SSC17-S2-07

Nanosatellite Communication Constellation Testbed for Autonomous Scheduling Algorithms to Enable Mission Performance Analysis and Demonstration

Alonzo E. Jenkins, Peter J. Yoo, and Cherry Y. Wakayama
SPAWAR Systems Center Pacific
San Diego, CA 92152; 619-767-4432
alonzo.jenkins@navy.mil

ABSTRACT

We describe a nanosatellite (nanosat) communication constellation testbed (NCCT) for evaluation and demonstration of autonomous nanosat message delivery scheduling algorithm. The testbed includes multiple flight computers with flight software and subsystem simulator representing nanosats and virtual ground nodes within a workstation that are connected via Ethernet forming a network constituting a nanosat communication constellation. The ability to prototype and evaluate autonomous scheduling algorithm interacting with the flight software at a mission level without on-orbit operation offers significant advantages to the nanosat system development since it is typically complex, hard to test and expensive using real systems. The NCCT provides interfaces to various nanosat subsystems including power module, radio transmitter and receiver and flight software, virtual ground nodes, and other simulation modules including message generation and nanosat orbit simulator. The configuration for the nanosat communication network simulation is entered by the user through a graphical user interface. We are currently developing a prototype testbed to evaluate a store-and-forward nanosat communication network including multiple nanosats and multiple ground nodes.

1 INTRODUCTION

The effort invested into the development of nanosatellite (nanosat) technology has resulted in increased operational usage of nanosats for various missions. With nanosats having a low cost, small size, low power, and rapidly deployable space operations, they have gained a significant amount of interest from both the government and commercial sectors to satisfy growing needs including remote sensing, scientific research and communications. Subsequently, constellations of nanosats have been proposed for reliable and resilient communications networks. There are various efforts on optimizing mission performance through individual subsystem development and algorithm development and the broader integration of serval subsystems that make up the nanosat. However, currently, there is no known hardware-in-the-loop (HiL) mission-level testbed for evaluating and demonstrating a nanosat communication constellation. This work is based on creating a low-cost, risk-reducing, HiL mission-level testbed called Nanosat Communication Constellation Testbed (NCCT) to provide a simulation environment for nanosat space-to-ground communication mission analysis. The NCCT platform will be used to support research and development of autonomous constellation management including scheduling and routing algorithms for communications and networking applications.

The NCCT is designed to simulate a nanosat communications constellation consisting of ground nodes and nanosats. To design and implement the baseline testbed, we use commercial off-the-shelf (COTS) hardware and existing government-owned development flight software. The hardware consists of commercial embedded computing platform called BeagleBone Black (BBB) for flight computer emulation, Ethernet switches and cables for User Datagram Protocol (UDP) device communication, and a computer workstation with virtual machine software to emulate ground nodes and establish connectivity with the BBBs. The flight software running on the BBBs is based on the flight software developed for DoD 6 U multi-mission nanosat bus by Pumpkin Inc. The flight software runs on Debian and the Python programming language is used to interface applications with the flight software to provide any necessary nanosat information to the applications. The commercial software, VMWare, is used to setup virtual Linux-based ground nodes where graphical user interfaces (GUI) are developed and utilized to configure simulation parameters and obtain mission analysis output.

Our efforts in building the NCCT will simplify the process of developing and demonstrating various autonomous scheduling algorithms. Such an integrated testbed framework lays the foundation for the integration and testing of future algorithms and could help researchers to demonstrate the value of nanosat

autonomy, perform parametric trades, and optimize mission approaches before we employ the real system. To fully verify the operational testbed, we will demonstrate an autonomous nanosat message delivery scheduling system for a store-and-forward nanosat communications architecture using the proposed testbed simulation environment. Figure 1 illustrates a store-and-forward nanosat communication architecture, where nanosats receive messages of different sizes and priorities from remote user and central user ground nodes and each nanosat scheduling system determines when to deliver messages to associated destinations (remote user/central user ground nodes). The autonomous scheduling system requires nanosat telemetry such as priorities and sizes of messages, orbit information such as position and velocity, and remaining battery energy level to derive message delivery scheduling decisions.

