Simulating Delay Tolerant Networking for CubeSats
Communication/Telemetry, Integration and Testing

Paul Muri and Janise McNair
Wireless and Mobile Systems Lab
Electrical and Computer Engineering Department
University of Florida
Gainesville, FL, 32611, USA

Corresponding Author: Paul Muri - pmuri@ufl.edu

ABSTRACT

Delay Tolerant Networking (DTN) is a communication protocol suite that addresses many of the communication challenges CubeSats encounter with space exploration including low transmission power, intermittent connectivity, long delay, and high bit error rate. Most research for DTN space communication relies on simulation for protocol validation and integration since real deployments are often either very expensive or impossible. This paper describes a virtual space inter-networking environment to control test bed nodes of Linux-based hardware candidates for CubeSats. For this test bed, a Network Simulator 3 (NS-3) module modeled physical DTNs up to the data link layer as WiFi-type given topology and mobility patterns, ranges, and data rates. At the networking stack’s higher layers, the simulation platform emulates DTN on nodes external to NS-3. Nodes include virtual machines, Linux containers, or external hardware nodes as candidates for an experimental payload for CubeSats. For experimentation, DTN models for CubeSat clusters were designed and compared to UDP/IP clusters and constellations. When compared with UDP/IP cluster, the DTN metrics outperform with higher data-rate, lower bundle-drop ratio, less overhead, and longer transmission windows for the given mobility models. The test bed proves that a DTN inter-networking solution overcomes CubeSat communication limitations for space exploration.

1. INTRODUCTION

Delay Tolerant Networking (DTN) is a computer network architecture that addresses challenging environmental conditions that create delay, intermittent connectivity, and high bit errors. Examples of these challenged network environments include sensor networks on remote land regions, water, aerial, and space networks.

The power and space limitations of CubeSats have hindered the advancement of its communication system. In addition to power and space limitations, CubeSats have exclusively launched in low earth orbits, which gives limited
time to link with ground stations. Depending on the orbital parameters and the location of the ground station, a single CubeSat may only downlink two to three times a day for ten-minute periods. Thus, a communication protocol is needed to burst downlinked data efficiently when a connection with the ground station occurs.

In general, aside from proprietary protocols, CubeSats have operated over the AX.25 link layer protocol at point-to-point links. There have been projects proposed using Digital Smart Technologies for Amateur Radio (DSTAR) on CubeSats and AeroCube has used its own proprietary protocol (Muri, 2012). DTN protocols have been successfully tested for space communication on larger spacecraft such as the International Space Station, Epoxi, Earth Observing-1 satellite, IntelSat 14, and UK-DMC. For CubeSats, DTN can run on top of AX.25, a DTN convergence layer adapter (CLA) has been implemented for the popular link layer protocol AX.25 (Ronan, 2010).

This article presents a simulation tool to validate DTN protocols with an 802.11g-2007 convergence layer for CubeSat cluster topologies and a comparison to UDP/IP based satellite constellations. Section 2 discusses multi-CubeSat mission topologies. Section 3 provides background on DTN simulations. Section 4 discusses the design of this DTN test bed. Section 5 details the experiment run on the testbed with results in section 6. Then section 7 concludes and details future work.

2. MULTI-CUBESAT MISSIONS

A DTN protocol suite can take advantage of how multiple clusters of a dozens of CubeSats rideshare on rockets by inter-satellite linking CubeSats to larger, high powered relay satellites future interplanetary missions. The DTN protocol allows multiple nodes to store and forward data allowing for efficient payload data downlinking and ultimately more opportunities to link with a ground station.

2.1. CubeSat Constellations

An earlier study was done to compare how orbital constellations would extend the opportunities to link with a ground station (Muri, 2011). A CubeSat orbital constellation is shown in figure 1; eight CubeSats are in a sun-synchronous orbital plane for observation. A single satellite shown orbiting in an equatorial repeating ground track orbit would encounter the ground station more frequently could act as a sink satellite to relay the polar CubeSats’ observation data. However, all CubeSats would need propulsion capabilities to be placed in this constellation.
Not needing propulsion, a more realizable CubeSat mission is that of the CubeSat cluster. An example of the mission includes QB50 program that has called for a cluster of 50 CubeSats for a magnetosphere study (Bridges, 2011). For QB50, CubeSats do not transmit through inter-satellite links. However, an inter-satellite networking capability in such a mission could take advantage of the cluster topology for distributed processing and higher data relay to a ground station. This topology is shown in figure 2.

