Space Network (SpaceNet) Testbed – Development of a Multi-Functional Testbed for Simulating Space Communication Networks

John Downs *, Bruce Barbour †, Alexander Kedrowitsch ‡, Deven Mhadgut §, Suryansh Aryan ¶, and Samantha Parry Kenyon‖
Virginia Polytechnic Institute and State University, Blacksburg, VA, 24060

Low Earth Orbit mega-constellations make up a large portion of modern developments in space communications. The ability to simulate and properly research these complex systems is currently being developed to answer questions on the capabilities of the rapidly expanding industry. The development and improvement of a simulation-based platform for satellite network testing can help research efforts to enable industry and government entities to work together in the growing enterprise. The ongoing development of the SpaceNet Testbed is investigated, detailing its use of orbit-based constellation dynamics modeling, software-based emulation, and a hardware-in-the-loop integration design. An overview of the testbed's software infrastructure design is described alongside details regarding the make-up of its hardware components. Use cases are presented comparing the differences in performance between the satellite network of an actual Starlink mega-constellation currently in orbit and a custom constellation with the same quantity of satellites, but with ideal node spacing and initial orbital positioning. Results of these use cases are then discussed, focusing on the latency of the data traffic and how it differs when varying the testbed's user-defined configurations. In the future, resiliency testing and ground station to satellite link behavior analysis can be included into the testbed. The testbed has potential to help lead efforts in simulating complex space communication systems.

I. Introduction

Low Earth Orbit (LEO) mega-constellation networks continue to provide flexibility and accessibility in satellite communications. However, in many aspects, traditional research can find itself outpaced by ongoing industry developments and the rapid deployment of large LEO network satellites [1]. While there has been substantial research regarding LEO satellite communications in general, investigations into mega-constellations and their expected levels of data traffic are only just beginning [2][3]. LEO space networks already need to adapt to satellites' high speeds, intersecting orbits, and position variability over time, but these issues are made even more difficult by the size of today's mega-constellations.

Satellite internet has been available through networks existing in geosynchronous equatorial orbits (GEO), but these networks have limited capabilities due to high network latency and low bandwidth [4]. Current LEO mega-constellations like Starlink and the growing networks of OneWeb and Kuiper aim to provide high speed internet across the globe, with goals of significantly lowering network latencies compared to their GEO predecessors [5]. However, the lower altitudes of these networks require a number of satellites in the thousands. The large number of high mobility nodes requires the network to have near-constant topology, link, and route changes.

Finding new ways to efficiently simulate and test the behaviors of these complex satellite networks can improve how well these systems can be designed and understood. Developing a multi-functional testbed to simulate the capabilities and traits of mega-constellations creates room for a variety of satellite network testing without having to use or modify an existing network. One such testbed is Hypatia [1], which is specifically designed for LEO networks, but uses NS-3 to handle network simulation, which can have issues regarding scalability and is meant more for smaller networks [6].

*Graduate Student, Department of Aerospace and Ocean Engineering
†Graduate Student, Department of Aerospace and Ocean Engineering
‡Graduate Student, Department of Computer Science
§Graduate Student, Department of Aerospace and Ocean Engineering
¶Graduate Student, Department of Aerospace and Ocean Engineering
‖Research Assistant Professor, Department of Aerospace and Ocean Engineering

Many more testbeds are being developed to further expand the research field's ability to conduct both realistic and large-scale simulated experiments [7][8][9].

Currently, the Space Network Testbed (SpaceNet, formerly NeTSat [10]) is being developed to serve this role of a multi-functional LEO network simulator. The testbed works with a hybrid structure, combining a software-based emulator with a FlatSat-inspired, hardware-in-the-loop (HIL) integration design; it also makes use of Mininet, a network emulator that can use predefined network topologies to achieve large-scale functionality [11]. SpaceNet is the first of its kind to use Mininet while featuring software defined networking, a HIL hybrid design, and weather effects in its ground station modeling. The testbed's basic functions include data performance measurements of emulated satellite constellation networks and link creation between emulated networks and real, hardware-based nodes, but additional functions are currently in development involving constellation and orbit design experiments, network resiliency tests, and integrating emulated terrestrial nodes into the non-terrestrial network. SpaceNet capabilities will be shown here to compare the emulated network performance of a custom-made constellation and an existing section of the Starlink mega-constellation.

II. Testbed Overview

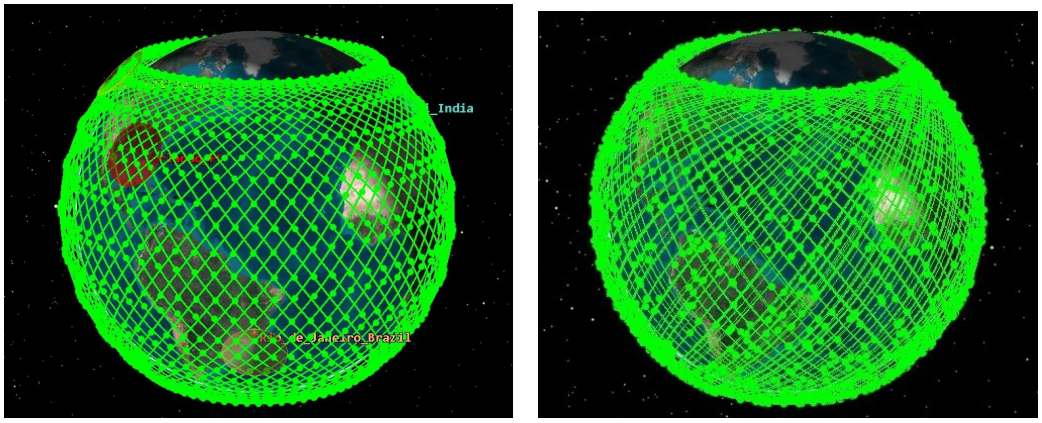
The software side of the SpaceNet testbed is split into two phases: simulation (Phase 1) and emulation (Phase 2). Phase 1 primarily involves sorting satellite data received in the form of two-line element (TLE) sets, simulating each satellite's orbital motion, and using each satellite's simulated positional data to predefine network link characteristics and routing tables. Phase 2 then uses this link and routing data with Mininet to create a virtual network of nodes representing each satellite in the constellation and each ground station that links to it. This emulation phase handles the delivery of data packets from node to node and the monitoring of network performance characteristics. The hardware portion of the testbed works as a hardware-in-the-loop (HIL) extension of Phase 2. It uses hardware-based nodes (emulating additional space network nodes) to communicate with the active virtual nodes and to investigate the link between a massive space network and real, independent devices.

A. Phase 1 (Simulation)

1. Actual vs Custom TLEs

The main functionality of the testbed is to simulate a network topology for a satellite constellation network; the network being simulated can either be a custom-made constellation or an actual satellite constellation with a real-life irregular distribution such as Starlink, OneWeb, etc. To use a custom-made constellation, the user must provide a collection of satellite TLE data for their constellation. The testbed has the ability to generate constellations formed in the Walker Delta pattern; the notation for this pattern primarily requires the user to input the total number of satellites in the constellation, the total number of orbital planes, and an orbital inclination value common to every satellite [12]. When a user provides these properties, the testbed can generate individual TLE sets for every satellite in the constellation. These custom-generated constellations will evenly distribute the satellites within their orbital planes, which are also equally spaced around the Earth. Additionally, the generated TLEs give the satellites a second derivative of mean motion of zero and a constant drag coefficient, giving the satellites ideal initial conditions (once orbit propagation begins, these ideal conditions will fade over time). Fig. 1(a) shows a custom constellation designed to match Starlink's original first shell of 1584 satellites. These 1584 satellites are spread across 72 orbital planes (i.e. 22 satellites per orbital plane).

When simulating an actual satellite constellation, the TLE datasets are extracted from a satellite tracking database (e.g. *Celestrak*) and provided to the testbed. The testbed then filters out the relevant satellites from the TLEs that are to be simulated based on user-defined constellation specifications. The TLE dataset acquired from the Celestrak TLE file for this set of experiments contains a total of 6403 satellites, which also includes the latest Gen2 Starlink satellites [13]. Currently, the testbed is specified to only simulate Starlink's LEO satellites with an operational altitude and inclination of 540 km and 53.2°. Using SpaceX's latest proposed reorganization for the Starlink constellation [14] and the satellite distribution that is tabulated in Table 1, the testbed effectively filters out the satellites that match the requested parameters, which totals to 1584. Fig. 1(b) visualizes these satellites in their orbits and works as a comparison between to the custom constellation in Fig. 1(a). Unlike the custom constellation, the Starlink satellites do not have ideal initial conditions and are not evenly distributed throughout the constellation's mesh-patterned grid. The differences in network performance between these two constellations can be found using SpaceNet's emulator and are crucial to understanding the testbed's capabilities.



