable and intermittent link in a flight test; this is detailed in the next sections.

4. Flight Setup

The deleterious channel effects in space communications links generally include three characteristics: delays, disruptions, and disconnections. As link range increases, time of light delay also increases. The many minutes to hours of link delay inherent with deep space missions make it difficult if not impossible to close feedback control loops with Earth. Link fading is also exacerbated with increases in link range, and the resulting decrease in signal to noise ratio due to beam spread causes a proportional decrease in data rate. Finally, most missions whether near Earth or deep space remain disconnected most of the time, with scheduled links occurring once or a very few times per day (the exception being those mission utilizing relay services such as TDRSS). In addition, unscheduled disconnections may occur as the result of temporary occlusion, single event upsets from radiation, or other such equipment glitches and failures.

Each of these conditions may be emulated in the laboratory to evaluate HDTN's performance, but the relevant environment is often better approximated under field test conditions. An experiment of opportunity arose to fly a high-rate laser communications payload on the NASA Glenn Research Center's de Havilland DHC-6 'Twin Otter' aircraft, and incorporate HDTN as end-state terminal nodes to exchange data across the free space optical link. The plane is shown in Figure 4. The two flights taken are shown in Figures 5 and 6. The physical layer portion of the communications system consisted of a small optical aperture integrated into a spherical turret residing in the aft belly of the aircraft. This rotating turret

[2]DTNME is the modernization of DTN2 and is also open source; see `https://github.com/nasa/DTNME`

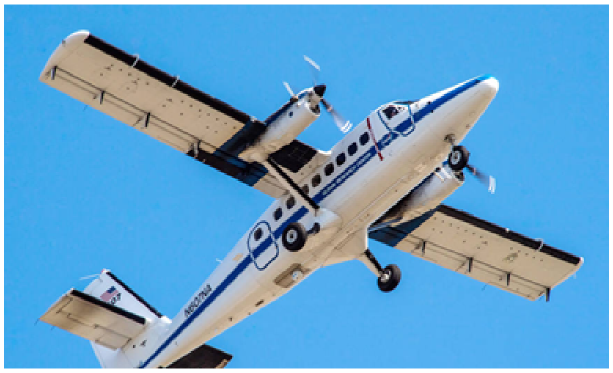


Figure 4. NASA Twin Otter aircraft with laser terminal

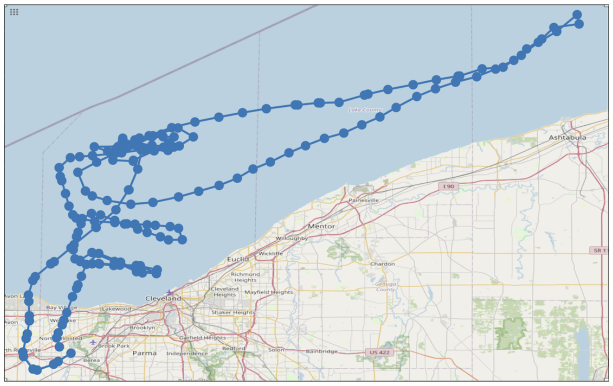


Figure 5. First aircraft flight track over Lake Erie[3]
Latitude: $41.345°$ to $42.1375°$
Longitude: $-82.0081°$ to $-80.7667°$

contained both the laser transmitter and the optical detector for establishing tracking and bidirectional communications between the terminals.

The ground portion of the laser communications system consisted of similar hardware located near the GRC hangar. The modems at both the flight and ground terminals supported rates approaching gigabit per second (Gbps), and featured logging capabilities to capture several timestamped link parameters such as connectivity, data rate, and bundle rate (bundles per second). The ground station setup is shown in Figure 7. Two HDTN nodes were implemented at each end of the communications systems. The ground computer is referred to as `Tantalus`, and the flight computer is simply `NUC`.

On the flight side an Intel Core i7 Next Unit of Computing (NUC) small form factor computer was configured with 2 TB of M.2 storage and 32 GB of Double Data Rate 4 Synchronous Dynamic Random-Access Memory (DDR4 SDRAM). The `NUC` is shown in Figure 8. In preparation for flight testing several convergence layers were implemented for evaluation including TCPCL, STCP, and LTP over UDP. To boost reliability in the Bundle layer, custody transfer was also prepared. A data payload containing 9,429 hyperspectral image files totaling over 34 GB was loaded into storage to represent the science traffic across the link.

[3]Map credit: ©OpenStreetMap contributors.

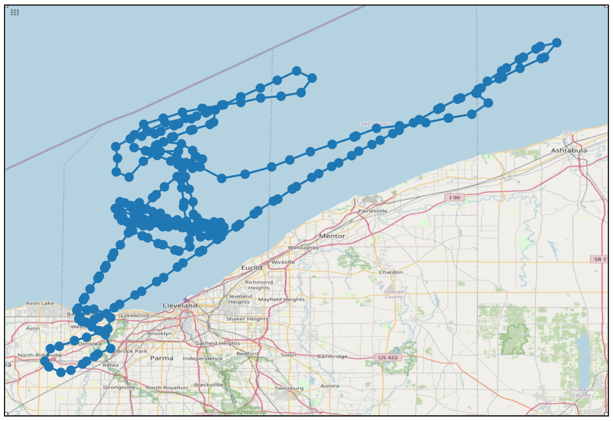


Figure 6. Second aircraft flight track over Lake Erie
Latitude: $41.3508°$ to $42.1128°$
Longitude: $-82.0067°$ to $-80.8194°$

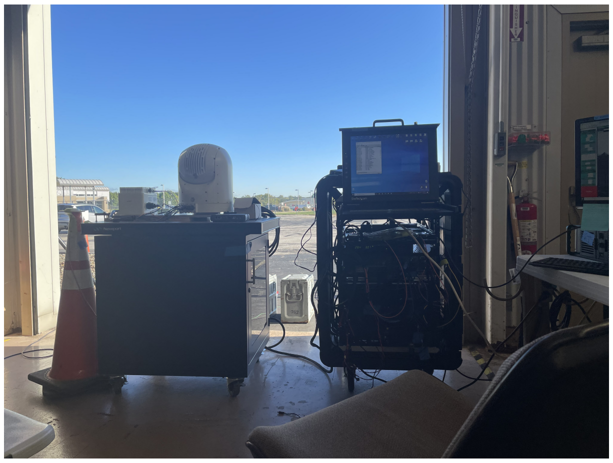


Figure 7. Ground setup of optical modem and transceiver

The test plan was to exchange these files with the ground terminal under varying link conditions to compare and contrast the performance of the various CLs configured under HDTN. In addition checksums were implemented to ensure data integrity after each file transfer case. The flight profile consisted of a northerly departure from NASA Glenn located adjacent to Cleveland Hopkins International Airport (KCLE) with a series of pattern maneuvers below 10,000 ft altitude over Lake Erie, and remained within 120 km of the ground station.

The varying slant range link distances across the lake resulted in channel fading effects and concomitant data rate fluctuations over time similar to spacecraft operations encountered during a communications pass. The full Gbps rate of the modems tested HDTN's ability to process bundles at significant speed, and the flight patterns demonstrated HDTN's performance under degraded and variable link conditions. Depending upon the pose and location of the aircraft, the line of sight was obscured by landing gear, trees, and light poles. This caused sporadic and temporary link disconnections, which offered an opportunity to examine HDTN recovery performance once service was restored. It also exercised the variable-rate capability of the link; note that the optical

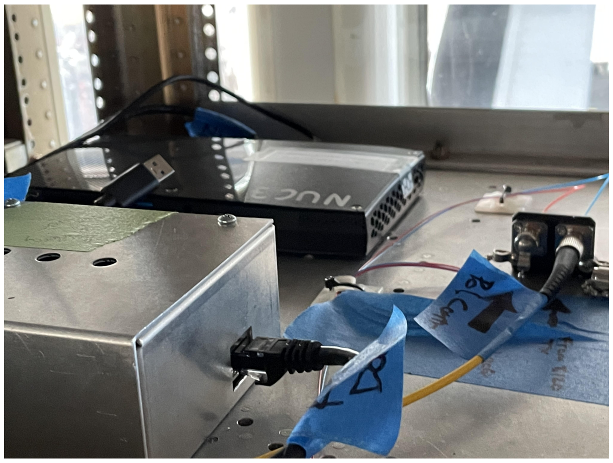


Figure 8. NUC installed on the Twin Otter

link could be viewed as a "variable rate" wire between the flight switch and the switch on the ground, and did not send any in-band telemetry. This challenging and unpredictable environment was used to stress-test and compare several implemented CLs in the DTN protocol while trying to achieve reliable network performance across the optical link.

