Multi-Spacecraft Network for Ground-Space VPNs: Architecture, Radio Specifications and Testbed

Marcos Bergamo, Ph. D.
BBN Technologies,
10 Moulton St,
Cambridge, MA 02138.
Email: marcos.bergamo@bbn.com

David Lapsley, Ph. D.
BBN Technologies,
10 Moulton St,
Cambridge, MA 02138.
Email: david.lapsley@bbn.com

***Abstract*— BBN Technologies (BBN), under complementary tasks for NASA, designed the High-Throughput Distributed Spacecraft Network (HiDSN), developed a laboratory prototype network testbed and, currently, is integrating support for virtual-private-networks (VPN) over such a testbed (SpaceVPN). The ultimate project goal is to demonstrate the use and the performance of standard-based IPSec VPNs for secure ground-experimenter access to emulated low-earth orbit sensor satellites in which ground-space-sensor connectivity is achieved using the multi-hop routed space links of HiDSN.**

This paper describes architectural options for establishing ground-space virtual private networks (i.e., SpaceVPNs), the network self-formation mechanisms, and the network architecture developed to extend the (terrestrial) Internet to space. In the testbed, both the physical layer processing and the media-access control protocol (i.e., MAC layer) were implemented using software-defined radio (SDR) techniques based on the ones developed for the GNU radio. In this paper, we also describe key decisions made to enable the use of SDR techniques for the spacecraft radios.

I. Introduction

The network architecture and the software-defined space-radio developed for the *High-Throughput Distributed Spacecraft Network (HiDSN)*[1] and its follow-on *Space-based Virtual Private Network (SpaceVPN)*[2] are the first to integrate time, code and spatial multiplexing with ad-hoc networking techniques to create a flexible feature-rich infrastructure for ground-space and in-space communications. The HiDSN system provides a self-forming vertically integrated network infrastructure for establishing and maintaining high-throughput multi-hop communications between ground and space and/or among multiple spacecraft operating in diverse orbits. The SpaceVPN builds upon the HiDSN infrastructure while focusing on problems related to enabling real-time, continued, secure experimenter access to instruments and data on-board of sensor satellites using the Internet and IPSec-based Virtual Private Networks (VPNs). Fig. 1 shows a high level view of a SpaceVPN network.

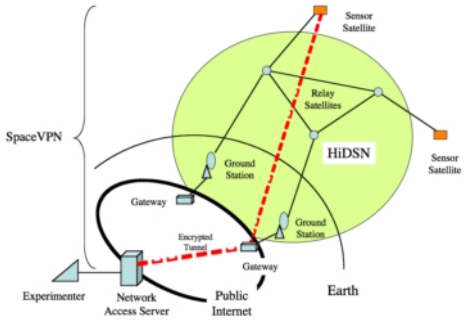


Fig. 1. SpaceVPN Network

This paper is structured in four parts. In the first part, we describe the overall HiDSN/SpaceVPN System Architecture. In the second part, we describe the testbed that was constructed to validate the system architecture. In the third part, we present and discuss some results from the testbed. Finally, we discuss our main conclusions and future work.

II. SpaceVPN System Architecture

In the SpaceVPN system, the HiDSN network provides the self-forming multi-hop network. It integrates the predictability of orbital movement to establish and maintain cross-links and multi-hop routes, ad hoc networking techniques to autonomously discover "new" neighbors, spatial multiplexing with dynamic digital beam forming to maximize re-use of the allocated spectrum, and variable-rate cross-links with multi-code spread spectrum to maximize network connectivity under large inter-spacecraft distances and distance differentials.

Transmission is performed in bursts, with typically one packet per burst. The spectrum is fully re-used on cross-links that are spatially isolated form each other. The spectrum is shared using time and/or time-code multiplexing during the times in which the spacecraft become "aligned." Multi-code encoding is used to adjust both the maximum effective symbol rate of a cross-link and, combined with multi-bit modulation,

[1]The authors gratefully acknowledge the support of NASA's Advanced Cross-Enterprise Technology Development for NASA Missions program under contract number NNC04CB16C.

[2]The authors gratefully acknowledge the support of NASA's Advanced Information Systems Technology (AIST) program under contract number NAS3-01101.

the effective instantaneous bit rate in which each packet is transmitted over such a cross-link. The inter-spacecraft communication strategy enables allocation of resources to meet QoS requirements on a per packet basis. The number of codes used in the encoding of each packet burst is selected to meet both bit error rate (BER) and delay requirements of each flow that compose the aggregate packet traffic over each cross-link. For each spacecraft, the number of in-range neighbors, and the distances and link performance (e.g., signal to noise ratio or SNR) of each of the cross-links change over time.

The network (self) organizes itself into a tree-like hierarchy for scalability. The elements that participate in the network are called, generically, nodes. Nodes can have different combinations of mobility, routing and transmission characteristics.

In terms of mobility, nodes are classified, generically, as in-orbit nodes (e.g., spacecraft in multiple-altitude LEO, MEO, GEO or elliptical orbits), air-mobile nodes (e.g., aircraft and air-mobile sensors), and ground-based nodes. Using orbital parameters, the relative position and mobility of in-orbit and ground-fixed nodes, albeit arbitrary, can be accurately predicted over time. The relative position and velocity of air-mobile and surface-mobile nodes have to be "discovered" and "tracked" over time.

Nodes that perform routing functions (e.g., spacecraft in low, medium and planet synchronous orbits), called router nodes (or simply routers), form a mesh network at the highest level. Nodes that are only sources or destinations of traffic, called endpoints, may be affiliated with a router. The endpoints themselves may be arranged in a two-level hierarchy (thereby making a total of potentially three levels) with some endpoints acting as a bridge endpoint for others. For instance, an aircraft could act as a bridge for a surface-exploration robot on the surface of the planet.

All nodes, regardless of its size, power and level in the hierarchy, have identical physical layer, media access control (MAC) layer and subnetwork (i.e., below IP) capabilities. The RF power, antenna gain and beamforming agility may vary from node to node.

Common capabilities, grouped per layer, are summarized in the following sections.

A. PHY-Layer

The PHY-Layer includes the following capabilities:

- *Transmission at a High-Frequency-Band (e.g., Ka):* Enables transmission at high data rates using small-size/ high-gain antennas/arrays.
- *Null Steered Burst Transmission:* Enables bust-by-burst transmission without causing interference to neighbor spacecraft.
- *Null Steered Burst Reception:* Enables a spacecraft to receive without interference from neighbor nodes and full reuse of the available spectrum in every spatially separable cross-link.
- *Variable Rate Cross-Links:* Enables spacecraft to maintain connectivity over a wide distance range (e.g., from 100 km to 10,000 km) by adapting the effective cross-link data rate to the varying link attenuation (e.g., from 100 Kbit/s to 1 Gbit/s).
- *Constant-Envelope RF:* Enables efficient use of the available RF power by allowing the operation of the array antenna power amplifiers at or near saturation

B. MAC-Layer

The MAC-Layer includes the following capabilities:

- Receiver-directed frame-based communications: Enables interference-free code multiplexing and time-slot based spatial multiplexing. Provides support for space-time synchronization (i.e., Direction of Arrival and propagation delay measurements), burst transmission of packet traffic with isolated (isolated datagrams), intermitting (bursty packet traffic) and recurring (stream traffic flow) characteristics, and for the discovery of new neighbors.
- Channel Access Mechanism that integrates Spatial-Division, Time-Division and Code-Division Multiple Access leveraging the direct-sequence orthogonal code multiplexing capabilities of BBN's TDMA with CDMA-encoding Multiple Access (TCeMA).
- A QoS-oriented Bandwidth Allocation Mechanism that integrates support for long-term bandwidth allocation for rate-based and volume-based application traffic, and for packet flows with different delay and delay-jitter characteristics.