This paper is organized as follows. We describe an overview of the testbed architecture in Section 2. Our testbed implementation and progress is discussed in Section 3. The paper concludes with current and future work on the testbed development and demonstration in Section 4.

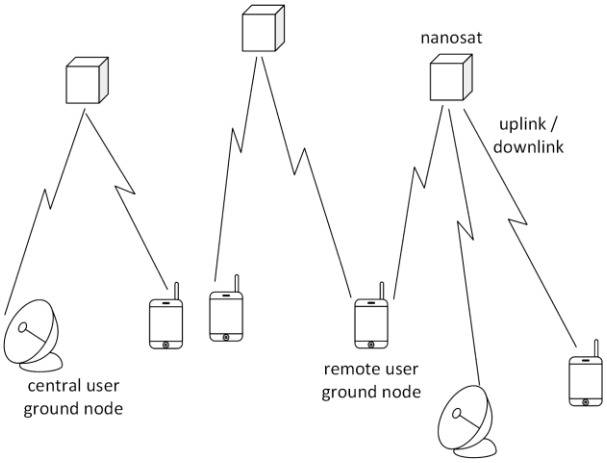

Figure 1: A store-and-forward nanosat communication architecture

2 TESTBED ARCHITECTURE

In this section, we describe the NCCT architecture (Figure 2) and the three main modules making up the NCCT structure namely: (1) Ground Node Module, (2) Nanosat Space Module, and (3) Ethernet Switch. The Ground Node and Nanosat Space Modules can be further broken down into defining subsystems which are described in this section.

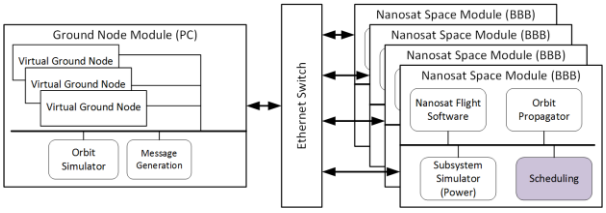

Figure 2: Testbed Architecture Block Diagram

2.1 Ground Node Module

The Ground Node Module implemented on a personal computer (PC) workstation is broken down into submodules including Virtual Ground Nodes, Orbit Simulator, and Message Generation. The Virtual Ground Node submodule simulates message transmission and reception at central user ground nodes and remote user ground nodes in different geographic areas. The Orbit Simulator submodule provides the scenario data including contact time windows between distributed ground node locations and nanosats for a specified nanosat constellation. As currently implemented, the contact time windows data are generated using Systems Tool Kit 11 (STK 11) by Analytical Graphics, Inc. (AGI). The Message Generation submodule creates random messages of different sizes, priorities and destination ground nodes over time.

2.2 Nanosat Space Module

The Nanosat Space Module implemented on a BBB is broken down into four submodules which include Nanosat Flight Software, Orbit Propagator, Subsystem Simulator and Autonomous Scheduling. Each of these submodules are described in the following.

Nanosat Flight Software

The Nanosat Flight Software is a government-off-the-shelf (GOTS) product developed by the US Naval Research Laboratory (NRL) [1, 2]. It is a Debian-based, flight-ready software. The software architecture was used successfully in previous government nanosat missions. It uses Consultative Committee for Space Data Systems (CCSDS) compliant message structure called "Space Packet" for all telemetry, commands, interface control requests and acknowledgements [3]. For communicating with the software in a spacecraft, it utilizes an IP-based/Ethernet between the Flight Computer and Payloads. The definition of a payload in our flight software core is very loosely defined, such that any components with UDP interface can be a payload. This means that a ground node can also be payload. This makes it easier for payload developers to integrate their payloads. The flight software provides nine services for interfacing between the nanosat bus

and payload, mainly for sending commands or receiving telemetry [4].

The payload controls which nanosat bus services are used. Each service supports data flow with acknowledgement for flow control and/or data receipt verification. A short description for each bus service is described below:

- Payload Command Forwarding – Send a command to a payload without interpretation to the content. The bus is essentially providing a bent-pipe service to transfer commands from a source to the designated payload. E.g., Power-on/-off.