(a) (b)

Fig. 1 (a) Starlink Custom constellation (b) Starlink Actual constellation from extracted TLEs

Altitude	Inclination	Orbital Planes	Satellites per plane	Total satellites
540 km	53.2°	72	22	1584
550 km	53°	72	22	1584
570 km	70°	36	20	720
560 km	97.6°	6	58	348
560 km	97.6°	4	43	172

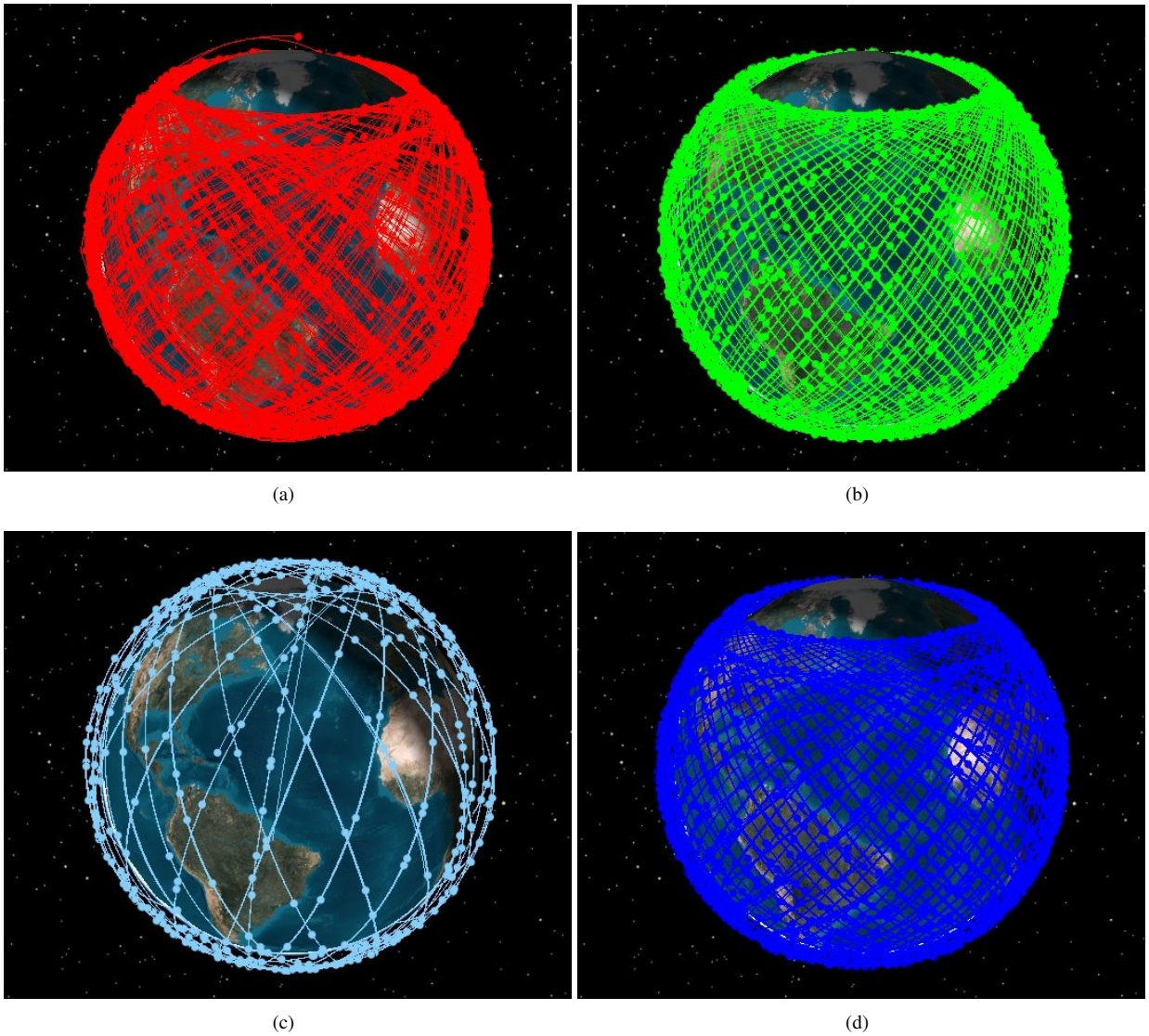
Table 1 Starlink Shell Configurations [14]

2. Orbit Sorting and Propagation

Once TLE datasets have been generated or acquired, the primary portion of SpaceNet's Phase 1 has begun. The testbed uses a YAML-formatted configuration file as a user interface; this configuration file is used to define experimental parameters such as simulation length, simulation time-step count, time-step length, simulation start time, and a variety of constellation specifications. When simulating a custom-made constellation, irrelevant TLEs can be removed beforehand and the constellation specifications (i.e. total number of satellites, number of orbital planes, etc.) listed in the configuration file simply need to match what was already generated. When an actual constellation using outside-sourced TLEs is simulated, the testbed uses the constellation specifications to select the relevant satellites from the TLEs.

Since there are over 6000 satellites in the TLE dataset used in these experiments, there are multiple groupings of satellites that must be sorted out of that initial pool. As of September 2024, the satellite distribution in an actual Starlink TLE dataset includes their Gen1 and V2 Mini satellites with a significant amount of them in extremely low orbits (<350km) intended for initial system checks. Moreover, the TLEs also provide sufficient knowledge on the distribution of satellites beyond the considered operational altitude of 540 km that essentially can be divided into two groups, one with an operational inclination of 53° and the other with a relatively higher operational inclination of 70° and 97.6°. These groups are depicted in the in Fig. 2, with Fig. 2(b) depicting the constellation being kept and tested.

After removing irrelevant satellites, the testbed sorts individual satellites to their respective orbital planes based on the orbital elements contained in the TLE sets, since the TLE data does not specify satellite groupings within orbital planes. The accuracy of the testbed's orbit determination plays a crucial role in setting up the initial conditions for the propagator that would simulate the entire constellation for a user-specified time duration and granularity. Therefore, the testbed leverages the high-fidelity measurements already provided in the TLE data to assign satellites to their respective orbits based on their 72 distinct operational Right Ascension of Ascending Node (RAAN) values according to Starlink's latest FCC report [14]. The orbital plane classification segment of the testbed promptly deals with this continuous distribution of RAAN values using a data clustering algorithm called the Fisher-Jenks algorithm [15]. Like any other similar algorithms, Fisher-Jenks classifies data to achieve low variance and set clear intervals (natural breaks) between the clusters. Moreover, compared to other clustering algorithms like K-means, it is relatively less computationally



(a) (b)

(c) (d)

Fig. 2 (a) Starlink satellites (<540 km; 43°**,** 53°**) (b) Starlink satellites (=540 km;** 53.2°**) (c) Starlink satellites (>540 km;** 70°**,** 97.6°**) (d) Starlink satellites (>540 km;** 43°**,** 53°**)**

expensive for large datasets with uneven distributions. Consequently, the orbital plane classification for custom TLEs would also follow the same process where the Fisher-Jenks algorithm is able to find natural breaks even faster because of the evenly spaced values of orbital elements. After designating satellites to their respective orbits, the testbed assigns unique IDs to individual satellites which the testbed exploits to recognize each of them in an orbital plane at any point in the simulation phase.

Each satellite is propagated using the Simplified General Perturbations Model (SGP4) provided by the Skyfield Python package. The SGP4 propagator considers the perturbation data for LEO spacecrafts already encoded in the TLEs, such as the atmospheric drag and the Earth's significant low-order zonal harmonics. Moreover, due to TLEs containing the mean orbital elements for the satellite, the SGP4 propagator is the ideal propagation model considered for the testbed application. Each testbed simulation is split into individual time-steps across its total simulation length, so each new data point in the propagation is separated by a user-specified time-step length. Additionally, the satellite propagation is a part of the testbed's supporting mobility utilities; These utilities involve: updating inter-satellite links (ISLs) and ground-station-to-satellite links (GSLs) for topology generation, assigning link characteristics, and generating network routes. The testbed also has a multiprocessing capability that can exploit the computational power of the host machine

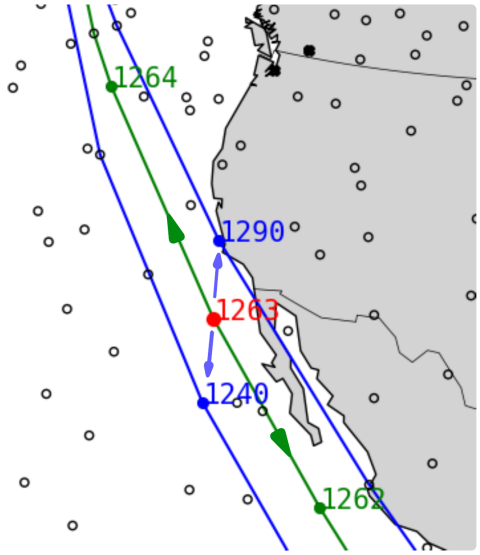


Fig. 3 PlusGrid Representation of Testbed constellation

to establish parallel mobility management processes for each time-propagated constellation instance.

3. Link and Route Generation

The topology generation involves constructing ISLs and GSLs as a precursor to generating a routing table collecting all of the network routes for the whole constellation specific to a pair of source and the destination ground stations, also called target nodes, provided by the user in the configuration file. Each pairing of nodes that create a link are saved in a file, with every link in the network eventually being used to form a connectivity matrix. Additionally, the GSL association criteria and the ISL grid structure can be defined by the user to determine the network topology. For the demo examples' topology generation, the association criteria for the target node's GSLs are configured to have each ground station establish links with the four satellites at the shortest distances within their fields of view. This allows the routing scheme to choose the most optimal satellite for the target node's pathing, rather than simply connecting to the closest satellite. The ground station's field of view (FOV) is determined by the satellite's operational altitude and the ground station's minimum sight angle and is used to determine a valid GSL connection with nearby satellites. The ISL connections for each satellite in the constellation are established with the satellites just ahead of and behind it in the same orbital plane as well as each satellite from the neighboring adjacent orbital planes with the shortest latency; the algorithm making these link selections is referred to as PlusGrid [16].

