[SSC23-IX-08]

Space Networking Implementation for Lunar Operations

Nathan Drzadinski, Purdue University, ndrzadinski@gmail.com
Stephanie Booth, Blake LaFuente, Daniel Raible, NASA Glenn Research Center

Abstract

The High-Rate Delay Tolerant Networking (HDTN) project at NASA has developed a performance optimized and open-source Delay Tolerant Networking (DTN) implementation. The primary goal is to create a scalable networking solution to increase the scientific data return rate of space missions. To reach this goal, HDTN must span multiple edge cases in space networking by including tools and configurations to accommodate a wide range of space systems. Typically, HDTN evaluations are conducted on a laboratory emulation test bed, made up of hardware accelerated x86 based systems capable of data rates over 10 Gbps. HDTN must have an effective implementation process on a wide range of systems to increase the sustainability of the design.

One important implementation option is with low-level embedded systems which could be used on small robotic missions. This paper details the implementation process, benchmark testing, and performance results of HDTN in multiple configurations on Raspberry Pi 4 devices. By implementing HDTN on a Raspberry Pi 4, a process for building HDTN onto ARM processors was developed and utilized to conduct benchmark tests in multiple network configurations, achieving a data rate performance exceeding 600 Mbps. Based on these results, HDTN proved to run on small ARM based systems with slight modifications to the build procedure. These results were then extended to evaluating an implementation of the HDTN software parsed across several Raspberry Pi 4 nodes. To test this capability, HDTN was configured in a simplified cut-through setup and distributed among multiple Raspberry Pi 4 processors. This distributed architecture was benchmark tested in a similar fashion to the testing of a singular HDTN implementation. The results from the benchmark testing are used to examine how these implementation options and capabilities can expand the use cases for DTN, and particularly with small robotic missions.

Introduction

The High-Rate Delay Tolerant Networking (HDTN) project at NASA aims to be capable of supporting high-rate communications in space, an imperative quality for a future space internet and increasing the data throughput of all space missions. At a high level, a single HDTN node is made up of 5 main services (see Figure 1): Ingress, Storage, Scheduler, Router, and Egress.[3] Each service interacts and communicates as internal ZMQ sockets. ZMQ is a concurrency framework built to create distributed applications while minimizing latency and maximizing reliability.[11] During this particular research and testing, a cut-through version of HDTN was used. This bypasses the Storage, Scheduler, Router, Contact Graph Routing Client, and Contact Plan. The cut-through setup eliminates any potential decreases in data rates by avoiding a solid-state drive (SSD) storage system which could limit rate performance.

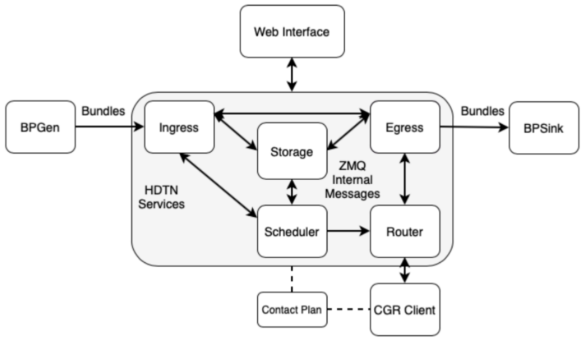

Figure 1: High Level HDTN Architecture

The HDTN software is open-source and is publicly accessible on the HDTN GitHub.[6] To achieve its goals, HDTN must have robust implementation capabilities with an array of different computer systems, such as supporting ARM processor systems and a multitude of edge cases. This is essential as NASA and other entities start colonizing the moon and increase lunar surface robotic presence.