C. Subnetwork-Layer

The Sub-Network Layer is highly oriented to maintaining the network connected (i.e., a single "island") and to controlling the connectivity degree (i.e., number of neighbors) in the constellation. It includes the following capabilities:

- Neighbor Discovery Protocol: Enables a node to advertise itself, find other nodes, and achieve frame, time-slot and initial frequency synchronization with any other node that that happens to be within range.
- Network Synchronization Protocol: Enables each node in the network to, over time, achieve global time-and-frequency synchronization.
- Position Based Routing: Maintains at each node, a topology database with information of the characteristics of each active link and leverages predictable positioning and dynamics of nodes in space to make proactive link state dissemination and routing decisions.
- Node Affiliation Protocol: Enables endpoint nodes to find a router node that can relay traffic on its behalf, and perform dynamic hand-off as necessary as the network topology and geometry changes over time.
- Packet Forwarding Protocol: Makes decisions, when a packet arrives, of what should be done with it (i.e., consume, relay or drop).

D. IP Services and Interfaces

The proposed architecture does not require each satellite node in space to support IP services, but does require sensor satellites to provide IP services for their onboard instruments in order to allow researchers to access those instruments. We expect that sensor satellites will usually affiliate as terminal nodes with a HiDSN satellite, and that there will be one or more HiDSN satellite nodes between that sensor satellite and the ground station supporting it any point in time. It is also possible that a ground station may connect directly to a sensor satellite without any intermediate HiDSN satellites, depending on the relative position and distance of the sensor satellite

The IP that supports the connection from the researcher to the instrument can span multiple satellite links, depending on the number of HiDSN satellites along the path, but those links need not support IP services. The router at the ground station and the instrument on the sensor satellite are the only two required IP nodes in the path between the ground station and the instrument.

When a sensor satellite connects directly to a ground station router, the sensor satellite IP interface must support neighbor discovery between the sensor IP interface and the ground station router IP interface as well as packet forwarding between the instrument and the ground router. The term "connects directly" in this sentence refers to the IP network-level connection. Of course, the direct connection requires an existing link-level connection between the ground station and the sensor satellite, as well as local network connections to the ground station router and local satellite connections to the instrument.

When a sensor satellite connects to the ground station router through intermediate HiDSN satellites, neighbor discovery and packet forwarding services between the instrument and the ground router are still required, but these are carried out through a subnetwork layer on the HiDSN satellites. HiDSN satellite routers use subnetwork functions to select the best available path between HiDSN nodes to connect the sensor satellite and ground station router IP interfaces.

E. Planned VPN Protocol Support

For SpaceVPN IPSEC-based VPNs, in addition to IP, the spacecraft nodes must support all the protocols required for IPsec with a PKI using certificate-based keys (ESP, AES, IKE, ISAKMP, etc.). The requirement for certificate-based keys implies that the sensor satellite nodes must also support interfaces to Certificate Authorities (on the ground for this stage of the architecture), including key management protocol for over the network key distribution and revocation. The recommended CA includes X.509 certificates on X.500 directory servers accessed via Lightweight Directory Access Protocol (LDAP). The exact choice of protocols will depend on the Certificate Authority that NASA selects (NASA has used an Entrust CA internally in the past) for this architecture, because some CA functions, particularly those related to monitoring and management, still vary between vendors. Because certificates will be crucial to implementing and managing VPNs, the CA interface will have to support multiple primary and backup CAs. The space-based sensor nodes will access these CAS via ground stations at multiple Internet addresses as a satellite's ground connection moves from one ground station to another during orbit.

F. Support for Ground-Space VPNs (SpaceVPN)

A Virtual Private Network (VPN) is a private network running over a shared public infrastructure such as the Internet. Several technologies form the building blocks of Virtual Private Networks:

- *Tunneling:* makes the network between VPN endpoints look like simple links, even if it actually involves complex connections through multiple infrastructures. IPsec, Point-to-Point Tunneling Protocol (PPTP), Layer 2 Tunneling Protocol (L2TP), and Multi-Protocol Label Switching (MPLS) are examples of tunneling protocols.
- *Authentication:* verifies the identities of VPN users and devices and ensures that all data transmitted on the VPN originates only from those authenticated sources. Public Key Infrastructure (PKI), RADIUS, and Smartcards are examples of authentication mechanisms. Access Control provides ways to manage the authorized use of resources. X.500 directory servers and Access Control Lists (ACLs) are technologies that organize and distribute access control policies.
- *Data Security:* ensures that the data transmitted over the VPN is not visible to anyone except the participants in the VPN. Cryptographic algorithms such as Triple Data Encryption Standard (3DES) and Advanced Encryption Standard (AES) are used to encrypt and decrypt VPN data.
- *Data Provisioning:* ensures that network resources will provide adequate delivery for VPN packets, so that measurable performance characteristics like delay and throughput suit the applications transmitting data over the VPN. QoS and Traffic Engineering are examples of techniques used to provision network resources for VPNs.

Implementing interfaces for each of the VPN building blocks is complicated, from both performance and security standpoints, and the solutions are computationally intensive compared to those used for providing unprotected network access. Special purpose VPN devices such as VPN gateways and special-purpose software such as VPN clients are often necessary to provide adequate support for users, particularly for high-speed data connections such as those to many NASA satellites.

A VPN gateway has two or more network interfaces. The interface on the public Internet side accepts only packets destined for established secure tunnels and specific protocol messages needed to negotiate and establish those tunnels. The interface on the private network side accepts only packets that

conform to policy rules on the gateway about what kinds of traffic should travel through secure tunnels (e.g., traffic with a destination address listed in the VPN gateway's policy table). The VPN gateway sets up a secure connection to another VPN device at an appropriate destination on behalf of the user whose traffic was received on the gateway's private network interface, unless a tunnel for that traffic already exists. Once a tunnel is in place, the gateway forwards the packet from the private network over the tunnel, applying appropriate data security and authentication processing before the packet emerges on the public interface. In addition to the user traffic just described, a VPN gateway usually accepts monitoring and control traffic on a secure management port associated with one of its interfaces, depending on how the gateway is deployed.

A VPN client connects a single private workstation to a remote VPN device over the workstation's network interface, providing the same tunneling, authentication, access control and data security functions.

VPN tunneling functions are most frequently implemented at layer 2 (Medium Access Control (MAC) and layer 3 (network) of the OSI model. Layer 2 solutions are most often used for dial-in services where the Point to Point Protocol (PPP) is tunneled through the Internet to reach an office network, so that the user does not have to pay for a long-distance phone call to the office. Layer 2 VPNs have generally proven less secure and less manageable than layer 3 VPNs, so we recommend layer 3 tunneling for NASA solutions, specifically tunnels that use the IPsec suite of protocols (IETF RFCs 2401-2412).

The IPsec protocol suite allows the endpoints of a VPN to establish a Security Association (SA) that determines how these IPsec peers will exchange data securely on that VPN. The protocols define ways that peers can identify each other, exchange key information, create shared secret keying material, and negotiate the SAs that are required to implement IPsec tunnels. IPsec provides many options for performing network encryption and authentication. Each IPsec connection can provide encryption, integrity, authenticity, or all three services.

IPsec requires that the endpoints of the tunnel first authenticate each other, for example by using digital certificates. To implement the security services, the two IPsec peers must determine exactly which algorithms to use (for example, AES or 3DES for encryption; MD5 or SHA-1 for integrity), and then share session keys that use these algorithms. The IPsec peers use the Security Associations that they establish for each VPN to determine whether a particular packet belongs to a VPN or not, and how to process the packets that do. If the VPN includes authentication, the peers will authenticate each packet using AH or ESP protocols in tunnel mode.

The IPsec VPN that supports the secure connection from the researcher to the instrument on board the sensor satellite can span multiple satellite links, depending on the number of HiDSN satellites along the path, but those links need not support IP services.