For longer interplanetary links, DTN’s Licklider Transport Protocol LTP, and ION and JPL could be used. LTP takes care of high latency, and asynchronous channels because data could be downlinked without many ACKs from the destinations.
3. SIMULATING DELAY TOLERANT NETWORKING

In 2007, IRTF published RFC 5050 that defined a standard protocol for DTN known as the Bundle Protocol (BP). Bundles defined in BP contain semantic data that helps a network application more than single small packets of TCP or UDP. BP operates with non-IP and IP transport layers, known as convergence layers, in a store and forward method between nodes. The architecture of BP works as an overlay network and uses Endpoint Identifiers, or EIDs, as a naming scheme to identify ultimate destination nodes.

3.1. Issues with DTN Network Simulations

Simulators such as DTN, DTNSim2, and the Opportunistic Network Environment (ONE) have been used in previous studies. DTNSim2 is no longer supported and ONE was developed between 2008 and 2010. Often, these simulation platforms require high computing resources.

Also, the effectiveness of DTN simulators for space inter-networking is sensitive to the level of realism. Many DTN simulators do not currently implement realistic channel models or networking stack. There is also an issue of cross-simulator comparability. Researchers often create their own simulators to test algorithms, so it can be difficult to compare a new algorithm with existing ones unless the new protocol is implemented on a variety of simulators. Thus, a goal for our testbed was to use standardized, open source, physical and link layer simulation modules, and have the higher layers as standard DTN implementations on virtual or real hardware nodes.

3.2. DTN Implementations

Many DTN implementations have been written for various platforms. For our testbed, we choose the Network Simulator 3 (NS-3), a widely available and capable open-source simulator, for the channel and link layer simulation. Then, on top of the simulated link layer, one of following DTN implementations could be run.

3.2.1. DTN2

For a Bundle Protocol reference implementation, the Delay Tolerant Networking Research Ground (DTNRG) created DTN2. DTN2 is described in the Internet Research Task Force (IRTF) standards organization’s RFC5050 specification.

3.2.2. ION

For a Licklider Transport Protocol (LTP) implementation, the NASA Jet Propulsion Laboratory (JPL) created the Interplanetary Overlay Network (ION) implementation (Wang, 2011). Specifically, ION implements LTP found in IRTF RFC5325 to 5327 and BP RFC5050.
3.2.3. IBR-DTN

Built by the Institute of Operating Systems and Computer Networks at Technische Universität Braunschweig, IBR-DTN is a lightweight DTN implementation of the Bundle-Protocol RFC5050. IBR-DTN is particularly useful for an embedded systems involved in intermittent sensor networks.

3.2.4. JDTN

Cisco Systems created a DTN implementation in java known as JDTN for the mobile android platform. JDTN supports BP RFC5050 and LTP RFC5326 (Schildt et al., 2011)

3.2.5. ByteWalla

The TSLab at the KTH School of ICT in Kista, Stockholm created ByteWalla as an adhoc DTN mobile Android application to bring connectivity to remote regions using BP. The application uses mobile phone as data carriers, similar to JDTN.

3.2.6. N4C

Networking for Communication Challenged Communities (N4C) created a DTN2 simulation platform that uses NS-3 to model the physical RF nodal connections (Hołubowicz et al., 2011) and can run emulation of DTN at the network layer.

4. DESIGN OF TESTBED

The N4C implementation described in section 3.2.6 was picked for the test bed because it had automated network connectivity scheduling. Automated scheduling allowed for simple integration of a DTN network with CubeSat topologies.

The test bed simulated the physical channel as wireless 802.11g-2007 given parameters for topology, mobility patterns, data transmission ranges, and data rates. For the higher layers, the tool bridges connections to network nodes, external to NS-3, in the form of Linux container (LXCs) or virtual machines. The network status or netstat of the virtual machines can be viewed in a real-time terminal as seen in figure 3.
Figure 3. The simulation platform screen capture of the network status (netstat) of the LXC terminal window in real time.

