IMPLEMENTING DELAY-TOLERANT NETWORKING AT MOREHEAD STATE UNIVERSITY

A Thesis

Presented to

the Faculty of the College of Science

Morehead State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Nathaniel J. Richard

April 28, 2017

Accepted by the faculty of the College of Science, Morehead State University, in partial fulfillment of the requirements for the Master of Science degree.

____
Jeffrey A. Kruth
Director of Thesis

Master's Committee: ________________________________, Chair
Dr. Charles D. Conner

____
Dr. Benjamin K. Malphrus

____
Kevin Z. Brown

____
Date

IMPLEMENTING DELAY-TOLERANT NETWORKING AT MOREHEAD STATE UNIVERSITY

Nathaniel J. Richard
Morehead State University, 2017

Director of Thesis: ____
Jeffrey A. Kruth

Implementing Delay-Tolerant Networking at Morehead State University consists of the demonstration of delay-tolerant communication between Morehead's ground station and a small spacecraft telemetry emulator. Delay-tolerant networking or DTN is defined as a set of communication protocols that allow for the extension of Internet-like communication to systems that would otherwise operate on an ad-hoc basis because of disruptions in communication or operations over distances that are measured in light-seconds. A small spacecraft telemetry emulator is defined as one that utilizes a standard the same set of commands and telemetry responses as the complete spacecraft but exists as either a piece of software or a set of components on a benchtop. DTN has been tested before with spacecraft, such as NASA's EPOXI mission and the International Space Station, but these have only served as technology demonstrations and have not been an integral part of the spacecraft's operation. Implementing DTN at Morehead will show the capability of using DTN on a small spacecraft as an integral part of the spacecraft's operations. Furthermore, NASA is exploring having DTN as an integral part of spacecraft operations and has contracted Morehead to serve as a test case and serve to reduce risk. The NASA-developed implementation of DTN called the Interplanetary Overlay Network (ION) will serve as the primary means of using DTN at Morehead. Initial work has been performed locally and with the DTN Experimental Network (DEN) to test the functionality of operating DTN at Morehead's. A trade study and experimentation was done to select a network emulator that would simulate the connection between the ground station and the spacecraft. With support from the Jet Propulsion Laboratory (JPL) work is being done to test if it is possible to interface DTN with Morehead's ground station software. Development of the spacecraft emulator is still ongoing, but the requirements for it have been defined and Morehead's Lunar Ice Cube spacecraft will serve as a basis for the development of the emulator. The ongoing work to put in place DTN at Morehead will prove the possibility of operating a small spacecraft via DTN and serve as a low-cost risk reduction for future NASA capabilities.

Accepted by: ______________________________, Chair
Dr. Charles D. Conner

____
Dr. Benjamin K. Malphrus

____
Kevin Z. Brown

ACKNOWLEDGEMENTS

I would like to thank Leigh Torgerson, Greg Miles, and Scott Burleigh at JPL for all their help getting the DTN node setup and running, without them this project would have taken a lot longer. I also need to thank Dennis Iannicca for his help connecting the node to the DEN and tolerating all my networking questions. I owe Dr. Malphrus, Jay Wyatt, and Tim Pham for the internship at JPL that led to this project. I want to thank my family for all their love and support throughout the years. Lastly, I want to thank anyone I may have forgotten that helped me during this project.

Terminology

AMMOS	Advanced Multi-Mission Operations System
BP	Bundle Protocol
CCSDS	Consultative Committee on Space Data Systems
CFDP	CCSDS File Delivery Protocol
CGR	Contact Graph Routing
cm	centimeter
CRC	Cyclic Redundancy Check
DEN	DTN Experimental Network
DINET	Deep Impact Network Experiment
DIXI	Deep Impact eXtended Investigation
DSOC	Deep Space Operations Center
DSN	Deep Space Network
DTE	Direct to Earth
DTN	Delay/Disruption-Tolerant Networking
EM-1	Exploration Mission-1
EPOCH	Extrasolar Planet Observations and Characterization
EPOXI	EPOch and diXI
GEO	Geostationary Earth Orbit
GHz	gigahertz
GT	Ground Terminal
ION	Interplanetary Overlay Network
IP	Internet Protocol
ipn	Interplanetary Network
IPsec	Internet Protocol Security
ISS	International Space Station
JAXA	Japanese Aerospace Exploration Agency
JPL	Jet Propulsion Laboratory
kg	kilogram
LEO	Low Earth Orbit
LTP	Licklider Transmission Protocol
mm	millimeter
MOC	Mission Operations Center
MSPA	Multiple Spacecraft Per Aperture
NASA	National Aeronautics and Space Administration
PDU	Payload Data Unit
RF	Radio Frequency
SDR	Software Defined Radio
SELinux	Security-Enhanced Linux
SLS	Space Launch System
TCP	Transmission Control Protocol
UT	Unitdata Transport
UDP	User Datagram Protocol
VPN	Virtual Private Network
WAN	Wide Area Network
X-band	radio frequency that ranges from 8 to 12 GHz

Table of Contents

CHAPTER I

INTRODUCTION

1.1 General Area of Concern

In the past decade, the use of CubeSats has exploded with many applications in low Earth orbit, from simple experiments to constellations mapping the globe. Now many are looking beyond low Earth orbit to the Moon and elsewhere for applications of CubeSats. These CubeSats would join the family of spacecraft already exploring our solar system that is supported by NASA's Deep Space Network (DSN). The DSN is busy communicating with spacecraft across the solar system and the possibility of sending CubeSats beyond low Earth orbit will start to strain the DSN capabilities because a single launch can carry many CubeSats. So, JPL is looking at several ways to expand the DSN's capacity to support all these new spacecraft.

NASA is taking a three-prong approach to expanding the DSN's capacity. The first avenue is increasing automation of the DSN's capabilities. It starts with improving human interaction between the operators and the systems they are managing (Wyatt & Malphrus, 2016). By having displays that summarize the data coming, operators can better react to the information they have at hand and have a more hands-off approach. The DSN's automation infrastructure is 25 years old and needs updating, so it will be augmented to reduce an operator's workload (Wyatt & Malphrus, 2016). The last part of the DSN's automation is scheduling operators based on how many simultaneous passes can be run by an operator without taxing their abilities (Wyatt & Malphrus, 2016). These automations with help with additional capabilities being added to the DSN to support small spacecraft, especially CubeSats.

The next part of the DSN upgrades is adding to capabilities to the DSN to support small spacecraft. One part of increasing the DSN capabilities is adding university partners with antennas to the DSN. Morehead State University is one such university with its 21-meter antenna (Wyatt & Malphrus, 2015). Next, the DSN will be adding a capability called Opportunistic Multiple Spacecraft Per Aperture, Opportunistic MSPA. It is the ability to downlink data from multiple spacecraft with the view of one antenna. One spacecraft is defined as the "host" spacecraft, it is the one that is scheduled with the DSN (Wyatt & Malphrus, 2016). All spacecraft within the view of the antenna transmit down to the receiver and those transmissions are recorded to be later processed for telemetry extraction (Wyatt & Malphrus, 2016). The last part of the new capabilities is creating a fully-automated prioritized scheduling system for the DSN. The new scheduling system will allow mission critical phases to take priority over standard spacecraft operations (Wyatt & Malphrus, 2016). All these new capabilities will allow the DSN to reliable downlink telemetry from CubeSats without increasing strain on the DSN, but these capabilities can only go so far.

The last part of expanding the DSN's capacity is to add support for networked space missions and emerging communication standards (Wyatt & Malphrus, 2016). Networked space missions would mean spacecraft could be treated as another computer on a network instead of how they are treated now where each spacecraft is a unique entity that must have direct communication with Earth. The problem with creating networked space missions is the spacecraft is not always in view of Earth and transmissions can take a significant amount of time to propagate between Earth and the spacecraft. To solve these problems a new method of networking is being developed, called delay/disruption tolerant networking which can handle these problems inherent in space communication. NASA is supporting the development of this method of networking the underlying protocols needed for it to function.