The visual representation of PlusGrid in Fig. 3 shows the optical laser-based ISL connections from the central satellite to its neighboring satellites that sufficiently guarantee the total connectivity of the network topology of a densely populated Starlink constellation. The numbers represent the satellite indices that the testbed assigns during the simulation, where an arbitrary satellite ID 1263 was found to have connections with satellite ID 1264 and satellite ID 1262 in the same orbital plane and satellite ID 1240 and satellite ID 1290 in the adjacent orbital planes. Specifically, a satellite would establish a two-way (bidirectional) connection with its neighbors on the same orbital plane that effectively facilitates the inter-hemispheric flow of the network traffic. Additionally, it also establishes another set of two-way connections with neighboring satellites in the adjacent orbits that promote the traffic flow along the latitudes. Consequently, when a satellite with such a ISL configuration forms a constellation, the overall optimal route from the source to destination nodes is computationally less expensive and is decently optimal in terms of metrics like latency or the number of hops, moreover favorably maintaining this optimal route for a prolonged period of time. An important remark on the application of PlusGrid regards a significant difference in how the algorithm works for the custom and actual TLEs; in the custom TLE case, all the satellites are bound to have exactly 4 ISLs at any given time interval, but this is not observed in the actual TLEs. Due to the uneven distribution in the actual constellation, the satellites have a significant chance of establishing ISLs with a neighboring satellite whose 4 neighbors are already determined. Ultimately, this would force certain satellites to have more than 4 ISL connections, given that all the satellites are

purposely required to perform two-way ISL connections. The testbed currently implements a PlusGrid ISL scheme for the entire constellation while considering range limitations on the ISLs and though it might provide fairly accurate routes and avoids frequent expensive link changes, there are plans to use a more adaptive grid algorithm for higher efficiency where the testbed simulator can leverage the time-based variations in satellite proximity with changing latitudes.

After the connectivity matrix generation is complete, link characteristics are assigned to each existing connection in the network topology. This includes the calculation of latency and throughput matrices for both ISLs and GSLs. The GSL throughput values are accurately computed by considering the density of users, the channel width, and the Signal-to-Noise ratio (SNR). The SNR also updated with weather condition data (using OpenWeather's Weather API [17]) for each ground station at every time-step in the simulation. The link characteristics in these matrices are primarily used as initial values for the emulation phase to build off of.

The users can specify the metrics that would be used to construct the routing table. After the topology is generated and link characteristics are determined, the next stage of the simulation is to execute a routing algorithm that computes the routing tables based on the metrics provided by the link characteristics. More specifically, the testbed exploits the NetworkX graphing API[18] to assign satellites and ground stations as graph nodes, the connectivity matrix provides connections as graph edges, and the metric values assigned to each edge are determined by the link characteristics. This construction of the constellation graph is followed by a routing scheme that computes every network route and ultimately constructs the routing table for the whole constellation. The current default routing scheme that the testbed utilizes is Floyd-Warshall's Shortest Path algorithm which handily calculates the shortest path for each pair of nodes (both satellites and ground stations) in the graph based on the decided metric [19]. If ground stations apart from the target nodes are also available to the testbed, the resultant routing table exhibits all possible paths data traffic can flow between any combination of target nodes among the provided ground stations. Along with the connectivity matrices, routing tables are generated for each time-step within the simulation, as each time-step will feature a unique network topology.

B. Phase 2 (Emulation)

1. Mininet Virtual Network

To emulate the modeled constellation network in real-time, SpaceNet employs the open source Software-Defined Network emulator to generate a massively parallel virtual network topology that leverages the full Linux network stack for realistic network protocol behaviors. The software side of SpaceNet is also run within a VirtualBox virtual machine (VM); this is the recommended approach by Mininet's developers [20] and it helps to prevent Mininet from potentially disrupting the rest of the user's machine and to focus it solely on the functions of the testbed. In Mininet, SpaceNet designates each satellite and ground terminal as a "Linux Router" device, allowing each node to operate independently to service network traffic as needed. SpaceNet has successfully emulated network constellations with over 1500+ nodes and 3000+ inter-satellite links. Fig. 4 provides a visual overview of Phase 2's operations.

The link topology and routing tables generated during SpaceNet's Phase 1 are provided as inputs to Phase 2, along with additional configurations specified in the simulation configuration file. As the version of Mininet employed in SpaceNet does not support the creation or destruction of links after the start of emulation, link topologies for all time increments are used to construct a "meta-topology" in Mininet where only links available for a specified time period are enabled with the rest disabled. SpaceNet allows the Mininet emulator to execute for the length of the current time period. After the specified time period elapses, SpaceNet updates link states to match the new instance topology while providing routing table updates to all network nodes as needed.

To aid scalability and researcher access, SpaceNet can build "optimized" Mininet topologies, consisting of only nodes and links identified as servicing application network data and relevant to paths between the source and destination target nodes. This optimized topology reduces topology complexity by orders of magnitude, permitting researchers to perform studies with SpaceNet on commodity systems and run experiments in a more timely manner.

By default, SpaceNet acts as a centralized coordinator, using Mininet's API to send commands to network nodes or modify the topology conditions. However, SpaceNet may also have each network node run independent commands or Python scripts, permitting decentralized and autonomous node execution. SpaceNet may still send and receive commands from network nodes through a secondary management network using Google's Remote Procedure Call (gRPC) framework. gRPC is an open-source network communication protocol that is fast, efficient, cross-platform, and well-adopted across the Internet [21].

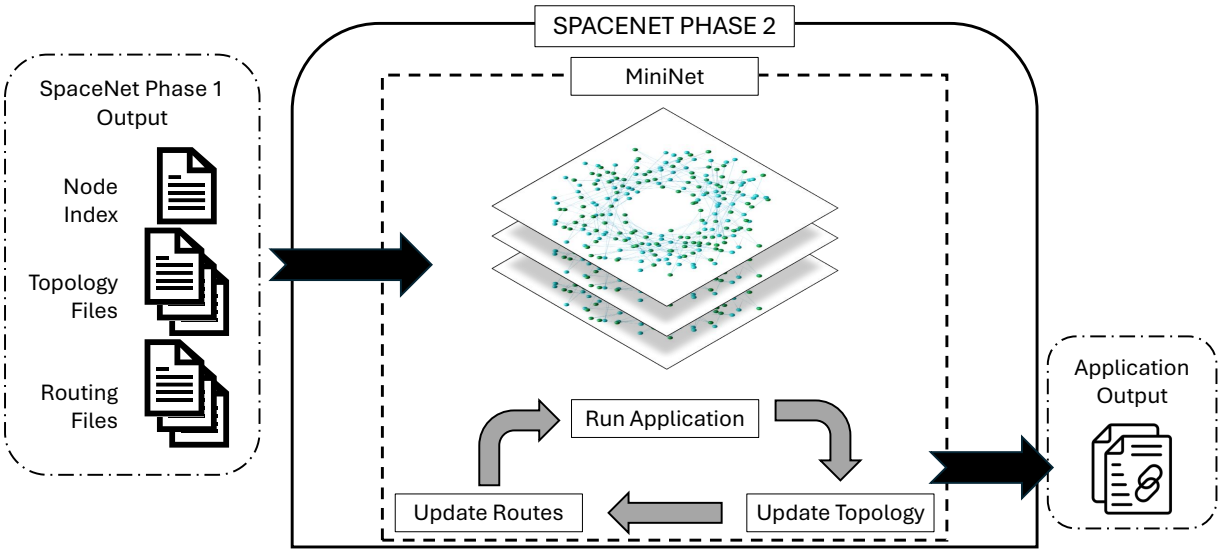


Fig. 4 SpaceNet Phase 2 uses Mininet to emulate a constellation topology for every time increment and produce application output for analysis.

2. Network Performance Measurement

After SpaceNet creates the network topology and starts the Mininet emulator, network performance data is collected by executing various Linux command line utilities and applications. Currently, SpaceNet fully supports the Linux Ping and iPerf command line utilities, but has the capacity to support future applications as needed. The ping utility simply sends ICMP packets from a specified source node to a specified destination node, and SpaceNet logs the round trip latency of the ping response (i.e. the amount of time it takes the packet to back and forth between the source and destination nodes). The iPerf utility consists of configuring an iPerf "server" on the specified destination node and having an iPerf "client" send TCP or UDP traffic from the specified source node to the destination. SpaceNet logs iPerf activity at both the source and destination nodes to gather various performance statistics such as data throughput, packet loss, and jitter.

With default configurations, Phase 2 applications are executed continuously with network topology and routing table changes occurring in the background at the appropriate time increments. However, depending on the size of the constellation or number of changes necessary from one time instance to another, application results may be skewed during periods of time the network is in a transitional "indeterminate" state. While this may be beneficial for specific research studies that are interesting in capturing this phenomenon, SpaceNet also supports running applications in discrete time periods, excluding "indeterminate" network transitions, and compiling individual application outputs into a single output file.

C. Hardware-in-the-Loop Integration

An experimental flatsat setup has been developed which configures Raspberry Pi (RPi) nodes to emulate a satellite constellation network, enabling bidirectional communication through software-defined radios (SDRs). GNU Radio is utilized to facilitate robust interactions and data transmission between these nodes, effectively simulating satellite-to-satellite communication pathways within the constellation. This configuration, as shown in Fig. 5 provides a controlled environment in which signal propagation can be systematically observed and tested, establishing a practical framework for initial experimentation. A simplified schematic of the hardware-in-the-loop testbed is presented in Fig. 6.