III. Testbed Overview

We constructed a testbed to validate the system architecture described in the previous section. The testbed constructed integrates the following key technologies:

- Software Defined Radio (SDR)
- HiDSN
- TCeMA
- Emulation of Null-steered spatial multiplexing
- Orbit-predictable spacecraft integrated with ad-hoc techniques for neighbor discovery and self-formation
- Standard IPsec VPNs
- Position-based routing

The SpaceVPN Testbed leverages standard practices for SDR and GNU-principles while providing for future interoperability. Architectural features include:

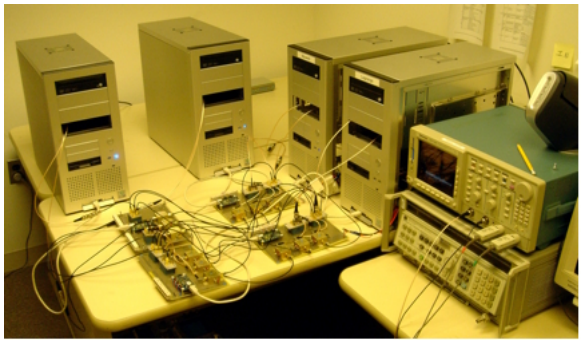
- Clean separation between PHY/MAC and Hardware layers PHY and MAC layer interface
- Separate Hardware Abstraction Layer process
- Standard interface between Python and C/C++ SDR modules Flexibility to move C++ code modules to DSP platform to support significantly higher data rates
- Significantly compressed development cycle through use of Python/C++ paradigm, software loopbacks and block-level implementation and testing
- Event driven MAC with synchronous interface to the hardware.

In its current form, the proposed architecture is the result of over five years of development at BBN Technologies [1]–[8]. The feasibility of the architecture has been verified through extensive simulation. To validate the architecture, we have implemented a hardware and software testbed. In this paper, we describe the testbed implementation and present some results.

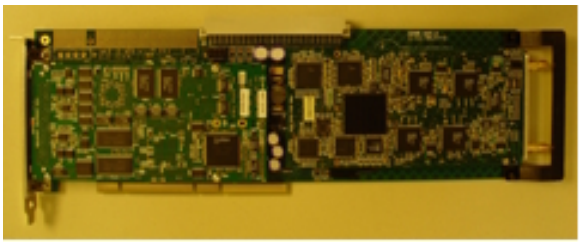
A. Hardware

The testbed consists of 4 Intel-based Linux Servers shown in Fig. 2(a). Each of the servers is equipped with a Red-River WaveRunner PMC 301 Radio Frequency Board (Fig. 2(b)) used for 70 MHz baseband transmission/reception, a Pentium IV processor running at 2 GHz, and 1 GB of RAM. The servers each run the RedHat 9.0 Linux distribution. The choice of distribution and version was determined by the WaveRunner driver software.

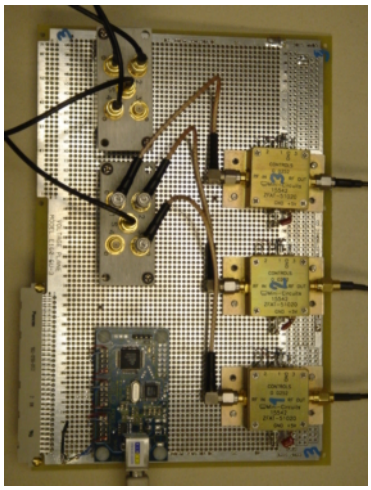
Each of the servers is equipped with a Dynamic Beam Emulation (DBE) Board designed and constructed at BBN. The DBE boards are also used for mobility management (as described in Section III-D and III-E). The DBE boards are equipped with voltage-controlled variable attenuators that allow each server to control the level of transmit side attenuation to each of the other servers and hence emulate dynamic beam forming and path loss variation between nodes. The transmit



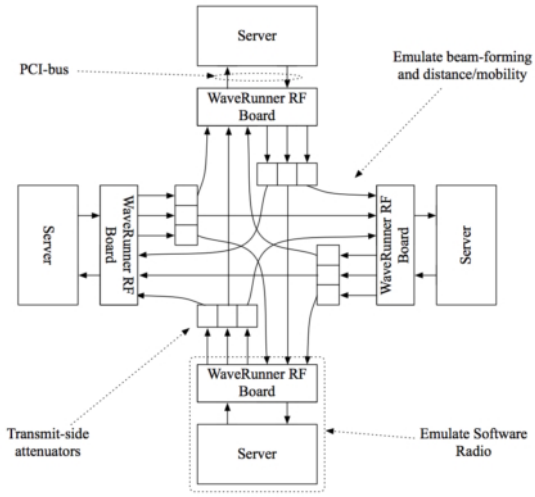
(a) Testbed Overview



(b) WaveRunner RF Board



(c) Beam Emulation and Mobility Management Board



(d) Testbed Connectivity

Fig. 2. Testbed

signal from each server's WaveRunner board is split and sent to each of 3 attenuators. Commands from each server are transmitted over a USB interface to a USB controller on the DBE board, where they are then converted to TTL levels that are used to control the attenuation levels of each of the attenuators. On the receive side, the signal fed into a server's WaveRunner receive interface is simply the sum of the transmit signals of all of the other servers. Fig. 2(c) shows one of the DBE boards.

The four nodes are connected in a mesh topology via the dynamic beam/mobility emulation boards as shown in Fig. 2(d). Fig. 2(a) shows the actual testbed.

B. Software Overview

GNU Radio [9] is the defacto standard Software Defined Radio (SDR) architecture in the open source community. GNU Radio uses a modular, block-based architecture to implement complex signal processing chains in a convenient and efficient manner. These signal processing chains are capable of generating or receiving radio signals from a variety of sources: IEEE 802.11 access points, AM/FM broadcast radio, Television, Celestial Radio sources, and so on. GNU Radio uses a hybrid Python/C++ architecture where blocks are implemented in C++, which is suitable for computationally intensive processing, but also provide interfaces to the higher level Python [10] programming language. Python is a higher level language that provides a convenient mechanism to control the configuration of large numbers of blocks and the high level program flow of the SDR.

The SpaceVPN physical layer processing and Media-Access Control protocol (i.e., MAC layer) are implemented using SDR techniques based on the ones developed for the GNU Radio. The Physical and MAC protocols are both implemented using an architecture similar to that of GNU Radio. Functionality in both the transmit and receive paths is implemented in modular blocks that are then connected into processing graphs. Data and metadata are input to these graphs and the output of the graphs is complex I- and Q- samples for transmission, in the case of the transmit graph, or user data, in the case of the receive graph. In the SpaceVPN testbed, this block-based architecture has been implemented from scratch and includes a number of extensions to the existing GNU Radio architecture:

- Extension of basic block-based architecture from PHY to PHY and MAC layer
- Extension to include state machines incorporated in a configurable manner
- Dynamic processing chain configuration
- Designed for discrete event driven systems
- No buffer limitation, pass pointers instead of buffers
- Dynamic memory management: blocks are able to dynamically vary the size of buffers
- High visibility and exchange of information between modules
- Support for frame structure
- Support for flexible, acknowledgeable asynchronous event notification
- Parallel data and control streams
- Configurable MAC layer
- Support for time-based burst transmission

In addition to these enhancements, we have developed hardware and software that enable us to perform dynamic beam emulation and emulation of node mobility.