1.2 Objectives

The objectives of this project are:

1. Configure and setup a DTN node at Morehead State University
2. Demonstration DTN communication between at node at Morehead and a node at JPL
3. Provide risk-reduction for NASA's future implementation of DTN within the DSN

1.3 Significance of the Study

EM-1 is the first launch of the SLS and it will be carrying CubeSats as thirteen secondary payloads (Evans, 2016). Morehead State is one of those payloads (Evans). To support its payload Morehead will be using its 21-meter antenna. Morehead's mission, Lunar Ice Cube, requires ranging information to enter lunar orbit, but the 21-meter antenna does not have that capability, so it is being upgraded to DSN capabilities (Wyatt & Malphrus, 2015). Morehead is now amid upgrading it to be DSN compatible. The process of becoming DSN compatible includes upgrading the RF feed to be able to transmit and receive at X-band frequencies, adding RF equipment capable of measuring the distance to spacecraft, connecting the antenna's systems to the NASA's network so information received at Morehead can be transferred to their destination, and acquiring a hydrogen MASER to provide precise timing signals to the new equipment. In addition to upgrading the RF capabilities of the antenna, a DTN node will be added to Morehead's antenna to provide DTN capabilities.

Lunar Ice Cube will be carrying an SDR capable of operating in X-band called IRIS. Because IRIS is an SDR it can be programmed to have the capabilities required for the mission it is on. In this case, Lunar Ice Cube will be serving as a demonstration of DTN on a CubeSat, so IRIS will be delivered with the capability to serve as a DTN node. It will not be an integral part of the mission, but it will be part of the secondary mission goals. For the DTN demonstration to be successful, there needs to be a DTN node on the ground. Setting up one at Morehead's antenna, an asset Morehead controls, would allow the demonstration to have a higher chance of success because both assets are well understood at Morehead, which would allow for quicker problems resolution.

1.4 Definition of Terms

CubeSats are those satellites that have a mass of under 1.33 kg per 10 cm x 10 cm x 10 cm volume otherwise known as One Standard Unit, or 1U (CubeSat Design Specification (CDS) REV 13, 2014). This satellite classification was developed at California Polytechnic State University (Cal Poly) in 1999 as a means of standardizing small satellite architectures across the entire small satellite industry. This served to facilitate reduced costs and time associated with the development of small satellite missions, thus allowing for organizations that would have

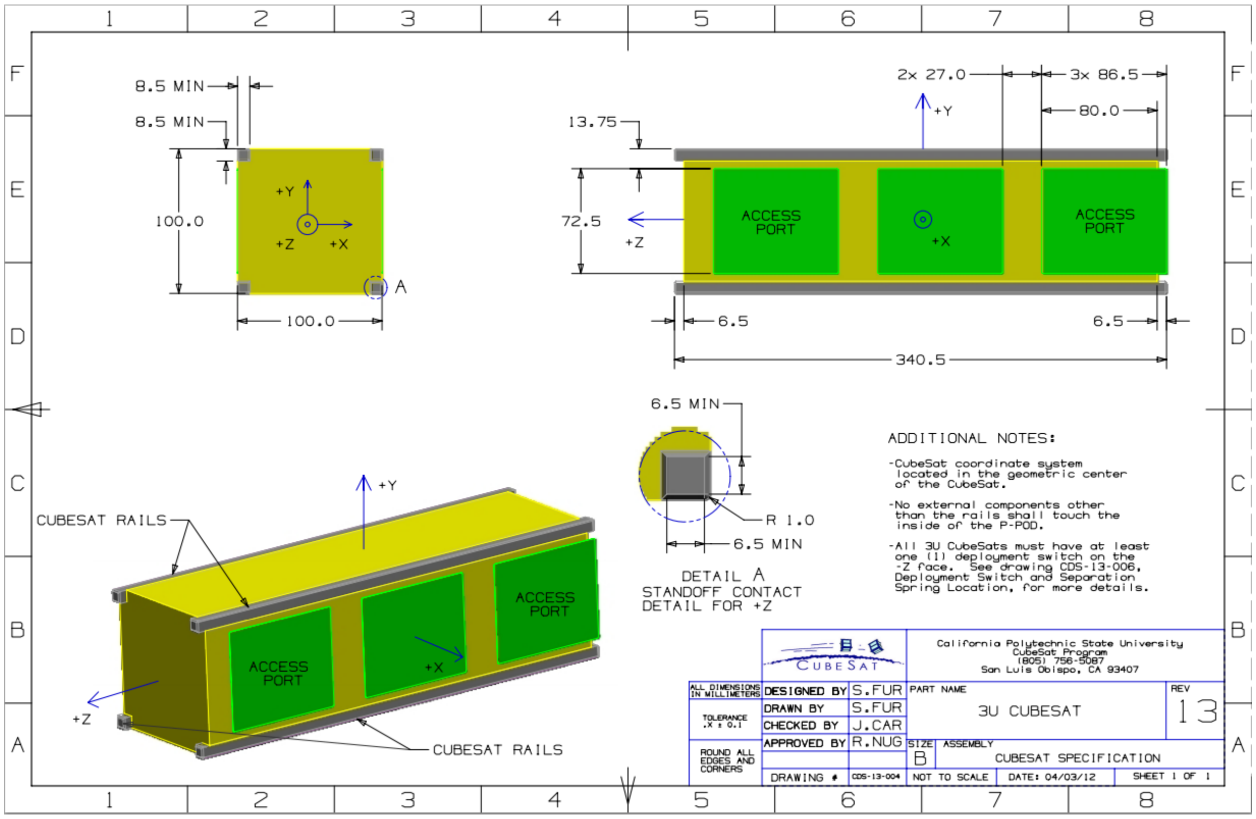
8.5 MIN
8.5 MIN
100.0
100.0
+Y
+Z
+X
A
13.75
72.5
+Z
2x 27.0
3x 86.5
80.0
+Y
ACCESS PORT
+X
ACCESS PORT
6.5
6.5
340.5
6.5 MIN
R 1.0
6.5 MIN
DETAIL A
STANDOFF CONTACT
DETAIL FOR +Z
ADDITIONAL NOTES:
-CubeSat coordinate system located in the geometric center of the CubeSat.
-No external components other than the rails shall touch the inside of the P-POD.
-All 3U CubeSats must have at least one (1) deployment switch on the -Z face. See drawing CDS-13-006, Deployment Switch and Separation Spring Location, for more details.
CUBESAT RAILS
+Y
ACCESS PORT
ACCESS PORT
+X
+Z
CUBESAT RAILS
CUBESAT
California Polytechnic State University
CubeSat Program
(805) 756-5087
San Luis Obispo, CA 93407
ALL DIMENSIONS IN MILLIMETERS
TOLERANCE .X ± 0.1
ROUND ALL EDGES AND CORNERS
DESIGNED BY S.FUR
DRAWN BY S.FUR
CHECKED BY J.CAR
APPROVED BY R.NUG
PART NAME 3U CUBESAT
REV 13
SIZE B
ASSEMBLY CUBESAT SPECIFICATION
DRAWING # CDS-13-004
NOT TO SCALE
DATE: 04/03/12
SHEET 1 OF 1

Figure 1: 3U CubeSat Architecture

previously not been able to develop and launch small spacecraft (Such as Universities and Privately Funded Corporations) to launch scientifically significant, impactful, low-cost missions. Nanosatellites come in several different sizes, ranging from 1U to 6U. An example of 3U nanosatellite architecture, as defined by the *CubeSat Design Specification Document, Rev 13* is shown in Figure 1. Though the vertical dimension of each configuration depends on its type, the width of any CubeSat is limited to 100 mm, thus imposing a limit on the size that a given nanosatellite can occupy (CubeSat Design Specification (CDS) REV 13, 2014).

A DTN node can refer to any number of things. DTN has the capability to run on anything from cell phones to Martian rovers, it can even be setup to be used with flash drives. It is a computer that takes bundles from another computer that is part of a delay-tolerant network and then if there is a link available will send it to the next part of the network, otherwise, it will hold onto the bundles until a link becomes available. Throughout this paper whenever a DTN node is mentioned, it will mean a Linux server running CentOS 7 with DTN running on it.

1.5 Summary

Chapter I provides an introduction into this thesis project be describing the general area of concern for implementing DTN at Morehead State University, the objectives of the project, and the projects significance. Also, provided in Chapter I was a brief outline of terms that the reader will encounter in this thesis report.

CHAPTER II

REVIEW OF LITERATURE

2.1 Background

What has allowed the Internet to become so ubiquitous is a standard set of protocols everyone must follow to transfer data. A couple of problems with those protocols is they were not designed to handle any disruptions in communication or operate over distances that are measured in light-seconds. Because of these problems deep space missions need to communicate directly through whichever DSN antenna complex is visible. DTN tries to overcome these problems and offer a way to bring the Internet to deep space.