- Payload Telemetry – Report health and status telemetry of a payload to the bus. E.g., Payload temperature, voltage, etc.

- Bus Command – Send a command to the bus. If the commands are destined for the bus, the bus will execute the commands. E.g., Flight software reset, telemetry log, turn off/on GPS, etc.

- Bus Telemetry Packet – Get telemetry packets from the bus. All telemetry packets stored by the bus are available by request. E.g., Get ADCS (attitude determination and control) orientation data, GPS lock data, power status, command and fault counters, etc.

- Bus Telemetry Stream – Stream bus telemetry down to ground using RF. E.g., Get ADCS orientation data, GPS lock data, power status, command and fault counters, etc.

- Payload Data Storage – Store payload data on bus upon request. E.g., Log sensor data, save images.

- Payload Data Downlink – Downlink payload data. E.g., Sensor data, images.

- Payload Data Load – Transfer data from a file stored on the bus to the payload. E.g., Software update.

- Time Service – Send time messages when a valid time and position message is received from the GPS. E.g., GPS update.

Orbit Propagator

In real satellite systems, the orbit propagator is often used to estimate future satellite positions and velocities based on current measured values such as GPS input and Two Line Element (TLE). The orbit propagator interacts with the nanosat flight software to obtain GPS and TLE and also outputs its estimated values to the flight software to control radio transmissions.

Subsystem Simulator

The subsystem simulator in NCCT provides simulated values found in a typical nanosat to other nanosat applications. The following nanosat subsystems will be considered in our testbed:

- Power subsystem including solar array, battery and power distribution

- Communications subsystem including radio, antenna, power amplifier and low-noise amplifier

- Attitude determination and control (ADCS) subsystem including reaction wheels, IMU, magnetometers and star tracker

- Guidance and navigation control (GNC) subsystem including GPS receiver

These four main subsystems will be simulated in the nanosat space module based on its on-orbit parameters obtained for a specified nanosat orbit generated using the orbit simulator. In particular, the testbed will simulate battery levels, satellite orientations, radio conditions, and GPS data. As the development progresses, we plan to incorporate higher fidelity simulation models to obtain these parameters as these models will enable more accurate evaluation of autonomous scheduling algorithm performance using this testbed.

Scheduling

The scheduling submodule represents an autonomous message delivery scheduling system for store-and-forward communications nanosats. In [5], an autonomous message delivery scheduling strategy for nanosats is developed. In particular, the scheduling strategy considers message priorities, onboard energy, and contact time windows with destinations to generate an optimal message delivery schedule. The scheduling submodule interacts with the orbit propagator and the subsystem simulator through the nanosat fight software to obtain information on future contact time windows with message destinations and nanosat energy levels.

2.3 Ethernet Switch

The Ethernet switch and cables establish connectivity among the virtual ground nodes on the PC and the nanosats on the BBBs forming a nanosat communication constellation network. The UDP/IP are

used to transport content from each of the device to any other device connected in the network.

3 TESTBED IMPLEMENTATION

Following the architecture development phase, we begin the implementation phase using COTS products for low-cost rapid prototyping. The hardware consists of widely-used commercial embedded computer called BealgeBone Black (BBB) as a nanosat flight computer (also referred to as Nanosat Space Module), Ethernet switches and cables for User Datagram Protocol (UDP) device communication, and a computer workstation with virtual machine software to emulate ground nodes.

BBB is a low-cost, community supported development platform with the following specifications [6]:

- Processor: AM335x 1GHz ARM Cortex A8
- 512MB DDR3 Ram
- 4GB 8-bit eMMC flash storage
- USB host
- Ethernet
- Debian, Android and Ubuntu OS support

It is currently used in various unmanned systems such as quadcopters, small UAVs and nanosats. Various organizations analyzed and tested use of BBB as a flight computer for a nanosat. The result seems promising, however, it has not yet flown on operational nanosats.

Any ethernet switches, cables and computer workstations can be used to develop the NCCT. For our testbed, we used a 4-port ethernet switch. However, based on the mission scenario, the ethernet switch needs to support at least the total number of nanosats and a workstation which will host multiple ground nodes.