As previously mentioned in Section II.B.1, the software phases of the testbed are managed on a VirtualBox virtual machine (VM) running on a host machine. The VM is set up to have three bridged network adapters. With bridged networking, VirtualBox allows the host computer to intercept and inject data into the physical network through software. This creates a virtual network interface, allowing the VM to behave as another node on the host network with its own IP address. The first adapter is used to connect to the internal network and is configured to have Promiscuous mode ON, which allows the adapter to accept multiple MAC addresses, enabling all incoming traffic to reach the virtual network

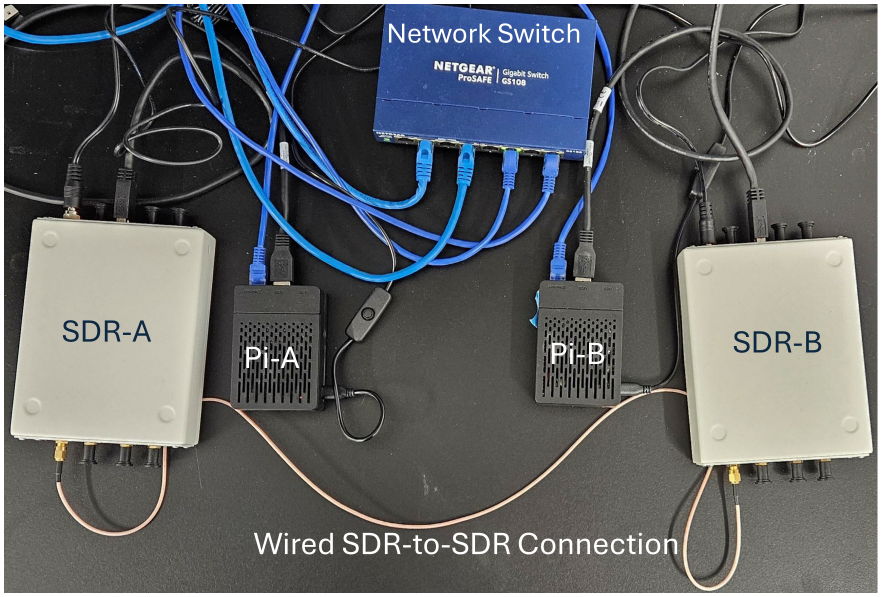


Fig. 5 Hardware Device Setup

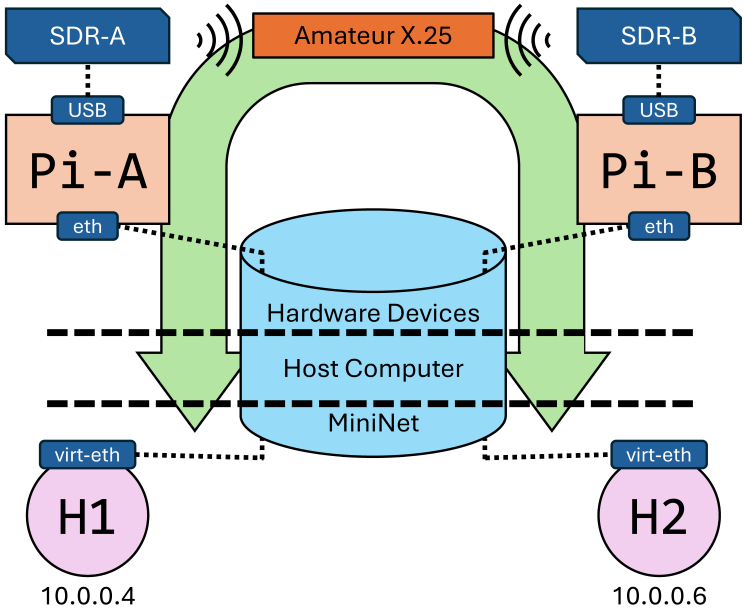


Fig. 6 Hardware-in-the-Loop Schematic

adapter of a VM, even if the traffic is not specifically addressed to it. The second adapter connects the VM to the internet. Finally, a third unconfigured adapter is also created to allow the VM to talk to the Raspberry Pis separately.

Each RPi and SDR pair operates as a satellite node, communicating using the Amateur X.25 (AX.25) protocol. GNU Radio shell scripts are used to configure the SDRs, assign static IP addresses, and display real-time TX/RX waterfall plots to monitor communication dynamics (shown in Fig. 7). On the RPis, key system configurations enable seamless packet transmission: packet forwarding is activated (sudo sysctl -w net.ipv4.ip_forward=1), and reverse path filtering is disabled (sudo sysctl -w net.ipv4.conf.all.rp_filter=0 and sudo sysctl -w net.ipv4.conf.default.rp_filter=0) to allow bidirectional communication through diverse network paths.

In Mininet, a virtual topology was designed using MiniEdit (Mininet's GUI), consisting of two hosts (H1 and H2) connected via an Open vSwitch (OVS). These virtual hosts are connected to the Raspberry Pis via virtual Ethernet (virt-

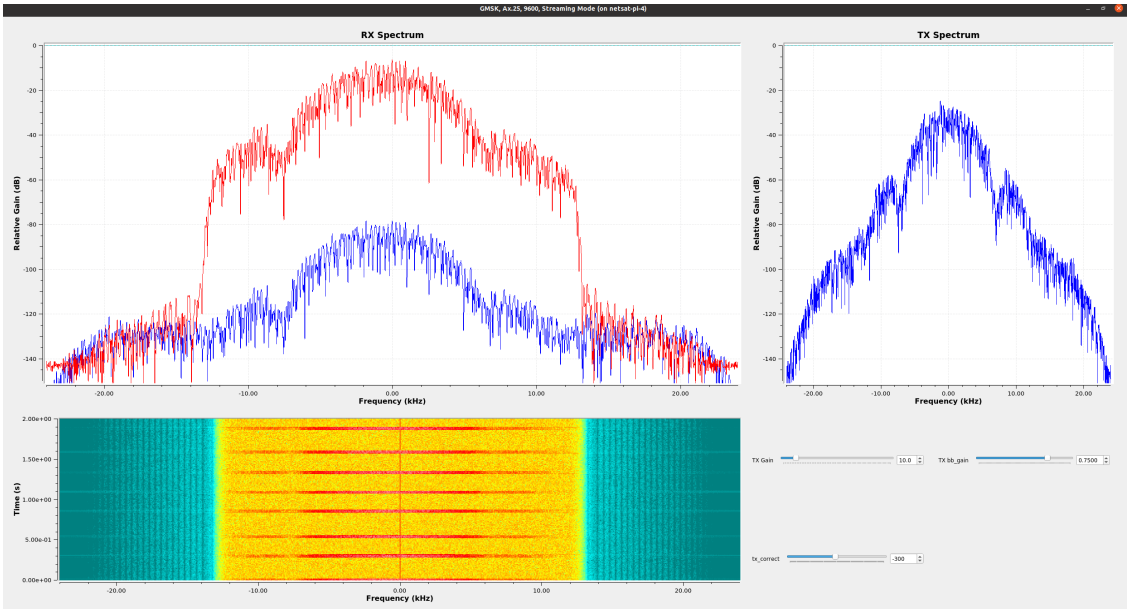


Fig. 7 GNURadio Waterfall Plots for Ping Tests

eth) interfaces. Static IPs (192.168.30.xx, where $xx > 30$) were assigned to the $eth0$ interfaces of the hosts, and the virtual OVS switch was linked to the external physical network switch. The command ovs−ofctl add−flow s1 action=normal configured the OVS to allow traffic to flow through the switch without additional filtering, facilitating communication between the Mininet environment and external hardware. To simplify network management, we configure the interfaces of both hardware and virtual nodes to reside within a single subnet. This setup presents a challenge when routing packets between same-subnet IPs through external relay nodes, which we overcome by assigning unique /32 (single-host) addresses to the Mininet host loopback (lo) interfaces and enforce policies that use these addresses as the source for all outbound packets. This configuration mirrors the host addressing practices employed in our larger SpaceNet topologies, supporting a unified static route configuration method. It also enables seamless changes to which virtual link is represented by hardware-in-the-loop during SpaceNet topology updates.

Routing tables were carefully configured to ensure seamless communication between the Mininet hosts and the RPis, forming a hardware-in-the-loop setup. For example, static routes were set on Mininet hosts to direct traffic intended for SDRs through the external switch, and reciprocal routes were defined on the RPis to forward packets toward the virtual environment. These configurations allowed us to conduct comprehensive testing of the constellation's communication protocols, including monitoring latencies and verifying successful packet exchanges (ping) across the hardware and simulated network. The ping packets can be seen as the horizontal red lines in the time-frequency waterfall plot as well as the relative gain spike in the TX Spectrum flowgraph in Fig. 7. The average latency recorded for a bidirectional ping transmission between the Mininet hosts was around 270 ms. Additionally, a custom Netcat utility script was used to transfer User Datagram Protocol (UDP) packets from one host to the other over the external hardware network. An average throughput of around 2020 Kbps was noted over a duration of 10 seconds. This hybrid approach demonstrated the feasibility of integrating physical SDR nodes into a scalable software simulation framework for testing and validating large-scale satellite constellations. Continued expansion of this hardware-in-the-loop design and more robust use cases are a vital part of the future development of the testbed.

III. Methods

For each experimental use case, the following parameters in the configuration file were varied: source and destination ground station nodes, simulation date and time, and constellation and orbit specifications. Orlando, US was used as a source ground station and three GS-to-GS communication scenarios were considered based on the following destinations ground stations: Seattle US, Rio de Janeiro Brazil, and Delhi India. Each of those three cases was meticulously chosen such that the testbed's resulting optimal routes and network results reflect its capability to emulate intra-continental (Seattle), inter-continental (Rio and Delhi), cross-hemisphere (Delhi), and latitude-parallel (Rio) traffic flows. For

each of source and destination pairs, two times of the chosen simulation day (September 27, 2024) were analyzed: 12:30:00 UTC and 22:15:00 UTC. The chosen satellite constellation for the simulation was Starlink's 540km altitude operational shell (1584 satellites, 72 orbital planes, 53.2° inclination) as referred to in Section II.A.1 and listed in Table 1. This constellation was simulated as two different constellations in the testbed: a custom-made constellation with the orbital parameters and Walker Delta pattern matching Starlink's orbital parameters and the actual constellation whose positional data was pulled from *Celestrak*. The TLE datasets for both of these constellations were generated and acquired, respectively, before performing any experiments. Combining the three source and destination paths, the two times of day, and the two constellations, the number of use cases totals to twelve.