The Internet uses an architecture that consists of several levels sometimes called a stack because of its graphical representation. It consists of four levels; the application level, the transport level, the network level, and the physical level (Mohr, 2009). The application level is where the user operates, such as sending emails or chatting online. The next level is the transport level, this level handles transportation of the data, ensuring all data arrives at its destination without errors (Mohr). The protocol the Internet uses is called Transmission Control Protocol (TCP) (Mohr). Underneath the transport level is the network level. The network level handles the routing of the data, and it ensures that all data arrives at the correct network address (Mohr). Internet Protocol (IP) is the network level protocol used by the Internet (Mohr). The final level is the physical level; this level is what physically connects the computers to each other (Mohr). It is sometimes referred to as the bit level, where the ones and zeroes exist. The transport and network layers together are referred to as the TCP/IP protocol (Mohr). DTN shares many similarities with the TCP/IP protocol.

The DTN stack is comparable to TCP/IP protocol stack, except below the application layer is another protocol called the bundle protocol (BP). The DTN protocol stack compared with the Internet protocol stack can be seen in Figure 2 (Burleigh, 2016). BP serves as an overlay protocol to connect other networks including networks using TCP/IP and networks using space communication protocols (Burleigh). Packets in a DTN are referred to as bundles and they serve the same purpose as IP packets, moving data between BP endpoints (Burleigh). To ensure transmission reliability, like TCP, DTN uses LTP (Burleigh). LTP offers reliability over delayed links such as communicating with deep space spacecraft. The approach to using these protocols is different than the TCP/IP protocol.

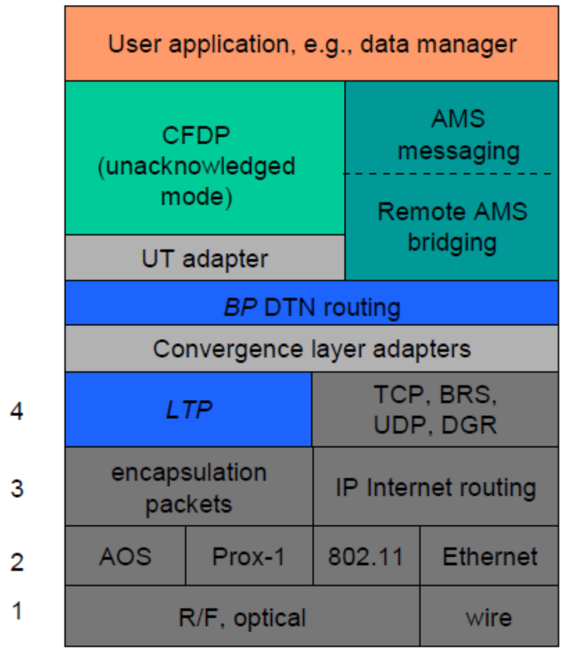

Figure 2: DTN Protocol Stack

While TCP/IP and DTN share similar protocol architectures, DTN has a different implementation. TCP is conversational because there are many messages passing between the source and destination as data is transmitted (Warthman, 2012). Acknowledgment messages and data transfers are handled by the source and destination with the intermediate nodes handling only the routing (Warthman). Because of the nature of delay-tolerant networks, the direct communication between the source and destination might not be possible. To ensure that data is reliably transmitted from the source to the destination, DTN implements store-and-forward message switching (Warthman). Each node in a delay-tolerant network not only routes data but also stores it when the next node in the route is unavailable. So, when a node receives data but does not receive an acknowledgment from the next node in the route it will hold it until it can forward the data. To provide reliability to this method the bundle protocol uses custody transfer, which is shown in Figure 3 (Warthman). When the node sends a bundle to the next node, a

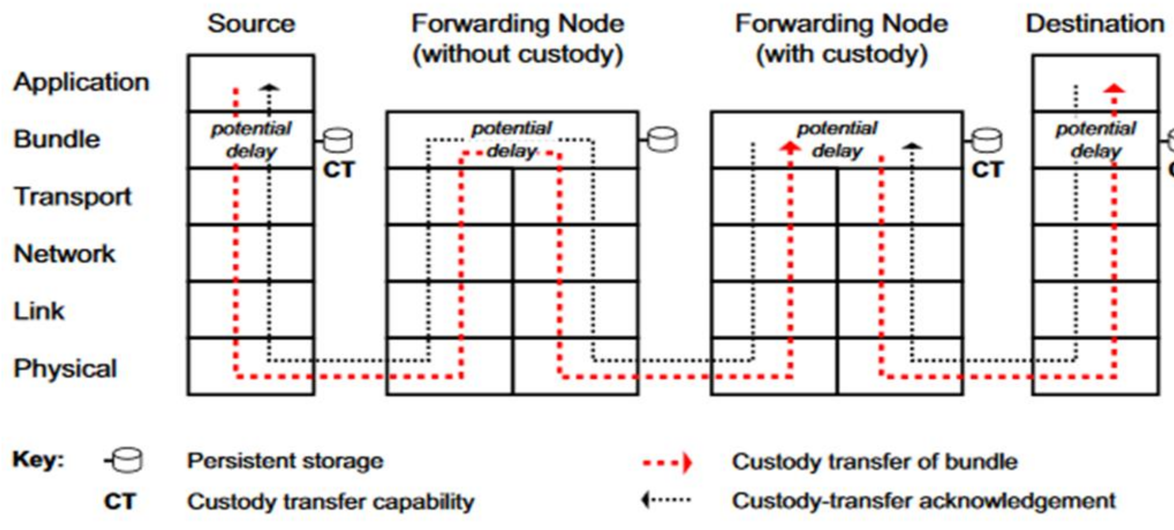
Source
Forwarding Node (without custody)
Forwarding Node (with custody)
Destination
Application
Bundle
Transport
Network
Link
Physical
potential delay
CT
Key:
Persistent storage
CT Custody transfer capability
Custody transfer of bundle
Custody-transfer acknowledgement

Figure 3: DTN Custody Transfer

custody transfer is requested and a time-to-acknowledge retransmission timer starts (Warthman). If custody is accepted, the receiving node sends an acknowledgment to the sending node (Warthman). If the sending node does not receive an acknowledgment before the time-to-acknowledge timer expires, it will retransmit the bundle (Warthman). The store-and-forward capability also serves as a backup, if the data is lost in transit the receiving node can request the data again without asking the source. DTN nodes can serve in two different capacities. They can be a source/destination node or a forwarding node (Warthman). The forwarding node can also act in two different capacities. It can just route data from one network to another with both using the same underlying protocols or it can serve as a gateway where it transfers the data from one network to another, but the networks use different underlying protocols (Warthman). Requirements needed at each node location will affect which type of node is deployed there.

NASA’s implementation of DTN is called ION. The basis of ION is both BP and LTP, but there are several other protocols included with it. One of the protocols is CFDP, it is a file delivery protocol that transfers files using the underlying DTN protocols to provide reliability. The file is broken up into payload data units, PDUs, and transmitted via the underlying DTN protocols to the destination to be reassembled into the file (Burleigh, 2016). There is a protocol included with ION that is in its early stages called the bundle security protocol that provides security to the bundles to prevent those who are not the intended recipient from reading the bundles. Along with these other protocols, ION has something called contact graph routing, CGR. CGR defines what DTN nodes are available for a node to contact, how long that contact will last, what data rate is between the two nodes, and how long the light time delay is between the nodes. A CGR is a predefined configuration file, which is fine for small networks but large networks will cause the configuration file to grow quite large. Since, storage space on DTN nodes can be sometimes be limited, such as on spacecraft, large configuration files are not desired. So, something called opportunistic CGR is being developed to support larger networks. In its current implementation, CGR is deterministic where the contact is known when it will happen with 100% certainty. Opportunistic CGR will include contacts that may or may not happen mean they have a probability of making contact less than one (Araniti, et al., 2015). Then copies of the bundles are forwarded to the nodes of all the opportunistically discovered routes that increase the probability of the bundles being delivered by more than a predefined threshold (Araniti, et al.). ION has already seen limited use in space with spacecraft and the International Space Station.