In the testbed, we are using the GOTS flight software that was used in DoD 6U multi-mission bus developed by Pumpkin Inc. The Core flight software is developed using C and it is hosted in BBB to perform command and data handling functions. Most of the modules connecting to the Core flight software was developed using Python programming language.

The commercial software, VMWare, is used to setup multiple virtual Linux-based ground nodes where graphical user interfaces (GUI) are developed and utilized to configure simulation parameters and obtain mission analysis output. The reason for VMWare is to minimize hardware required to develop in the testbed. Figure 4 shows a picture of the testbed setup.

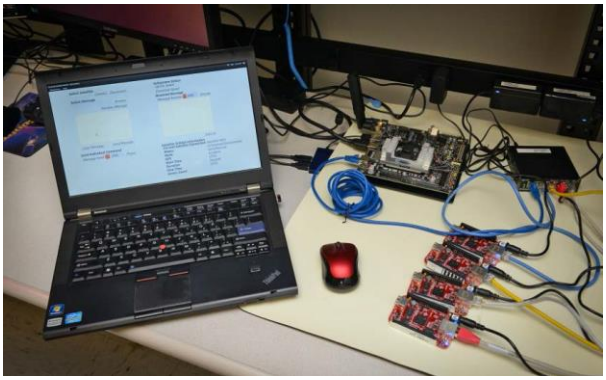

Figure 4: Testbed Setup

3.1 Interface to User-developed Software

The scheduling submodule situated in the nanosat space module is essentially a software payload located in the BBB. A software wrapper coded in Python allows for both interfacing with the core nanosat flight software services and rapid implementation of user-developed software [7]. Data transport is achieved with the wrapper which provides receive and send capability between ground nodes and nanosats. The interface with the core nanosat bus services makes telemetry data available for the scheduling submodule to use during message delivery scheduling decision computation.

3.2 Scenario Simulation GUI Design

The envisioned scenario is to have *M* number of ground nodes and *N* number of nanosats to emulate a nanosat communications constellation. The functions included for the scenario include random message generation, message discretization and message delivery. A database of ground nodes, nanosats, contact time windows, and sunlight time windows were collected via STK simulation. The database is the reference for location of ground nodes and information pertinent to communicating with the satellites, i.e. the access times to each ground node and the umbra, penumbra, and total eclipse times for each individual satellite. The scenario simulation is also an interface to run user defined scheduling strategies. In particular the testbed is set up to demonstrate different message scheduling strategies including highest priority rule strategy and other proposed strategies as defined in [5]. A high-level GUI was developed to incorporate the full functionality of the described scenario. With the GUI a ground operator or user has the ability to select the message source and destination ground node, generate messages of different describing metrics, upload databases

containing satellite characteristics including access times and sunlight data, and analyze the performance of the algorithm. The following figure shows the screen capture of the core functionality of the GUI.

