DSNS: The Deep Space Network Simulator

Joshua Smailes
University of Oxford
joshua.smailes@cs.ox.ac.uk

Filip Futera
University of Oxford
filip.futera@cs.ox.ac.uk

Sebastian Köhler
University of Oxford
sebastian.kohler@cs.ox.ac.uk

Simon Birnbach
University of Oxford
simon.birnbach@cs.ox.ac.uk

Martin Strohmeier
armasuisse Science + Technology
martin.strohmeier@armasuisse.ch

Ivan Martinovic
University of Oxford
ivan.martinovic@cs.ox.ac.uk

***Abstract*—Simulation tools are commonly used in the development and testing of new protocols or new networks. However, as satellite networks start to grow to encompass thousands of nodes, and as companies and space agencies begin to realize the interplanetary internet, existing satellite and network simulation tools have become impractical for use in this context.**

We therefore present the Deep Space Network Simulator (DSNS): a new network simulator with a focus on large-scale satellite networks. We demonstrate its improved capabilities compared to existing offerings, showcase its flexibility and extensibility through an implementation of existing protocols and the DTN simulation reference scenarios recommended by CCSDS, and evaluate its scalability, showing that it exceeds existing tools while providing better fidelity.

DSNS provides concrete usefulness to both standards bodies and satellite operators, enabling fast iteration on protocol development and testing of parameters under highly realistic conditions. By removing roadblocks to research and innovation, we can accelerate the development of upcoming satellite networks and ensure that their communication is both fast and secure.

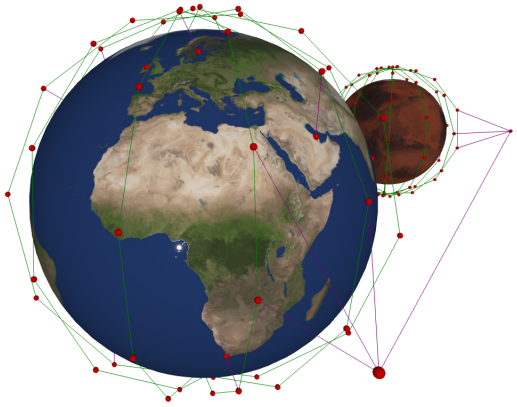


Fig. 1. An interplanetary network simulated in and visualized by DSNS, with communication between Earth and Mars constellations via a relay link.

I. Introduction

As space becomes a critical component of global infrastructure, there is an increasing interest in new paradigms of communication to support the scale and complexity of upcoming networks. Protocols are being developed to support the delay-tolerant nature of communication, and space agencies such as ESA and NASA are starting to introduce Lunar communication [1–3] and interplanetary networks [4] (cf. Figure 1 for a visualization of such an interplanetary network). The presence of sporadic long-distance relay links in these networks, alongside the inherent difficulty of communication across highly distributed internet-scale networks in space, means new protocols and approaches must be taken.

Standards bodies and research organizations like the Consultative Committee for Space Data Systems (CCSDS) and Internet Research Task Force (IRTF) have been working to build standards supporting communication in the face of these new network paradigms. For example, the Bundle Protocol (BP) provides message forwarding and delivery in networks with interrupted links [5], and the Licklider Transmission Protocol (LTP) enables reliable message transmission across individual network segments [6] – however, it is becoming increasingly difficult to test these protocols to ensure their correct and efficient operation in the large-scale, highly distributed networks predicted to emerge in the coming decades. Simulation tools are a crucial component of protocol development and testing, but existing offerings do not meet all the requirements for this purpose in these new networks. There is thus a real need for new tools capable of simulating protocols at the scale of a future interplanetary internet.

A. Requirements

We identify the following core requirements for any network simulation tool in order for it to be able to effectively aid protocol development, with a particular focus on large-scale networks and interplanetary networking:

R1: Orbital simulation. The simulator must be able to simulate the movement and connectivity of satellite constellations, including LEO megaconstellations.

R2: Interplanetary network simulation. The simulator must be able to handle nodes orbiting different planets, and connectivity between them.

R3: Dynamic connectivity. Links in the simulation must be configurable based on distance, line of sight, and/or occlusion, to ensure realistic connectivity with interruptions.

R4: Dynamic timesteps. In order to support long-distance interplanetary links, in addition to short-distance local links, the simulator must be able to skip forward by

TABLE I
SUMMARY OF THE DIFFERENT SATELLITE NETWORK SIMULATORS CURRENTLY AVAILABLE.

Name	Created by	Language	License	Released	Maintained	Summary
ONE Simulator [7]	TKK	Java	GPLv3	2007	✗	Lightweight DTN network simulator. Focused on small numbers of nodes with random movement.
NOS3 [8]	NASA	C	NOSA	2019	✓	Small satellite operational simulator. Simulates flight and ground software for single missions with high fidelity.
SpaceSecLab/ NSE2 [9]	ESA	—	Unreleased	—	—	Containerized satellite simulator with integrated network simulation. Realistic and configurable, possible integration with real hardware.
Hypatia [10]	ETH Zürich	C++/Python	GPLv2/MIT	2020	✗	Extension to ns-3, provides LEO constellation mobility for fixed ISLs. Now part of "SNS3" [11].
Celestial [12]	TU Berlin	Python	GPLv3	2022	✓	LEO system testbed based on micro-VMs, supporting software emulation for LEO networks.
StarryNet [13]	Tsinghua	Python	MIT	2023	✓	Simulator for integrated space and terrestrial networks, combining orbital simulation with Docker and network emulation.
$x$eoverse [14]	Surrey	Python	Non-commercial	2025	✓	LEO megaconstellation simulator based on Mininet [15]. Real-time simulation of large networks including dynamic ISLs.
Stardust [16]	TU Wien	C#	Apache 2.0	2025	✓	Scalable 3D network routing simulator, plugins to extend functionality. Fast, supports many nodes.
SatScope [17]	NUDT China	Python	Non-commercial	2025	—	LEO constellation network simulator based on VTK with a focus on satellite internet routing/coverage.
DSNS	Oxford	Python	GPLv3	2025	✓	Scalable network simulator supporting arbitrary interplanetary networks. High-level protocol simulation to support development.

TABLE II
COMPARISON OF THE FEATURES PROVIDED BY EACH OF THE SATELLITE/DTN NETWORK SIMULATORS.