The configuration is shown in Figure 9. While the STCP, TCPCL, and LTP convergence layers are themselves reliable, *custody* was used to offer reliability at the bundle layer. For a primer on custody, the Aggregate Custody Signals (ACS), and the Custody Transfer Enhancement Block (CTEB), see [18]. In 9, the dashed arrows are enabled when custody is enabled.

The final portion of the test setup was to create a file-transfer application - one was quickly developed as the HDTN software currently does not currently support a CCSDS File Delivery Protocol (CFDP) file transfer utility. The BpSendFile utility reads a directory of files, sorts them alphabetically, breaks each file into bundles of a maximum specified bundle size (0.5MB for this test), prepends its own header to the bundle payload (consisting of total file size, fragment offset, fragment length, file name string length, and file name), and adds the file fragment to the bundle (unbuffered using `fread` calls). The BpReceiveFile utility receives these bundles in any order and using similar logic from LTP, reassembles the file fragments (see HDTN's util/FragmentSet.cpp in the GitHub link above) and closes the file when all file fragments have been received. No checksum mechanism has been implemented into the file transfer utility for this test, but SHA256 hashes of the files were verified later after the test was completed.

5. Configuration

There are four sets of configuration files through this whole process: on the sending side, we have the `NUC` HDTN with the BPSendFile config files, and on the receiving side we have the `Tantalus` HDTN with the BPReceiveFile config files. For the HDTN config files, we need to update the induct and outduct vectors with the correct port numbers and final destination nodes. BPSendFile has an outduct over local host to the HDTN and BPReceiveFile has a localhost induct, both of which are over STCP convergence layer. For tests with custody transfer, additional inducts and outducts are needed in the HDTN config files: two additional outducts are needed for the custody to go back to the senders (BPSendFile and HDTN `NUC`) and two additional inducts are needed for the custody coming from the receivers on the Ground (HDTN `Tantalus` and BPReceiveFile). The configurations for the flight node and the ground node are shown in Figures 24 and 25 respectively in the appendix.

For the storage we allocated 82 GB striped across two different files. The storage configuration is also shown in the appendix, in Figure 26.

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.

We set `ltpMaxRetriesPerSerialNumber` to 500 retries to make sure the Egress will keep bundles for 20 minutes of disconnection time and prevent bundles from being dropped. This was calculated as in Formula 1, in milliseconds, which gave the maximum time of disconnection for a bundle before it is dropped as:

$$(2,400 \text{ ms roundtrip time}) \times (500 \text{ retries}) = 1,200 \text{ s.}$$

$$\begin{aligned}\texttt{Maximum disconnection time} = {} & 2\left(\texttt{oneWayLightTimeMs} + \texttt{oneWayMarginTimeMs}\right) \\ & \times \texttt{ltpMaxRetriesPerSerialNumber}.\end{aligned} \tag{1}$$

The `ltpCheckpointEveryNthDataSegment` field had to be updated to enable accelerated retransmission. For the tests where accelerated retransmission was disabled this value was set to 0, and when accelerated retransmission was used it was set to 10; this made LTP reconcile its lost packets faster.

Finally, UDP kernel parameters were tuned to prevent packet-dropping for the LTP tests. They are as follows:

```
net.core.rmem_default=26214400
net.core.wmem_default=26214400
net.core.rmem_max=26214400
net.core.wmem_max=26214400
```

6. Flight Testing

The day of testing (September 30, 2021) offered two opportunities to conduct flights beyond the northeast shores of Ohio over Lake Erie. Weather conditions were very clear. The free space optical link between the aircraft and the ground exhibited very continuous and repeatable degradation as a function of range. At the end of testing over 578 GB of data were transferred successfully over several hours with over 99.99% of the data files being transferred error-free.

Below, graphs of the data rates during testing are shown and discussed briefly. Each graph has the same format: the horizontal axis is time, in seconds. The vertical axis is rate -

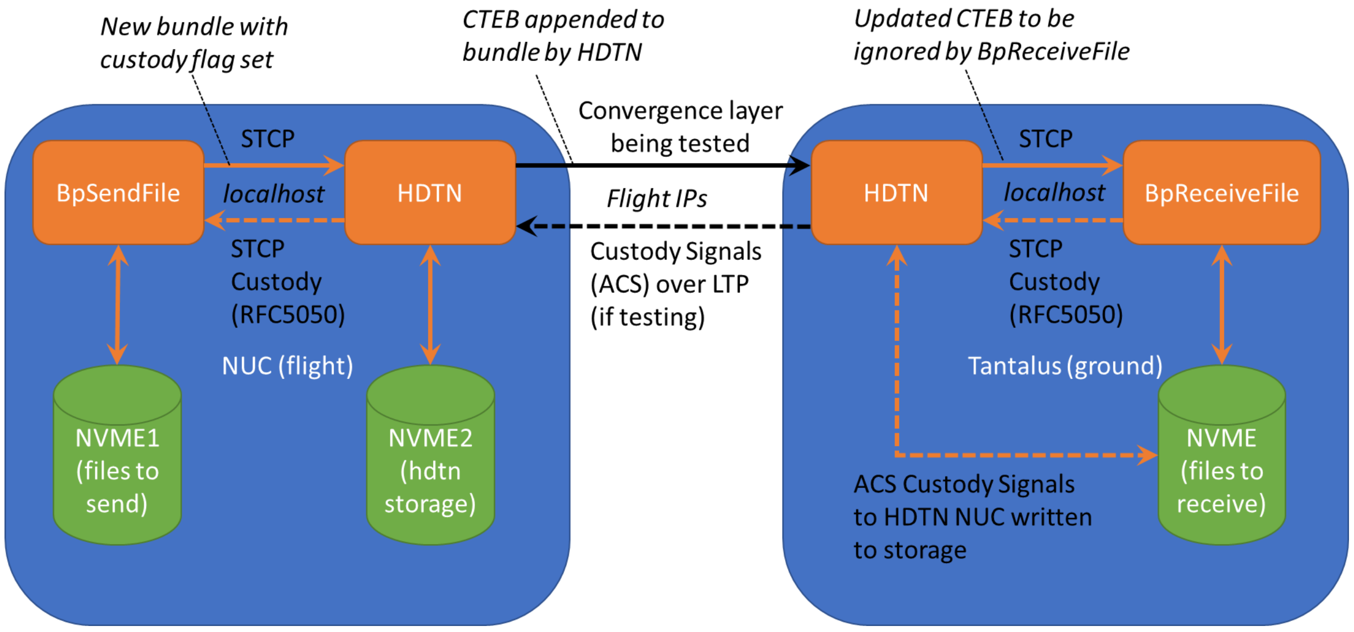

New bundle with custody flag set
CTEB appended to bundle by HDTN
Updated CTEB to be ignored by BpReceiveFile
Convergence layer being tested
STCP
BpSendFile
localhost
HDTN
STCP Custody (RFC5050)
NUC (flight)
NVME1 (files to send)
NVME2 (hdtn storage)
Flight IPs
Custody Signals (ACS) over LTP (if testing)
HDTN
STCP
localhost
BpReceiveFile
STCP Custody (RFC5050)
Tantalus (ground)
NVME (files to receive)
ACS Custody Signals to HDTN NUC written to storage

Figure 9. Flight test setup

red shows bundles per second, and blue shows megabits per second. Connectivity was measured using a time-stamped log of pings, and is denoted by a green dot.