The Raspberry Pi is a low cost, credit-card sized single board computer that contains a quad-core Cortex-A72 (ARM v8) processor. The Raspberry Pi Foundation claims that the Raspberry Pi has the capabilities of a typical desktop computer while being accessible to people of all ages and levels of knowledge.[10] Because of this, Raspberry Pis are moving into a plethora of industries and applications, including space. One example of this is the Raspberry Pi Foundation's strong partnership with the European Space Agency (ESA) with their Astro Pi challenges. The challenge inspires young Raspberry Pi users by giving them an opportunity to have their code run on Raspberry Pis(Astro Pis) aboard the ISS.[1] Another example is how Raspberry Pis are being implemented aboard CubeSats, such as Utah State University's Get Away Special Passive Attitude Control Satellite (GAS-PACS) mission, operational during 2022.[9] Raspberry Pis have become prevalent too the point where NASA JPL has a guideline specifically for using Raspberry Pis in space applications.[2] Due to the ease of use, projected relevance, and differences from the typical HDTN test benches, Raspberry Pis were an obvious choice for testing HDTN's software.

The Raspberry Pi 4 was used throughout integration and testing. Each device ran the Ubuntu Desktop 64-bit operating system (OS) because HDTN has the most heritage on the Ubuntu OS. Therefore it was presumed, the installation, build, and testing processes' issues would minimized. Once HDTN was built on its first ARM system, performance evaluations in multiple network architectures and three convergence layers configurations were performed. To standardize all benchmark testing, each benchmark test followed these specifications:

- 5 minutes run time,
- Cut-through version of HDTN,
- No Linux optimizations

Note, running optimization would cause all the tested rates to greatly improve but were not necessary for this initial benchmark testing.

1 Simple End-to-end Raspberry Pi Architecture

1.1 High-Rate Delay Tolerant Networking on an ARM Processor

HDTN was built on large x86 processors, so the established build process was not able to be executed. The steps for building HDTN had to be rewritten specifically for Raspberry Pis. Within the build process for HDTN, the typical CMakeCache.txt edits must be made alongside additional modifications. Since the Ubuntu Desktop ran in 64-bit and not 32-bit during our testing, a position independent command line option (-fPIC) must be added to the `CMAKE CXX FLAGS RELEASE:STRING=-O3 -DNDEBUG` line in the CMakeCache.txt file. Also, HDTN was developed on machines running x86 processors which use a Complex Instruction Set Computer (CISC) CPU design while ARM processors use a Reduced Instruction Set Computer (RISC) CPU design. To alleviate this problem, the `x86_HARDWARE_ACCELERATION` and the `LTP_RNG_USE_RDSEED` must be set to `BOOL=OFF` inside the CMakeCache.txt file. The last modification that must be made is due to the cpuid.h library used to conduct a CPU Flag Detection test, one of the unit-tests for HDTN. However, this library only works with x86 processor systems, and therefore the CPU Flag Detection test fails. There are two places where the CPU Flag Detection test has to be commented out of the script from the unit-test call and the common utility function. Once these modifications are performed, the Raspberry Pi 4s will pass all unit tests and be ready to run benchmark tests.

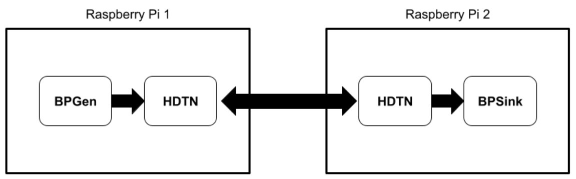

Figure 2: Raspberry Pi to Raspberry Pi Architecture

Convergence Layer	TCPCL	STCP	LTP over UDP
Payload Only Rate (Mbits/sec)	508.2815833	682.5734	107.7537833
Total Rate (Mbits/sec)	508.6373833	683.0511833	107.8292333
Highest Rate (Mbits/sec)	557.9048	841.7317	111.6745
Lowest Rate (Mbits/sec)	412.9462	466.9799	95.8367
Bundle Count	197674.3333	268352.3333	42731.66667
Bundle Rate (Bundles/sec)	658.914444	894.5077778	142.4388889

Table 1: Data Table of Initial Raspberry Pi Benchmark Testing Averages

1.2 Testing

The test architecture used two Raspberry Pis, (1) a sender Pi hosting BPGen and a HDTN node and (2) a receiver Pi hosting a second HDTN node and BPSink. BPGen is the bundle generator, and BPSink is the bundle receiver (see Figure 2). The tests were configured with each of the convergence layers: TCPCL v4, STCP, and LTP over UDP. Each configuration benchmark test was performed three times to detect outliers, as well as an additional three times reversing the Raspberry Pis (i.e., switching sender/receiver roles) to detect Raspberry Pi specific board discrepancies. In addition, each test measured the payload exclusive data rate, the total data rate, the highest recorded rate, the lowest recorded rate, the bundle count and the bundle rate.