Simulator	Orbital Simulation (**R1**)	Interplanetary (**R2**)	Dynamic connectivity (**R3**)	Dynamic Timesteps (**R4**)	Extensible (**R5**)	Scalable (**R6**)	Abstracted network stack (**R7**)
ONE Simulator [7]	○	○	●	●	●	◐	●
NOS3 [8]	○	○	◐	○	◐	○	○
SpaceSecLab/NSE2 [9]	◐	◐	◐	●	●	◐	○
Hypatia [10]	●	○	○	●	●	◐	◐
Celestial [12]	●	○	●	○	◐	◐	○
StarryNet [13]	●	○	●	○	◐	◐	◐
$x$eoverse [14]	●	○	◐	○	◐	◐	○
Stardust [16]	●	○	●	○	●	●	◐
SatScope [17]	●	○	●	○	◐	◐	○
DSNS	●	●	●	●	●	●	●

large timesteps when no activity is occurring, while also supporting fine-grained simulation of traffic over short-distance links.

R5: Extensible. It must be straightforward to extend the simulator to support new protocols, routing strategies, constellations, etc.

R6: Scalable. The simulator must be able to handle large numbers of nodes (at least hundreds, if not thousands or more), with large traffic volumes.

R7: Abstracted network stack. In service of **R5**, and to enable faster protocol development and testing, simulation tools benefit from supporting abstracted or reduced network stacks (potentially in addition to full-stack simulation or emulation).

B. Contributions

In this paper we present the Deep Space Network Simulator (DSNS): a new network simulator optimized for interplanetary networks that satisfies all the requirements established above. In contrast to existing offerings, explored in Section II, DSNS scales well to large numbers of nodes, supports arbitrary network topologies with interplanetary links, and uses an underlying event-based simulation to enable simulations to run faster than real time. Furthermore, DSNS is easily extensible thanks to its modular architecture – new protocols can be added, removed, and swapped out, and new layers of the network stack can be implemented by simply defining new sets of rules upon which messages and events are matched. To facilitate future research, DSNS has been made fully open source, released under the GNU GPLv3 license.[1]

II. BACKGROUND

Simulators are used in the development and evaluation of protocols in satellite and interplanetary networks, as they enable testing in networks much larger than otherwise possible, under a wide range of configurations, and without risking damage to real-world systems. Depending on the use case, simulations may be performed purely in software, paired with hardware simulation, or connected to real-world hardware.

Several satellite network simulators are already in use; in Table I we describe each at a high level, and in Table II we assess them against the requirements established in Section I-A.

[1]The source code and documentation can be found at https://github.com/ssloxford/DSNS.

We justify these assessments below. In addition to these, there are also some more generalized network simulation tools, such as ns-3 [18] and OMNeT++ [19]. However, these do not scale well to large numbers of nodes with many connections, so we do not consider them by themselves in this paper.[2]

The "ONE Simulator" [7] is a lightweight network simulator designed with DTNs in mind, primarily supporting scenarios with randomly moving nodes connecting based on proximity, with a well-defined extension framework. However, it does not natively support orbital simulation or satellite movement, and scalability is limited, with tested configurations limited to approximately 1000 nodes.

More recently, NASA have released their "NOS3" simulator, designed to be a satellite digital twin for developing and testing onboard software [8]. Although highly suited to this purpose, it is not feasible to run it at scale to test network protocols between many nodes. ESA are also planning to release their "Network Simulation and Emulation Environment (NSE2)" as part of their "SpaceSecLab" [9], providing a Docker-based network simulation that supports highly realistic emulation of the network stack and connectivity for small numbers of nodes. This is useful for testing implementation-specific details and small-scale mission control, but is not as practical for large satellite networks.

There has also been recent academic interest in satellite network simulators, with a range of newly released simulators since 2020 – summarized in Tables I and II. Notably, the vast majority of these simulators either use fixed timesteps or tie the simulation to a real-world clock. This makes the simulation of interplanetary networks particularly difficult, since they include short-distance local links alongside very long interplanetary links – to support both of these, a very small timestep will need to be chosen, increasing the simulation runtime to impractical levels. Event-driven simulators including "Hypatia", "ONE Simulator" and "NSE2" do not do this, but still struggle when large amounts of events are generated in a short timespan, triggering unnecessary position updates. This can be improved upon by only recomputing connectivity graphs once a minimum time delta has passed.

Also notable is that many of the simulators make use of virtualization or emulation to simulate real network stacks – this can be useful when protocols are being tested at the implementation level, but is not necessary for the vast majority of research and development tasks, slowing down development by requiring a full implementation of the protocol even at the earliest stages of testing. Furthermore, interplanetary networks often involve new protocols across the whole stack which do not always have mature implementations, so it may not even be possible to achieve a full network stack, especially as many components will differ from the traditional IP stack. This is of particular concern to simulators like "$x$eoverse" and "Celestial", as their underlying virtualization platforms make use of the host network stack – this is unlikely to match the tested networks. Similarly, "Stardust" cannot be used for traffic simulation or emulation, since it focuses instead on deploying computation tasks for edge computing cases.

[2]There is also an extension to ns-3 called SNS3 [11] which provides satellite spot beam simulation and an implementation of the DVB-S2 protocol; however, it does not solve its scalability issues, so it is only suitable for simulating smaller networks.

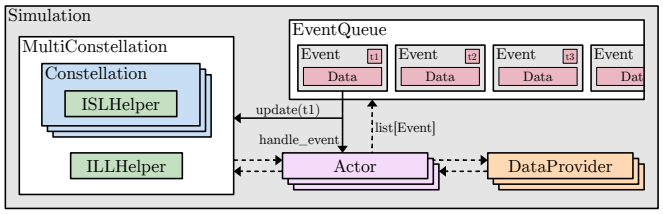


Fig. 2. Overall structure of DSNS and its high-level operation.

Finally, we highlight that none of the existing simulators support interplanetary networks and many of them only have limited scalability. Both of these features are critical to ensure communication protocols and management systems can handle the high latencies and frequent interruptions involved in interplanetary settings, and adapt to the predicted scale of these networks in coming decades.

Although these simulators are highly capable for the tasks they were designed for, none provide all the features required for effective interplanetary network simulation. Some struggle to scale to large numbers of nodes, some do not simulate mobility, and others do not support dynamic links or multi-planet systems, severely limiting the range of network configurations that can be tested. The simulators that support some of these features often require simulation of the entire network stack, even when a much smaller number of layers is sufficient. Others do not support simulation of protocols that deviate too far from the IP stack – which are those that we are most interested in testing and optimizing.

In contrast, the Deep Space Network Simulator supports arbitrary mobility and connectivity models, including fixed networks, LEO constellations, and interplanetary networking. The links between nodes can be defined from a fixed list of edges or programmatically, based on distance, line of sight, angle of elevation, or planetary occlusion. Its event-based architecture with optional minimum timesteps enables large spans of time to be fast-forwarded when messages travel a long distance alongside simulating large numbers of messages across shorter links within the same simulation scenario. It is easily extensible through a simple Python interface, scalable to many thousands of nodes (demonstrated in Section V-B), and simulates an abstracted network stack to improve efficiency and simplify development and iteration, while also supporting full-stack implementation of protocols if needed.