TCPCL

The only TCPCL test conducted is illustrated in Figure 10; the summary is shown in Table 1. Perhaps in some sense

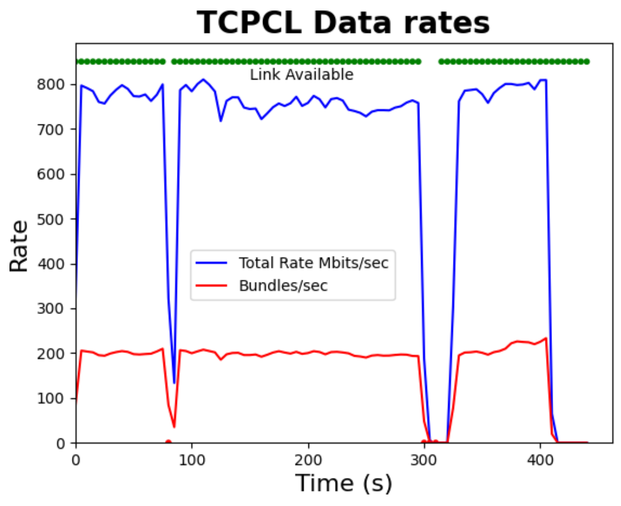


Figure 10. TCPCL Test

TCPCL can be considered "cheating," and one might not expect TCP to be used over such a link. What this test gives us is a notion of the upper-bound of the optical setup used, while retrieving 100% of the data. One can clearly see where availability dropped, likely due to obscuration as this test was conducted with the aircraft relatively close (~20km) to the ground station.

Stats %	Test
Bundles received	98.90
Bundles sent to Storage	23.95
Bundles sent to Egress	76.03
Correct checksums	97.90

Table 1. Results of the TCPCL test

STCP

In this section, we consider STCP tests. The first test took place during the flight's furthest distance (~120km) from the ground station close to the Pennsylvania border; this test's data is shown in Figure 11. Due to this distance, the link was unavailable for nearly the entire test and few bundles were able to be transmitted. The Storage module was unable to store the large amount of data to send when the link was reestablished. In all, five STCP tests were run; the results are shown in Table 2.

Stats %	Test1	Test2	Test3	Test4	Test5
Bundles received	7.27	19.46	48.95	99.99	99.99
Bundles sent to Storage	43.68	92.11	NA	0	16.90
Bundles sent to Egress	56.25	7.87	NA	99.99	83.09
Correct checksums	NA	NA	NA	99.96	99.97

Table 2. Results of the STCP test.

STCP Tests 2 and 3, illustrated in Figures 12 and 13 respectively, took place at a much closer and consistent distance of about 50 kilometers. Test 2 ran well for the first two minutes, achieving a data rate of over 500 Mbits/sec for most of that time. At about 125 seconds into the test, the link was lost and after several minutes of failing to reestablish a connection, the test was ended. The ground station received 19.5% of the

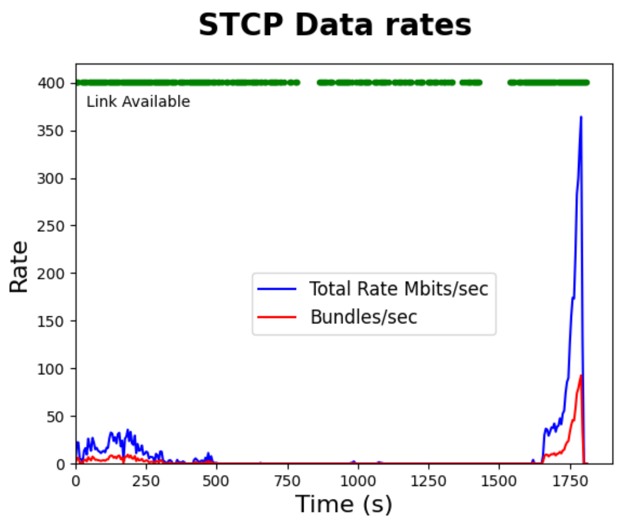


Figure 11. STCP Test 1

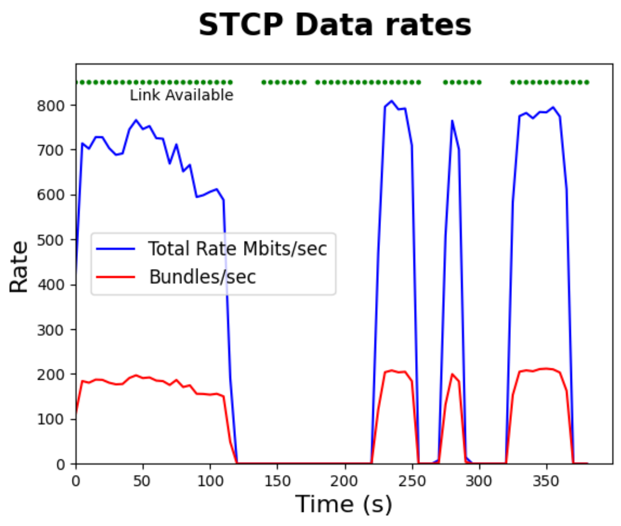


Figure 13. STCP Test 3

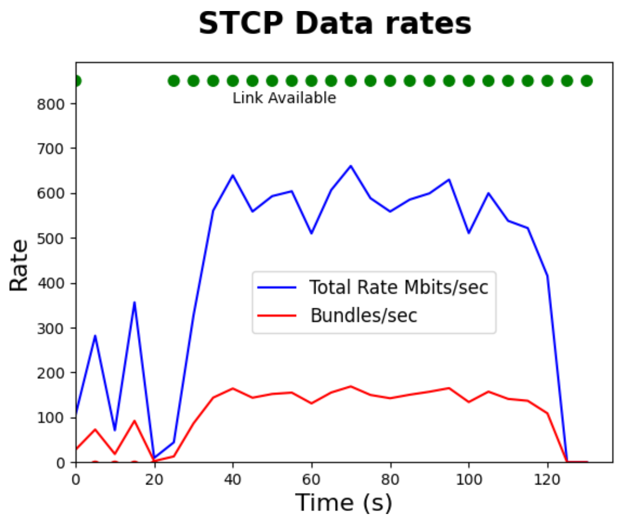


Figure 12. STCP Test 2

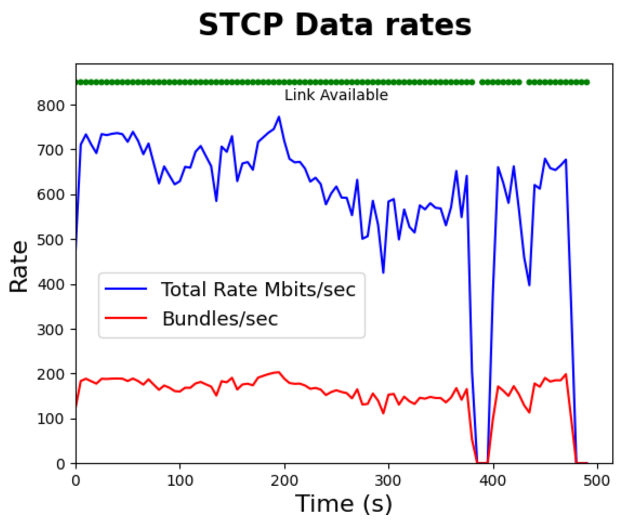


Figure 14. STCP Test 4

expected bundles.

Test 3 was started immediately after Test 2 and at about 120 seconds into the test link was lost. Once the link was reestablished, bundles were not received until about 240 seconds into the test. Twice more the link was interrupted and the ground terminal was able to receive bundles after reconnecting. After losing the link for a fourth time, bundles were unable to be received and after waiting several minutes, the test was ended. The ground station received 49% of the expected bundles. An error with the logs occurred on the flight computer resulting in missing data on whether bundles were sent to Storage or Egress.

In Tests 1, 2, and 3, the Storage module was only configured to only store 8 GB of data which was too low of a threshold. Thus in all three tests, once the Storage Module reached its maximum limit for stored bundles, the remaining bundles were lost. This is the expected cause for the ground station failing to receive all the bundles. The percentage of correct checksums were not calculated as less than half of the bundles were received.