C. SpaceVPN Software Radio

There are three main architectural features implemented as part of our SpaceVPN Software Radio:

- *Active Objects:* [11], [12] describe active objects. These are objects that run in their own thread of execution and are able to pass messages to each other in a thread-safe manner. Active objects may implement both asynchronous and synchronous methods. Synchronous methods are blocking methods that do not return until the underlying operation has completed. Asynchronous methods are non-blocking methods that take as a parameter a callback thread-safe message queue. Asynchronous methods return immediately, and notify the caller when an operation is complete by sending a message to the callback queue. Active objects are one of the building blocks of the SpaceVPN Software Radio.
- *Message Passing:* information is passed between blocks as discrete chunks of information consisting of data and control. Data usually consists of bytes, bits, M-ary QAM symbols, TCeMA spread multi-codes or samples. Control information usually consists of transmit or receive profiles that supply parameters to the blocks in a particular processing chain. These parameters could include, for example, modulation/demodulation scheme, Forward Error Correction scheme, transmit power level, and so on.
- *Control Plane:* in the GNU Radio model, only data is moved between blocks. In our implementation, data and control information are moved between blocks in lockstep. Each data path has a parallel control path. Each data message that is sent between blocks is accompanied by a control message that traverses the parallel control path. The use of a control path enables us to dynamically reconfigure our processing blocks.

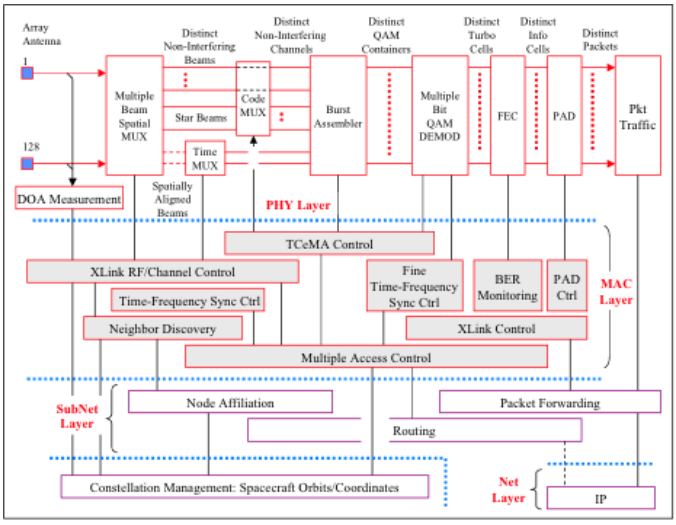


Fig. 3. HiDSN/Space VPN Radio Layers, Functions and Interfaces

- *Asynchronous request model:* enables event driven radio operation with full visibility (e.g., by higher layers) of processing progress at individual modules, including actual transmission times of frame-scheduled bursts and status changes in the SW/HW interface. The use of profiles for passing configuration information between layers and between processing blocks provides a high degree of configurability.

Fig. 3 illustrates the functional architecture of the radio and network protocol subsystems we have been developing throughout the project. Fig. 4 shows an overview of the SpaceVPN software functions, interfaces and architecture. Also illustrated are the interfaces between such component subsystems. The detailed characterization of these interfaces was shown to be key for creating a modular, expandable and flexible radio subsystem.

Fig. 4 shows an overview of the software architecture. The MAC layer is implemented purely in Python and consists of a number of active objects each implementing various functionals. These functionals include neighbor discovery, neighbor synchronization, neighbor link maintenance, mobility manager, scoreboard scheduler and so on. The MAC layer communicates with the PHY layer below it via the use of the HiDSN PHY Layer API [4]. This interface is implemented by the HiDSN Task object shown in the figure. Round objects are implemented in C++, while all other objects are implemented in Python. Within the PHY layer, both the transmit and receive chains are implemented using a hybrid Python/C++ model similar to that used by GNU Radio: computationally intensive software is written in C++ and exposes high level Python interfaces for ease of access and configuration at the Python layer.

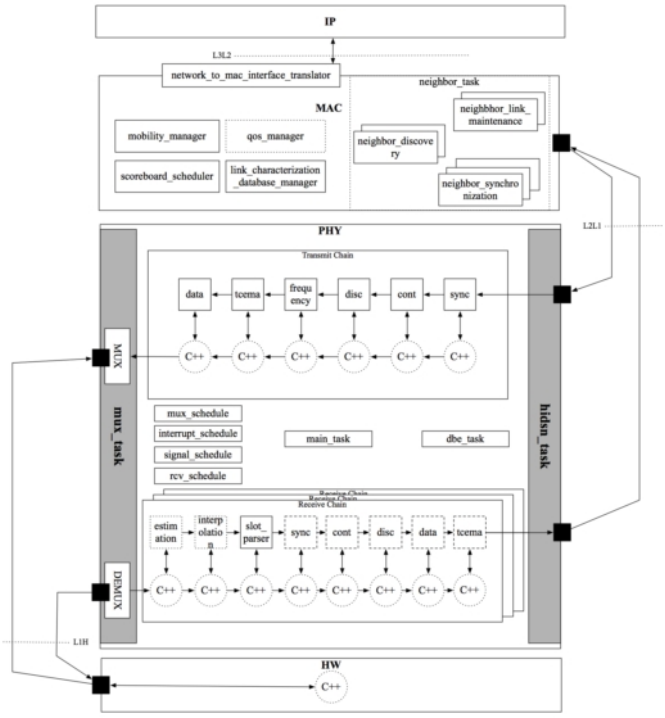


Fig. 4. Software Architecture

D. Dynamic Beam Emulation

The SpaceVPN Testbed includes four dynamic beam emulation boards. Each beam emulation board (Fig. 2(c)) is associated with a single RF board and host combination. The dynamic beam emulation boards enable a host to split its RF output into three separate beams. Each of these beams can be individually attenuated via the use of computer controlled attenuators. The attenuation can be varied from 4.0 dB up to 35.0 dB. By varying the attenuation, it is possible to emulate both null-steering and distance variation between nodes. On the receive side of the boards, the RF output from all neighbors is multiplexed and directed into the RF input of the computer associated with the board.

The boards are controlled via a USB I/O 24 TTL microcontroller. This provides an interface between the USB board on the host computer and the TTL logic required to drive the attenuators. The dynamic beam emulation driver software is implemented in Python.

Software has been developed and integrated that allows burst-by-burst attenuation control. Each burst to be transmitted has an associated "dynamic beam emulation" profile. This profile specifies the attenuation required on each of the three beams for transmission of the burst. This effectively allows the host to control the attenuation on two of such beams (i.e., null-steering direction emulation) and the attenuation in the direction of a target destination node (i.e., combined beam-gain and path loss emulation).

Fig. 5 shows sample oscilloscope traces that show the result of burst-by-burst beam emulation and variable link attenuation (mobility emulation). Fig. 5(a) illustrates the capability of the beamforming emulator software to generate burst-by-burst attenuation controls that are synchronized with the time instants in which the corresponding data burst becomes available for transmission. Fig. 5(b) illustrates the capability of creating an emulated null-steered beam composed of an emulated high-gain/low-attenuation (4 dB) beam towards a target neighbor node and an emulated null/high-attenuation (35 dB) beam towards a non-intended neighbor.

E. Mobility Management

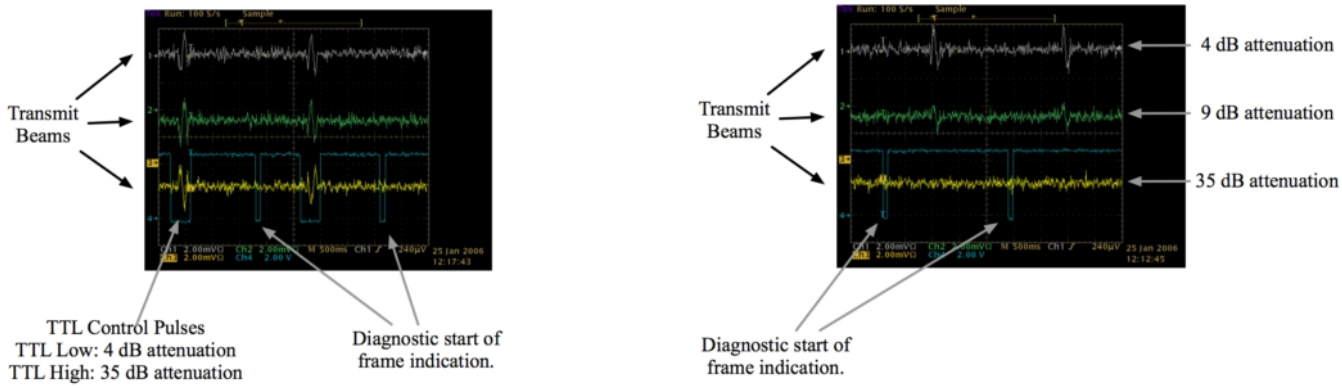
Mobility management also utilizes the dynamic beam emulation/mobility boards, but in a slightly different manner than the dynamic beam emulation software. Mobility management operates in a global manner and allows a centralized "mobility manager" to control the attenuation on all of the attenuators in the system. In doing so, the mobility manager is able to dynamically create network topologies where the emulated distance between nodes varies with time.