III. Simulator Design

In this section we describe the design of the Deep Space Network Simulator, demonstrating its underlying functionality, usage, and the ways in which it fulfills each of the requirements established in Section I-A.

A. Architecture

Figure 2 shows the overall architecture of DSNS. The simulation is highly modular, underpinned by a simple event-based simulation: components can be switched out to alter behavior, or entirely new components can be added for additional functionality. Mobility and connectivity are managed by the `MultiConstellation` class and its components, and all other functionality is handled by an event-based simulation and associated `Actors` and `DataProviders`. Everything in the simulation, including message creation/routing/delivery, link changes, routing table updates, and scenario-specific changes, are stored in a priority queue as `Events` with attached timestamps. These events are processed by `Actors`, which implement all complex functionality by matching and processing events – for example, a single instance of the `MessageRoutingActor` handles message routing and delivery for all nodes in the simulation.

At each step of the simulation, the topmost event is removed from the queue and passed to all actors, which use pattern matching to decide whether to process the event, and optionally add further events to the queue. They may also query data providers to gain additional information (e.g., routing tables), or the mobility model to get distances between nodes and the state of links. This process repeats until the simulation terminates. The modular implementation of complex functionality and protocols makes it easy to add new features without requiring a deep understanding of the rest of the simulation.

This architecture is highly efficient, as only the required components of the stack are simulated and simulation time can be advanced by large steps if needed. The latter is particularly useful in interplanetary settings as these often have long periods of little to no activity while a relay link is unavailable. To further improve efficiency for high-traffic scenarios, we also provide an optional minimum time delta: with this setting, mobility and routing models will not be updated more than once within this timespan, reducing the amount of unnecessary updates, thus satisfying **R4**. The actor model also makes DSNS highly extensible (**R5**), as we demonstrate through our implementation of LTP in Section III-D – we also explore further extension opportunities in Section VI.

B. Mobility

Mobility and connectivity are handled by the `MultiConstellation` class, which simulates any number of constellations, the Inter-Satellite Links (ISLs) within constellations, and the Inter-Layer Links (ILLs) between constellations. Constellations can be created from fixed points, Walker constellation parameters, or by importing Two Line Element (TLE) sets for full orbital simulation using the SGP4 propagator [20]. Each of these inherits from the `Constellation` class, defining the positions of satellites in the constellation or segment over time. Each constellation also has an `OrbitalCenter`, defined as the parent around which satellites orbit, enabling complex movement and simulation of satellites around many different planets or other bodies, without loss of precision when communicating over short distances. This satisfies **R1** and **R2**.

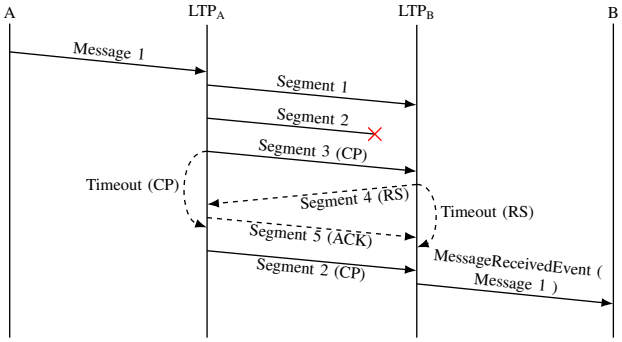


Fig. 3. Sequence diagram for LTP message retransmission. Segment 2 is lost in-transit and retransmitted to guarantee the message reaches its destination.

Inter-satellite links can be fixed (for ground systems or Walker constellations) or dynamic (connecting to satellites within view), by using one of the pre-built `ISLHelper` classes or defining the links manually. The `ILLHelper` works almost identically, but defines instead the links between different constellations or planets. Each time the simulation time is updated, positions of satellites and connectivity of links are updated to reflect the new state – satisfying **R3**.

C. Message Delivery

Message routing and delivery in DSNS is modeled realistically by simulating propagation along each link in the network, for both point-to-point and broadcast messages. This is handled for all nodes in the simulation by a `MessageRoutingActor` that, when a message is received, figures out the next hop in its path and forwards it on. Propagation delays are modeled using speed-of-light distances between nodes, and the `LinkTransmissionActor` manages link bandwidth and transmission delays, queueing messages if the link is busy or buffering them if the link goes down. All parameters including routing strategy, bandwidth, and error rate can be customized on a global or per-link basis.

Routing is handled by a `RoutingDataProvider`, which builds a connectivity graph for the network and computes optimal next-hop routing. We provide routing systems for best-effort and store-and-forward delivery – the latter looks ahead to future states of the network, enabling messages to be stored until the link becomes available. We discuss in Section VII how future work can build upon this system to provide realistic implementations of current and proposed routing protocols.

D. Protocol Design

It is easy to extend DSNS to support new protocols (**R5**); we demonstrate this by implementing the Licklider Transmission Protocol (LTP) for reliable per-hop message transmission, demonstrated in Figure 3 [6, 21]. All functionality is contained within the `LTPActor` – when enabled, this actor breaks the underlying message into green (unreliable) followed by red

(reliable) `LTPDataSegments` and queues them for transmission. This segmentation follows a configurable maximum segment size. The last red data segment (if any) is marked as a checkpoint. This aligns with the mandatory checkpoint requirements of the protocol, while support for optional discretionary checkpoints is left to future work.

Each segment is received through a `LTPSegmentReceivedEvent` and buffered until message reassembly is invoked. This in turn occurs when all expected red `LTPDataSegments` are received, or in the case of an all-green message, an end-of-block green `LTPDataSegment` is received. At that point, the underlying message is reassembled and a `MessageReceivedEvent` is emitted. Although standard LTP uses block offsets and lengths to compute missing byte ranges, our implementation abstracts this by having the checkpoint segment explicitly list the UIDs of sent `LTPDataSegments`, simplifying the receiver logic. Upon receiving a checkpoint segment, the receiver responds with a `LTPReportSegment` listing the UIDs of the `LTPDataSegments` it received. When the sender receives a `LTPReportSegment`, it sends a `LTPReportAcknowledgementSegment` and retransmits any missing `LTPDataSegments`.

Both the checkpoint and report segments have configurable timeouts, which trigger retransmissions if the corresponding responses are not received within the allotted time. The actor can also be configured with a maximum number of retransmission attempts for each checkpoint or report segment. If this limit is exceeded, the session is canceled and the message is dropped at the sender (`MessageDroppedEvent`) and/or has its reception aborted at the receiver (`MessageReceptionCanceledEvent`).