Also, the test bed topology, mobility, transmission link directions, and data rate can be visualized in real-time and logged using NS-3’s python-based pyviz application shown in figure 4. Note that visualization shows nodes in two-dimension, however node topology is set to three-dimensions using Cartesian x, y, z coordinates.

Figure 4. The simulation platform screen capture of the python-based visualizer (pyviz) that shows the data rate and node topology. The bottom window shows the network status (netstat) of the LXC terminal window in real time.

4.1. Architecture of the Host Machine

The entire test bed can be run from one host machine. The test bed components include NS-3 and virtual machines with the DTN2 network software stack and DTN applications, subject to testing and integration, run on top.

Using the network simulator in conjunction with the networking capabilities of the host operating system, including bridging and routing between the virtual machines, gives more flexibility in configuring the simulated network and its operation. The host operating system’s networking stack can be used to set up and tear down connections, and shape the network traffic with Linux’s advanced routing subsystem.
NS-3 models communication channel properties such as delays, transmission rates, errors, and packet loss distributions with detailed scheduling. Also, NS-3 allows configuration of mobility patterns of wireless stations, networking device properties at the physical and link layer, logging and packet tracing. If required, new models can be constructed and used in simulations.

The simulation platform developed generally does not use models of nodes that participate in simulated networks, although this is possible and could complement the current DTN simulation to generate some specific traffic or model non-DTN nodes. Virtual node hardware gives the possibility of installing a regular operating system and an almost native environment on which to install DTN software. Thus, it has been preferred to modeling nodes within the simulator. These virtual nodes can be replaced with real flight hardware candidates in the form of laptops and smart phones. Flight hardware candidates with DTN implementations connect to the simulated channels through an access point created by the simulation host machine. Figure 5 presents the main architectural components of the simulation platform together with virtual machines and LXC's participating in a simulation.

**Figure 5.** Diagram of the host system's architecture connecting virtual machines or LXC's to NS-3 and VPN tunnels to remote desktop hardware devices. An 802.11 access point can also be setup for CubeSat flight hardware candidates such as laptops and smart phones (Credit of N4C, Hołubowicz et al., 2011).
4.2. Virtual Machines

The virtual machines create a native environment for the DTN software. Thus, they provide virtual hardware on which operating systems can be installed and configured according to the DTN software requirements with minimal impact to the underlying host. Of the various virtual platforms available Oracle’s VirtualBox has been selected as the primary because there are available open source editions, released under the GNU General Public License V2. Modification of VirtualBox is quite feasible, but other possibilities do exist such as VMWare Player and VMWare Server.

The network connecting the virtual nodes is modeled by the network simulator with only minimum engagement of the host operating system’s networking stack. The host’s networking stack provides only basic connectivity between each individual virtual machine and the simulator, not between the virtual machines. This might be illustrated as an analogy to a switch or router connecting workstations: the simulator plays the role of the switch or router, while the host operating system provides only the patch-cords (cables) between the workstations and the switch (router). Using a virtual machine allows the user to observe the behavior of a full DTN application in real-time.

4.4 LXC

As stated, lightweight virtual nodes have also been implemented in the form of Linux Containers (LXC). Nodes hosted on Linux containers use the operating system kernel of the hosts instead of stand-alone installations in virtual machines. However, LXC can use resource isolation and control based on the control groups (or cgroup) feature of the Linux kernel. Key system resources such as network interfaces, process namespaces, selected parts of file systems are isolated between the containers. The resulting isolation resembles what the chroot command provides with respect to the file system, but extended to processes and logical network interfaces. Read-only resources common to more than one container (directories with common files) can be shared. Processor instructions are executed natively without a need for any special interpretation.

Due to all these features, the overhead of the LXC method is significantly lower than that of full virtual machines. Setting up hundreds of virtual nodes is feasible while still providing the appearance of a guest operating system to the applications run in a container. As a result, a lightweight virtual node, though limited to using the same operating system as the host, requires much less RAM and disk space than a full virtual node and can therefore be replicated in great numbers. In practical tests for the simulation platform around 80-90 nodes at maximum were run.
They are useful more in simulating bundle transport through a large network. Studying the behavior of particular DTN applications is better observed on full virtual machines.