To benchmark the effectiveness of the HDTN nodes, a total of 18 tests were conducted. To remove skewed data caused by startup and shutdown, the first and last data points were removed from the analyzed data set. The average data for all the 18 tests can be seen in Table 1. To see how well HDTN functions, look to the payload data rate and bundle rate. The highest and lowest rates give the range that would be valid in a new implementation. From this data, it was clear that the direction the data traveled had no effect on the data rate.

2 Distributed Raspberry Pi Architecture

2.1 Parsing HDTN

A possible use case for HDTN could be where a single node of HDTN is distributed among multiple processors. This would be done by splitting apart HDTN at the ZMQ level. A system like this would most likely run on multiple low-level computers where a single processor might struggle handling the entirety of a HDTN node but be capable of handling a single sub-service. To test this type of implementation, it made sense to use the Raspberry Pis as the test bed. While the Raspberry Pi 4 proved it could handle an entire HDTN node, it is still a low-level ARM system that will allow for easier extrapolation to other low-level systems. To set this up, the individual HDTN json ZMQ configuration files had to be edited (see Section 10 of the User Guide[4]). The ingress has to port its output to the other Raspberry Pi running the egress and the egress had to accept the ingress from the other Raspberry Pi.

To benchmark the effectiveness of the distributed HDTN nodes on the Raspberry Pis, similar tests were conducted as with the simple end-to-end tests. Each test measured the payload data rate, the total data rate, the highest recorded rate, the lowest recorded rate, the bundle count, and the bundle rate. The tests were configured with each of the convergence layers: TCP, STCP, and LTP over UDP. Each configuration was operated three times and had their data results averaged.

2.2 Testing

A few parsed architectures were tested when exploring the possibilities of a distributed version of HDTN. First, a parsed version of HDTN was attempted in the most simple structure. Two Raspberry Pis host a single HDTN node, one ran BPGen and the HDTN Ingress while the other ran HDTN Egress and BPSink (see Figure 3). This was done to minimize the complexity of the set up and focus on making sure a parsed version was possible without inhibiting HDTN's functionality. A total of 9 tests were conducted. Again, the first and last data points were removed to eliminate discrepancies due to start-up and shutdown transients, and the resultant data of the individual configuration tests were averaged (see Table 2).

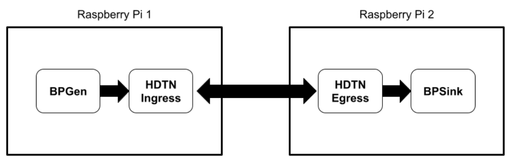

Figure 3: 2 Raspberry Pi Distributed Benchmark Architecture

Convergence Layer	TCPCL	STCP	LTP over UDP
Payload Only Rate (Mbits/sec)	520.8909333	735.717866	109.3312667
Total Rate (Mbits/sec)	521.2556	736.2328667	109.4078
Highest Rate (Mbits/sec)	553.1031	887.3295	112.4759
Lowest Rate (Mbits/sec)	481.2641	595.314	106.3956
Bundle Count	209263.6667	267692.3333	42175.66667
Bundle Rate (Bundles/sec)	697.5455556	892.3077778	140.5855556

Table 2: Data Table of Figure 3 Benchmark Testing Averages

Comparing the distributed values obtained to those from the simple end-to-end tests (see Table 1), the payload data rates experienced negligible change with a minor exception of STCP which increased by about 7%. These data rates prove the viability of a parsed system within a HDTN network. Looking closer at the data, an increase of the data rate was to be expected from only running a single HDTN node instead of two. This could point to STCP speeds being more effected by the number of nodes in a HDTN network than TCPCL and LTP over UDP. The testing produced respectable performance results; however, running BPGen or BPSink on the same Pi as the parsed HDTN node does not reflect how the system would work in real world scenarios. A second architecture using 4 devices running each piece separately would make for a stronger characterization of the parsed system. Three Raspberry Pis were used along with a laptop. The laptop ran BPGen while one Pi ran HDTN Ingress, a second Pi ran HDTN Egress, and a third Pi ran BPSink (see Figure 4). The laptop running BPGen would not be processing any data, thus not impacting the tests results. The same benchmark tests that were run for the previous parsed architecture were used here.