E. Visualization

DSNS also contains a visualization tool using the "pyrender" library, enabling a 3D view of any interplanetary network to provide a better understanding of how satellite network topologies evolve over time, and assist in the construction of new constellations or simulations. This tool was used to create the visualization of an Earth-Mars network in Figure 1.

IV. Reference Scenarios

In this section we describe and implement a number of reference scenarios to demonstrate the capabilities and performance of DSNS, which can be used as a starting point for protocol development and optimization. These scenarios define a network topology, bandwidth limitations, traffic models, and an error model, each of which can be mixed and matched, modified, or extended.

Each scenario is constructed from a simple Python script, examples of which are given in Appendix A. Upon completion, they produce a log file detailing all events generated during the simulation (or aggregate statistics), from which further information can be extracted. Metrics include message latency and hop count, bandwidth usage, and link saturation. These can be used to assess the performance of different protocols or configurations, or to see how the characteristics of a given network topology change over time.

Finally, we also demonstrate the performance and scalability of DSNS itself (satisfying **R6**), by running simulations with large numbers of nodes and high levels of traffic, and comparing to performance figures quoted for other simulators.

A. CCSDS Reference Scenarios

We start by implementing the network topologies specified in the DTN reference scenarios proposed by CCSDS [22].[3] In these scenarios, the nodes, links, data rates, and traffic types are all well-defined, enabling consistent implementation for testing and development purposes. The document specifies three scenarios:

- **Earth Observation**: A payload control center and mission control center are each connected to two ground stations, which connect to an Earth observation satellite.
- **Lunar Communication**: A lunar base, rover, two relay satellites and a lunar gateway communicate with control centers and ground stations based on Earth.
- **Mars Communication**: Two rovers and three relay satellites are based on and around Mars, communicating with Earth-based control centers and ground stations.

B. Custom Scenarios

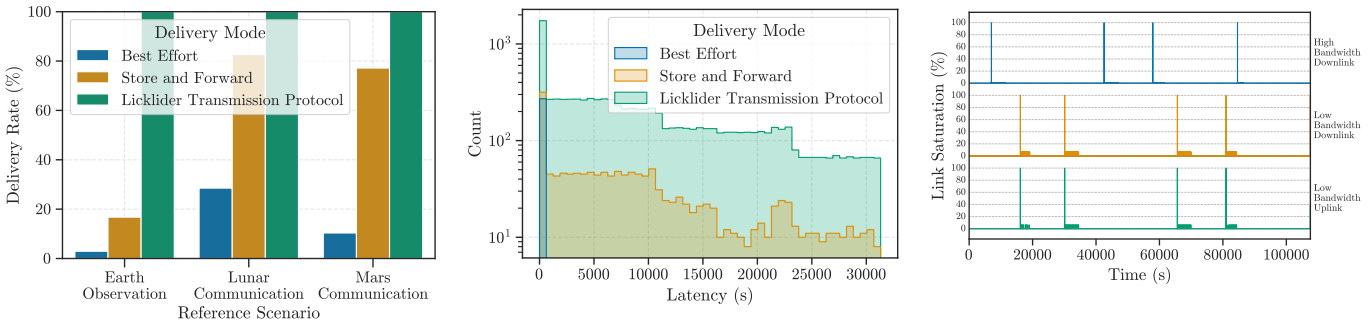
Although useful for testing protocols against long- and short-distance communication, the above reference scenarios are very small in scale. We therefore provide our own custom scenarios alongside these, focusing in particular on the scalability of DSNS applied to interplanetary networks – the number of devices in each scenario can be scaled to suit the use case. We implement the following scenarios:

- **Walker Constellation**: A Walker constellation in LEO around Earth is connected to 12 ground stations and within itself via ISLs in a plus-grid topology. By default we use 66 satellites matching the Iridium constellation.
- **CubeSat Constellation**: Using data from CelesTrak [23] and the SGP4 orbital propagator [20], we construct a federated network composed of the 98 CubeSats currently in orbit, connected to 12 ground stations, and to each other via opportunistic ISLs with a range of $2500\,\mathrm{km}$.[4]
- **Lunar-Mars Communication**: The "Walker Constellation" scenario is augmented with Lunar connectivity, with a Walker constellation of 8 satellites connected to 12 Lunar ground stations, and a single relay satellite connecting to three ground systems on Earth. A network around Mars is also added, with 66 satellites and 12 ground stations, and its own relay satellite.

For these scenarios, we assume a uniform data rate of $25\,\mathrm{Mbit/s}$ by default, matching the reported bandwidth of Iridium's ISLs [24] – as with the CCSDS scenarios, this

[3]These scenarios are currently in draft form; our implementations in DSNS will be updated to reflect the final document.

[4]We use data provided on 2025-06-27 and propagated from 2025-06-27T00:00:00Z. Up-to-date TLEs can be substituted in if needed.



(a) Message delivery rate under each of the three message delivery systems.

(b) Distribution of the latency of delivered messages, under the Earth Observation scenario, for each message delivery system.

(c) Saturation of the three ground-to-space links over time, under the Earth Observation scenario with store-and-forward delivery.

Fig. 4. A selection of results from the reference scenarios, demonstrating delivery rates, message latency, and link saturation over time.

can be configured on a global or per-link basis to match the parameters of a particular mission.

To ensure the scenarios are as versatile as possible, the traffic generation strategy can also be configured. We provide the following options:

- **Point-to-point Communication:** Pairs of nodes send data between each other at a constant rate for the full duration of the simulation. The specific pairs of nodes, rate of communication, and size of generated messages are all configurable. By default, 10 pairs of nodes send a 1 MB block of data every second.
- **Random Communication**: Nodes randomly generate messages to send between each other. The message rate, source, destination, and size are all drawn from user-configurable distributions – by default, we support constant, uniform, Gaussian, and Pareto distributions.

C. Message Delivery

When running any of the above reference scenarios, we must also choose the message delivery system which the simulation uses to route and deliver messages. We have implemented three message delivery systems:

- **Best-effort delivery:** Following the IP paradigm, messages are routed according to instantaneous route availability, and are dropped if no route is available.
- **Store-and-forward delivery:** Using a BP-style strategy, the routing system looks forward to the state of the network in the future to ensure messages arrive at their destination, storing messages if needed while waiting for a link to become available.
- **Store-and-forward with LTP:** The same store-and-forward strategy is used, but with Licklider Transmission Protocol implemented at the data link layer to ensure reliable data transport and realistic transmission queues.

Each of these can be swapped out with a single option. We focus on demonstrating store-and-forward (with and without LTP) in our results, as best-effort delivery is not well-suited to networking in space, where links are often unavailable.

D. Error Model

Finally, we provide an error model with a configurable message loss rate. By default, this is set to 5% across all links, but this can be adjusted globally or on a per-link basis. These message errors usually result in failed delivery, unless reliable data transport is used – in our results, we demonstrate how the introduction of LTP prevents message loss.