The configuration file was also used to specify a simulation time of 120 seconds with an individual time-step length of 10 seconds (totaling to twelve individual time-steps and simulation instances) for each of the above use cases. For the routing table generation, the routing metric chosen was the assigned latency of each ISL and GSL, so routes were determined based on the lowest total predefined latency between every node. For Phase 2-specific configurations, each use case was ran "optimized", so the only nodes set up by Mininet were those found to be relevant to each route between the two target nodes (as specified in Section II.B.1). Additionally, ping tests were selected to collect latency data for each of the use cases; the ping tests were also configured to run for the length of each time-step, but pausing in-between each to allow time for link changes during the emulator's "indeterminate" state, as mentioned in Section II.B.2. The resulting data was configured to output as an array of latency values, with each value corresponding to a successfully delivered data packet where one data packet was sent out every second.

IV. Results and Discussion

Presentation and discussion of the data collected has been split up by destination ground station. For each of the cases, the data is presented in tables displaying each use case's average round trip latency and standard deviation, and in scatter plots with four sets of data per figure (Fig. 8 is one such scatter plot). In addition to the raw data points, also included in the figures are solid lines plotting the average latency value at each time step and standard deviations setting the error bounds in the colored shaded regions. Since most of the outlying data points are extreme highs, the negative standard deviation sometimes imply too low of a possible latency. To fix that in the presentation of data, the minimum possible latency, which is found with the speed of light and the exact distance between the two ground stations, is included in the figure and used as a lower error bound in some cases of extreme outliers. Specific sections of interest in the scatter plots are further investigated using various maps showing the routes for each use case (e.g. Fig. 9). These route maps provide additional evidence to support the reasoning behind certain trends in the data.

A. Demo 1: Orlando to Seattle

	12:30pm	10:15pm
Actual TLE	47.917/34.182	71.318/11.210
Custom TLE	94.189/171.48	60.675/28.353

Table 2 Mean/standard deviation values of round trip latency (in ms) for Orlando to Seattle

Table 2 shows the average value and corresponding standard deviation of the round trip latency data for the four datasets of the Orlando to Seattle use case. The averages show that neither the actual or custom TLE datasets had significantly higher or lower latency values, which was expected, as the the custom constellation was meant to serve as an "ideal" form of the actual one. The results also show that the testbed results are dependent on the positioning of the satellites at specific times, as there are significant differences in the results at the two different times. However, the standard deviation for the first of the custom cases show that the average has been greatly affected by some extreme-high outliers. This can be seen in Fig. 8, which features the raw data points, average latency, and standard deviation error bounds for the first use case. The immediate takeaway from these results is that the custom TLEs lead to more consistent latency values compared to the actual TLEs. The reason for the frequent outliers for the custom TLEs and not for the actual TLEs may require more investigating, but some possible reasons are discussed.

Fig. 9 shows the first routes for each of the times and TLE sets in the Orlando to Seattle demo, but other than the visual difference in the satellite spacing, there is not a large difference between the two datasets. More interesting evidence can be found when looking at the routes one time step at a time. As seen in Fig. 8, in the 12:30pm custom TLE

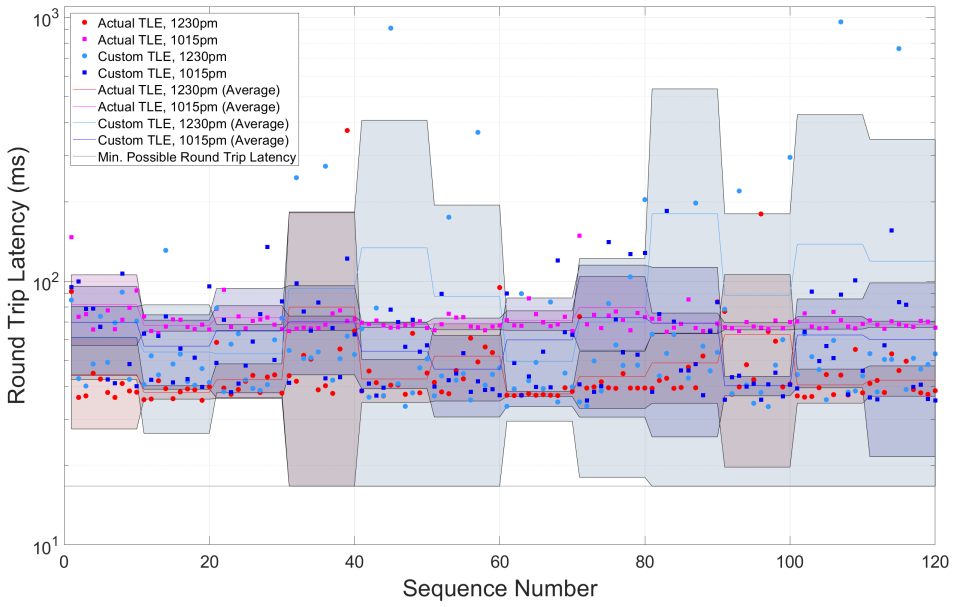


Fig. 8 Orlando to Seattle Ping Results

dataset, there is a jump in latency values from the 9th to 10th time-steps that remains high for the rest of the simulation length. The jump is primarily caused by some high outliers not characteristic of the behavior of the end to end link, as there are lower data points in the same time-step.

The two routes used during the 9th and 10th time-steps in the 12:30pm custom TLE dataset can be seen in Fig. 10. As can be seen in these routes, which ground-station-to-satellite link (GSL) is being used changes from the 9th to 10th time-steps for both the source and destination GS. Anytime a GS changes GSLs, the nodes that are being used in that route must change. This causes Mininet to change which links are being used to send data back and forth; if more nodes need to be changed from one route to another, then it takes Mininet longer to change all of the relevant links. Cases like this with many extreme outliers can be found throughout the datasets, and many of them line up with route changes like these. Additionally, since these cases are more likely to have extreme outliers when the custom TLEs are being used, the spacing of the satellites is likely a factor. With the custom TLEs, the satellites are neatly and equally spaced in their orbits, creating rows of satellites, with each row also equally spaced from each other. However, these spaces between rows are relatively large and that can cause short hops between nodes within these rows, but the hops outside of these rows can be quite large at times, creating some inconsistencies between time steps in both node counts and the distances between nodes. Both factors can lead to higher latency values. Additionally, the increased spacing within the custom TLE grid can cause significant routing changes to occur from one time step to the next. This causes Mininet to need to change every node and link between two time steps in that scenario, possibly causing the round trip latency result outliers.

For the scenario with actual TLEs, the unequal spacing of the satellites within and across orbits leads to more consistently short hops between satellites and leads to most routes having a similar number of nodes over time. These routes do tend to zig-zag over the map, as opposed to remaining as a straightforward, uniformly curved path (examples of this are explored later in the Rio and Delhi cases), but the actual TLEs still have a bit more consistency in their latency data points. Overall, when links need to be changed, the actual TLEs show more subtle changes over time, while the custom TLEs remain stagnant for longer but change more completely when necessary.

The stark difference of route from one time step to another for the custom TLE cases may cause issues with how Mininet handles the link changes and the network as a whole, which could be another possible explanation for why the custom TLE cases have an increased number of outliers and a larger standard deviation. Mininet has been found to cause data jumps in ping results due to leftover data artifacts. This can especially be prevalent in the types of cases being executed here, where the data collected is from a single trial and the testbed is being run on a general purpose operating system rather than a real-time system. Collecting data from multiple trials and running the testbed on a more

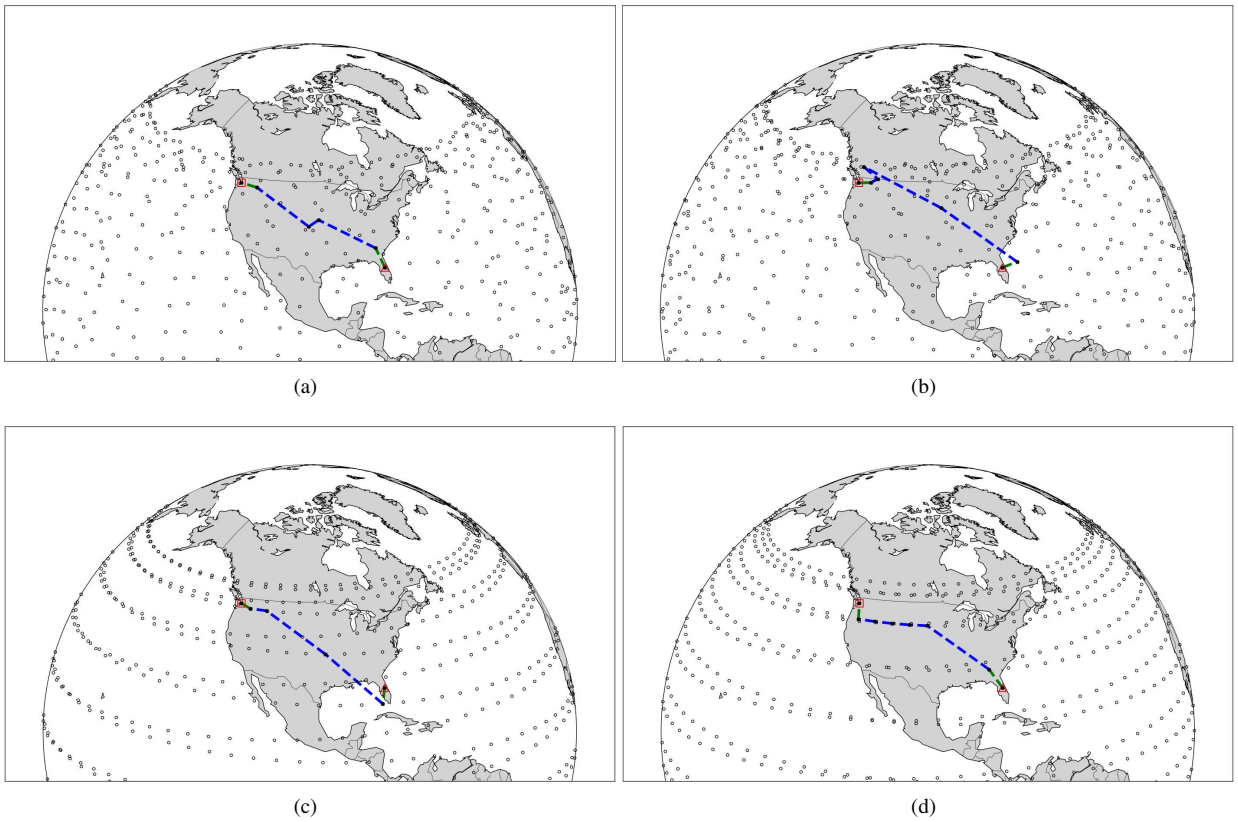


Fig. 9 First Time-Step for: (a) Orlando to Seattle, Actual TLE, 12:30pm (b) Orlando to Seattle, Actual TLE, 10:15pm (c) Orlando to Seattle, Custom TLE, 12:30pm (d) Orlando to Seattle, Custom TLE, 10:15pm