This configuration was changed in the period before the second flight of the day and Tests 4 and 5 (Figures 14 and 15) were much more predictable. Both took place at about an 18 km distance from the ground station. Both tests were able to achieve very high data rates and recovered quickly following an interruption in the link. In the end, both Tests 4 and 5 were able to successfully transfer 99.99% of their bundles.

LTP/UDP without Custody

In this section, LTP was used without custody transfer. Four tests were run, the results of which are shown in Table 3. Tests 1, 2, and 3 (Figures 16, 17, and 18) took place during the first flight and included the same configuration that was used in the STCP tests in which the Storage Module was only configured to hold 8 GB of data. Tests 1 and 3 reached that limit. As a result of the ground station receiving less than half of the expected bundles for those tests, the correct checksums were not calculated.

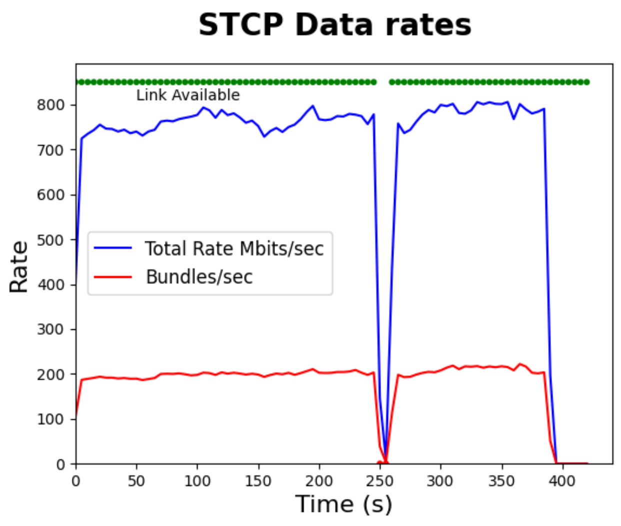


Figure 15. STCP Test 5

Stats %	Test1	Test2	Test3	Test4
Bundles received	22.13	99.99	43.95	99.98
Sent packets lost	3.59	9.67	2.55	6.00
Bundles sent to Storage	97.35	6.52	78.24	0.06
Bundles sent to Egress	2.63	93.74	21.74	99.92
Correct checksums	NA	NA	NA	99.95

Table 3. Results of the LTP without Custody test

Test 1 was the first of all the flight tests performed. The flight computer began sending bundles from a distance of 35 km, but the link was lost shortly after the ground station established a connection and started receiving bundles. Shortly after Test 1 started the link was unavailable which caused the bundles to be sent to storage right away at a high BPSendFile transfer rate (5240 Mbits/sec) which exhausted the storage and caused it to run out of space very fast. The link was reestablished about six minutes later and bundles were once again able to be received. Many bundles were lost once the storage limit was reached and the test ended with 22.13% of bundles received.

Test 2 (Figure 17) took place right after Test 1 (Figure 16 starting at the same distance of 35 km with the plane flying away to a distance of 50 km. As expected, the data rate steadily declined from a high of 821.98 Mbits/sec to a rate of 678.7 Mbits/sec before the link dropped. The test completed once the link was reestablished with a 99.99% successful bundle transfer. An error occurred resulting in a loss of the checksum file for this test.

Test 3 (Figure 18) took place after Test 2 at a steady distance of about 40km. During the test, the link was interrupted and the Storage module reached the maximum limit of data storage causing bundles to be lost. Once the link was reestablished, stored bundles were sent and the test completed with 43.95% of bundles being received by the grounds station.

Test 4, shown in Figure 19, took place during the second flight and after the storage configuration was modified from storing a maximum of 8 GB to a maximum of 80 GB. Data

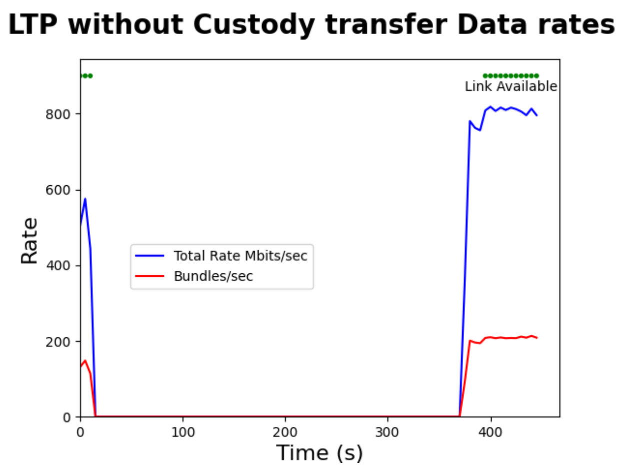


Figure 16. LTP/UDP without Custody Test 1

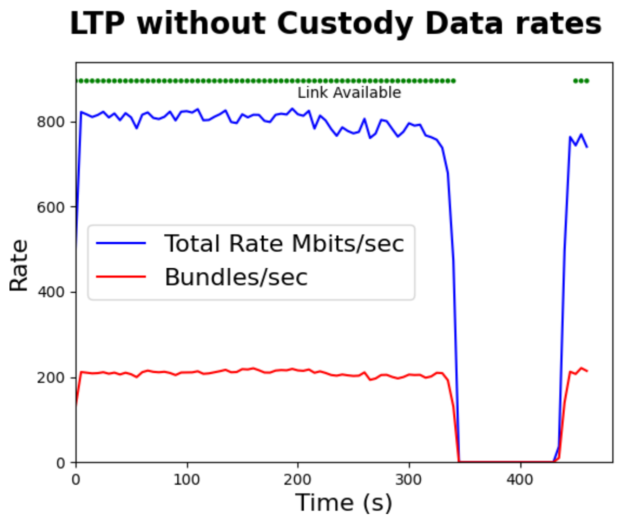


Figure 17. LTP/UDP without Custody Test 2

was transmitted from a variable distance of 27 to 35 km with only a momentary loss of connection. The test completed with 99.98% of bundles received by the ground station.

LTP/UDP with Custody

In this section, LTP was used with custody transfer. The only other change was in the storage configuration to allow the maximum data to be 80 GB. All other settings were the same as those for LTP without custody. These tests had a 100% file transfer success rate, which involved sending approximately 27 million 1360-byte UDP packets. The data are in Table 4. An maximum (also modal) bundle size of 0.5 MB was used, and each bundle was one “fully red” LTP session; recall that red means reliable. A maximum of 5 concurrent LTP sessions were used.

The first test, with characteristics illustrated in Figure 20, was run without accelerated retransmission, and had the final checkpoint expire on 12 out of 76,251 0.5-Megabyte bundles (0.016% loss). Tests 2, 3, and 4 (shown in Figures 21, 22, and 23 respectively) had accelerated retransmission enabled and had a checkpoint on every 10^{th} UDP packet. Test 2 dropped