V. Evaluation

In this section we run a number of reference scenarios to demonstrate the capability of DSNS in a wide range of contexts, and showcase its performance and scalability to large-scale constellations.

A. Reference Scenarios

By running simulations using the reference scenarios outlined above, we can demonstrate the effectiveness of DSNS in a wide range of contexts. For this section we focus on the CCSDS reference scenarios; full results are in Appendix B.

1) Delivery Rate and Latency: We look first at the delivery rate of messages under each message delivery system; these results are summarized for each of the CCSDS scenarios in Figure 4a. We can see that, as expected, delivery rate is much lower under best-effort delivery, since any messages that are generated when a link is not immediately available are discarded. Delivery rates under store-and-forward delivery are better, but messages are still lost due to the 5% error rate. However, using LTP successfully reduces the number of failed deliveries to 0%, since any lost data is retransmitted.

We gain further insights looking at the distribution of the latency of delivered messages, shown in Figure 4b. Latency under best-effort delivery is very low, since the only delivered messages are those for which no buffering is required. LTP results in higher latencies than store-and-forward, since more messages are delivered instead of dropped, and additional overhead is incurred by retransmissions and acknowledgments.

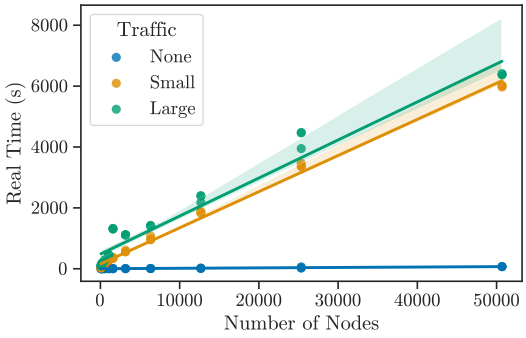


Fig. 5. Time taken to run the Walker constellation scenario in DSNS as the number of nodes in the constellation increases.

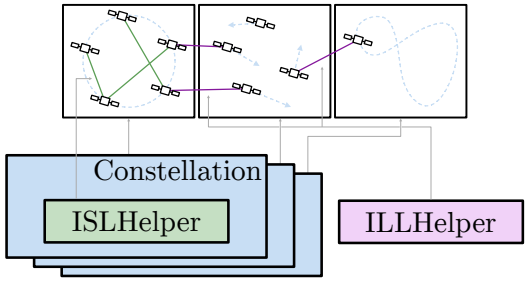


Fig. 6. DSNS can be extended to support new mobility and connectivity models by building new instances of the `Constellation`, `ISLHelper`, and `ILLHelper` classes.

2) Link Saturation: Next, we look at how the saturation of links changes over time, shown in Figure 4c for the Earth Observation scenario under store-and-forward delivery, looking in particular at the ground-to-space links. As expected, whenever a link comes online, we see a spike in saturation as all the buffered messages are sent, followed by the newly generated messages, resulting in a more consistent amount of low saturation. In large-scale networks, this analysis is useful for locating the most saturated links in a network and tracking down its root cause.

B. Performance

Finally, we use these scenarios to demonstrate the performance and scalability of DSNS (**R6**), and compare to other network simulators. This analysis can be repeated for any network topology and traffic configuration; for the sake of simplicity we choose to focus on the Walker constellation with point-to-point communication, running for 100 minutes of simulation time. In order to get the best understanding of the overhead imposed by DSNS itself, we configure the simulator to use best-effort delivery with no lookahead in routing – the topology of the Walker constellation does not change significantly, so it is not necessary to use lookahead.

We start with a simulation of 66 nodes, gradually increasing the number of nodes to 50 688. For each constellation, we run simulations with three different traffic configurations: no communication between nodes, low traffic communication (10 MB every 11.5 seconds, matching the EOS scenario), and high traffic communication (100×10 MB every 11.5 seconds). All experiments were executed on a single CPU core, using an Intel Xeon E5-2660 processor running at 3.3 GHz. No simulation used more than 2.1 GB of RAM – further details are in Appendix B.

The time taken to run each simulation is summarized in Figure 5. We see that even on the larger traffic configuration, on networks whose scale far exceeds any existing satellite network, the simulation still runs faster than real time. On smaller networks, or with smaller volumes of traffic, simulations are even faster, with many running tens or hundreds of times faster than real time. This exceeds the performance of even the best-performing simulators in related work: "Stardust" takes just over 6000 seconds to simulate 20 600 satellites [16], and does not perform full traffic simulation, and "StarryNet" takes over 3000 seconds to simulate 1000 satellites under the same conditions [16]. "Hypatia" has a slowdown rate of between 2 and 50 000 on a network of just over 1000 satellites [10] depending on traffic volume, compared to DSNS' slowdown rate of less than 1 under all tested configurations.

Furthermore, since DSNS is single-threaded, many simulations can be executed at once on a single machine with no impact on performance for each simulation. This is highly useful in practice, enabling multiple tests under different network topologies or protocol configurations to be executed at the same time and compared afterwards.

VI. Extensibility of DSNS

One major use case for DSNS lies in its extensibility (**R5**) – each of the reference scenarios can be modified, extended, or rewritten to support the needs of a particular experimental goal. By using the same reference scenarios and metrics across all experiments, we can be sure that new protocols or network topologies provide concrete improvements, rather than merely performing better as an artifact of the experimental setup.

Due to its modular design, adding new functionality to DSNS is a highly straightforward process. We demonstrate this by outlining how a user might add support for a new network layer above or below those already simulated, a new routing strategy, and new constellations within the mobility and connectivity model.

Since the behavior of each component is defined by Python functions, rather than a domain-specific language, customization options are almost unlimited: as long as the desired functionality can be expressed in code, it can be supported in DSNS. This enables a choice of different levels of abstraction based on need: for the majority of the examples expressed in this paper we have focused on abstracted implementations of protocols, allowing us to test functionality with minimal computational impact. However, we have also seen through our implementation of LTP that protocols can be implemented with greater fidelity. If required, this can even be taken down to the bit level for a given protocol – and importantly, this can

be done without requiring the rest of the stack to be simulated with such high precision, unless specifically required.

A. Mobility and Connectivity

Firstly, we consider extensions to the mobility and connectivity model used by DSNS, illustrated in Figure 6. These are the easiest parts of the simulator to modify, as they are unaffected by the rest of the simulation. To add a new type of orbit, a user simply extends the `Constellation` class to provide the desired functionality, defining `update_positions` to correctly set their positions at each timestep. These can be fixed, follow a given curve or orbital propagation model, move randomly, or follow any other pattern that can be expressed in code. Similarly, `ISLHelper` can be extended to provide new connectivity models, with `get_isls` defining how satellites in a constellation are connected at any given point in time – by modifying this function, any configuration of links can be supported, from fixed topologies to more esoteric options. Finally, `ILLHelper` (and corresponding function `get_ills`) defines the connections between layers and planetary segments, and can be modified in the same way to enable arbitrary inter-layer and inter-segment connectivity.