The mobility management function is implemented in two parts: a central "mobility manager" that controls the overall network topology and host-based "mobility agents" that communicate with the mobility manager, obtain topology updates from the mobility manager and then individually adjust the attenuation on their associated dynamic beam emulation boards. Thus, the mobility manager exercises global topology control by directing individual hosts to attenuate their beams appropriately. The overall topology is defined in a configuration file that is read by the mobility manager. This configuration file specifies a sequence of instructions to be sent out to each host in the network. All hosts in the network are updated at the same time. The instructions include how much attenuation to apply to each beam as well as instructions on which neighbors are in range and which neighbors are out of range. Hosts send an acknowledgement to the mobility manager upon the completion of a specific event (e.g. neighbor discovery). Once the mobility manager has received acknowledgments from all of the hosts in the network, it issues instructions for the next topology.

F. TCeMA Containerization

Data transmission is performed in packets of arbitrary length (in bits). Packets are encapsulated as illustrated in Fig. 6. They are segmented in fixed-length chunks; chunks are FEC protected in turbo chunks; turbo chunks are M-ary QAM modulated (and compressed) into possibly fewer modulated chunks; modulated chunks are grouped into fixed-size payloads; payloads are transmitted in containers formed by pre-pending a fixed-size BPSK header to each payload.

The payload size is fixed and dimensioned to carry exactly eight 1920 M-ary QAM symbols. Distinct packets are carried in distinct payloads. The modulation level (i.e., bits/symbol) is selected on a per packet basis and, consequently, there is no



(a) Beam Emulation.

(b) Mobility Emulation.

Fig. 5. Beam and Mobility Emulation Results

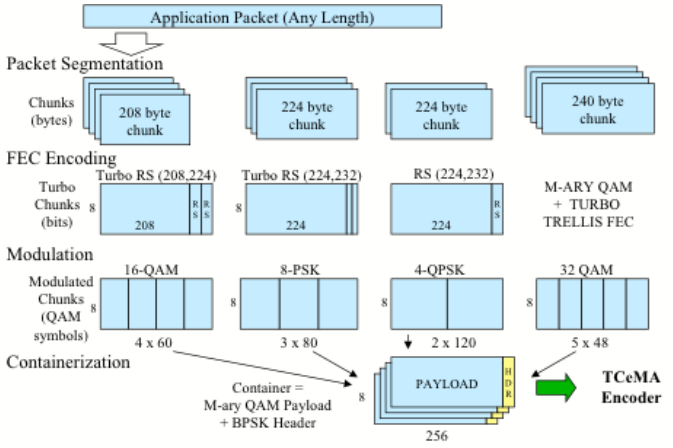


Fig. 6. TCeMA Containerization: Packets can be transmitted using a variety of modulation levels, but are always carried in fixed-size payloads of 1920 QAM-symbol each.

mix of modulation levels in a payload. The modulation level used in the payload is specified in the container header. The number of modulated chunks that fit in one payload varies with the modulation level selected for the packet.

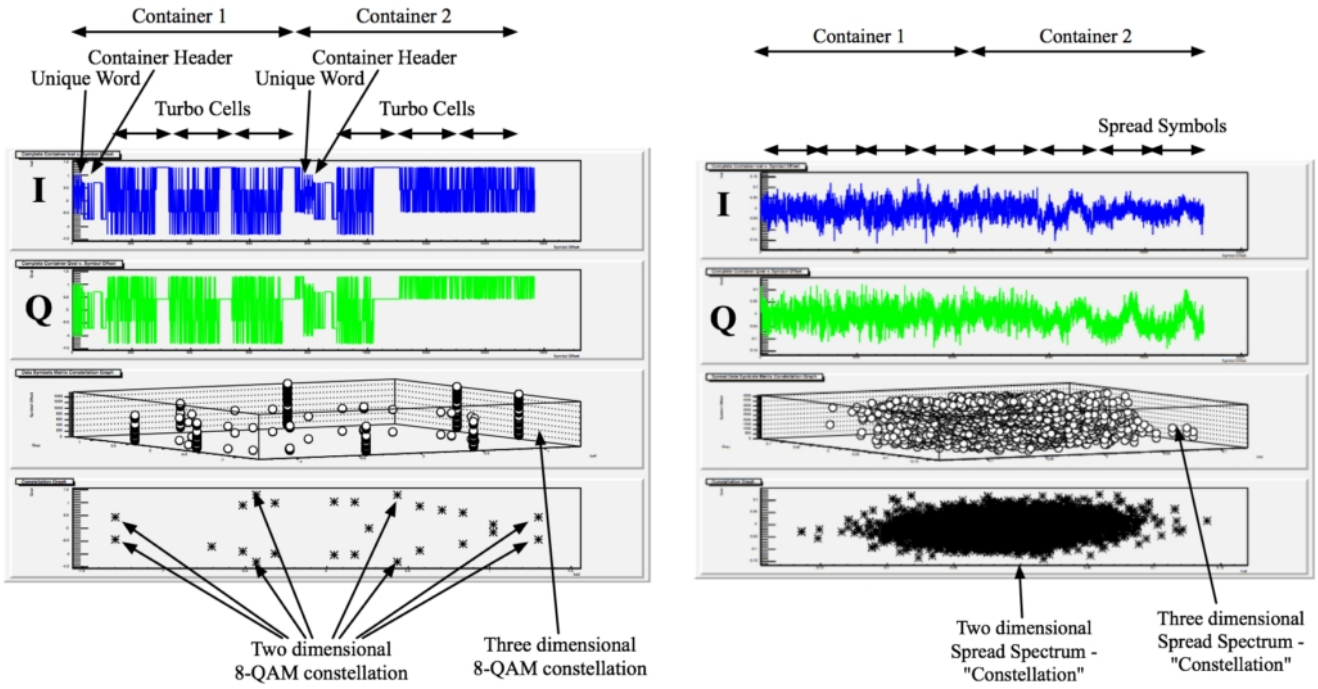
Fig. 7 shows the final stage in the TCeMA containerization process. Fig. 7(a) shows two containers worth of I- and Q- samples (along with the three-dimensional and two dimensional constellation diagrams). Fig. 7(b) shows the two containers *after* TCeMA spreading.

G. Sample Offset Estimation and Interpolation

In practice, the transmit/receive clocks of spacecraft communicating using SpaceVPN would be similar, however, not identical. In order for two spacecraft to communicate, and, more importantly, to enable a spacecraft to receive simultaneously from multiple neighbors, they must first synchronize their clocks. SpaceVPN provides three mechanisms to do this: coarse-grained timing and frequency synchronization during neighbor discovery, fine-grained timing synchronization through the synchronization slot, and extremely fine-grained timing synchronization though the sample offset estimation and interpolation process shown in Fig. 8.