5. SETUP OF EXPERIMENT

As a main network simulation and server Linux platform, Fedora Linux 64-bit distribution was selected as the operating system both for the host. Ubuntu Linux distribution was chosen due to the DTN2 reference implementation for the guests. Hardware of the host can be any standard PC or notebook compatible with the operating system, with sufficient size of at least 8GB of RAM, and disk space to allow for many virtual machines. It is important to note that a virtual node on this platform cannot model different hardware platforms of interest such as ARM, Broadcom, or Atheros used in WiFi routers. Instead, they can be integrated by physically connecting them into the system through an access point.

NS-3 comes with a number of channel and device models, including CSMA/CD, WiFi, and WiMax. The simulation setups are written mainly in C++ and make use of components from the simulator library. User simulation scripts use the Python language. The DTN2 systems and application software that run on the virtual nodes come from the DTNRG.

5.1. Bridging Connections of Virtual Devices

Each virtual machine in the host machine’s network has a virtual Ethernet interface (veth), known as eth0 in the virtual machine. Virtual Ethernet is visible to its guest operating system, and has been allocated a unique simulation address space IP address. This Ethernet interface is bridged by the virtual machine engine to a virtual interface on the host computer, which the host operating system sees as \(vm_1, vm_2, \ldots, vm_n\).

The host operating system maintains another set of virtual network interfaces, of a tap type, created by the `tunctl` command. A network tap interface provides a way to monitor the traffic flowing between two network nodes non-obtrusively. The tap has three ports an A port, B port, and monitor port, but does not need an IP address. The resulting network configuration is shown in figure 6.

In the experiment setup, taps are named \(i_1, i_2, \ldots, i_n\). Each tap corresponds logically to a virtual machine, although technically taps do not have anything common with the virtual machines, until specific commands are executed by the host operating system.
Figure 6. Network Configuration Diagram for the bridging of DTN virtual Ethernet devices to the NS-3 wireless transmission simulation (Credit of N4C, Hołubowicz et al., 2011).

The commands to create tap virtual interfaces and bring them up without IP addresses are the following:

```
tunctl -t i1
ifconfig i1 0.0.0.0
ifconfig i2 0.0.0.0
```

The NS-3 script creates, for a specific runtime, virtual network devices as internal data structures. The script associates with the host operating system interfaces $i_1, i_2, ..., i_n$ as tap bridge objects, such that each of the taps forms a bridge with a corresponding network device in the simulator.

The commands creating bridges on the host to bring them up without IP addresses:

```
brctl addbr b1
ifconfig b1 0.0.0.0
brctl addbr b2
ifconfig b2 0.0.0.0
```

In the networking subsystem of the host operating system, virtual interfaces $vboxnet_1, vboxnet_2, ..., vboxnet_n$ are bridged by a brctl command with the interfaces $i_1, i_2, ..., i_n$. So, the interfaces eth0 of the guest operating systems are bridged to the corresponding network devices in the simulator.

To connect the tap interfaces to the virtual machine interfaces through the bridges:

```
brctl addif b1 i1
```
Finally, the simulator connects internal networking devices to an internal communication channel. This connects all the virtual nodes. Only the interfaces eth0 on the virtual nodes need IP addresses. All other network interfaces are configured as a layer-2 connection.

For the network frames to actually reach their destinations, they must not be impeded by packet filtering mechanisms in the kernel. Either the rules manipulated by tools like iptables, arptables, and ebtables have to accept them to be forwarded, or the rules by the kernel has to be disabled for bridges by zeroing the relevant entries in the directory found through the path: /proc/sys/net/bridge/bridge-nf-call. Note that although a bridge is a layer-2 device, the forward chain of the IP kernel packet filter accessed by iptables can block packets traversing the bridge.

When running the simulation, the network status or “netstat” can be viewed in real time for each virtual machine or LXC such as figure 1 by typing the command:

```
netstat -t -u -c
```

### 5.2. Networking Protocols Tested

The Wifi and CSMA network models are built into the NS-3 source code. Wi-Fi is considered the common type of link for a DTN tested. CSMA, as a basic model of wired Ethernet, is a candidate for comparison.

