OpenSN: An Open Source Library for Emulating LEO Satellite Networks

Wenhao Lu[1], Zhiyuan Wang[1,2,3], Shan Zhang[1,2,4], Qingkai Meng[1], Hongbin Luo[1,2,4]
[1]School of Computer Science and Engineering, Beihang University
[2]Zhongguancun Laboratory, Beijing, China
[3]State Key Laboratory of Virtual Reality Technology and Systems
[4]State Key Laboratory of Software Development Environment

ABSTRACT

Low-earth-orbit (LEO) satellite constellations (e.g., Starlink) are becoming the necessary component of future Internet. There have been increasing studies on LEO satellite networking. It is a crucial problem how to evaluate these studies in a systematic and reproducible manner. In this paper, we present OpenSN, i.e., an open-source library for emulating large-scale satellite network (SN). Different from Mininet-based SN emulators (e.g., LeoEM), OpenSN adopts container-based virtualization, thus allows for running distributed routing software on each node, and can achieve horizontal scalability via flexible multi-machine extension. Compared to other container-based SN emulators (e.g., StarryNet), OpenSN streamlines the interaction with Docker command line interface and significantly reduces unnecessary operation of creating virtual links. These modifications improve emulation efficiency and vertical scalability on a single machine. Furthermore, OpenSN separates user-defined configuration from container network management via a key-value database that records the necessary information of SN emulation. Such a separation architecture enhances the function extensibility. To sum up, OpenSN exhibits advantages in efficiency, scalability, and extensibility, thus is a valuable open-source library that empowers research on satellite networking. Experimental results show that OpenSN can construct mage-constellations 5X-10X faster than StarryNet, and update link state 2X-4X faster than Mininet.

CCS CONCEPTS

• **Networks → Network protocol design**.

KEYWORDS

LEO constellation, Satellite network emulator, Open-source library

This work was supported by National Natural Science Foundation of China under Grants 62202021, 62225201, and 62271019, in part by the National Key Research and Development Program of China under Grant 2022YFB4501000. *Zhiyuan Wang is the corresponding author.*

APNET'24, August 3-4, 2024, Sydney, Australia

ACM ISBN 979-8-4007-1758-1/24/08
https://doi.org/10.1145/3663408.3663430

ACM Reference Format:
Wenhao Lu, Zhiyuan Wang, Shan Zhang, Qingkai Meng, Hongbin Luo. 2024. OpenSN: An Open Source Library for Emulating LEO Satellite Networks. In *The Eighth Asia-Pacific Workshop on Networking (APNET'24), August 3-4, 2024, Sydney, Australia.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3663408.3663430

1 INTRODUCTION

1.1 Background & Motivation

In recent years, we have witnessed the rapid development of low-earth-orbit (LEO) constellations. Different from the geostationary satellites orbiting at the altitude 35768 km, the orbital altitude of LEO satellites is around 300-2000 km. The lower orbital altitude leads to a smaller latency for the ground-satellite links (GSLs), but also reduces the coverage area of a single satellite. Consequently, it is indispensable to deploy hundreds or thousands of LEO satellites to achieve global coverage. Many commercial enterprises have launched their mega-constellation plans. For example, Starlink Shell I consists of a total of 1584 satellites on 72 orbital planes. At the early stages, LEO satellites are only used as repeaters between two ground stations (GSes), which is also known as the bent-pipe architecture. Recently, many LEO constellations have been equipped with (or are deploying) inter-satellite links (ISLs). For example, Iridium Next has been using ISLs in L-band [13]. Starlink has launched satellites with laser-based ISLs way back in 2021 [30]. The adoption of ISLs creates the satellite network (SN) in space.

Although SN provides opportunities for extending Internet services to where the terrestrial networks cannot reach, it also leads to new challenges on satellite networking. Due to the mobility nature and the time-varying communication links, the topology characteristics of LEO satellite networks are quite different from those of terrestrial Internet. There have been increasing studies on LEO satellite networking, focusing on the low-latency advantages (e.g., [1, 11, 12]), topology design (e.g., [2, 20, 36]), intra-domain routing (e.g., [18, 21, 29, 34, 35]), and inter-domain routing (e.g., [7, 10, 15]).

Despite the wide investigation, how to evaluate these new architectures, protocols, and algorithms in a systematic and reproducible manner has been an open problem. Different from terrestrial Internet, it is costly to carry out experiments in a real-world SN. Even for those giants like SpaceX and Amazon, it also takes a few years to deploy their LEO constellations. Therefore, most existing studies on SN still rely on simulation or emulation. The evaluation platforms for SN could be classified into three categories:

(1) Orbit Analysis: The initial step of simulating SN is to obtain the trajectory data of LEO satellites. STK [32] is an orbit analysis tool, but does not support network simulation.

(2) Discrete-Event Simulation: The discrete-event simulators of SN will construct the constellation topology based on the trajectory data obtained from STK, and then simulate the networking events at different levels. Flow-level SN simulators (e.g., StarPerf [17]) cannot capture the fine-grained packet forwarding behaviors. Packet-level SN simulators (e.g., [5, 14, 33]) usually rely on a third-party discrete-event simulation platforms (e.g., ns3 [25] and OMNeT++ [26]).