In order to correct for these timing and frequency offsets, we must first estimate the magnitude of the offsets, then choose an optimal interpolation vector that will compensate and finally interpolate the received waveform with this vector to ensure that the sample points are again at optimal portions of the interpolated waveform.

Fig. 8 shows a detailed flow chart of the interpolation process. Data is transmitted from the transmitter and over the channel to the receiver. The channel will delay and distort the transmitted symbol vector. The *rcv-symbols-vector* represents the transmitted signal, as seen at the receiver having been passed through the channel. The receiver oversamples the *rcv-samples-vector* by a factor of 4. These samples are then passed into block 4, which reconstructs the HiDSN container from the received samples. Container reconstruction involves using cyclic correlation to extract the appropriate set of codes from the received samples. Each timeslot consists of 8 spread symbols, with each spread symbol containing up to 512 codes. The 8 code "stripes" in each timeslot are extracted and then "spliced" together to form a HiDSN container. The *reconstructed-container-samples-vector* is then sent to block 5 for timing offset estimation and to block 6.2 for optimal downsampling. Block 5 determines the timing offset. This is a measure of the temporal offset between the received sampled signal waveform and the optimal sampling point. Block 5 calculates the *sample-time-offset* (this time), the *closest-sample-phase* (which identifies which of the 4 oversample phases is the closest to the optimal phase) and the *first-sample* to use. These three quantities are forwarded to block 6.1 which then chooses the optimal interpolation vector, while the *closest-sample-phase* is sent to block 6.2 which then downsamples the *reconstructed-container-samples-vector* by



(a) TCeMA Containerization: Penultimate step; container in time domain.

(b) Final Stage of TCeMA Containerization: Final step; spreading of containers just prior to transmission.

Fig. 7. TCeMA Containerization and Spreading

selecting only those samples that correspond to the closest-sample-phase. Finally, the *optimal-interpolation-vector* and the *downsampled-reconstructed-container-symbols-vector* are forwarded to block 7, where they are convolved to produce the interpolated waveform. This *interpolated-downsampled-reconstructed-container-symbols-vector* is then forwarded to block 8 for demodulation.

Fig. 9 shows an example of a sampled received signal waveform that would be received at block 4. Fig. 9(a) and Fig. 9(b) show the In-phase (I) and Quadrature (Q) samples time series respectively. Fig. 9(c) shows the magnitude of the sampled waveform. Fig. 9(d) shows the constellation diagram formed by plotting the in-phase versus quadrature component. By looking closely at Fig. 9(d) it is possible to discern a number of interesting features. There is a faint diagonal line that is aligned diagonally starting at the bottom left of the square formed by the samples and ending at the top right.

This line corresponds to the header samples of the container which use 45 degree Binary Phase Shift Keying (BPSK). In this case, the data portion of the container is using 16-QAM. It is possible to make out the faint 16-QAM constellation. This corresponds to a 4 x 4 grid of data points. Due to the delay inherent in the transmission and propagation process and the oversampling, there is a lot of noise due to intersymbol interference. This shows up as random scattering of samples about the nominal 16 positions. Ideally, we would see the grid as a distinct set of points, with all samples located on one of the 16 points of the grid. In practice, the observed signal would not be able to be decoded regardless of how high the received signal to noise ratio (SNR) . However, using the procedure outlined above, we are able to practically eliminate the intersymbol interference and allow the system to achieve the BER corresponding to the actual received SNR, including processing gain from cyclic correlation.

Fig. 10 shows the interpolated symbols obtained by convolving the optimal interpolation vector with the *reconstructed-container-vector*. This figure is in the same format as Fig. 9. There are a number of differences. First of all, the in-phase and quadrature components of the signal in Fig. 10(a) and Fig. 10(b) show a number of discrete levels. Compare this with Fig. 9(a) and Fig. 9(b) that have samples distributed seemingly randomly over the range of amplitude. Similarly for the magnitude in Fig. 10(c) versus Fig. 9(c). The most striking and important difference is between the constellation diagrams in Fig. 10(d) and Fig. 9(d). In Fig. 10(d) we can clearly see the distinct points that form the 4 x 4 16-QAM constellation. Note the clear separation between points and the lack of any random points between or around grid points. This represents a much higher SNR (due to the elimination of intersymbol interference) than shown in Fig. 9(d) and will result in a much lower BER after demodulation. It is also possible to see points that correspond to the 45 degree BPSK that is used in the header, as well as a few points spread around the constellation that correspond to the Unique Word that is

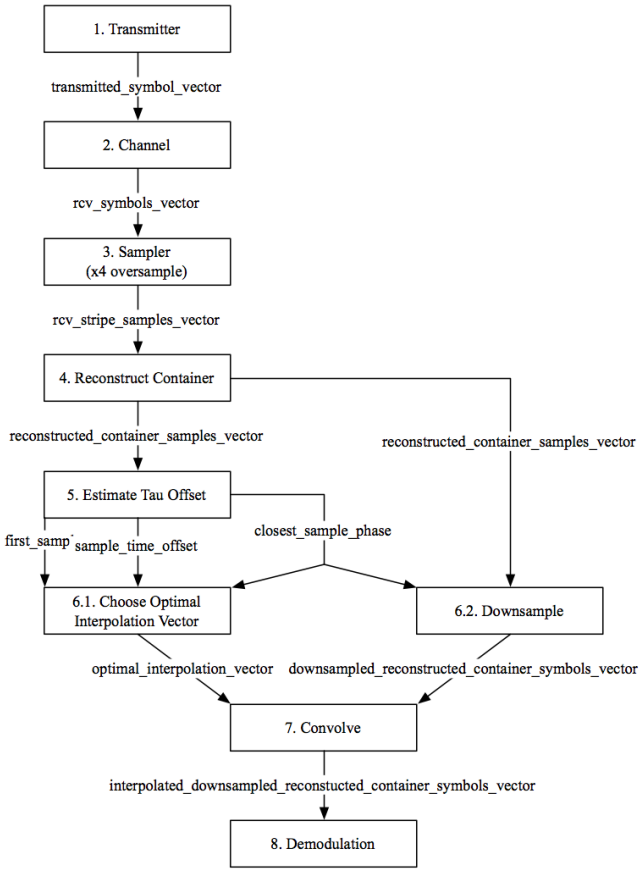
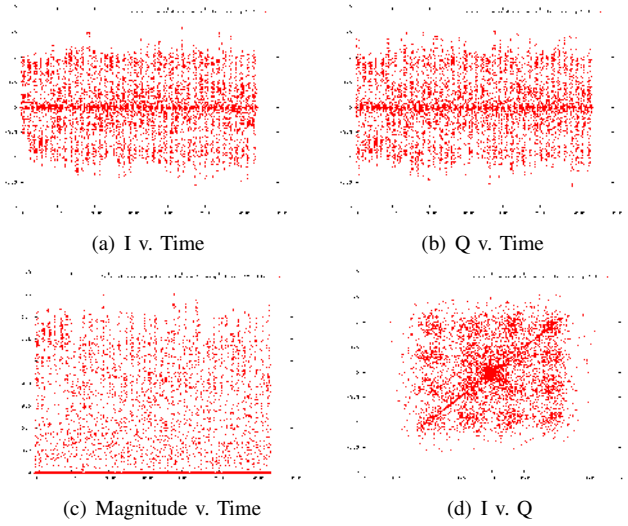