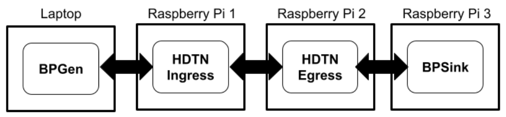

Figure 4: Isolated 2 Raspberry Pi Distributed Benchmark Architecture

Convergence Layer	TCPCL	STCP	LTP over UDP
Payload Only Rate (Mbits/sec)	503.3108667	640.9643333	106.8426333
Total Rate (Mbits/sec)	503.6631667	641.413	106.9174
Highest Rate (Mbits/sec)	517.5834	761.4155	108.7951
Lowest Rate (Mbits/sec)	452.6731	554.7908	103.6755
Bundle Count	194982.3333	248444.6667	41414
Bundle Rate (Bundles/sec)	649.9411111	828.1488889	138.0466667

Table 3: Data Table for Figure 4 Benchmark Testing Averages

This second architecture produced interesting results. The TCPCL and LTP over UDP again had negligible payload data rate effects, and the STCP's payload rate decreased by about 13%. This similarly proved the viability of a parsed system in a HDTN network with negligible impact on the performance. Analyzing further, the STCP payload decrease seems to be correlated to the number of devices running HDTN in addition to the number of HDTN nodes. Continuing with testing, it made sense to run an architecture which would be more directly comparable to the initial simple end to end HDTN on two Raspberry Pis. This third architecture used 3 Raspberry Pis: one running BPGen and Ingress, one running just the Egress and one running a full HDTN node and BPSink (see Table 5). The same benchmark tests were run as the previous two parsed architectures.

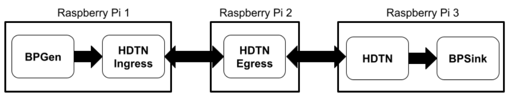

Figure 5: 2 Raspberry Pi Distributed to Raspberry Pi Benchmark Architecture

Convergence Layer	TCPCL	STCP	LTP over UDP
Payload Only Rate (Mbits/sec)	502.8962333	666.868	108.7951
Total Rate (Mbits/sec)	503.2482333	667.3348	108.8712333
Highest Rate (Mbits/sec)	550.5435	787.659	112.3157
Lowest Rate (Mbits/sec)	465.7473	554.5297	102.5557
Bundle Count	193100.3333	259185	42917
Bundle Rate (Bundles/sec)	643.6677778	863.95	143.0566667

Table 4: Data Table of Figure 5 Benchmark Testing Averages

These tests found that parsed HDTN had negligible differences in payload data rate from the initial two whole HDTN node tests. This solidified the concept of parsed HDTN having negligible impact on network performance. The biggest difference was in the STCP tests with a decrease in speed by 3%. As seen with the previous two architecture tests, this is probably attributed to the one additional device operating in the network.

3 Documentation for Implementation

The general Raspberry Pi setup requirements are accessible from the Raspberry Pi Foundation website.[10] These are: download the desired OS image (Ubuntu) to the Micro SD card, boot up the Raspberry Pi, and complete the set up instructions given to the user from the OS. Before downloading the HDTN code from GitHub and performing the build, make sure to update and upgrade the Pi. The process for building a single Raspberry Pi HDTN node, as explained in section 1.1, is documented in full at the publicly accessible HDTN GitHub[6] and the user guide found there.[4] This will provide the necessary package installations, the list of package download commands with directory locations, the cmake build process for HDTN, and all the necessary lines to edit. To make sure the system is working properly, run the unit tests also documented on the GitHub. In case of errors, there is an added extra section that details how to fix some common and reoccurring problems. The process for editing the configuration files to set up architectures such as the tested parsed systems are also located in these documents.