2.2 Space Communication Literature Review

DTN has had a limited demonstration in space with a demonstration with the EPOXI spacecraft, a demonstration with JAXA’s GEO relay satellite, and providing some support to experiments on the ISS. EPOXI was a JPL mission that reused the Deep Impact spacecraft, which had previously visited comet Temple 1, to visit another comet, Hartley 2 (Jpl.nasa.gov, 2017). The name EPOXI originated from the name of the two missions that would be conducted during the cruise and the flyby of Hartley 2, Extrasolar Planet Observations and Characterization (EPOCh) and Deep Impact eXtended Investigation (DIXI) (Discovery.nasa.gov, 2017). The demonstration of DTN with the EPOXI spacecraft was called Deep Impact Network Experiment, DINET, and took place during the cruise between Temple 1 and Hartley 2 (Wyatt, et al., 2009). JPL used it as a technology validation experiment of JPL’s DTN implementation, ION. The experiment was designed to have a minimal impact on EPOXI’s primary mission, so the software was installed on the backup flight computer within the backup software partition. EPOXI was the only on orbit node in DINET; the rest were simulated on Earth. The topology of DINET are shown in Figure 5 (Wyatt, et al.). It consisted of two surface assets, one on Mars and the other on Phobos; an orbital relay satellite, EPOXI; and the Earth ground station. Image files would be sent from “Mars” or “Phobos” and then relayed via EPOXI to Earth. As Figure 5 shows,

Figure 4: EPOXI Spacecraft

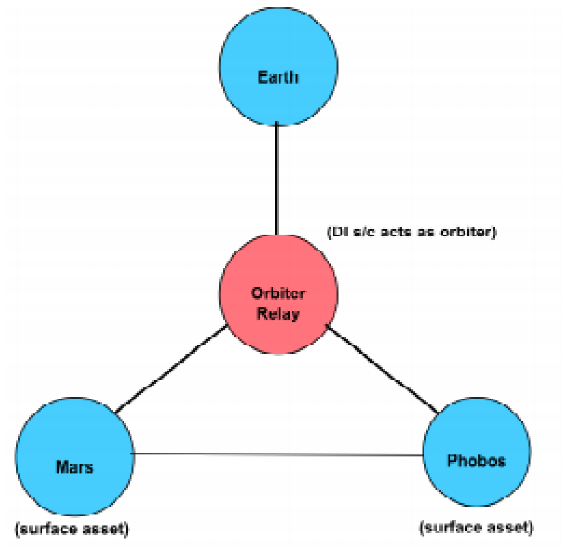

Figure 5: DINET Topology

there was also a crosslink that would be available at times for “Mars” and “Phobos” to communicate with each other. The EPOXI demonstration has so far been the only deep space demonstration of DTN and it was with artificial telemetry.

DTN has had one other demonstration with spacecraft in GEO with JAXA’s Data Relay Test Satellite. There were seven nodes which can be seen in Figure 6 on the right (Araniti, et al., 2015). It consisted of a relay spacecraft and a LEO spacecraft with two direct to Earth connections were the LEO spacecraft communicated with a ground station and two ground terminals that communicated with the relay satellite. This demonstration was used to show how CGR can deliver data to its destination and conform to the contact plans that were defined.

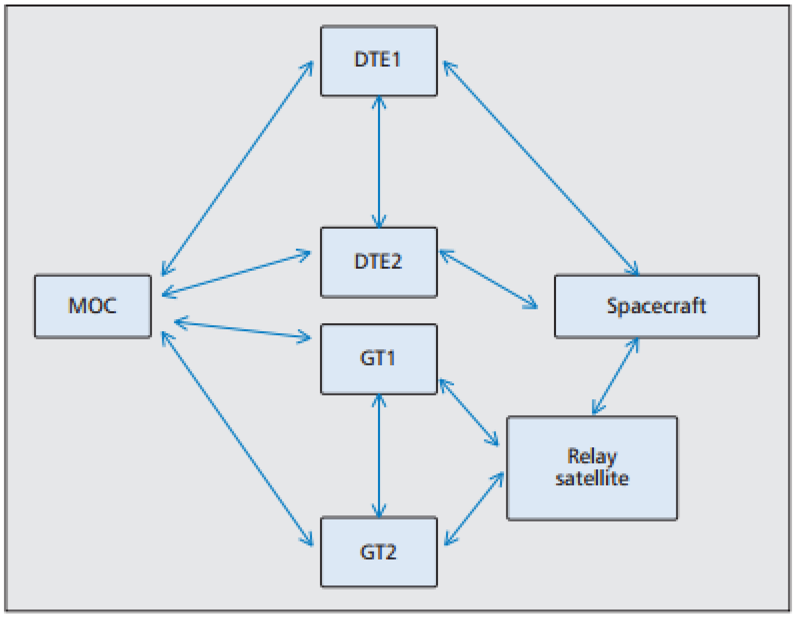

Figure 6: JAXA DTN Topology

While in space demonstrations have improved the technology readiness level of DTN, using DTN for actual missions shows how capable it is. DTN is currently being used on the ISS to support some science experiments. DTN has been implemented across all parts of the ISS program from flight and ground systems to testing and simulation (Willman & Davidson, 2014). It provides increased reliability to payload data transfers during signal acquisition/loss transitions (Willman & Davidson). DTN also provides better automation for data transfers from payloads on the ISS (Willman & Davidson). Using DTN relieves the support required to plan data transfers during signal acquisition/loss transitions and when those transfers require operators (Willman & Davidson). DTN has come to allow payload developers to retrieve their experiment data from the ISS like they would a file from Dropbox.

2.3 Summary

This chapter provides a technical background of how DTN operates and differentiate it from the architecture used by the Internet. It also tries to explain DTN in a way that someone unfamiliar with networking can understand. Chapter II also shows how NASA has already demonstrated DTN and its current use in space.

CHAPTER III

METHODOLOGY

3.1 Overview

To demonstrate the DTN interface is two part. The first part is showing file transfers are possible. This is a simple test to demonstrate that it is possible to transfer a file comparable to what Lunar Ice Cube would do while orbiting the Moon. The second part of the test would demonstrate telemetry exchange capability between applications that can write Space Packets to a socket, and applications that can read Space Packets from a socket, Space Packets are described below. This test would seek to demonstrate both the ability to send commands and receive streamed telemetry from Lunar Ice Cube. While these tests are straight forward there are several things to be done before the tests could be carried out.

There were several things that had to be done to prepare before any DTN testing could be done. To simulate light-time distances that would be experienced during spacecraft operations a WAN emulator had to be selected that could simulate the delays the data being transferred would experience. The node that simulates JPL's DSOC is at JPL and it requires a secure connection. NASA supports a DTN experimental network, known as the DEN, and will provide a secure connection to the JPL. Connecting to the DEN requires a VPN tunnel to it. So, a VPN tunnel had to be setup for the Morehead node. The IRIS radio will be the node on Lunar Ice Cube and it expects the data to be formatted into Space Packets. Space Packets are a CCSDS standard that was developed to transfer spacecraft telemetry and commands over a space link (Space Packet Protocol, 2003). To generate these Space Packets would either require Lunar Ice Cube or code that would generate faux Space Packets like what Lunar Ice Cube would generate. Lunar Ice Cube is not in at a point to generate telemetry, so code needed to be written to generate faux

Space Packets. In addition, the ground station software used by Morehead, AMMOS, needed to be installed. With these components in place, the project could continue.

3.2 System Overview

3.2.1 Project Architectures

3.2.1. File Transfer

The file transfer test only required ION and a file to be transferred. The format of the file does not matter. Figure 7 shows the connections for the demonstration. Once ION is started on both nodes, the file transfer can be run with an included command called *cfdptest*. Before a file can be sent *cfdptest* must be told the destination node, the file name on the source node, what the file should be named on the destination node, and the light-time delay between the nodes. *cfdptest* is not required to be run on both nodes, but running it on the destination node serves as a check on receiving the file.

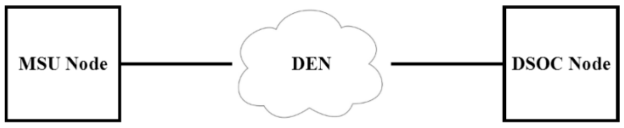

Figure 7: File Transfer Demonstration

3.2.1.2 Telemetry Exchange

The telemetry exchange test requires AMMOS to be installed on both nodes. Figure 8 shows the connection for the demonstration. The JPL node uses the command *chill_send_to_socket* to send some pre-generated telemetry to the Morehead node. The command requires the port number that the telemetry will be sent out, the IP address of the Morehead node, and the name of the file to be sent. Since the JPL node will be treated as a client sending to a remote server, it must also be specified to run *chill_send_to_socket* in client mode. The Morehead node uses the command *chill_get_from_socket* to receive the telemetry. The requirements for this command are the same

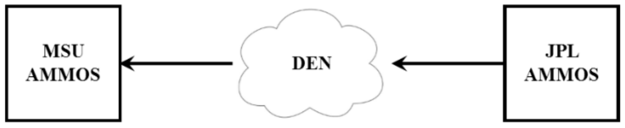

Figure 8: Telemetry Exchange Demonstration

as *chill_send_to_socket*, it requires the port number that the telemetry will be coming from, the IP address of the JPL node, and what to name file that will be received. It has the option to run in the default mode of running as a server or it can be run as a client. The Morehead node is treated as the server, so the default mode was used.