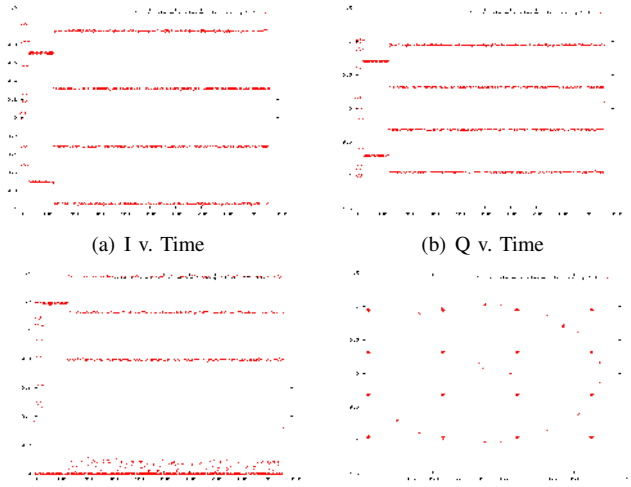
Fig. 8. Interpolation Process



(a) I v. Time (b) Q v. Time

(c) Magnitude v. Time (d) I v. Q

Fig. 9. Received Samples Waveforms



(a) I v. Time (b) Q v. Time

(c) Magnitude v. Time (d) I-values v. Q-values

Fig. 10. Interpolated Samples Waveforms

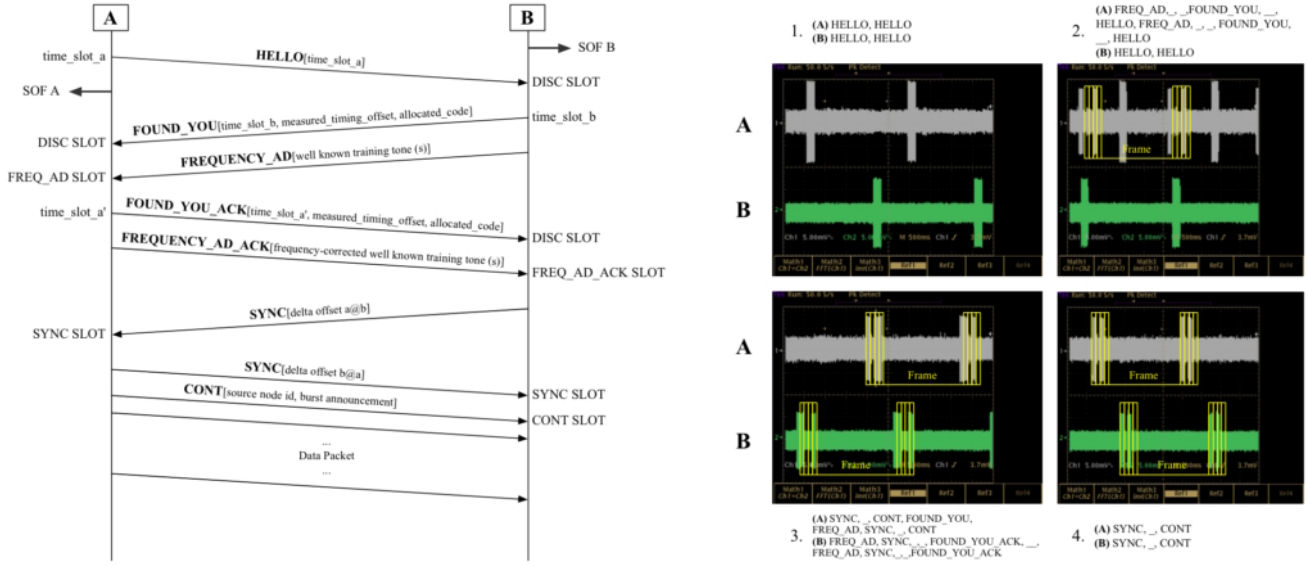
used within the header of each packet.

By enabling the extremely fine-grained timing and frequency synchronization of spacecraft transmit/receive clocks, SpaceVPN enables spacecraft to spacecraft communication. In addition, SpaceVPN enables a single spacecraft to receive and decode simultaneous or time overlapped transmissions from multiple spacecraft with similar, but non-identical transmit/receive clocks.

H. Network Self Formation

Although in a typical space-based system one can assume that the relative position (i.e., coordinates) of the various nodes can be calculated based on orbital equations, one cannot and possibly should not assume that all information required to establish a cross link will be available at all times at all nodes. In the SpaceVPN system, such an initial connection is established through a Neighbor Discovery protocol that assumes minimum or no knowledge about the eventual presence and 3D-location of a potential neighbor node. In our developed Neighbor Discovery protocol, illustrated in Fig. 11(a), the only information we assume known (but not required) by the node that performs neighbor discovery is the approximate relative angular direction of a potential neighbor. This angular direction was used in our laboratory prototype tests and demonstrations to transmit *HELLO* bursts using (emulated) null-steered beamforming in a way that the new neighbor can be discovered and a receiver-directed synchronized data link can be established without causing any interference with already established neighbors. The neighbor discovery process is essential to creating a network with self-forming characteristics.

In the SpaceVPN such neighbor discovery is performed using three types of control bursts, each carrying increasing levels of information: *HELLO* bursts are transmitted "whenever possible" while scanning the transmitting node frame. A *HELLO* burst, when received in a specific time slot of the receiver's frame called *DISCovery* time slot cause the transmission of a *FOUND-YOU* burst that includes detailed timing and carrier frequency synchronization information. This



(a) Network Self-formation sequence diagram

(b) Network Self-formation results

Fig. 11. Network Self-formation with inter-node time and frequency synchronization

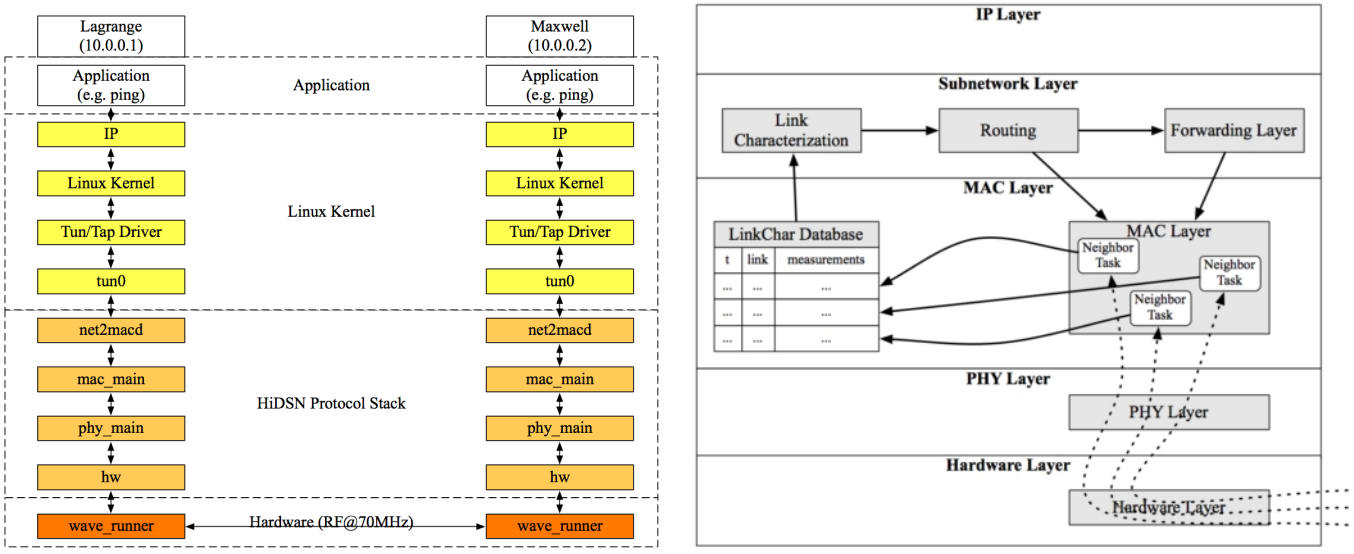
information can be in the form of actual parameters encoded in the burst or information that can be measured at the receiving node. Under normal circumstances the *FOUND-YOU* burst is received in the *DISCovery* time slot of the node that transmitted the *HELLO* burst and triggers the transmission of an acknowledgement burst called *FOUND-YOU-ACK*. The exchange of *FOUND-YOU* and *FOUND-YOU-ACK* is sufficient for the two involved nodes to start exchanging *SYNC* bursts. *SYNC* bursts are used to measure the SNR quality of the cross-link and to define maximum achievable bit rates (given desired packet BER). These SNR measurements are used to keep the link alive (or not) and to periodically update (at least one per multiframe) time and frequency synchronization as well as relative angular orientation. The relative angular orientation is important for non-interfering transmissions performed with null-steered beamforming. For this, *SYNC* bursts are received with two forms of multiplexing: code multiplexing and null steered beam multiplexing. Comparisons between these two forms of multiplexing enables compensating for Radio Frequency (RF) energy "leaks" that may occur when one uses actual array antennas (with non negligible coupling among neighbor array elements). In Fig. 11(b) we illustrate a typical neighbor discovery burst exchanged between previously unknown but neighbor nodes used in network self-formation lab demonstrations with emulated null-steered beams. In the figure, we also illustrate (yellow overlay) the control time slots of the SpaceVPN frame: *SYNC*, *CONTention*, *ACTivity* and *DISCovery*. The CONT and *ACT* time slots are used to inform the receiving node about the relative time-code location of incoming data bursts. The function of these *CONT* and *ACT* is particularly important as it enables flexible multiplexing of multi-application data burst, possibly with different QoS requirements, over space links with long delays.