Once these have been defined, the simulator handles all low-level functionality, solving for routes and generating link up/down events as links become available and unavailable.

B. Routing Strategies

Implementing a new routing strategy in DSNS is straightforward at its core, requiring an extension to the `RoutingDataProvider` class with the new strategy. However, this obscures some underlying complexity – the existing routing strategies rely upon global knowledge of the network state, and do not require message passing between nodes.

To implement a routing protocol that involves an exchange of information between nodes, additional functionality must be integrated into the data provider. This could be build on top of the existing message delivery system, using manually routed messages to pass information between adjacent nodes.

The message delivery system can also be extended to support new functionality; for instance, if the data provider returns more than one next hop, it could be configured to deliver messages along multiple paths to increase the chance of successful delivery. This flexibility enables the implementation of a wide range of routing protocols and strategies.

C. Higher Layer Protocols

It is easy to add new protocols at higher levels by extending the `BaseMessage` to add additional fields, encapsulating the original message as a field within the new message. However, developers need to take care that layers are appropriately ordered by ensuring each layer correctly pattern matches on messages from the layer above it, and that a `MessageCreatedEvent` is only generated for the lowest layer event. For more complex simulations with multiple layers, a layer management actor might be used to make sure layers are correctly ordered and handle events between them.

D. Physical Layer

Alongside higher layer functionality, DSNS also supports implementing additional features at lower layers. This might involve adding additional parameters such as message size, priority, or physical layer modeling for more realistic message loss. To modify the message structure, `BaseMessage` can be extended once again to add the new fields. Following this, the `MessageRoutingActor` can be extended to modify physical layer delivery mechanisms – for example, the `handle_message_sent_event` function can be modified to support bandwidth limiting or message loss modeling.

E. Security

Finally, we discuss the ways in which DSNS can be used to enable security research in this area. In the source code for the simulator we provide mechanisms for simulating the following attacks:

- **Global message loss:** We provide configurable randomized message loss across the whole network through the `LossConfig` class, enabling the testing of reliable delivery and rerouting mechanisms under realistic loss conditions.
- **Targeted message loss:** The `LossConfig` also supports per-link loss rates, enabling simulation of more targeted attacks and ensuring redelivery and rerouting still works under direct attack.
- **Link flooding attacks:** Finally, we provide tools to simulate targeted flooding attacks, denying service by exhausting the available bandwidth on a link. This can be accessed via the `TrafficFloodActor`, and can be configured to start and end at a specific time, target a single link or a group of links, and use different amounts of bandwidth.

Furthermore, the simplicity and extensibility of DSNS makes it easy to build further attack mechanisms, making it particularly useful to assess the behavior of different protocols and strategies under a wide range of attacks.

VII. Discussion

In previous sections we have demonstrated that DSNS is efficient, scalable, supports any required network topology, and can be easily extended to support new protocols. The combination of these features makes it immensely useful for future research, development, and testing. Standards bodies like the CCSDS can use the simulator to run their DTN communication reference scenarios, with fast iteration on parameters for upcoming protocols and recommended standards. They will also be able to run simulations at a much larger scale than these small examples, using configurations built upon the custom scenarios explored in this paper to test those same protocols on significantly larger networks without sacrificing performance. In doing so, we can ensure standards are well-informed by simulations which match real-world configurations.

Alongside these standards bodies, satellite developers and operators will also benefit through making data-driven decisions prior to deployment. Alongside optimizing their planned

network topology, they will also be able to test protocols to ensure their configuration provides optimal performance.

Although the performance of DSNS is already sufficient for the vast majority of cases, there is always scope for further improvements in this area. For example, the impact of message routing could be reduced by precomputing routes on paths that are known to be fixed, and components of the mobility and connectivity model could be written in a more performant language such as Rust or C++. Thanks to the modularity of DSNS combined with its thorough documentation, it will be possible to replace components with higher-performance variants without impacting usage or results.

Future work might also consider implementing the JSON/CSV specifications used by the CCSDS reference scenarios, enabling them to be implemented and tweaked directly using the same format as the ESA simulation platform. By running the same simulations on multiple different platforms, we can gain the best of both worlds – the SpaceSecLab platform provides better support for full-stack network simulation, whereas DSNS supports larger constellations and faster iteration on protocol design, so both can be used together for effective testing across the board. However, implementing this specification is not trivial, as DSNS makes use of full orbital simulation to calculate rendezvous times, whereas the CCSDS specification uses a list of fixed connection times. Parity between the two can be ensured by either using DSNS to generate the rendezvous times for the CCSDS scenarios, or by modifying the simulation configuration for the reference scenarios to instead take a fixed list of connection times.

VIII. Conclusion

In this paper we have presented the Deep Space Network Simulator (DSNS), and demonstrated how it can be used to enhance the research and development necessary for upcoming LEO megaconstellations and interplanetary networks. Through its modular architecture, event-based simulation, and a simple and flexible interface, DSNS offers improved scalability, configurability, and extensibility compared to existing tools. We have demonstrated its capabilities through the implementation of existing protocols and reference scenarios, and shown that its performance exceeds existing state-of-the-art tools.

As satellite networks continue to expand in scale and capability, DSNS is well-positioned to enable future innovation in this critical area, bringing us closer to the ultimate goal of a unified interplanetary internet.

Acknowledgments

We would like to thank armasuisse Science + Technology for their support during this work. Joshua was supported by the Engineering and Physical Sciences Research Council (EPSRC). Sebastian and Simon were supported by the Royal Academy of Engineering and the Office of the Chief Science Adviser for National Security under the UK Intelligence Community Postdoctoral Research Fellowships programme.

References

[1] David J Israel, Kendall D Mauldin, Christopher J Roberts, Jason W Mitchell, Antti A Pulkkinen, D Cooper La Vida, Michael A Johnson, Steven D Christe, and Cheryl J Gramling. "LunaNet: A Flexible and Extensible Lunar Exploration Communications and Navigation Infrastructure". In: 2020 IEEE Aerospace Conference. IEEE, 2020, pp. 1–14. ISBN: 1-72812-734-3.

[2] NASA. *LunaNet Interoperability Specification Document*. 2022. URL: https://www3.nasa.gov/sites/default/files/atoms/files/lunanet_interoperability_specification_version_4.pdf (visited on 06/11/2025).

[3] European Space Agency. *Moonlight*. 2024. URL: https://www.esa.int/Applications/Connectivity_and_Secure_Communications/Moonlight (visited on 06/11/2025).