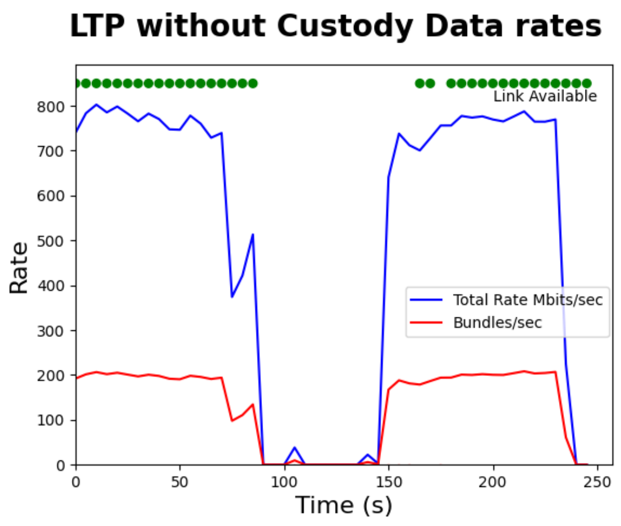


Figure 18. LTP/UDP without Custody Test 3

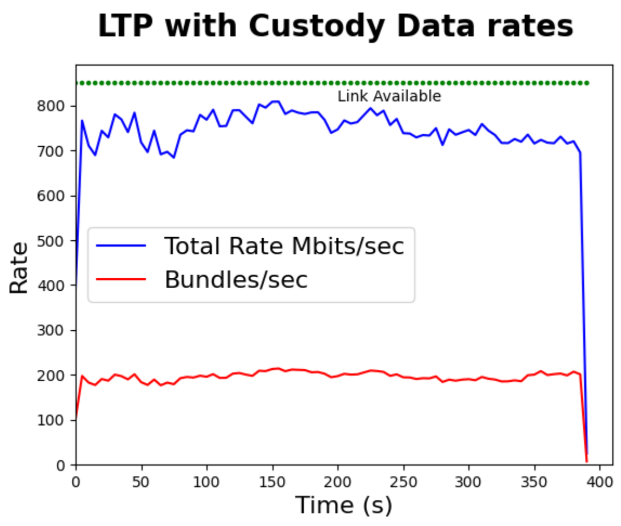


Figure 20. LTP/UDP with Custody Test 1

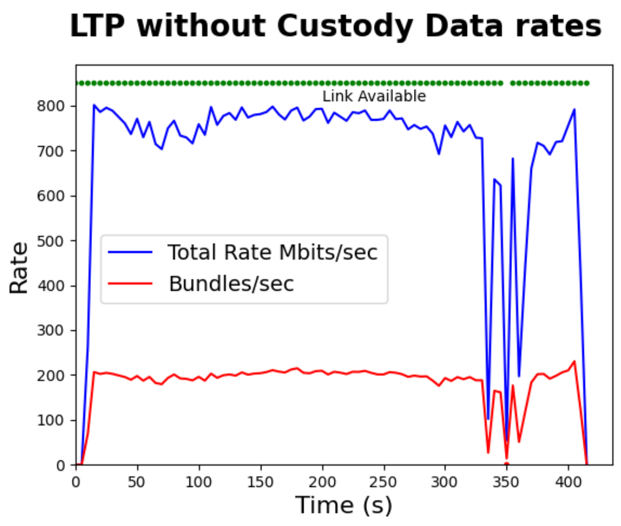


Figure 19. LTP/UDP without Custody Test 4

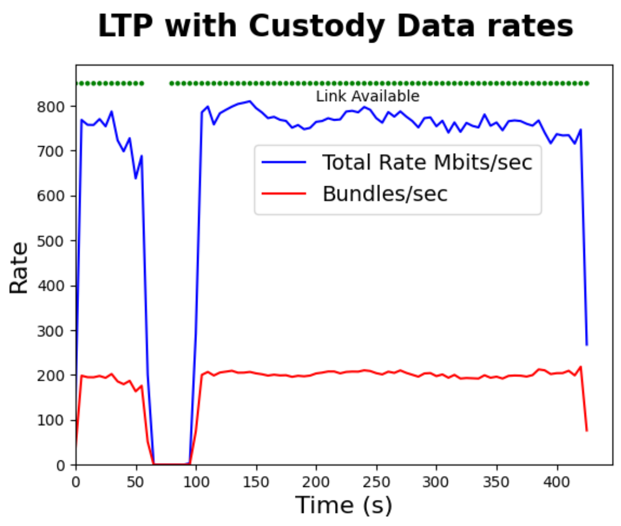


Figure 21. LTP/UDP with Custody Test 2

Stats %	Test1	Test2	Test3	Test4
Bundles received	100	100	100	100
Bundles sent to Storage	100	100	100	100
Bundles sent to Egress	0	0	0	0
Sent packets lost	0.01	0.04	0.09	0.11
Correct checksums	100	100	100	100

Table 4. Results of the LTP with Custody test.

only 94 out of approximately 2.7 million checkpoint packets (0.004% loss). Test 3 dropped 2434 out of approximately 2.7 million checkpoint packets (0.09% loss). This test shows a drop in data rate followed by an increase. This was due to the plane traveling to a distance of 50km and returning.

Test 4 dropped 3010 out of approximately 2.7 million checkpoint packets (0.11% loss). This test starts at a low data rate and experiences large losses of signal due to it taking place when the plane traveled its furthest flight of 120 km and then returning.

7. Flight Results

All four LTP Custody transfer tests performed had 100% success rate of sending all bundles across the link, while the ten non-custody tests performed dropped on average approximately 9 out of the 76,251 file data bundles for a 99.99% success rate. The reason for this loss was due to a bug in the cut-through path of HDTN, in which if a link was available and custody was not requested, Storage was skipped completely. This was caused by a hard-coded 2-second timeout (which is the bug) in the Ingress portion in which if the link went down and the bundle could not be sent from the Egress portion within two seconds, Ingress dropped the bundle rather than sending it to storage. In addition, there was a configuration issue in the first three non-custody tests in which the minimum storage size needed (approximately 34 GB) was not allocated to these tests (only 8 GB were

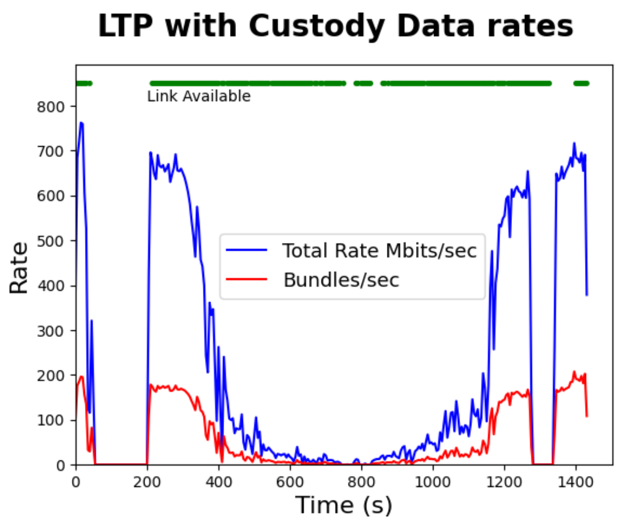


Figure 22. LTP/UDP with Custody Test 3

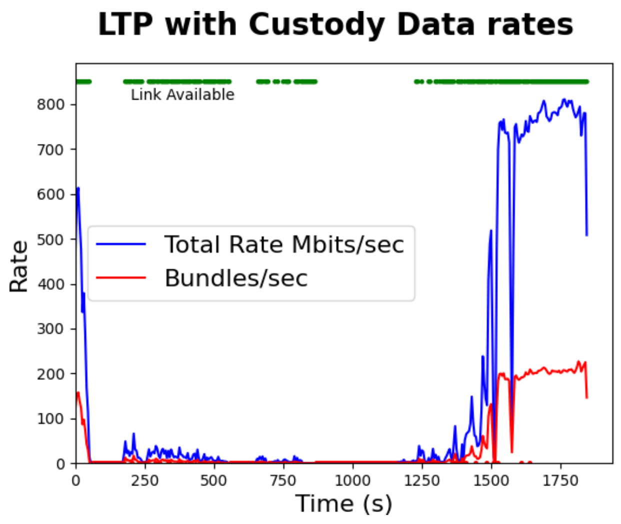


Figure 23. LTP/UDP with Custody Test 4

allocated), and the link-down event caused HDTN to run out of storage space for some time.

There were two anomalies that will require further investigation. One was that old LTP data segments were being received after the LTP session had been closed. The other was that in two of the custody transfer LTP tests, approximately 450 out of 76251 custody transfers were lost (5 to 6 Aggregate Custody Signal bundles were lost out of approximately 900 ACS bundles sent).