3.2.2 VPN Tunnel

The VPN tunnel is required to connect the Morehead node to the DEN. The SG-2220 was selected to serve as the gateway between the Morehead node and the DEN based off recommendation from a colleague at NASA because it is like hardware used there. It was configured to use IPsec to provide the VPN tunnel. In addition to configuring the VPN tunnel, a route had to be added to allow the use of both Ethernet ports on the Morehead node. The route was added using the Linux *route* command. It defines that all packets destined for the DEN need to go through one Ethernet port and all other packets go through the other Ethernet port. Related to the VPN tunnel was turning off a Linux security feature, called SELinux, included in the node's version of Linux. With it on DTN bundles could not go through the node's firewall.

Figure 9: SG-2220 Gateway for VPN tunnel

3.2.3 WAN Emulator

Some research was done to select a WAN emulator that would work for the project. A hardware solution would require at least $1,000, which the project did not have. So, a software solution was explored. There are a few software solutions available, but the one selected was already included with the Linux installation on the Morehead node. The WAN emulator is a built-in Linux command called *netem* it can add delay, packet loss, and other characteristics to packets going out a selected Ethernet port. A script was adapted to make it easier to use the command that adds delay and loss to packets leaving an Ethernet port. The script requires the input arguments of which Ethernet port to use, the packets destination that require the delay and loss, how long to delay the packets, and the percentage of packets to lose. See Appendix A for the code. During testing it was found to fail after having it run for days, but a cause could not be determined. The best course of action was to limit the length of the test runs.

3.2.4 Space Packets

The data format that is sent to the IRIS radio is in Space Packets. Figure 10 shows the structure of a Space Packet. It consists of a primary header, secondary header, and the user data field which contains the telemetry and science data. The total size of a Space Packet is 65.54 kilobytes including the header. At the time of this project Lunar Ice Cube was not using the secondary header. So, the header consists of 6 bytes and the user data field is rest of the packet. Figure 11 shows what is in the header. Code was written based on C++ header files to create Space Packets in the formats seen in Figures 10 and 11. It fills a user

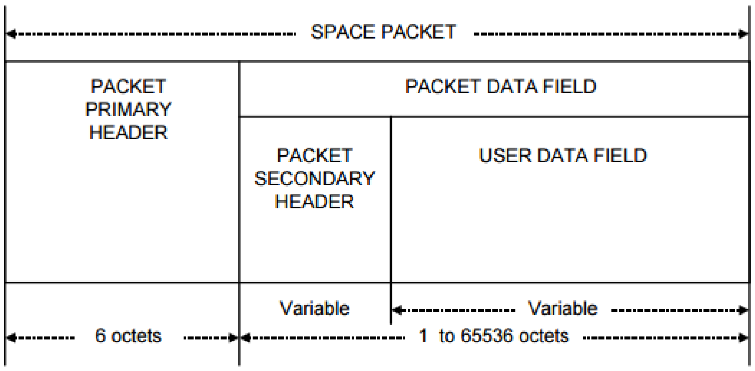

Figure 10: Space Packet Structure

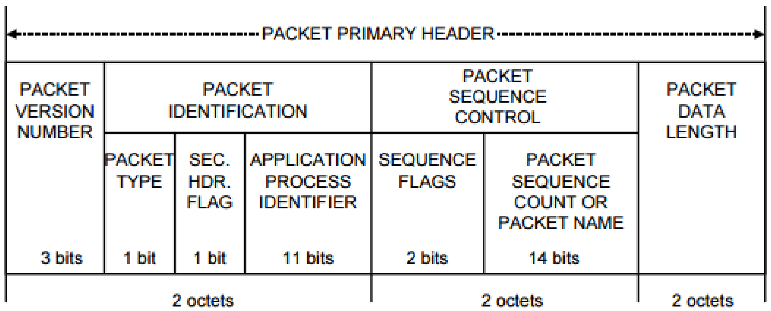

Figure 11: Space Packet Header Structure

specified number of packets with random numbers and sends it out a port using UDP. See Appendix B for the code.

3.3 Summary

This chapter covered several aspects of the configuration and setup of the project. It described the setup for the two tests and what was needed to support them. Also described is this chapter was how the support equipment was setup and the code that was needed to support the tests.

CHAPTER IV

FINDINGS AND ANALYSIS

4.1 Overview

With ION installed and the hardware configured both tests could be run. Part of running a node is the configuration files. The configuration files define the contacts available to the node, what underlying protocols to use with the contacts, and defining the storage for the node. The configuration file for a node consist of nine files and two scripts that start and stop the node. These configuration files are management commands that are to their respective administration interfaces, for example the commands contained in a .bprc file are passed to *bpadmin* which manages the BP operations for the node. It is important to have a standard naming convention for all the configuration files, especially if multiple nodes exist on one machine. For this project, all the configuration files were called node2, e.g. node2.bprc. The commands used for Morehead's node are discussed in the following section. Refer to Appendices C1 and C2 for the ION start and stop scripts.

4.2 Results

4.2.1 DTN Node Configuration Files

4.2.1.1 bprc

```
1
a scheme ipn 'ipnfw' 'ipnadminep'
a endpoint ipn:2.0 x
a protocol ltp 1400 100
a induct ltp 2 ltpcli
a outduct ltp 1 ltpclo
r 'ipnadmin node2.ipnrc'
w 1
s
```

The first config file to consider is the .bprc file that commands *bpadmin*, the process that runs the BP functions. It is broken up into several commands, which a portion of the .bprc file can be seen to the right. The rest of the .bprc file can be found in Appendix C3. The first command of the file is *1* this initializes *bpadmin* so

BP operations can function. The second command adds a scheme, which creates a naming scheme for endpoints similar to a socket. So, *a scheme* is the command, *ipn* is the naming scheme, *ipnfw* starts the daemon, or process, that will forward bundles, and *ipnadminep* will process custody signals and bundle status reports. The next command, *a endpoint*, creates an endpoint called ipn:2.0 and x ends the command. The endpoints are what are used to transfer to other nodes. There can be multiple endpoints. After the endpoint command is the define a protocol command, *a protocol*, which is what type of transmission protocol it should expect to receive. In the case above it is expecting LTP packets and it should expect them to contain 1400 bytes of payload per frame and 100 bytes of overhead per frame. The other protocols are TCP and UDP. *a induct* tells *bpadmin* to treat node 2 as an ingress point using the LTP convergence layer adapter. *a outduct* tells *bpadmin* to treat node 1 as an egress point using the LTP convergence layer adapter. The next .bprc command tells *bpadmin* to run the node2.ipnrc with *ipnadmin*, *r* can be used with commands not defined for *bpadmin*. *bpadmin* can display status characters, known as watch characters, to the command line and to initialize that function the commands *w 1* are used. See Appendix D for the watch characters. The last command *s* starts everything that was initialized before it. It must be the last command.

4.2.1.2 cfdprc

```
1
w 1
m requirecrc 1
s bputa
```

Some configuration files are simpler than others. The .cdfprc file is one such file, the whole file is on the right. The first command of the file is *1* this initializes *cfdpadmin* so CFDP operations can function. The next command, *w 1*, will display watch characters on the command line when CFDP is used. *m requirecrc 1* is the command to enable

CRC, which is have CRCs on all PDUs that are sent by the local node. The final command, *s bputa*, starts the UT-layer service that adapts it to BP.

4.2.1.3 imrc

The .imrc file is not currently used, but it is included for future use.

4.2.1.4 ionconfig

```
configFlags 13
heapWords 50000000
pathName /tmp/
sdrWmSize 500000000
wmSize 10000000
```

The .ionconfig file defines the ION parameters for the local node. The first command defines which configuration of the simple data recorder (SDR) database to use for the local node. For this node, the SDR is implemented in a region of shared memory, transfer of data to/from the SDR are written ahead to a log, which makes the them reversible, and updates to the SDR heap are not allowed to cross object boundaries. *heapWords* defines the number of words (64 bits each on Morehead’s node) of non-volatile storage to use for the SDR’s database. The next command defines the path were the file to be used as heap space for the SDR is located and the file to be used to log the database updates to reflect the transfer of data to/from the SDR. *sdrWmSize* defines how large the dynamic memory in bytes will be for the SDR’s private working memory. *wmSize* defines how large the dynamic memory in bytes will be for the node.