[4] NASA Science. *The Mars Relay Network Connects Us to NASA's Martian Explorers*. 2021. URL: https://mars.nasa.gov/news/8861/the-mars-relay-network-connects-us-to-nasas-martian-explorers (visited on 06/11/2025).

[5] Scott Burleigh, Kevin Fall, and Edward J. Birrane. *Bundle Protocol Version 7*. Request for Comments RFC 9171. Internet Engineering Task Force, Jan. 2022. 53 pp. DOI: 10.17487/RFC9171. URL: https://datatracker.ietf.org/doc/rfc9171 (visited on 06/11/2025).

[6] Scott Burleigh, Stephen Farrell, and Manikantan Ramadas. *Licklider Transmission Protocol – Specification*. Request for Comments RFC 5326. Internet Engineering Task Force, 2008. 54 pp. DOI: 10.17487/RFC5326. URL: https://datatracker.ietf.org/doc/rfc5326 (visited on 06/30/2025).

[7] Ari Keränen, Jörg Ott, and Teemu Kärkkäinen. "The ONE Simulator for DTN Protocol Evaluation". In: Proceedings of the 2nd International Conference on Simulation Tools and Techniques. 2009, pp. 1–10.

[8] NASA Technology Transfer Platform. *NASA Operational Simulator for Small Satellites (NOS^3)*. 2025. URL: https://software.nasa.gov/software/GSC-17737-1 (visited on 06/11/2025).

[9] Daniel Fischer, Mariella Spada, and David Koisser. "SpaceSecLab: A Modular Environment for Prototyping Space-Link Security Protocols". In: *14th International Conference on Space Operations*. 2016, p. 2391.

[10] Simon Kassing, Debopam Bhattacherjee, André Baptista Águas, Jens Eirik Saethre, and Ankit Singla. "Exploring the "Internet from Space" with Hypatia". In: Proceedings of the ACM Internet Measurement Conference. 2020, pp. 214–229.

[11] Jani Puttonen, Budiarto Herman, Sami Rantanen, Frans Laakso, and Janne Kurjenniemi. "Satellite Network Simulator 3". In: *Workshop on Simulation for European Space Programmes (SESP)*. Vol. 24. 2015, p. 26.

[12] Tobias Pfandzelter and David Bermbach. "Celestial: Virtual Software System Testbeds for the LEO Edge". In: *Proceedings of the 23rd ACM/IFIP International Middleware Conference*. 2022, pp. 69–81.

[13] Zeqi Lai, Hewu Li, Yangtao Deng, Qian Wu, Jun Liu, Yuanjie Li, Jihao Li, Lixin Liu, Weisen Liu, and Jianping Wu. "StarryNet: Empowering Researchers to Evaluate Futuristic Integrated Space and Terrestrial Networks". In: *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 2023, pp. 1309–1324. ISBN: 1-939133-33-5.

[14] Mohamed M. Kassem and Nishanth Sastry. "xeoverse: A Real-time Simulation Platform for Large LEO Satellite Mega-Constellations". In: *arXiv preprint arXiv:2406.11366* (2024). arXiv: 2406.11366.

[15] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. "Reproducible Network Experiments Using Container-Based Emulation". In: *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*. 2012, pp. 253–264.

[16] Thomas Pusztai, Jan Hisberger, Cynthia Marcelino, and Stefan Nastic. "Stardust: A Scalable and Extensible Simulator for the 3D Continuum". In: *arXiv preprint arXiv:2506.01513* (2025). arXiv: 2506.01513.

[17] Qichen Wang, Guozheng Yang, Yongyu Liang, Chiyu Chen, Qingsong Zhao, and Sugai Chen. "SatScope: A Data-Driven Simulator for Low-Earth-Orbit Satellite Internet". In: *Future Internet* 17.7 (2025), p. 278. ISSN: 1999-5903.

[18] George F Riley and Thomas R Henderson. "The Ns-3 Network Simulator". In: *Modeling and Tools for Network Simulation*. Springer, 2010, pp. 15–34.

[19] Andras Varga. "OMNeT++". In: *Modeling and Tools for Network Simulation*. Springer, 2010, pp. 35–59.

[20] David Vallado and Paul Crawford. "SGP4 Orbit Determination". In: *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*. 2008, p. 6770.

[21] Scott Burleigh, Stephen Farrell, and Manikantan Ramadas. *Licklider Transmission Protocol – Security Extensions*. Request for Comments RFC 5327. Internet Engineering Task Force, Sept. 2008. 11 pp. DOI: 10.17487/RFC5327. URL: https://datatracker.ietf.org/doc/rfc5327 (visited on 06/30/2025).

[22] CCSDS. *Reference Scenarios*. 2023. 29 pp.

[23] T.S. Kelso. *CelesTrak*. 1985. URL: https://celestrak.org/ (visited on 06/27/2025).

[24] Jun Yang. *The Inter-Satellite Link: Theory and Technology*. Springer Nature, 2025.

APPENDIX A
EXAMPLE CODE

In this appendix, we provide code snippets illustrating the ease with which scenarios can be constructed using DSNS, the reference scenarios described in Section IV as a base.

The following builds and runs the Earth Observation scenario:

```
constellation = EarthObservationMultiConstellation()

transmission_actor = \
EarthObservationTransmissionActor(
    constellation=constellation
)

traffic_actor = EarthObservationTrafficActor(
    constellation=constellation,
    update_interval=300,
)

routing_data_provider = LookaheadRoutingDataProvider(
    resolution=60,
    num_steps=600,
)
routing_actor = MessageRoutingActor(
    routing_data_provider,
    store_and_forward=True,
    model_bandwidth=True,
    #loss_config=LossConfig(
    #    seed=0,
    #    default_loss_probability=0.05
    #),
    #reliable_transfer_config=LTPConfig(),
)

simulation = Simulation(
    constellation,
    actors=[
        transmission_actor,
        traffic_actor,
        routing_actor,
    ],
    data_providers=[routing_data_provider],
    timestep=0.01,
)

simulation.initialize(time=0)
simulation.run(3600*24, progress=False)
```

To modify this scenario to use LTP, we simply uncomment the `LTPConfig()`, alongside the following LTP actor:

```
retransmission_config = RetransmissionConfig()
ltp_actor = LTPActor(
    config=retransmission_config,
    model_bandwidth=True,
)
```

Similarly, message loss can be added by uncommenting the `LossConfig()`.

We provide the helper scripts `ccsds_reference.py` and `custom_reference.py` to build and run all of the scenarios described in this paper and collect logs, with a range of options to customize the simulation. These can be used as they are, or as a starting point for modification.