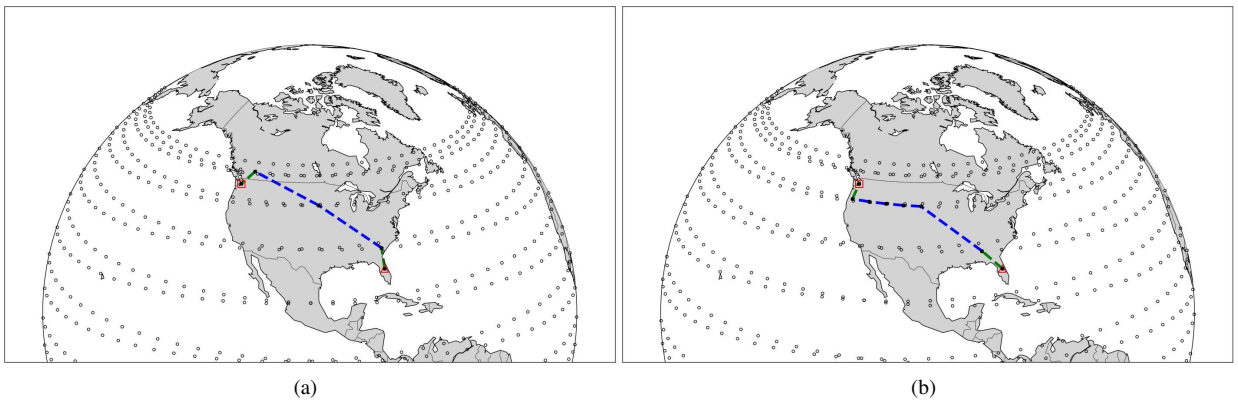


Fig. 10 (a) Ninth Time-Step for: Orlando to Seattle, Custom TLE, 12:30pm (b) Tenth Time-Step for: Orlando to Seattle, Custom TLE, 12:30pm

purpose-built, real-time system may lead to less inconsistencies. More investigation into how Mininet handles large networks is planned to confirm some of these ideas.

B. Demo 2: Orlando to Rio de Janeiro

The Orlando to Rio dataset does not have as many extreme outliers as the previous dataset (as seen in Fig. 11), but there are still some interesting trends to point out here, specifically how some drastic route changes can also occur with

	12:30pm	10:15pm
Actual TLE	72.078/21.686	86.733/44.878
Custom TLE	99.674/93.481	117.80/50.074

Table 3 Mean/standard deviation values of round trip latency (in ms) for Orlando to Rio

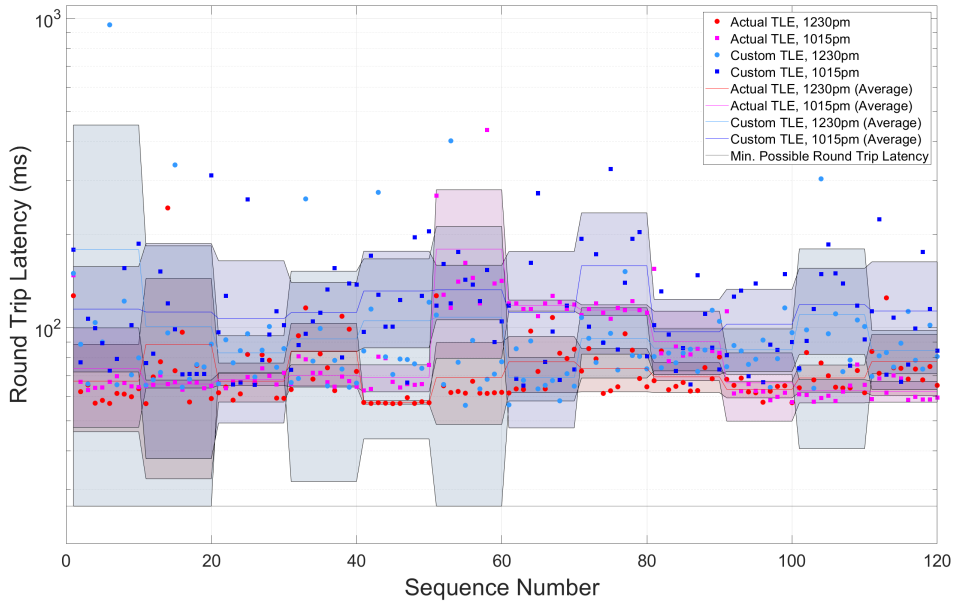


Fig. 11 Orlando to Rio Ping Results

the actual TLEs. For the most part, the routes in all four data sets in this case stay relatively similar to the routes seen in Fig. 12. However, the section at the sixth time-step in the 10:15pm actual TLE dataset does have a clear shift in its data (pink data points). This can be seen by the jump in raw data points, average latency, and the standard deviation error bounds. This is likely due to a routing change as seen in Fig. 13. This shows the route having more hops than the previous route and being significantly longer than the route seen in Fig. 12(b) with some zig-zag behavior over South America. The routing change holds for the next two time-steps before going back to a route similar to the one before. This behavior explains why the latency values are higher for these few time steps and why the overall average and standard deviation values for the 10:15pm actual TLE case are higher as well. This routing change is primarily due to the PlusGrid algorithm detailed in Sec. II.A.3. The algorithm creates a path like this due to satellite positions at this specific time; as seen in the 12:30pm case, this kind of path is not always generated when creating a route from Orlando to Rio. It is common for the constellations of actual TLEs to see more zig-zag path behaviors than their custom TLE alternatives. Alternate algorithmic approaches to link establishment as well as additional information regarding real-life satellite network routing determination should help to reduce the inconsistent path issues.

However, despite the longer route for part of the 10:15pm actual TLE dataset, the latency values for the custom TLEs were still found to be larger overall as seen in Table 3. However, the standard deviations in the same table show that the high outliers present in the custom TLE data still significantly impact the values. The routes seen in Figures 12(c) and 12(d) stay the same or close to the same for the simulation's full length, so changes in the custom TLE routing path shape is not the cause of the inconsistency. As stated with the Orlando to Seattle case, further investigations into the differences between the custom and actual TLEs as well as more details regarding how Mininet functions are required in determining the source of these outliers but could be due to changing GSL links or the increased spacing in custom TLE grids, among other reasons previously discussed.

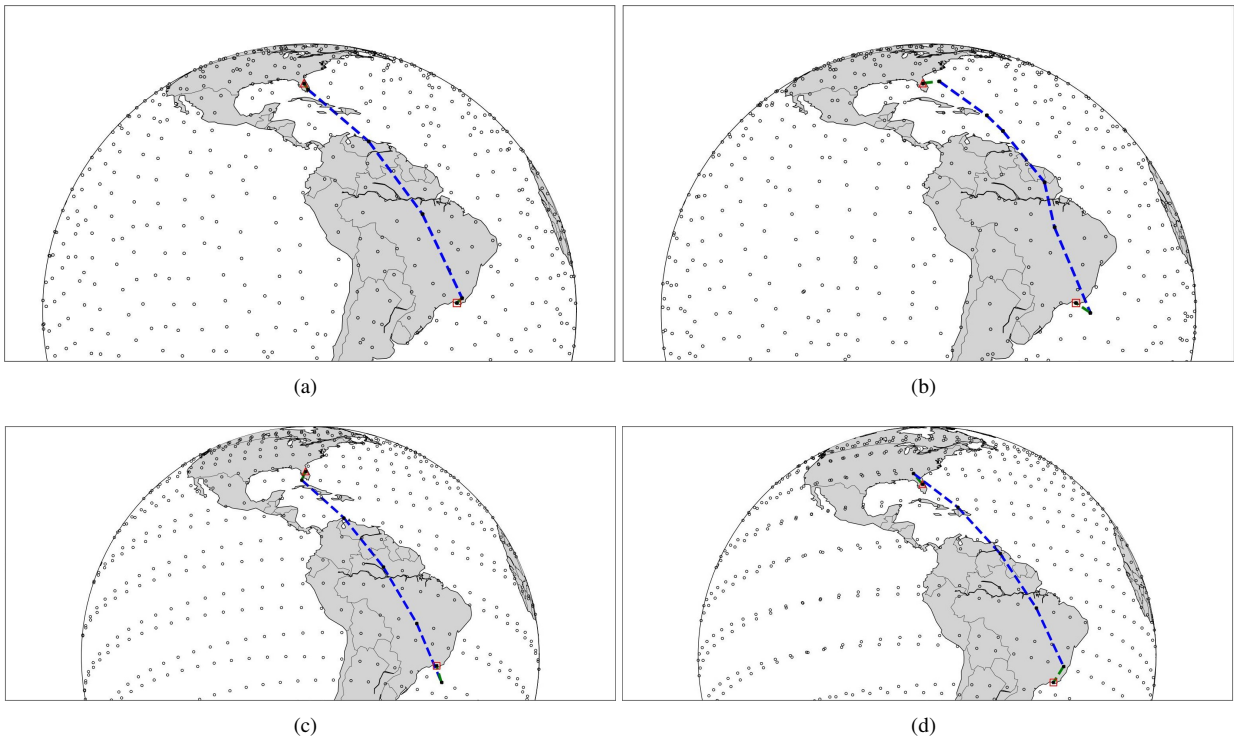


Fig. 12 First Time-Step for: (a) Orlando to Rio, Actual TLE, 12:30pm (b) Orlando to Rio, Actual TLE, 10:15pm (c) Orlando to Rio, Custom TLE, 12:30pm (d) Orlando to Rio, Custom TLE, 10:15pm