One of the basic characteristics of a DTN network one would try to model is an ad-hoc link, which can go up or down at different times. In the WiFi case the on/off link state changes can be modeled by defining an appropriate mobility model. When the distance between node locations becomes sufficiently large, the effect is loss of connectivity, which quite adequately models the real world link down event. In the CSMA case, there are two specific attributes of NetDevice objects, SendEnable and ReceiveEnable, which can be manipulated to get the on/off effect.

Three routing options were available for configuration: static, prophet, or flood. Flood was used for all current simulations. However, further work can perform comparisons between flood and prophet routing protocols for different topologies.

The standard slot time for 802.11g-2007 is 20µs. From the slot time the DCF Interframe Spaces (DIFS) can be calculated, shown in equation 1. Standard short inter-frame spacing (SIFS) for 802.11g-2007 is 10µs, however they
can be optimized for longer range Wifi (Khambari, 2011). For the simulation, SIFS are half of the slot time shown in equation 2.

\[ DIFS = \frac{5}{2} \times SlotTime \]  

\[ SIFS = \frac{1}{2} \times SlotTime \]  

For nodes 2000 km apart, radio signals propagate one way for a time of over 6 milliseconds. So, the slot times were extended to 15 milliseconds to account for high round trip times. Our DCF 802.11g-2007 specification for timing were calculated from the following equation 3.

\[ SlotTime = Air\ Propagation + CCA + \]
\[ Turnaround + Mac\ Processing \]  

There are other network characteristics that can be modeled by using attributes of the existing NS-3 classes, such as delay in CSMA models, or writing models for specific attributes such as propagation error models.

5.3. Mobility Model Used

A previous study by Bridges (2011) used a Java-based orbit propagation and visualization software called SatLauncher to model a CubeSat’s mobility upon launch of a cluster. The orbital parameters of each CubeSat were calculated given altitude, radius, rotation rates, mass, and launcher velocity. The velocity in the x-axis and z-axis was calculated by equation 4:

\[ \Delta V_x = \Delta V_s \cos(\varphi) - nR \sin(\varphi) \]
\[ and \]
\[ \Delta V_z = \Delta V_s \sin(\varphi) + nR \cos(\varphi) \]  

Where \( \varphi \) is the angle between the xy-plane, from the positive x-axis. \( V_s \) is defined as the vertical kickoff velocity from the launch vehicle surface. \( R \) is the radius of the launcher, and \( n \) is the rotation rate of the launcher about the y-axis. For the clusters in our test bed, satellites use the 1U CubeSat standard mass and volume, 1kg and 10cm\(^3\) respectively. Another goal was to create a cluster of tight formation but maintaining safe distances. The node topologies can be visualized using NS-3’s pyviz model in figure 4 or with wireless ranges shown in the network animator in figure 7.
5.4. Physical Channel Parameters

Configuring the 802.11g-2007 standard can allow for acceptable long-range performance. We calculated from equation 4 that a 30dbm transmission with 10db gain transmitting and receiving antennas, nodes with -116 dbm sensitivity have the range of 2000 km.

An assumption of 10 db gain for object in challenged networks was based on a study on how to point a small satellite and deploy a directional antenna to the intended destination (Muri, 2010) whether the destination be a ground station or other satellite. Table 1 shows the transmission range parameters used.

Table 1. Node Transmission Range Parameters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$ = Wavelength</td>
<td>.125 meters</td>
</tr>
<tr>
<td>$G_{rx} =$ Rx Antenna Gain</td>
<td>10 dB</td>
</tr>
<tr>
<td>$G_{tx} =$ Tx Antenna Gain</td>
<td>10 dB</td>
</tr>
<tr>
<td>$P_{tx} =$ Power Transmitted</td>
<td>30 dBm</td>
</tr>
<tr>
<td>$d =$ Transmission Distance</td>
<td>2,000 Km</td>
</tr>
<tr>
<td>$P_{rx} =$ Power Received</td>
<td>-116 dBm</td>
</tr>
</tbody>
</table>
When the parameters are used in the Friis transmission equation 5 the maximum range of a node amounts to 2,000 km.