I. IP Integration

The SpaceVPN Software Radio integrates with the native IP stack running on our Linux servers. Fig. 12(a) shows the complete protocol stack from the IP layer down to the hardware layer. The HiDSN layer sits above the physical hardware and implements all of the technologies discussed in Section III. A software daemon, *net2macd* provides the interface between the HiDSN layer and the Linux kernel's IP stack. This daemon communicates with the HiDSN layer via a UNIX named pipe (a form of Inter-Process Communication) and with the Linux kernel via the use of the Linux *tun* interface. This driver enables a user-space application to insert itself into the Linux protocol stack.

On transmit, IP packets from applications such as ping are forwarded to the Linux IP stack. IP packets are intercepted by the Linux kernel and forwarded to the HiDSN protocol stack via the Tun/Tap driver and *net2macd* daemon. Packets then traverse the HiDSN protocol stack and are transmitted via RF hardware. On the receive side, the reverse process is used.

Fig. 12(b) shows a simplified diagram of the planned final integrated application. The HiDSN PHY layer will forward link quality characteristics (received power, received noise power, frequency correction factor, fine-grain timing adjustment rate) to the data link/MAC layer. This layer will then store those in a trace record database. The trace record database is periodically queried by the link characterization module and returns the requested trace records. The link characterization module then calculates a link metric (s) based



(a) IP Integration

(b) (Planned) Full IP Integration

Fig. 12. IP Support

on the processed trace records and then forwards those metrics to the layer 2 routing module. The layer 2 routing module then calculates the next hop forwarding table and updates the forwarding modules table. The layer 2 routing module also calculates and forwards a set of radio profiles which are used by the physical layer to determine things such as the transmission power to use and the level of modulation to use for a given next hop neighbor.

Fig. 13 illustrates the support for IP demonstration performed to emulate the Internet access of an experimenter (Fourier) connected to the terrestrial Internet to an IP-addressable instrument on board a sensor satellite (Maxwell) through a dynamically established RF link established between a ground site (Lagrange) and the sensor satellite.

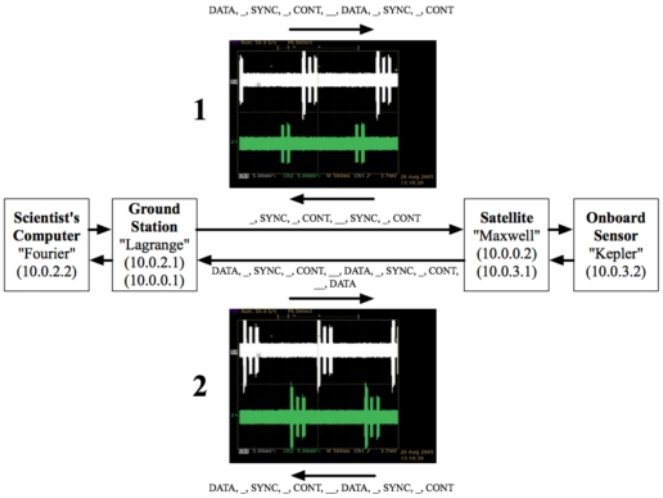


Fig. 13. IP Integration Results

IV. Conclusions

We have described architectural options for establishing ground-space virtual private networks (i.e., SpaceVPNs), the network self-formation mechanisms, and the network architecture developed to extend the (terrestrial) Internet to space. In the testbed, both the physical layer processing and the Media Access Control protocol (i.e., MAC layer) were implemented using Software Defined Radio (SDR) techniques based on the ones developed for the GNU radio. In this paper, we also describe key decisions made to enable the use of SDR techniques for the spacecraft radios.

The SpaceVPN testbed includes the following key technologies:

- Software Defined Radio
- HiDSN
- TCeMA
- Emulation of Null-steered spatial multiplexing
- Orbit-predictable spacecraft integrated with ad-hoc techniques for neighbor discovery and self-formation
- Position-based routing

Current results demonstrate the feasibility of the proposed architecture and technologies in providing end-to-end IP connectivity between ground-based scientists and sensors onboard LEO sensor satellites. Future work will involve the integration of standard IPSec VPNS into the testbed to demonstrate the feasibility of the overall SpaceVPN architecture.

ACKNOWLEDGMENT

The authors gratefully acknowledge the support of NASA's Advanced Cross-Enterprise Technology Development for NASA Missions program under contract number NNC04CB16C and NASA's Advanced Information Systems Technology (AIST) program under contract number NAS3-01101.

REFERENCES

[1] M. Bergamo, *Wireless Data Communications System,* U. S. Patent No. 6,104,708. U.S. Patent and Trademark Office, August 2000.

[2] ——, *Constant Envelope Signal System and Method,* patent pending. U.S. Patent and Trademark Office, November 2001.

[3] ——, "High-Throughput Distributed Spacecraft Network (HiDSN): TCeMA Scheduling: Algorithm for Burst Transmission Time Calculation," *Project Report, NASA Contract No. NAS3-01101*, 2003.

[4] ——, "High-Throughput Distributed Spacecraft Network (Hi-DSN): Physical-Layer API Specification," *Project Report, NASA Contract No. NAS3-01101*, 2003.

[5] ——, "High-Throughput Distributed Spacecraft Network (Hi-DSN): MAC-Layer Design," *Project Report, NASA Contract No. NAS3-01101*, 2004.

[6] M. Bergamo and H. Dempsey, "SpaceVPN Architecture: Space Network Environment, High-Throughput Distributed Spacecraft Network Services and Ground-Space VPN Deployment Options," *Project Report, NASA ESTO Contract No. NNC04CB16C*, 2004.

[7] M. Bergamo and R. Hain, "High-Throughput Distributed Spacecraft Network (Hi-DSN: A Full Capability Simulation Study Using OPNET," *Project Report, NASA Contract No. NAS3-01101*, 2004.

[8] M. Bergamo and S. Ramanathan, "High-Throughput Distributed Spacecraft Network (Hi-DSN): System Design," *Project Report, NASA Contract No. NAS3-01101*, 2003.

[9] GNU Radio. [Online]. Available: http://www.gnu.org/software/gnuradio/

[10] The Python Programming Language. [Online]. Available: http://www.python.org/

[11] D. C. Schmidt, M. Stal, H. Rohert, and F. Buschmann, *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*. John Wiley and Sons, 2000.

[12] D. C. Schmidt and S. Huston, *C++ Network Programming: Mastering Complexity with ACE and Patterns*. Addison-Wesley Longman, 2003.