The rates from BpSendfile to HDTN over STCP localhost on the `NUC` showed a maximum of 5 Gbps. When this rate exceeds the optical maximum link rate of 800 Mbps, this means that HDTN Storage is being used, which is always the case when custody is enabled. The storage is required in that case even if an immediate forwarding opportunity exists because the bundle layer has accepted a custody transfer and must therefore 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. When the rates in data were below the optical maximum rate, meaning that Storage is bypassed due to disabled link availability and custody, BpSendFile is blocked by the HDTN bundle pipelining mechanism. Nevertheless, the 5 Gbps rates imply that there is no software bottleneck when it comes to writing bundles to disk, but rather it is constrained by M.2 NVME speeds and/or operating system write-caching speeds. It is possible that the 5 Gbps speed could be exceeded if a second NVME disk was added to the Storage configuration file, as HDTN Storage module supports its own striping implementation across any number of disks.

8. Theory

DTN, with HDTN included, offer mechanisms to unify broad means of communications at lower layers. This extends from custom point-to-point links to TCP overlays. The penalty, as it were, is that the complexity increases. As noted above, scheduling is expensive and the amount of scheduling needed will only increase. In order to achieve true networked operations for the so-called Solar System Internet (SSI), there must also be practical network tools, such as routing algorithms and policy engines, which act on the DTN locally and globally.

To answer these questions, a mathematical framework for disconnected networking is being actively investigated and discovered by the GRC HDTN research team. This begins with the generalization of tools and structures familiar to networking. It has been noted that *networks are sheafy*[4], where *sheaves* are the formal data structure for gluing local data into global data - one can immediately refer to link state routing for intuition. For example, one might take Dijkstra's algorithm and recast it in a more powerful language - that of sheaves - so that it can be applied to more general structures than graphs - such as temporal graphs[19].

The HDTN effort has laid the very beginnings of this work in [20], which continued in the aforementioned paper [19] and was continued through [21]. With our ZeroMQ approach to interprocess communication (IPC), a Python version of contact graph routing (PyCGR) has been modified to act on HDTN[22]. As future algorithms are developed, they can be tested against HDTN in similar ways. For example, it has recently been discovered that contact graphs can in fact admit routing loops, and new approaches using multigraphs are proposed in [23].

9. Conclusion and Future Work

In this paper, we detailed the current version of NASA Glenn's implementation of DTN (RFCs 4838 and 5050), as well as integration tests over the SDIL - demonstrating interoperability with ION and DTNME, and over disrupted links - which demonstrated real-world applicability. Together, these demonstrated the core function of HDTN, to be a high-rate implementation of DTN for modern communications scenarios (e.g. Integrated LCRD Low-Earth-Orbit (LEO) User Modem and Amplifier Terminal - ILLUMA-T).

As these and future tests reduce the risk of integrating HDTN, the project moves towards a goal of mission integration. The Satellite Hosting Atmospheric and Littoral Ocean Water Sen-

[4] Attributed to Rob Ghrist.

sors (SHALLOWS) mission has had HDTN on the manifest since day one to handle reliable data transmission from a hyperspectral camera on a cubesat to a ground station; from a data point of view, this means collecting over 8 to 16 GB of data at 5 Gbps into bundles, storing them, and then transmitting them over a 300 Mbps X-band link. Future version of this project may feature constellations, stressing the networking aspect. Another goal is operating HDTN, at least experimentally, on the ISS - this is an opportunity to truly test a multi-hop, multi-path network with disconnection.

While these tests were strong indicators of HDTN's progress to date, they also uncovered multiple opportunities for improvement, several of which were based on configurations. Thus we conclude with areas for future work.

Future Work

- Continue and extend testing. The HDTN team has built a virtualization environment that is capable of tests with over 512 nodes. Using various emulators and simulators, we can implement and test discovery mechanisms in conjunction with traditional scheduled routing.
- Implement security. As BPSec becomes more matured, it is necessary to support for true integration and collaboration[24].
- Further integrate Contact Graph Routing. Communicating with PyCGR over ZMQ has enabled scheduling the best possible routes using Dijkstra's algorithm[22]. Further CGR development will involve a custom implementation using C++.
- Push DTN research. By emphasizing the mathematical and computer scientific aspects of DTN, we expect to continue discovering the foundations of disconnected networking. By improving our understanding and increasing the rigor of DTN discourse, we can build confidence and more attractive standards and software.
- Use Software-Defined Networking (SDN) to push the limits of high-rate DTN. For example, SDN can be used for load-balancing in a transparent manner, or also to reimplement anycast routing.
- Increase our connection with cognitive networking. Many of the above concepts such as DTN (Bundle Protocol, Contact Graph Routing, proactive/reactive fragmentation), SDN, policy-based services, and anycast have been identified as building blocks of cognitive networking technologies. HDTN can be used as the starting point in a incremental approach to develop autonomous network implementations that sense and adapt to optimize network and system level performance.

It is the authors' hope that HDTN has established a reliable and performance-driven DTN implementation that also enables DTN research.

10. ACKNOWLEDGEMENTS

The authors would like to thank the NASA Space Communications and Navigation (SCaN) program, and in particular Philip Baldwin and Brenda Lyons for their support. The flight tests would not have been possible without the Glenn Research Center Flight Research Crew. The SDIL testing depended on support from Tom Basciano, Bill Pohlchuck, and many others at the Sonny Carter Training Facility. Guidance from Lee Pitts and Josh Deaton at the Marshall Space Flight Center is much appreciated. Finally, we acknowledge John Lekki, Roger Tokars, and the hyperspectral imaging team for providing the traffic data.

REFERENCES

[1] Nibedita Mohanta. *How many satellites are orbiting the Earth in 2021?* May 2021. URL: `https://www.geospatialworld.net/blogs/how-many-satellites-are-orbiting-the-earth-in-2021/`.

[2] Jonathan McDowell. *Starlink Statistics*. Oct. 2021. URL: `https://planet4589.org/space/stats/star/starstats.html`.

[3] NASA. *Delay/Disruption Tolerant Networking*. Sept. 2020. URL: `https://www.nasa.gov/directorates/heo/scan/engineering/technology/disruption_tolerant_networking`.

[4] V. Cerf et al. "RFC 4838, Delay-Tolerant Networking Architecture". In: *IETF Network Working Group* (2007). URL: `https://tools.ietf.org/html/rfc4838`.

[5] K. Scott and S Burleigh. "RFC 5050, Bundle Protocol Specification". In: *IETF Network Working Group* (2007). URL: `https://tools.ietf.org/html/rfc5050`.

[6] Alan Hylton and Daniel E. Raible. "High Data Rate Architecture (HiDRA)". In: *34th AIAA International Communications Satellite Systems Conference*. DOI: `10.2514/6.2016-5756`. eprint: `https://arc.aiaa.org/doi/pdf/10.2514/6.2016-5756`. URL: `https://arc.aiaa.org/doi/abs/10.2514/6.2016-5756`.

[7] Alan Hylton, Daniel E. Raible, and Gilbert Clark. "On the Development and Application of High Data Rate Architecture (HiDRA) in Future Space Networks". In: *35th AIAA International Communications Satellite Systems Conference*. DOI: `10.2514/6.2017-5415`. eprint: `https://arc.aiaa.org/doi/pdf/10.2514/6.2017-5415`. URL: `https://arc.aiaa.org/doi/abs/10.2514/6.2017-5415`.

[8] A. Hylton, D. Raible, and G. Clark. "A Delay Tolerant Networking-Based Approach to a High Data Rate Architecture for Spacecraft". In: *2019 IEEE Aerospace Conference*. Mar. 2019, pp. 1–10. DOI: `10.1109/AERO.2019.8742135`.

[9] Alan Hylton et al. "Rising Above the Cloud – Toward High-Rate Delay-Tolerant Networking in Low-Earth Orbit". In: *37th AIAA International Communications Satellite Systems Conference*. Oct. 2019, pp. 1–8.

[10] D. J. Israel, B. L. Edwards, and J. W. Staren. "Laser Communications Relay Demonstration (LCRD) update and the path towards optical relay operations". In: *2017 IEEE Aerospace Conference*. Mar. 2017, pp. 1–6. DOI: `10.1109/AERO.2017.7943819`.

[11] Carlos G. Parra. *Space Station software lab gets new home*. July 2003. URL: `https://roundupreads.jsc.nasa.gov/jscfeatures/articles/000000023.html`.