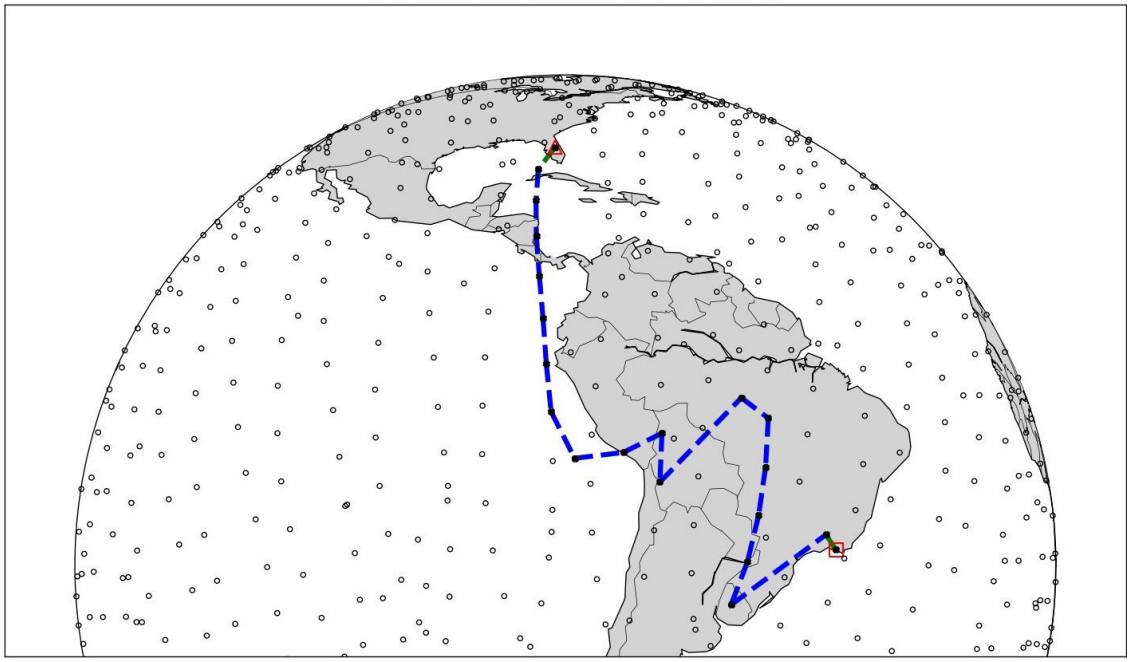


Fig. 13 Sixth Time-Step for: Orlando to Rio, Actual TLE, 10:15pm

	12:30pm	10:15pm
Actual TLE	196.84/66.705	152.41/58.331
Custom TLE	181.82/259.74	151.25/46.150

Table 4 Mean/standard deviation values of round trip latency (in ms) for Orlando to Delhi

C. Demo 3: Orlando to Delhi

The Orlando to Delhi datasets are fairly similar to the previous cases, showing similar trends. The two times of day and the two constellations have statistically similar results, with the custom TLE cases having an increased standard deviation than the actual TLE cases. As seen in Table 4 the Orlando to Delhi case has the closest averages between the actual and custom TLE datasets as compared to the Orlando to Seattle and Orlando to Rio cases previously discussed. There are still some variance between the two times of day, but the two 10:15pm datasets are the most similar cases when comparing the actual and custom TLEs. However, the 12:30pm custom TLE dataset does have an extremely large standard deviation.

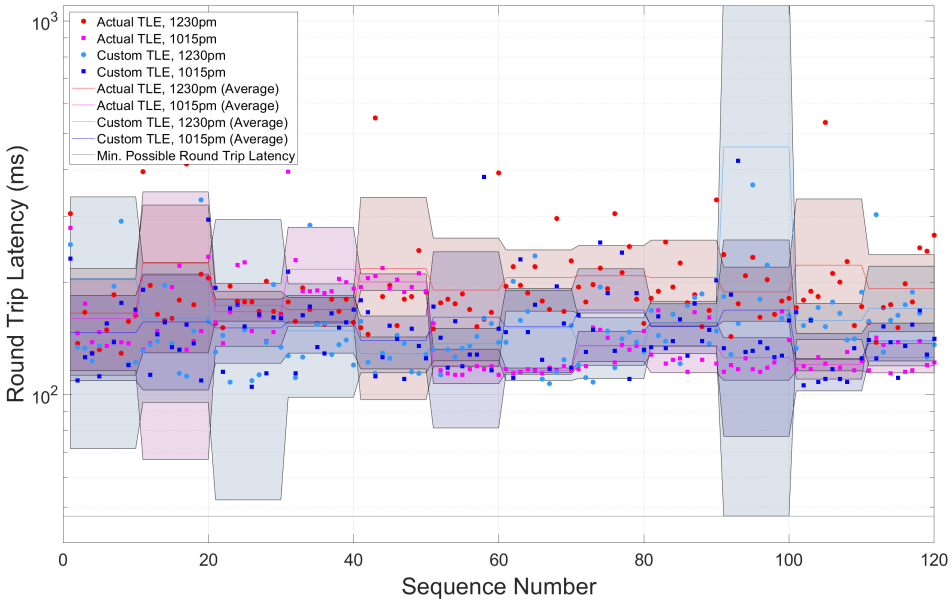
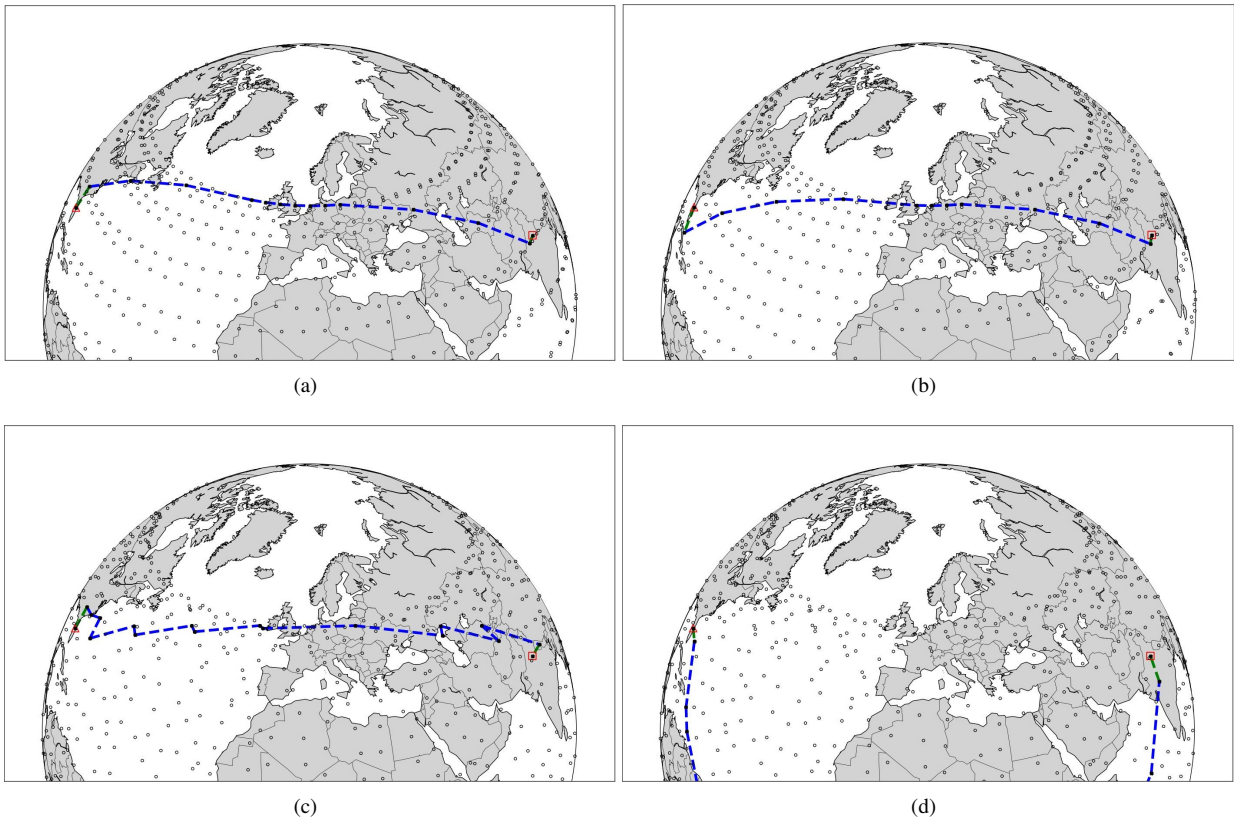


Fig. 14 Orlando to Delhi Ping Results

The 10th time step of this dataset (as seen in the third to last segment in Fig. 14) features the highest valued outlier, which significantly affects the standard deviation. The route used during this segment and the route of the segment before are displayed in Figures 15(a) and 15(b). There is a change in GSL for the source GS which can continue to explain some inconsistency, as long as the logic from the Orlando to Seattle case is still followed, but the routes still don't provide a clear explanation behind the magnitude of the latency outlier. The high spikes in the data may also be due to some of the previously mentioned complications related to Mininet and its handling of virtual networks. Again, more investigations into how Mininet handles these kinds of large networks in the context of a space testbed are planned.