4 Small Robotic Missions

Low-level ARM systems, like a Raspberry Pi, are suitable for use on robotic systems all over the planet today. As more interplanetary robotic missions are being produced, they will become highly integrated into a space communications network. One relevant example of this is in the developments and progression of lunar infrastructure to support the Artemis program. This will bring these systems to the lunar surface where they will play a major part in lunar habitats, transportation, and exploration. In the farther future, these devices will go to Mars and beyond. Enabling HDTN on these systems will assist in providing networked capability to the space environment.

5 Conclusion & Future Work

HDTN is a versatile technology that can work in most space applications. Through this testing process, HDTN proved it can operate on low-level ARM based systems and in a parsed architecture. This opens the door to more universal compatibility and mission infusion. HDTN can be operated on a variety of new systems and thus use cases, one being small robotic missions. It was deduced that, in an emulation environment, STCP is affected by the number of devices and number of nodes in a network, but not by direction of data flow. For TCPCL and LTP over UDP the direction of data flow, number of devices, and number of nodes in the network have negligible impact on payload data rates.

There are many more future exploration and investigation opportunities for HDTN implementations like these. Listing a few, a thorough investigation into how STCP payload data rates are affected by different architectures would be a good starting point. An opportunity exists for further data rate characterization. Second, a deeper exploration into larger networks running on low-level ARM systems such as Raspberry Pis would be insightful, and could probably go with the STCP investigation. Third, the small robotic test bed at GRC could have HDTN implemented on it. This test bed could be connected to the onsite HDTN network, take pictures, and transmit them on the emulation network. Next, as noted in the introduction, HDTN has not been optimized for these initial tests. A complete optimization could yield large increases in data rates, and show the maximum potential of a Raspberry Pi HDTN node. Finally, an investigation into determining the minimum requirements for a system running HDTN could be pursued as the Raspberry Pis were capable of handling the HDTN software.

Acknowledgments

The authors would like to thank the NASA Space Communications and Navigation (SCaN) program and the Delay Tolerant Networking (DTN) project for supporting this work.

References

[1] Astropi, astro-pi.org/.

[2] Guertin, Steven M. NASA, Raspberry Pis for Space Guideline, 2021, nepp.nasa.gov/docs/papers/2021-Guertin-Raspberry-Pi-Guideline-CL-21-5641.pdf.

[3] Hylton, Alan, et al. "IEEE Aerospace Conference 2022." NASA, New Horizons For A Practical And Performance-Optimized Solar System Internet, 1 Mar. 2022, ntrs.nasa.gov/api/citations/20220003634/downloads/

[4] LaFuente, Blake, et al. High-Rate Delay Tolerant Networking (HDTN) User Guide Version 1, Apr. 2023, ntrs.nasa.gov/api/citations/20230000826/downloads/ 2023000826.pdf.

[5] NASA's Moon to Mars Strategy and Objectives Development, www.nasa.gov/sites/default/files/atoms/files/m2m_st

[6] "NASA/HDTN: High-Rate Delay Tolerant Network (HDTN) Software." GitHub, github.com/nasa/HDTN.

[7] Raspberry Pi. "Raspberry Pi 4 Model B Specifications." Raspberry Pi, www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/.

[8] Schlieder, Sarah. "Pave the Way for Artemis: Send NASA Your Mini Moon Payload Designs." NASA, 9 Apr. 2020, www.nasa.gov/mini-moon-payload-designs.

[9] University, Utah State. "GAS-PACS CubeSat: Projects: Gas: Physics." Utah State University, www.usu.edu/physics/gas/projects/gaspacs.

[10] "What Is a Raspberry Pi?" Raspberry Pi, 20 Aug. 2015, www.raspberrypi.org/help/what-%20is-a-raspberry-pi/.

[11] ZeroMQ, zeromq.org/.