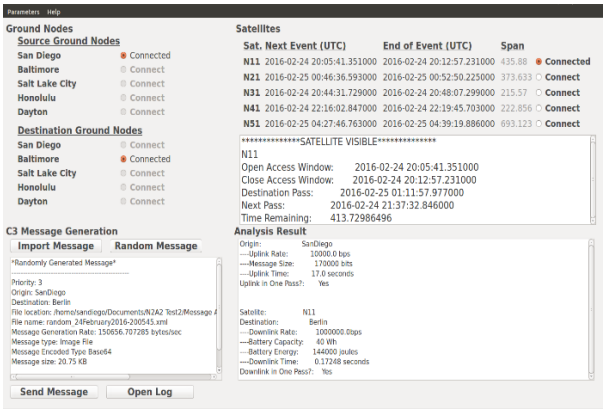

Figure 5: Testbed GUI design

As detailed in the preceding figure (Figure 5) the GUI is broken up into four main sections: Ground Nodes, Communications, Command and Control (C3) Message Generation, Satellites, and Analysis Result. In the Ground Nodes section, the user has the option of setting the message source ground node and message destination ground node through the use of radio buttons. The underlying code for this process defines the IP address and port base address of each location for UDP connectivity. The C3 Message Generation section is used to generate a message and send it to the destination ground node. There is code to either import a known message or to randomly generate a message. The message generated is an XML (eXtensible Markup Language) file that defines message origin and destination, message priority, message size, original message type (i.e. video, document, email), etc. The send message command button makes space packets from the XML file and 'uplinks' the packets to a nanosat flight computer using UDP. The Satellites section utilizes a pre-simulated STK database to show the access windows for the satellites. The user has the option to manually connect to a satellite if it is visible to the defined message source node. The Analysis Result section shows results pertinent to message transmission such as uplink and downlink rates and the performance of a user defined algorithm.

3.3 Testbed Message Flow

Utilizing a suite of open source and custom libraries in Python, the scenario simulation is a Python script that utilizes UDP to communicate with ground service equipment and flight software. Below is a breakdown of how the message flows in the testbed:

- When starting the simulation, a GUI is displayed and shows the options.
- For the demonstration, a user can manually define the message origin ground node and message destination ground node.
- Once both are chosen a table will populate with the access information for all satellites in the scenario. This is mainly as a visual aid.
- The user will have a choice to send a known message or randomly generate a message that will be uplinked from the origin ground node.
- A user defined ground scheduling algorithm is then computed to choose the best available satellite.
- When the chosen satellite is visible to the origin ground node the message will begin to be uplinked. Custom Python scripts are used to chunk the message and make it CCSDS compliant.
- If the entire message is uplinked in a single pass, the flight software automatically routes the message to the software payload where it is stored.
- Orbital propagation information, in the form of an STK database for the demonstration, will determine the location of the satellite and the access window of the destination ground node.
- Simultaneously the software payload will run a user defined nanosat scheduling algorithm to sort message delivery based on factors including, but not limited to, transmit energy available, message size, and message priority.
- Once the computation is finished messages will be downlinked to their appropriate destinations based on the results.

A fully downlinked message is then stored and analysis is run, thus ending the flow of a single message from origin to destination.

4 CONCLUSIONS AND FUTURE WORK

Our mission-level nanosat communication constellation testbed development work utilizing COTS hardware and GOTS flight software is in progress. Our next step in the testbed development process will be to integrate with user-developed ground and developed nanosat message scheduling software and perform mission-level demonstration and evaluation on a nanosat communications constellation with multiple nanosats and multiple ground nodes. Our development effort

thus far includes the HiL mission-level testbed framework architecture with submodule definitions, various Python scripts for defining flight software interfaces to user-developed software, and for scenario simulation user interfaces and various submodules of the NCCT. A testbed providing a mission-level analysis framework and demonstrating autonomous message delivery scheduling systems will be the outcome of this work.

Acknowledgments

This work was funded under SPAWAR Naval Innovative Science and Engineering (NISE) program.

References

1. Naval Research Laboratory, SUPERNOVA Flight Software, version 1-8-1 [Computer software].
2. Naval Center for Space Technology, Naval Research Laboratory, "SUPERNOVA Flight Software Developer's Guide," SN-DEVG-001 Revision 1.0, December 2016.
3. Consultative Committee for Space Data Systems, "Space Packet Protocol CCSDS 133.0-B-1 Cor. 2," Sep. 2012.
4. Naval Center for Space Technology, Naval Research Laboratory, "SUPERNOVA Bus to Payload Data ICD," SN-BPLICD-001 Revision 1.0, December 2016.
5. C.Y. Wakayama, P.J. Yoo, and Z.B. Zabinsky, "Energy-Cognizant Scheduling of Store-and-Forward Communications with Multiple Priority Levels in Nanosatellite Systems," Proceedings of the 30th Annual AIAA/USU Conference on Small Satellites, Logan, UT, 2016, SSC16-P2-03.
6. G. Coley, "BeagleBone Black System Reference Manual," Revision B, Jan. 2014.
7. Pumpkin Inc., "SUPERNOVA Payload Interface Control Document (ICD)", UM-16, January 2017.