While there may not have been as many standout sections of the Orlando to Delhi scatter plot, there were still some interesting network routes that can be discussed. Figures 15(c) and 15(d) show the routes for the third and fourth time-step of the 10:15pm, actual TLE dataset. The zig-zag pattern that the route forms in Fig. 15(c) is a semi-common occurrence with the actual TLEs. This is due to the more scattered distribution of the satellites compared to the custom TLEs. This zig-zagging is also more common at higher latitudes, as the satellites are more densely grouped in these regions compared to the satellites closer to the equator. Fig. 15(d) shows a dramatic change in the routing compared to the previous time-step. The route is partially cut off in the figure itself, but it still shows the network choosing a



(a) (b) (c) (d)

Fig. 15 (a) Ninth Time-Step for: Orlando to Delhi, Custom TLE, 12:30pm (b) Tenth Time-Step for: Orlando to Delhi, Custom TLE, 12:30pm (c) Third Time-Step for: Orlando to Delhi, Actual TLE, 10:15pm (d) Fourth Time-Step for: Orlando to Delhi, Actual TLE

satellite path that loops down toward the equator before coming back up to Delhi. This unusual route is likely due to a combination of the network changing GSLs at this time-step and the PlusGrid algorithm. PlusGrid's need to make these abnormal adjustments shows how the scattered distribution of these satellites can significantly affect these emulated networks. Improving the grid algorithm and learning more about how the real-life systems function continue to be important steps in accurately emulating networks using actual TLE datasets.

V. Conclusion and Future Work

SpaceNet has been presented as an adaptable testbed for simulating satellite constellation networks. The use of Mininet allows for the emulation of a space network with nodes in the thousands. The results of the initial use cases presented here illustrate that constellations composed of custom TLE have larger standard deviations in round trip latency time than constellations composed of actual TLE data. However, constellations composed of actual TLE data have some round trip latency outliers due to path zig-zag shapes which deviate from the standard smooth curved path seen in the custom TLE cases. A definite answer to the cause of the increased custom TLE outliers is not known, but it could be due to an increase in gaps between satellites, a Mininet usage error caused by an increased number of new satellites in a time step, or a product of artifacts in the computer system being used to run SpaceNet. The testbed has the capability to collect network performance data from the emulated space communication systems and will be used in the future to run additional use cases.

Additional experiments using existing pieces of the testbed can be performed to resolve some of the unanswered questions. In regards to the future developments on the software side of the testbed, a greater variety of experimental use cases is being explored. First, performing additional tests using the testbed's ability to collect bandwidth data would add another layer to the network performance analysis. Additional use cases include variations on the source

and destination target ground stations, experimenting with a wider variety of constellation patterns, and expanding network communication to other shells within an established constellation or to other constellations entirely. In regards to multi-shell capabilities, future efforts will also involve expanded inclusion of satellites in the real-life TLE datasets, rather than needing to exclude satellites that do not fit in broadly-defined orbital shells.

Current developments also include adding terrestrial-to-terrestrial node links to the emulated networks. Having terrestrial and non-terrestrial nodes work in tandem would allow the simulated networks to more closely resemble real-life global communication networks. Additionally, higher fidelity GSL modeling is in development; improvements to this portion of the testbed should allow for more robust testing regarding the role of user devices in the space network. The PlusGrid ISL connectivity can also be replaced with a more efficient and adaptive ISL grid where the decision of every neighbor ISL a satellite chooses for the connection would adapt based on parameters like current latitude or level of customer demand in various regions. Such a grid could be more resilient to route changes and would help to support the implementation of dynamic routing to the testbed's emulation phase. Dynamic routing would allow the testbed to use Mininet to make more realistic adjustments to the network without having to solely rely on the predefined networks generated in Phase 1 of the testbed. Additional experiments, including network resiliency tests, could also be implemented with the inclusion of dynamic routing.

On the hardware side of the testbed, future developments include adding additional hardware nodes to the HIL system, creating more complex routing scenarios, and making use of a programmable attenuator to replicate more realistic network conditions. Simply adding more RPi and SDR pairs would increase both the complexity and flexibility of the hardware network and its capabilities when communicating with the virtual network. Complex routing scenarios could be introduced through the use of multiple network switches; these switches would enable the emulation of dynamic network configurations, allowing exploration of the robustness and adaptability of communication protocols under varied conditions. This feature is essential for examining protocol resilience and network stability in the emulated constellation. Additionally, a programmable attenuator would allow the testbed to accurately replicate signal degradation effects typical of long-distance and interference-prone communication links. By adjusting signal strength, the setup could mimic scenarios affected by physical distance or interference, facilitating performance analysis in terms of key network metrics such as latency and bandwidth. Despite enhanced simulation capabilities, real-world constraints, including environmental variability, scalability challenges, and time delays, continue to influence efforts to refine and optimize the system.

Acknowledgments

Acknowledgement is made of NSF award #2235139 for sponsoring this research.

References

[1] Kassing, S., Bhattacherjee, D., Águas, A. B., Saethre, J. E., and Singla, A., "Exploring the "Internet from space" with Hypatia," *Proceedings of the ACM Internet Measurement Conference*, Association for Computing Machinery, New York, NY, USA, 2020, p. 214–229. https://doi.org/10.1145/3419394.3423635, URL https://doi.org/10.1145/3419394.3423635.

[2] Wang, N., Liu, L., Qin, Z., Liang, B., and Chen, D., "Capacity Analysis of LEO Mega-Constellation Networks," *IEEE Access*, Vol. 10, 2022, pp. 18420–18433. https://doi.org/10.1109/ACCESS.2022.3149961.

[3] Han, C., Xiong, W., and Yu, R., "A Hybrid Forecasting Model for Self-Similar Traffic in LEO Mega-Constellation Networks," *Aerospace*, Vol. 11, No. 3, 2024. https://doi.org/10.3390/aerospace11030191, URL https://www.mdpi.com/2226-4310/11/3/191.

[4] Singh, K., Nirmal, A., and Sharma, S., "LINK MARGIN FOR WIRELESS RADIO COMMUNICATION LINK," *ICTACT Journal on Communication Technology*, Vol. 8, 2017, p. 8. https://doi.org/10.21917/ijct.2017.0232.

[5] Shaengchart, Y., and Kraiwanit, T., "Starlink satellite project impact on the Internet provider service in emerging economies," *Research in Globalization*, Vol. 6, 2023, p. 100132. https://doi.org/https://doi.org/10.1016/j.resglo.2023.100132, URL https://www.sciencedirect.com/science/article/pii/S2590051X23000229.

[6] Renard, K., Peri, C., and Clarke, J., "A performance and scalability evaluation of the ns-3 distributed scheduler," 2011.

[7] Yang, Z., Li, H., Wu, Q., and Wu, J., "NPVT: Network Protocol Validation Testbed for Integrated Space-Terrestrial Network," *IEEE Access*, Vol. 7, 2019, pp. 46831–46845. https://doi.org/10.1109/ACCESS.2019.2906397.

[8] Pfandzelter, T., and Bermbach, D., "Celestial: Virtual Software System Testbeds for the LEO Edge," Association for Computing Machinery, New York, NY, USA, 2022, p. 69–81. https://doi.org/10.1145/3528535.3531517, URL https://doi.org/10.1145/3528535.3531517.

[9] Lai, Z., Li, H., Deng, Y., Wu, Q., Liu, J., Li, Y., Li, J., Liu, L., Liu, W., and Wu, J., "StarryNet: Empowering Researchers to Evaluate Futuristic Integrated Space and Terrestrial Networks," *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, USENIX Association, Boston, MA, 2023, pp. 1309–1324. URL https://www.usenix.org/conference/nsdi23/presentation/lai-zeqi.

[10] Barbour, B., Gibbons, R., Kenyon, S., McClure, J., Ridge, D., and Black, J., "Network Testbed for Small Satellites (NeTSat)-Distributed Space Adaptive Communications and Security for Multi-Constellation Networks," *AIAA SCITECH 2023 Forum*, 2023, p. 1502.

[11] de Oliveira, R. L. S., Schweitzer, C. M., Shinoda, A. A., and Prete, L. R., "Using Mininet for emulation and prototyping Software-Defined Networks," *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, 2014, pp. 1–6. https://doi.org/10.1109/ColComCon.2014.6860404.

[12] Walker, J. G., "Walker Satellite constellations," *Journal of the British Interplanetary Society*, Vol. 37, 1984, pp. 559–571.

[13] "CelesTrak," *CelesTrak*, Mar 2024.

[14] Anonymous, "SAT-MOD-20200417-00037," Tech. rep., FCC, 2020.

[15] North, M., "A Method for Implementing a Statistically Significant Number of Data Classes in the Jenks Algorithm," 2009, pp. 35–38. https://doi.org/10.1109/FSKD.2009.319.

[16] Bhattacherjee, D., and Singla, A., "Network topology design at 27,000 km/hour," 2019, pp. 341–354. https://doi.org/10.1145/3359989.3365407.

[17] OpenWeather, "Weather API," https://openweathermap.org/api, 2024.

[18] Hagberg, A., Swart, P., and S Chult, D., "Exploring network structure, dynamics, and function using NetworkX," Tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[19] Chumbley, A., Moore, K., Yang, J., Ross, E., and Khim, J., "Floyd-Warshall Algorithm," *Brilliant*, Accessed: 2024-07-05. URL https://brilliant.org/wiki/floyd-warshall-algorithm/.

[20] Lantz, B., and the Mininet Contributors, "Mininet v2.3.0," https://mininet.org/, 2021. [Online; accessed 2-December-2024].

[21] gRPC Authors, "gRPC," https://https://grpc.io/, 2024. [Online; accessed 2-June-2024].