APPENDIX B
EXTENDED RESULTS

In this appendix we expand upon the simulation results in Section V, providing additional statistics and insights.

A. Reference Scenarios

In addition to the CCSDS reference scenarios, we also ran simulations under the custom reference scenarios provided in this paper.[5] Results for all the scenarios are summarized in Table III,[6] and a selection of the results are given in Figure 7. We can see that when LTP is not used, the delivery rates are lower, even with the lowered error rate – this is due to the fact that messages must traverse many more links to reach their destination, causing the error rate to compound over each consecutive link. However, the inclusion of LTP counteracts this and ensures almost all messages are delivered successfully. We also see that delivery rates for best-effort and store-and-forward delivery are closer to one another, particularly in the Walker constellation scenario, since connectivity is largely fixed except for ground-to-space links, so the majority of losses are due to the error model.

B. Performance

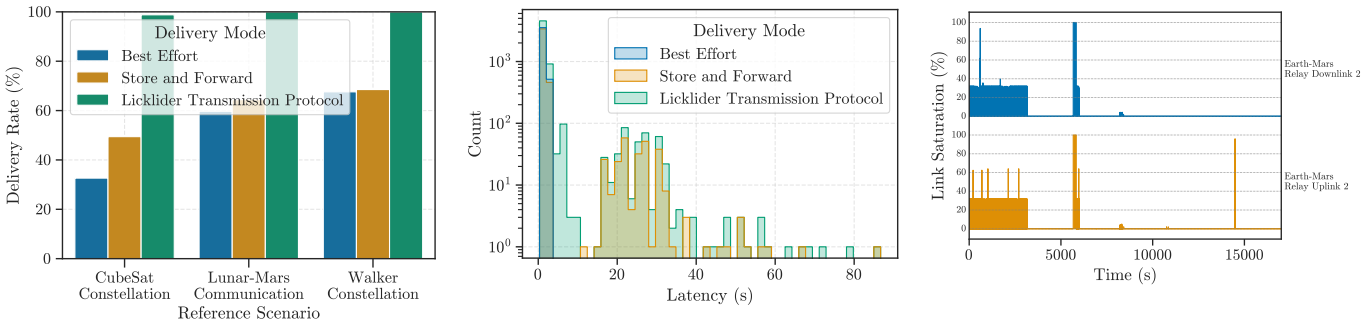
Figure 8 shows the memory used by DSNS for each of the Walker simulations. We can see that it increases linearly with the number of nodes in the simulation, due to the additional data associated with each node (position, routing tables, etc.) and increases slightly with the volume of traffic in the simulation. It is possible that memory usage could be reduced through the use of profilers and efficiency improvements, but it is already low enough for the vast majority of use cases – even on the largest simulations, usage did not exceed 2.1 GB.

[5] We adjust the loss rate for these scenarios to 0.5%.

[6] Any messages that are not delivered and not dropped (e.g., due to not being able to find a route for the message) are lost due to the scenario's error model.

TABLE III
AGGREGATED RESULTS FOR EACH OF THE TESTED REFERENCE SCENARIOS.

Configuration								
Scenario	Delivery Mode	Loss (%)	Delivered (%)	Dropped (%)	Mean Latency (s)	Mean Hops	Mean Link Utilization (%)	Max Link Utilization (%)
Earth Observation	Best Effort	5.0	2.93	83.35	0.10	2.00	1.48	6.40
Earth Observation	Store and Forward	5.0	16.76	0.00	8986.45	2.00	7.79	100.00
Earth Observation	Licklider Transmission Protocol	5.0	100.00	0.00	9704.86	2.00	9.29	100.00
Lunar Communication	Best Effort	5.0	28.51	60.03	1.31	3.35	0.99	17.07
Lunar Communication	Store and Forward	5.0	82.62	0.00	4707.89	3.70	3.22	100.00
Lunar Communication	Licklider Transmission Protocol	5.0	100.00	0.00	4733.25	3.71	1.83	100.00
Mars Communication	Best Effort	5.0	10.34	86.89	1259.66	4.65	1.42	51.20
Mars Communication	Store and Forward	5.0	77.16	0.00	14 732.93	5.05	13.06	100.00
Mars Communication	Licklider Transmission Protocol	5.0	100.00	0.20	14 517.87	4.97	9.65	100.00
Walker Constellation	Best Effort	0.5	67.58	0.35	2.01	4.73	28.47	67.59
Walker Constellation	Store and Forward	0.5	68.52	0.00	4.67	4.73	29.20	100.00
Walker Constellation	Licklider Transmission Protocol	0.5	100.00	0.00	3.42	4.96	15.25	100.00
CubeSat Constellation	Best Effort	0.5	32.63	42.13	2.64	5.70	28.23	100.00
CubeSat Constellation	Store and Forward	0.5	49.43	1.20	53.38	7.36	35.04	100.00
CubeSat Constellation	Licklider Transmission Protocol	0.5	98.80	1.20	78.25	10.02	21.03	100.00
Lunar-Mars Communication	Best Effort	0.5	59.72	8.65	102.52	5.12	27.64	67.59
Lunar-Mars Communication	Store and Forward	0.5	64.40	0.00	239.36	5.44	29.14	100.00
Lunar-Mars Communication	Licklider Transmission Protocol	0.5	100.00	0.00	421.20	6.04	16.17	100.00



(a) Message delivery rate under each of the three "custom" message delivery systems.

(b) Distribution of the latency of delivered messages, under the Walker Constellation scenario, for each message delivery system.

(c) Saturation of the Earth-Mars links over time, under the Lunar-Mars Communication scenario with LTP delivery.

Fig. 7. A selection of results from the reference scenarios, demonstrating delivery rates, message latency, and link saturation over time.

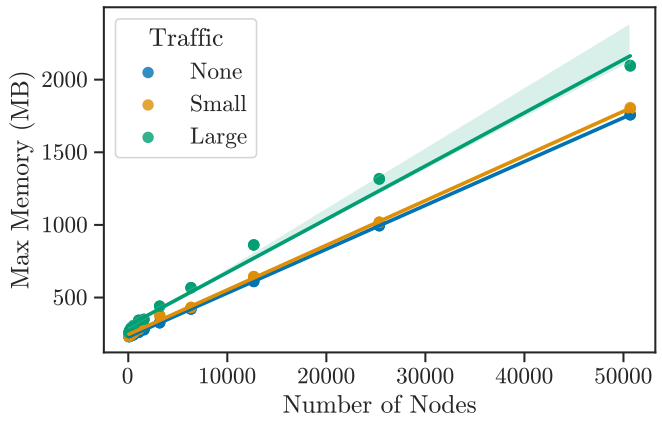


Fig. 8. Maximum memory used by DSNS while running the Walker constellation scenario as the number of nodes in the constellation increases.