[12] P. Hintjens. *ZeroMQ*. `https://zeromq.org/`. 2020.

[13] S. Burleigh. *Contact Graph Routing*. `https://tools.ietf.org/html/draft-burleigh-dtnrg-cgr-00`. 2009.

[14] A. Schlesinger et al. "Delay/Disruption Tolerant Networking for the International Space Station (ISS)". In: *2017 IEEE Aerospace Conference*. 2017, pp. 1–14. DOI: `10.1109/AERO.2017.7943857`.

[15] M. Demmer, J. Ott, and S. Perreault. "RFC 7242, Delay-Tolerant Networking TCP Convergence-Layer

Protocol ". In: *IETF Network Working Group* (2014). URL: https://tools.ietf.org/html/rfc7242.

[16] S. Burleigh. "Simple TCP Convergence-Layer Protocol ". In: *IETF Network Working Group* (2018). URL: https://datatracker.ietf.org/doc/html/draft-burleigh-dtn-stcp-00.

[17] M. Ramadas, S. Burleigh, and S. Farrell. "RFC 5326, Licklider Transmission Protocol - Specification ". In: *IETF Network Working Group* (2008). URL: https://datatracker.ietf.org/doc/html/rfc5326.

[18] Andrew Jenkins. *Aggregate Custody Signals*. Nov. 2011. URL: https://cwe.ccsds.org/sis/docs/SIS-DTN/Meeting%20Materials/2011/Fall%20(Colorado)/jenkins-sisdtn-aggregate-custody-signals.pdf.

[19] M. Moy et al. "Path Optimization Sheaves". 2020. URL: https://arxiv.org/abs/2012.05974.

[20] A. Hylton et al. "A Mathematical Analysis of an Example Delay Tolerant Network using the Theory of Sheaves". In: *2020 IEEE Aerospace Conference*. 2020, pp. 1–11.

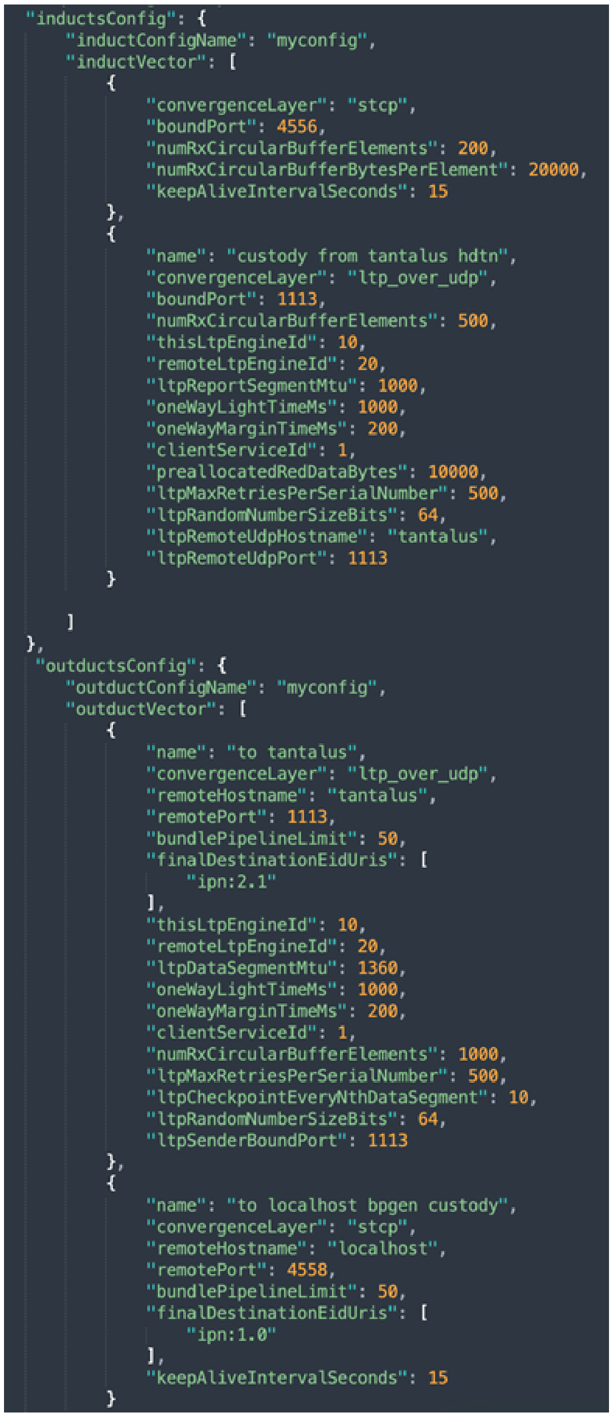
[21] Robert Short et al. "Towards Sheaf Theoretic Analyses for Delay Tolerant Networking". In: *2021 IEEE Aerospace Conference (50100)*. 2021, pp. 1–9. DOI: 10.1109/AERO50100.2021.9438167.

[22] Olivier De Jonckère and Juan A. Fraire. "A shortest-path tree approach for routing in space networks". In: *China Communications* 17.7 (2020), pp. 52–66. DOI: 10.23919/J.CC.2020.07.005.

[23] Robert Short et al. *What Type of Graph is a Contact Graph?* Technical Memorandum Upcoming. Glenn Research Center, Cleveland OH 44135, USA: NASA, 2021.

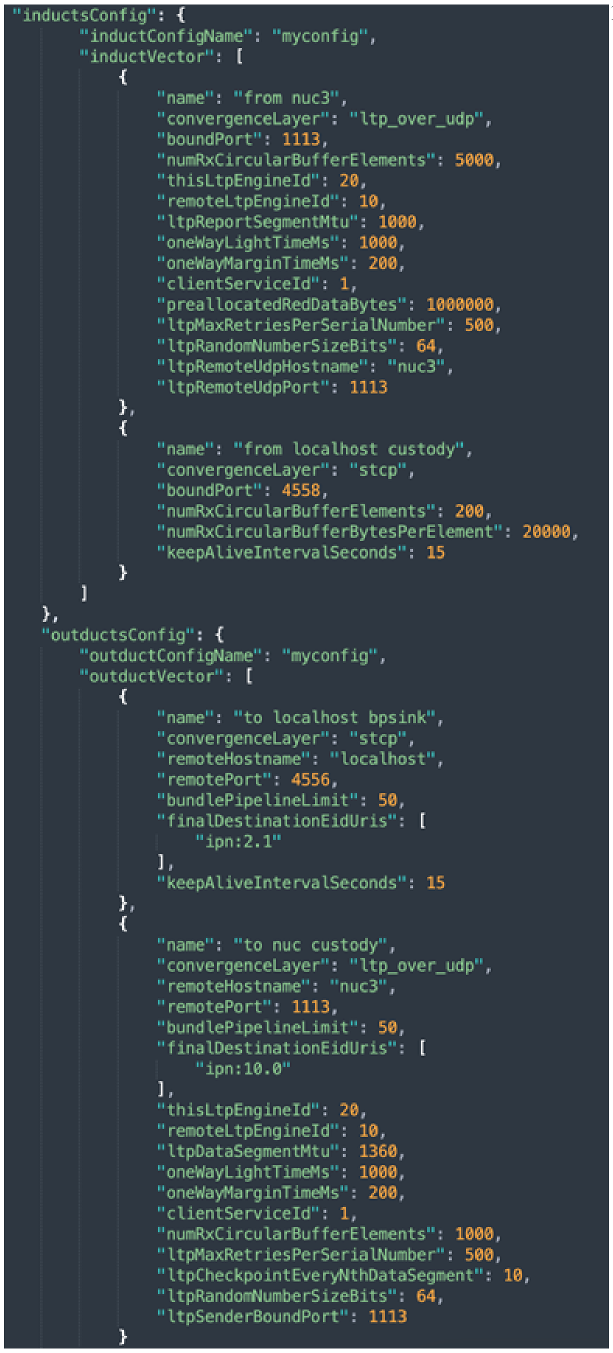
[24] Edward J. Birrane and Kenneth McKeever. *Bundle Protocol Security Specification*. Internet-Draft draft-ietf-dtn-bpsec-27. Work in Progress. Internet Engineering Task Force, Feb. 2021. 43 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-dtn-bpsec-27.