4.2.1.5 ionrc

```
1 2 node2.ionconfig
s
m horizon +0
```

The .ionrc is another simple configuration file. The first command initializes node 2 using the parameters defined in node2.ionconfig. The node starts with the *s* command. The final command starts a management command that searches for possible bundle congestions from the start of the node to infinity this is to prevent the filling up of bundle storage.

4.2.1.6 ionsecrc

This configuration file only consists of one command because it is not fully implemented in ION. The command is *1* and that initializes *ionsecadmin* to prevent an error being sent to the log file.

4.2.1.7 ipnrc

```
a plan 1 ltp/1
```

For this project the configuration file is simple with only one command because there is only one other node connected to it. This command defines a plan to use for the ipn scheme. In this case, all bundles destined to node 1 will be sent to the neighboring node 1 using LTP.

4.2.1.8 ltprc

```
1 4000
a span 1 2000 2000 1400 1 1
'udplso ip_addr 10000000'
w 1
s 'udplsi ip_addr'
```

A .ltprc configuration file defines how to manage LTP for the node. The first command initializes the node and tells the node how many export sessions it can have running. The next command, *a span*, defines a link, or span, between nodes. The command above says that the span for node 1 can have a max of 2000 export sessions and a max of 2000 import sessions, it can export blocks that are no larger than 1400 bytes at one time, there can only be one LTP packet placed in one block, after one second has passed the block is sent even if it is not full, and it sends a command to *ltpadmin* that it must use the UDP link-service output task to send to node 1’s IP address at a rate of 10000000 bits per second. After the *a span* command, the *w 1* command tells *ltpadmin* to send watch characters to the command line. The final command tells *ltpadmin* to start the node with the UDP service layer input task using the local nodes IP address.

4.2.1.9 global.rc

```
a range +0 +345600 1 2 2
a contact +0 +345600 1 1
10000000
```

The final configuration file is what defines the CGR. There is a portion of it to the right, see Appendix C4 for the rest. The first command tells the node that a contact will appear at the time the node starts and lasts for 345600 seconds. The contact is between 1 and 2 and there is a 2 second one way light time delay. The second command tells the node that during this contact communication node 1 can communicate with itself at 10000000 bits per second.

4.2.2 Test Results

The first test was file transfer, using the ION command *cfdptest*, was performed several times to verify repeatability. It was able to transfer a text file and a .seq file successfully. There were a few instances of the files not being sent. Those instances were either because of configuration file changes that occurred in between tests or the node firewall preventing packets from being sent. These problems were rectified by fixing mistakes in the configuration files and adjusting firewall rules by opening some ports. The file transfers were performed again. The second test was a file transfer with AMMOS. A file containing a set of example raw packets that would come from a spacecraft was sent through AMMOS. The file was transferred successfully from JPL node to the Morehead node. Then the received file was run through the AMMOS command *chill* to verify that the received packets were valid. These steps were performed several times to verify repeatability. One problem that occurred was a database that supports AMMOS on Morehead’s node would crash occasionally and the reason could not be identified.

It was possible to run the test without the database crashing during the test, but it would crash between tests.

4.3 Summary

Chapter IV described the configuration files that were created for this project. It described how the commands in each configuration file were used. It also described the results of the CFDP file transfer test and the AMMOS test and the problems that occurred during the tests.

CHAPTER V

CONCLUSION

5.1 Discussion of Results

The results of the tests show that the node can handle sending bundles to other nodes successfully and that AMMOS can be interfaced with node. Because the node can transfer files successful this demonstrates that the node was configured properly and that the connection to the DEN is functioning. With a functioning DTN node, the Morehead node could be connected to other nodes to expand the network and provide support to Lunar Ice Cube's flatsat once it has an IRIS radio that is DTN capable. The AMMOS test shows that Morehead's node can interface with AMMOS and AMMOS at JPL. It lays the ground work for integrating with DTN with AMMOS. DTN integrated with AMMOS means that telemetry can be sent with DTN directly to the mission operations center for processing and commands can be sent from the mission operations center to the spacecraft with DTN. The completion of these tests brings the node one step closer to providing significant risk reduction for NASA.

5.2 Future Work

The bulk of this project was taken up setting up the node, troubleshooting it, and getting it connected to the DEN. More work can be done to provide risk reduction for NASA. A socket to BP adapter will need to be developed to connect DTN with AMMOS to connect the mission operations center to DTN. A block diagram for this demonstration can be seen on the next page in Figure 12. To demonstrate DTN's capabilities with spacecraft it will need to be connected to Lunar Ice Cube's flatsat once it is ready. A good demonstration of a complete DTN link would be to connect the DTN node to the ground station once and the upgrades are complete and have Lunar Ice Cube transmit to the 21-meter antenna. A recommended intermediate step would be to connect just the node to the ground station and perform a loopback test to show the node interfacing with the ground station. Completing these tasks would be a good demonstration for NASA.

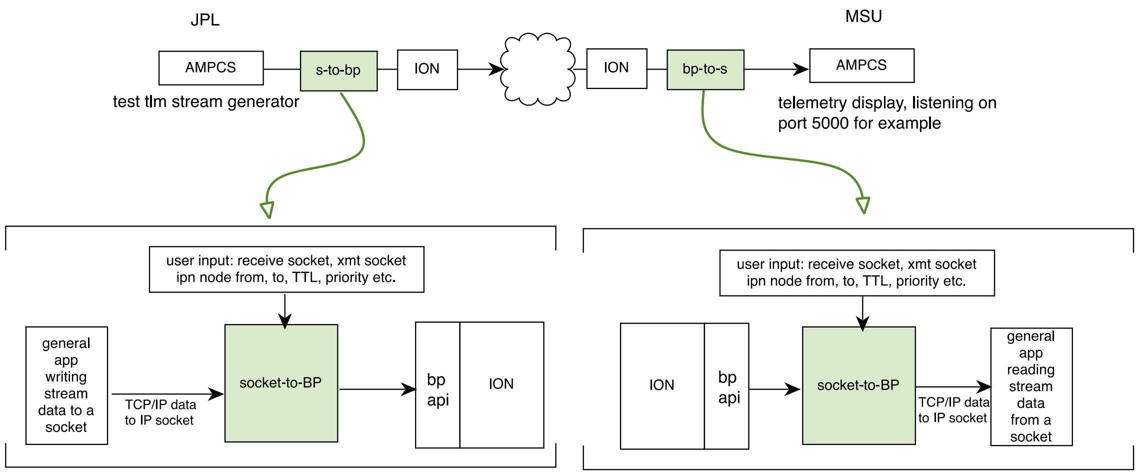
JPL
AMPCS
s-to-bp
ION
test tlm stream generator
ION
bp-to-s
AMPCS
MSU
telemetry display, listening on port 5000 for example
user input: receive socket, xmt socket ipn node from, to, TTL, priority etc.
general app writing stream data to a socket
TCP/IP data to IP socket
socket-to-BP
bp api
ION
user input: receive socket, xmt socket ipn node from, to, TTL, priority etc.
ION
bp api
socket-to-BP
TCP/IP data to IP socket
general app reading stream data from a socket

Figure 12: Future AES Demonstration

5.3 Summary

Chapter V discussed the results and what they mean for Morehead. It also discussed what future steps could be taken now that the node is running and what they would demonstrate. By presenting the methodology of the project along with the test results, this report can demonstrate why this project can serve as a basis for future risk reduction for NASA and Morehead's Lunar Ice Cube. The objective outlined in Chapter I of this report were achieved.

REFERENCES

Araniti, G., Bezirgiannidis, N., Birrane, E., Bisio, I., Burleigh, S., Caini, C., Feldmann, M., Marchese, M., Segui, J. and Suzuki, K. (2015). Contact graph routing in DTN space networks: overview, enhancements and performance. *IEEE Communications Magazine*, [online] 53(3), pp.38-46. Available at: https://www.researchgate.net/publication/273836227_Contact_Graph_Routing_in_DTN_Space_Networks_Overview_Enhancements_and_Performance.

Burleigh, S. (2016). *Interplanetary Overlay Network (ION) Design and Operation*. 3rd ed. [pdf] JPL, Caltech, pp.8, 22, 23. Available at: https://sourceforge.net/projects/ion-dtn/ [Accessed 11 Apr. 2017].

CubeSat Design Specification (CDS) REV 13. (2014). 13th ed. [pdf] The CubeSat Program, Cal Poly SLO. Available at: https://static1.squarespace.com/static/5418c831e4b0fa4ecac1bacd/t/56e9b62337013b6c063a655a/1458157095454/cds_rev13_final2.pdf [Accessed 11 Apr. 2017].