\[
P_{rx} = G_{tx} + G_{rx} - 20 \log \left( \frac{4\pi \times d}{\lambda} \right) + P_{tx}
\]

\[
= 10dB + 10dB - 20 \log \left( \frac{4\pi \times 2,000km}{.125m} \right) + 30dB
\]

\[
= -116dB
\] (5)

To address the Doppler shift, Jakes propagation loss model, built into NS-3 can be used. The model’s parameter for Doppler frequency shift, \(f_d\) [Hz] can be calculated from velocity \(V\) [m/s], transmission frequency \(f\) [Hz] and light speed \(c\) [m/s] in equation 5.

\[
F_d[Hz] = \frac{V[m/s]}{\lambda} = \frac{V[m/s] \times f[Hz]}{c[m/s]}
\] (6)

6. RESULTS

The default DTN configuration uses a UDP convergence mechanism, but to the simulation platform the traffic generated looks just as any IP traffic. Tests have been performed on mostly connected CSMA examples and Wi-Fi examples with virtual nodes and LXC’s. Tests of simple applications including dtnping, dtnsend, and dtnrecv show the platform can model DTN network connectivity. The results show the platform is capable of modeling various connectivity patterns in a manner transparent to IP packets.

Figure 8. Diagram of a small topology of close 802.11g-2007, DTN-enabled nodes for a starting spacing of 50 meters, average data rate for this topology is 755.84 Kbit/sec.

A small topology of close 802.11 nodes of 50 meters spacing is shown in figure 8. The average data rate for this topology is 755.84 Kbit/sec. For a cluster of Wi-Fi nodes with range of 2000 km, the average data rate is 10.44
Kbit/sec. To compare to an earlier study, a UDP/IP network of nodes at distances of 2000 km had a data rate of 0.08 Kbits/sec (Muri, 2011) shown in figure 9.

![Comparison of the average data rate between the three environments tested.](image)

**Figure 9.** Comparison of the average data rate between the three environments tested.

### 7. CONCLUSIONS AND FUTURE WORK

For experimentation, we evaluated CubeSat cluster mobility models, and topologies using the DTN metric of data-rate. When compared, the DTN data rate outperformed UDP/IP with higher data-rate for the given mobility models.

Future work includes integrating the simulation platform with more DTN implementations such as NASA’s Inter-planetary Overlay Network (ION) implementation with Licklider Transmission Protocol (LTP) (Wang, 2011). Future implementation can also consist of substituting virtual nodes by real-world flight hardware, provided that device has a working DTN stack compatible with its operating system. In addition, larger models can be built with three dimensions and varying channel conditions. It is straightforward to enlarge the DTN network modeled by defining additional virtual nodes.

Thus, topologies were designed for simulating virtual challenged environments to control test bed nodes of Linux-based hardware. This flexible and scalable simulation platform can aid in system integration by investigating networking performance of satellite clusters running native DTN software, as DTN network nodes. With these features the simulations could be applied to NASA projects involving challenged environments including PhoneSat-
V1, Desert Research and Technology Studies (RATS), Synchronized Position Hold, Engage, Reorient, Experimental Satellites (SPHERES), and the Laser Communication Relay Demonstration (LCRD).

ACKNOWLEDGMENT

We would like to thank NASA Office of Chief Technologist’s Space Technology Research Fellowship (NSTRF) grant #NNX11AM73H for supporting this work. This article’s conclusions and recommendations do not necessarily reflect the views of NASA.

REFERENCES


Available: http://dl.acm.org/citation.cfm?id=2050913

Khambari, M., Muhamad Z., Ariff, N., and Masud, M, (2011): Performance enhancements in IEEE 802.11 DCF MANET through variation of SIFS values in distance vector routing environment. 7th Proc. of the Information Technology in Asia (CITA 11), Kuching, Malaysia, pp.1-5,


Muri, P.; Challa, O.; McNair, J. (2010): Enhancing small satellite communication through effective antenna system design, Proc. of the IEEE Military Communications Conference, San Jose, CA pp. 347-352

Available: http://doi.acm.org/10.1145/1755743.1755757