Appendix



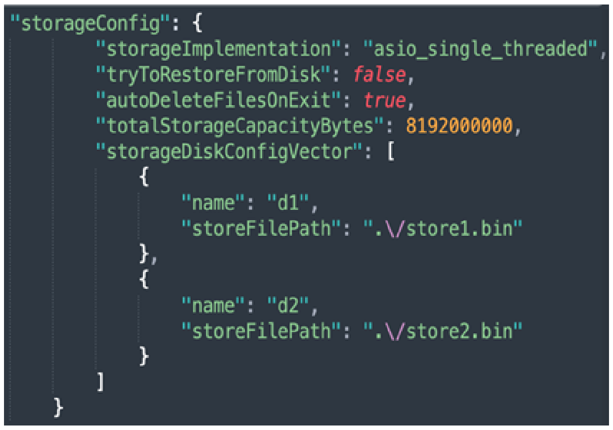
```
"inductsConfig": {
    "inductConfigName": "myconfig",
    "inductVector": [
        {
            "convergenceLayer": "stcp",
            "boundPort": 4556,
            "numRxCircularBufferElements": 200,
            "numRxCircularBufferBytesPerElement": 20000,
            "keepAliveIntervalSeconds": 15
        },
        {
            "name": "custody from tantalus hdtn",
            "convergenceLayer": "ltp_over_udp",
            "boundPort": 1113,
            "numRxCircularBufferElements": 500,
            "thisLtpEngineId": 10,
            "remoteLtpEngineId": 20,
            "ltpReportSegmentMtu": 1000,
            "oneWayLightTimeMs": 1000,
            "oneWayMarginTimeMs": 200,
            "clientServiceId": 1,
            "preallocatedRedDataBytes": 10000,
            "ltpMaxRetriesPerSerialNumber": 500,
            "ltpRandomNumberSizeBits": 64,
            "ltpRemoteUdpHostname": "tantalus",
            "ltpRemoteUdpPort": 1113
        }

    ]
},
 "outductsConfig": {
    "outductConfigName": "myconfig",
    "outductVector": [
        {
            "name": "to tantalus",
            "convergenceLayer": "ltp_over_udp",
            "remoteHostname": "tantalus",
            "remotePort": 1113,
            "bundlePipelineLimit": 50,
            "finalDestinationEidUris": [
                "ipn:2.1"
            ],
            "thisLtpEngineId": 10,
            "remoteLtpEngineId": 20,
            "ltpDataSegmentMtu": 1360,
            "oneWayLightTimeMs": 1000,
            "oneWayMarginTimeMs": 200,
            "clientServiceId": 1,
            "numRxCircularBufferElements": 1000,
            "ltpMaxRetriesPerSerialNumber": 500,
            "ltpCheckpointEveryNthDataSegment": 10,
            "ltpRandomNumberSizeBits": 64,
            "ltpSenderBoundPort": 1113
        },
        {
            "name": "to localhost bpgen custody",
            "convergenceLayer": "stcp",
            "remoteHostname": "localhost",
            "remotePort": 4558,
            "bundlePipelineLimit": 50,
            "finalDestinationEidUris": [
                "ipn:1.0"
            ],
            "keepAliveIntervalSeconds": 15
        }
```

Figure 24. Flight computer configuration: LTP with custody



```
"inductsConfig": {
       "inductConfigName": "myconfig",
       "inductVector": [
           {
               "name": "from nuc3",
               "convergenceLayer": "ltp_over_udp",
               "boundPort": 1113,
               "numRxCircularBufferElements": 5000,
               "thisLtpEngineId": 20,
               "remoteLtpEngineId": 10,
               "ltpReportSegmentMtu": 1000,
               "oneWayLightTimeMs": 1000,
               "oneWayMarginTimeMs": 200,
               "clientServiceId": 1,
               "preallocatedRedDataBytes": 1000000,
               "ltpMaxRetriesPerSerialNumber": 500,
               "ltpRandomNumberSizeBits": 64,
               "ltpRemoteUdpHostname": "nuc3",
               "ltpRemoteUdpPort": 1113
           },
           {
               "name": "from localhost custody",
               "convergenceLayer": "stcp",
               "boundPort": 4558,
               "numRxCircularBufferElements": 200,
               "numRxCircularBufferBytesPerElement": 20000,
               "keepAliveIntervalSeconds": 15
           }
       ]
   },
   "outductsConfig": {
       "outductConfigName": "myconfig",
       "outductVector": [
           {
               "name": "to localhost bpsink",
               "convergenceLayer": "stcp",
               "remoteHostname": "localhost",
               "remotePort": 4556,
               "bundlePipelineLimit": 50,
               "finalDestinationEidUris": [
                   "ipn:2.1"
               ],
               "keepAliveIntervalSeconds": 15
           },
           {
               "name": "to nuc custody",
               "convergenceLayer": "ltp_over_udp",
               "remoteHostname": "nuc3",
               "remotePort": 1113,
               "bundlePipelineLimit": 50,
               "finalDestinationEidUris": [
                   "ipn:10.0"
               ],
               "thisLtpEngineId": 20,
               "remoteLtpEngineId": 10,
               "ltpDataSegmentMtu": 1360,
               "oneWayLightTimeMs": 1000,
               "oneWayMarginTimeMs": 200,
               "clientServiceId": 1,
               "numRxCircularBufferElements": 1000,
               "ltpMaxRetriesPerSerialNumber": 500,
               "ltpCheckpointEveryNthDataSegment": 10,
               "ltpRandomNumberSizeBits": 64,
               "ltpSenderBoundPort": 1113
           }
```

Figure 25. Ground computer configuration: LTP with custody



```
"storageConfig": {
        "storageImplementation": "asio_single_threaded",
        "tryToRestoreFromDisk": false,
        "autoDeleteFilesOnExit": true,
        "totalStorageCapacityBytes": 8192000000,
        "storageDiskConfigVector": [
            {
                "name": "d1",
                "storeFilePath": ".\/store1.bin"
            },
            {
                "name": "d2",
                "storeFilePath": ".\/store2.bin"
            }
        ]
    }
```

Figure 26. Storage configuration

Biographies

Alan Hylton should probably be designing tube audio circuits, but instead directs Delay Tolerant Networking (DTN) research and development at the NASA Glenn Research 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.

Jacob Cleveland is a Senior studying Mathematics and Computer Engineering at the University of Nebraska at Omaha. They joined the Secure Networks, System Integration and Test Branch as a Pathways Intern at NASA Glenn Research Center in 2020. Since joining, they have performed several research projects applying pure mathematics to engineering problems such as networking in space, star tracking, and artificial neural networks.

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.

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.

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 communication. He holds a Master of Science degree in Computer and Information Science from the University of Michigan.

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 careers in the research and development of applied combustion systems and materials laser diagnostics, as well as specialized building infrastructure engineering.

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.

Robert Short earned his PhD in mathematics from Lehigh University in 2018. He worked as a Visiting Assistant Professor of Mathematics at John Carroll University until he joined the Secure Networks, System Integration and Test Branch at NASA Glenn Research Center in 2020. His research interests lie in the intersection of abstract mathematics and real world applications. Currently, his focus is on the foundations of networking theory and how to efficiently route data through a network using local information.

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.

Adam Wroblewski Wroblewski has been a research engineer in the Optics and Photonics Branch at the NASA Glenn Research Center for the past several years. He has supported and led many assignments at NASA ranging from jet nozzle noise reduction research, to supersonic parachute deployment testing for the Mars Science Laboratory (MSL), providing optical flow diagnostics and computational fluid dynamics analyses. Currently, Adam works in the areas of airborne laser communications to establish aircraft based laser-comm testbeds, and in and deep-space laser communication towards developing space based laser-comm flight prototypes. Dr. Wroblewski received his B.S., M.S., and Dr.Eng. degrees from Cleveland State University with focus in model-based and experimental rotordynamics, and robust control.