Discovery.nasa.gov. (2017). *DISCOVERY ::: EPOXI*. [online] Available at: https://discovery.nasa.gov/epoxi.cfml [Accessed 11 Apr. 2017].

Evans, B. (2016). *NASA Announces Payloads for First SLS Mission*. [online] AmericaSpace. Available at: http://www.americaspace.com/?p91175 [Accessed 11 Apr. 2017].

Jpl.nasa.gov. (2017). *Deep Impact - EPOXI*. [online] Available at: https://jpl.nasa.gov/missions/deep-impact-epoxi/ [Accessed 11 Apr. 2017].

Mohr, J. (2009). *Linux Knowledge Base and Tutorial*. [online] Linux-tutorial.info. Available at: http://www.linux-tutorial.info/modules.php?nameMContent &ob.page&pageid142 [Accessed 2 Aug. 2016].

pfSense, (2017). *SG-2220 pfSense® Security Gateway Appliance*. [image] Available at: https://store.pfsense.org/SG-2220/ [Accessed 12 Apr. 2017].

Space Packet Protocol CCSDS 133.0-B-1 Blue Book. (2003). 1st ed. [pdf] CCSDS. Available at: https://public.ccsds.org/Pubs/133x0b1c2.pdf [Accessed 11 Apr. 2017].

Warthman, F. (2012). *Delay- and Disruption-Tolerant Networks (DTNs) A Tutorial*. 2nd ed. [pdf] Warthman Associates. Available at: http://ipnsig.org/wp-content/uploads/2012/07/DTN_Tutorial_v2.05.pdf [Accessed 11 Apr. 2017].

Willman, B. and Davidson, S. (2014). *International Space Station (ISS) and Delay/Disruption Tolerant Networking*. [online] ipnsig.org. Available at: http://ipnsig.org/wp-content/uploads/2014/02/ISS-DTN-Presentation-IPNSIG.pdf [Accessed 11 Apr. 2017].

Wyatt, J. and Malphrus, B. (2016). *DSN DTN Integration Plans*.

Wyatt, J. and Malphrus, B. (2015). *Morehead State University 21 meter Antenna Upgrade to DSN Compatibility*. [online] deepspace.jpl.nasa.gov. Available at: https://deepspace.jpl.nasa.gov/files/dsn/IND_CubeSat_Comm_TIM Wyatt.pdf [Accessed 4 Aug. 2016].

Wyatt, J., Burleigh, S., Jones, R., Torgerson, L. and Wissler, S. (2009). Disruption Tolerant Networking Flight Validation Experiment on NASA's EPOXI Mission. *2009 First International Conference on Advances in Satellite and Space Communications*.

APPENDICES

A. WAN Script

```
#!/bin/sh
# root qdisc handle
r_handle=1
# netem qdisc handle
n_handle=2

interface=
delay=
dstip=
qdpresent=
loss=
delete=false
verbose=false

error() {
	printf "%s\n" "$1" >&2
}

log() {
	if [ $verbose == true ]; then
		printf "%s" "$1"

		if [ "$2" != false ]; then
			printf "\n"
		fi
	fi
}

# Ish...
isip () {
	if ! echo "$1" | grep -E '([0-9]{1,3}[.]){3}[0-9]{1,3}(/[0-9]{1,2})?' > /dev/null; then
		return 1
	fi

	return 0
}

iptohex () {
	printf '%02x' `echo "$1" | sed 's/\./ /g'`
}

usage() {
    echo "$0 -i <interface> -d <ip> -m <milliseconds> -l <loss percentage> [start|stop]"
    exit 1
}

while getopts ":hi:d:m:l:v" opt; do
    case "${opt}" in
        h)
            usage
            ;;
        i)
            interface="${OPTARG}"
            ;;
        d)
            dstip="${OPTARG}"
            ;;
        m)
            delay="${OPTARG}"
            if ! echo "$delay" | grep -E '^[0-9]+$' > /dev/null; then
                error "-m must be an interger value, got '$delay'"
                usage
            fi
            ;;
        l)
            loss="${OPTARG}"
            ;;
        v)
            verbose=true
            ;;
        *)
            usage
            ;;
    esac
done
shift $((OPTIND-1))

if [ "$interface" == '' ]; then
    error "No interface specified"
    usage
fi

if [ "$1" == 'stop' ]; then
    delete=true
elif [ "$1" != 'start' ]; then
    error "Invalid operation '$1', expected start or stop"
    usage
fi
```

```
#	Play nice with FQDNs too (IPv4 only)
if [ "$dstip" != '' ]; then
	if ! isip "$dstip"; then
		ret=`host $dstip`
		rsv=`echo "$ret" | tail -n 1 | grep -o -E '([0-9]{1,3}[.]){3}[0-9]{1,3}'`
		if [ $? -ne 0 ] || ! isip "$rsv"; then
			error "Failed resolving $dstip: $ret"
			exit 1
		fi
		dstip=$rsv
	fi
fi

#	Check if we have our queue discipline already added to the target inerface
#	let's hope nothing else if using this handle
log "Checking if root qdisc already added to $interface... " false
if tc qdisc show dev "$interface" | grep "qdisc prio $r_handle:" > /dev/null; then
	log "yes"
	qdpresent=true
else
	log "no"
	qdpresent=false
fi

#	Were we told to stop delaying packets?
if [ $delete == true ]; then
	if [ $qdpresent == true ]; then
		log "Removing qdisc with handle $r_handle... " false
		if tc qdisc del dev "$interface" root handle $r_handle:; then
			log "ok"
		else
			log "failed ($?)"
			exit 1
		fi
	fi
	exit 0
fi

#	Nope, first add the new root queue discipline if required
if [ $qdpresent != true ]; then
	log "Adding qdisc with handle $r_handle... " false
	if tc qdisc add dev "$interface" root handle $r_handle: prio; then
		log "ok"
	else
		log "failed ($?)"
		exit 1
	fi
fi

#	Add any IP filters these classify the traffic and limit what's delayed
if [ "$dstip" != '' ]; then
	log "Checking if IP is already in filter... " false
	if ! tc filter show dev "$interface" parent $r_handle:0 | grep -E 'match.*'`iptohex "$dstip"` 
> /dev/null; then
		log "no"

		log "Adding IP $dstip to filter... " false

		# Add a filter to device $interface
		# - attach it to qdisc $r_handle:0
		# - apply it to IP packets
		# - with a prio/pref (priority) of 1 (this is arbitrary as all filters have the same 
priority)
		# - use the u32 classifier
		# - match on ip dst $dstip
		# - forward matching packets to flowid $n_handle:1
		if tc filter add dev "$interface" parent $r_handle:0 protocol ip prio 1 u32 match ip 
dst $dstip flowid $n_handle:1; then
			log "ok"
		else
			log "failed ($?)"
			exit 1
		fi
	else
		log "yes"
	fi
fi

#	This is the destination for the filters we added above
#	Delay
if [ "$delay" != '' ]; then
	log "Checking if netem qdisc has been added (and has correct delay)... " false
	netem=`tc qdisc show dev "$interface" | grep "netem.*$n_handle:"`
	if [ $? -ne 0 ]; then
		log "no"
		log "Adding qdisc netem with handle $n_handle (delay ${delay}ms)... " false
		if tc qdisc add dev "$interface" parent $r_handle:1 handle $n_handle: netem delay 
${delay}ms loss ${loss}% 25%; then
			log "ok"
		else
			log "failed ($?)"
			exit 1
		fi
	elif ! echo "$netem" | grep "delay ${delay}.*ms" > /dev/null; then
		log "yes"
		log "Changing qdisc netem delay to ${delay}ms... " false
		if tc qdisc change dev "$interface" parent $r_handle:1 handle $n_handle: netem
delay ${delay}ms loss ${loss}% 25%; then
			log "ok"
		else
			log "failed ($?)"
			exit 1
		fi
	else
		log "yes"
	fi
fi

#	Loss
if [ "$loss" != '' ]; then
	log "Checking if netem qdisc has been added (and has correct loss)... " false
	netem=`tc qdisc show dev "$interface" | grep "netem.*$n_handle:"`
	if [ $? -ne 0 ]; then
		log "no"
		log "Adding qdisc netem with handle $n_handle (loss ${loss}%)... " false
		if tc qdisc add dev "$interface" parent $r_handle:1 handle $n_handle: netem loss
${loss}%; then
			log "ok"
		else
			log "failed ($?)"
			exit 1
		fi
	elif ! echo "$netem" | grep "loss ${loss}.*%" > /dev/null; then
		log "yes"
		log "Changing qdisc netem loss to ${loss}%... " false
		if tc qdisc change dev "$interface" parent $r_handle:1 handle $n_handle: netem
loss ${loss}%; then
			log "ok"
		else
			log "failed ($?)"
			exit 1
		fi
	else
		log "yes"
	fi
fi
```

B. Space Packets Code

```
#include <iostream>
#include "CCSDS.hh"
#include <vector>
#include <random>
#include <climits>
#include <algorithm>
#include <functional>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <iomanip>

#define SERVERPORT "51718"

using namespace std;

int apid = 0b10000000000;
int category = 0;
int aduCount = 0;
const char* ip = "127.0.0.1";
int16_t pktnumber;
int numbytes;
struct addrinfo hints, *servinfo, *p;
int rv;
int sockfd;
vector<uint8_t> smcpByteArray(2036);

using random_bytes_engine = independent_bits_engine<
    mt19937, 8, uint8_t>;

string createPacket(int16_t sequenceCount, int16_t maxpkts)
{
random_bytes_engine rbe;
generate(begin(smcpByteArray), end(smcpByteArray), ref(rbe));

   //constructs an empty instance
    CCSDSSpacePacket* ccsdsPacket = new CCSDSSpacePacket();
```

```
//set APID
ccsdsPacket->getPrimaryHeader()->setAPID(apid);

//set Packet Type (Telemetry or Command)
ccsdsPacket->getPrimaryHeader()->
   setPacketType(CCSDSSpacePacketPacketType::TelemetryPacket);

//set Secondary Header Flag (whether this packet has the Secondary Header part)
ccsdsPacket->getPrimaryHeader()->
   setSecondaryHeaderFlag(
      CCSDSSpacePacketSecondaryHeaderFlag::NotPresent
   );

//set segmentation information
if(sequenceCount == 0 && maxpkts > 1){
ccsdsPacket->getPrimaryHeader()->
   setSequenceFlag(
      CCSDSSpacePacketSequenceFlag::TheFirstSegment
   );
}

else if(sequenceCount > 0 && sequenceCount != (maxpkts - 1)){
ccsdsPacket->getPrimaryHeader()->
   setSequenceFlag(
      CCSDSSpacePacketSequenceFlag::ContinuationSegment
   );
}

else if(sequenceCount > 0 && sequenceCount == (maxpkts - 1)){
ccsdsPacket->getPrimaryHeader()->
   setSequenceFlag(
      CCSDSSpacePacketSequenceFlag::TheLastSegment
   );
}

else{
  ccsdsPacket->getPrimaryHeader()->
   setSequenceFlag(
      CCSDSSpacePacketSequenceFlag::UnsegmentedUserData
   );
}

//set Category
ccsdsPacket->getSecondaryHeader()->setCategory(category);
```

```
    //set secondary header type (whether ADU Channel presence)
    ccsdsPacket->getSecondaryHeader()->
        setSecondaryHeaderType(
            CCSDSSpacePacketSecondaryHeaderType::ADUChannelIsUsed
        );

    //set ADU Channel ID
    ccsdsPacket->getSecondaryHeader()->setADUChannelID(0x00);

    //set ADU Segmentation Flag (whether ADU is segmented)
    ccsdsPacket->getSecondaryHeader()->
        setADUSegmentFlag(
            CCSDSSpacePacketADUSegmentFlag::UnsegmentedADU
        );

    //set counters
    ccsdsPacket->getPrimaryHeader()->setSequenceCount(sequenceCount);
    ccsdsPacket->getSecondaryHeader()->setADUCount(aduCount);

    //set absolute time
    uint8_t time[4];
    ccsdsPacket->getSecondaryHeader()->setTime(time);

    //set data
    ccsdsPacket->setUserDataField(smcpByteArray);
    ccsdsPacket->setPacketDataLength();

    //get packet as byte array
    vector<uint8_t> packet = ccsdsPacket->getAsByteVector();

    string pkt = ccsdsPacket->toString();
    return pkt;
}

int main()
{

cout << "Expected number of packets: ";
cin >> pktnumber;

memset(&hints, 0, sizeof hints);
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_DGRAM;

if ((rv = getaddrinfo(ip, SERVERPORT, &hints, &servinfo)) != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
        return 1;
}

for(p = servinfo; p != NULL; p = p->ai_next) {
    if ((sockfd = socket(p->ai_family, p->ai_socktype,
         p->ai_protocol)) == -1) {
            perror("talker: socket");
            continue;
        }

    break;
}

for(int i = 0; i < pktnumber; i++){

 string pt = createPacket(i, pktnumber);
 cout << pt;
 const char* t =pt.c_str(); // Used to send packet, know it works, but only can see header of
conversion to txt
 //char buffer = sizeof(pt);
// char n = sprintf(buffer, pt);
 if ((numbytes = sendto(sockfd, t, strlen(t), 0,
      p->ai_addr, p->ai_addrlen)) == -1) {
        perror("talker: sendto");
        exit(1);
    }

}
close(sockfd);
}
```

C. Node Files

C1. ionstart

```
#!/bin/bash
# shell script to get node running
rm ion.log
sleep 1
ionadmin        node2.ionrc
sleep 1
ionsecadmin     node2.ionsecrc
sleep 1
ltpadmin        node2.ltprc
sleep 1
bpadmin         node2.bprc
```

```
sleep 1
cfdpadmin     node2.cfdprc
sleep 1
imcadmin      node2.imcrc
sleep 1
ionadmin      global.rc
sleep 1
bpecho ipn:2.3 &
```

C2. ionstop

```
#!/bin/bash
echo "IONSTOP will now stop ion and clean up the node for you..."
echo "bpadmin ."
bpadmin .
sleep 1
echo "cfdpadmin ."
cfdpadmin .
sleep 1
echo "ltpadmin ."
ltpadmin .
sleep 1
echo "ionadmin ."
ionadmin .
sleep 1
echo "global.rc ."
ionadmin .
sleep 1
echo "killm"
killm
echo "ION node ended. Log file: ion.log"
```

C3. node2.bprc

```
1
a scheme ipn 'ipnfw' 'ipnadminep'

# add ION utility endpoints
a endpoint ipn:2.0 x
a endpoint ipn:2.1 x
a endpoint ipn:2.2 x
a endpoint ipn:2.3 x
a endpoint ipn:2.4 x
a endpoint ipn:2.5 x
a endpoint ipn:2.6 x
a endpoint ipn:2.7 x
```

```
a endpoint ipn:2.8 x
a endpoint ipn:2.10 x
a endpoint ipn:2.11 x
a endpoint ipn:2.12 x

# add AMS endpoints
a endpoint ipn:2.9 x
a endpoint ipn:2.15 x

# add CFDP-1 endpoints
a endpoint ipn:2.64 x
a endpoint ipn:2.65 x

# add lgagent endpoint
a endpoint ipn:2.127 x

#add procotols/ducts

a protocol ltp 1400 100
a induct ltp 2 ltpcli
a outduct ltp 1 ltpclo
a outduct ltp 3 ltpclo

# load ipn parameters
r 'ipnadmin node2.ipnrc'

#start watch characters
w 1

#start the daemons
s
```

C4. global.rc

```
a range +0 +345600 1 2 2
a contact +0 +345600 1 1 10000000
a contact +0 +345600 1 2 10000000
a contact +0 +345600 2 1 10000000
a contact +0 +345600 2 2 10000000
```

D. Watch Characters

a	new bundle is queued for forwarding
b	bundle is queued for transmission
c	bundle is popped from its transmission queue
m	custody acceptance signal is received

w	custody of bundle is accepted
x	custody of bundle is refused
y	bundle is accepted upon arrival
z	bundle is queued for delivery to an application
˜	bundle is abandoned (discarded) on attempt to forward it
!	bundle is destroyed due to TTL expiration
&	custody refusal signal is received
#	bundle is queued for re-forwarding due to CL protocol failure
j	bundle is placed in ‘‘limbo’’ for possible future re-forwarding
k	bundle is removed from ‘‘limbo’’ and queued for re-forwarding
d	bundle appended to block for next session
e	segment of block is queued for transmission
f	block has been fully segmented for transmission
g	segment popped from transmission queue
h	positive ACK received for block, session ended
s	segment received
t	block has been fully received
@	negative ACK received for block, segments retransmitted
=	unacknowledged checkpoint was retransmitted
+	unacknowledged report segment was retransmitted
{	export session canceled locally (by sender)
}	import session canceled by remote sender
[	import session canceled locally (by receiver)
]	export session canceled by remote receive
p	CFDP PDU transmitted
q	CFDP PDU received