(3) Virtual-Network Emulation: The virtual-network emulators of SN could run real protocol stack in Linux kernel and support unmodified applications on host OS. Existing virtual-network SN emulators either rely on Mininet [23] or Docker container [6]. For example, LeoEM [4] is developed upon Mininet. StarryNet [16] and the emulator in [27] adopt the container-based virtualization.

Among these solutions, virtual-network emulation provides the most realistic and meaningful results. However, existing virtual-network SN emulators either are not efficient enough to emulate mega-constellations or have limitation on system scalability and function extensibility. To address these drawbacks, this paper present OpenSN, i.e., an open source library for emulating large-scale satellite networks. We believe that OpenSN could empower the research on LEO satellite networking in the future. OpenSN will be available at `https://opensn-library.github.io`.

1.2 Main Results and Key Contributions

This paper presents OpenSN, i.e., an open-source library for emulating LEO satellite networks. OpenSN is a virtual-network emulator, running real kernel and unmodified applications. Compared to other SN emulators, OpenSN achieves a better emulation efficiency, system scalability, and function extensibility. Next we introduce the key contributions, and leave the detailed comparison in Table 2.

Container-based Emulation: OpenSN adopts container-based virtualization for emulating SN with two considerations. First, the container-based emulation achieves a good horizontal scalability via flexible multi-machine extension. Second, it allows for running distributed software (i.e., routing software, observation applications) in large-scale constellation without manual configuration. The two aspects above enable OpenSN to outperform LeoEM (adopting lightweight virtualization via Mininet) in terms of horizontal scalability and function extensibility, respectively.

Separation Architecture: OpenSN mainly consists of User-defined Configurator, Container Network Manager, and Key-Value Database. The key-value database records the necessary information for SN emulation, which acts as a data forwarder between User-defined Configurator and Container Network Manager. Compared to other container-based SN emulators (e.g., StarryNet), such a separation architecture enhances the function extensibility of OpenSN. For example, OpenSN allows users to dynamically change the emulation parameters and customize the emulation rules (e.g., GSL handover policy and ISL failure model).

Efficiency Improvement: By investigating and comparing existing container-based SN emulators, we find that the emulation efficiency of the container network is mainly limited to the interaction with the Docker command line interface (CLI). To this end, OpenSN streamlines the interaction process with Docker. First, OpenSN's Container Runtime Manager replaces the CLI interaction with official SDK, and also adopts a distributed architecture to achieve better parallelism. Second, OpenSN's Virtual Link Manager skips Docker Network Manager, and directly controls Linux virtual devices, reducing the overhead of unnecessary operation. This way, OpenSN is capable of achieving almost the same emulation efficiency as the specialized network emulator Mininet.

The rest of this paper is as follows. We review existing SN emulators in Section 2. We then introduce OpenSN architecture in Section 3. We evaluate the performance of OpenSN in Section 4. Finally, we conclude this paper in Section 5.

2 LITERATURE REVIEW

Table 1 compares SN simulation/emulation platforms.

2.1 Discrete-Event Simulation for SN

The discrete-event simulation for SN captures the networking events at different levels:

Flow-level: The flow-level SN simulators could calculate some performance metrics given the constellation topology, but are not able to capture the fine-grained behaviours of each packet. Lai *et al.* in [17] present StarPerf, and evaluate the end-to-end latency under different LEO constellations.

Packet-Level: Existing packet-level SN simulators rely on the third-party discrete-event simulation platforms, e.g., ns3 [25] and OMNeT++ [26]. Kassing *et al.* in [14] propose Hypatia, i.e., a packet-level SN simulator based on ns3. The authors use Hypatia to investigate the impact of latency and link utilization on congestion control and routing. Cheng *et al.* in [5] present a packet-level simulator for space-air-ground integrated network based on ns3, which supports various mobility traces of space and aerial networks. Both the two SN simulators above focus on the classic TCP/IP architecture in ns3. Yan *et al.* in [33] develop NDN architecture on OMNet++, and conduct a comparative study of host-centric and information-centric routing for SN.

The aforementioned packet-level SN simulators have two drawbacks. First, it takes long time to complete a simulation due to the complexity of scheduling discrete events. Second, they are not able to run real protocol stacks. Hence some studies explore the virtual-network emulation for SN.

2.2 Virtual-Network Emulation for SN

The virtual-network emulators for SN either rely on Mininet (adopting lightweight virtualization) or adopt the container-based virtualization, thus enable us to run real applications over SN with real-time response.

Mininet-based: Mininet [23] enables users to create a realistic virtual network, running real kernel and applications code, but does not provide specific support for SN emulation. Furthermore, Mininet adopts lightweight virtualization (i.e., process-based), thus is not suitable to emulate large-scale SN running distributed software (e.g., routing software). Cao *et al.* in [4] develop LeoEM (available at [19]), which emulates SN via Mininet. Moreover, LeoEM

Table 1: Network simulators/emulators

Category	Tools	Granularity	Technology	Networking	Open-source
Discrete-event simulator	StarPerf [17]	Flow-level	STK [32] & Python	Route calculation	✓
	Hypatia [14]	Packet-level	ns3 [25]	Routing protocol simulation	×
	Yan *et al.* in [33]		OMNeT++ [26]		×
	Cheng *et al.* in [5]		ns3 [25] & Matlab		×
Virtual-network emulator	Mininet [23]	Real stacks	Virtualization (Namespace)	NA	Not for SN
	Etalon [24]		Container (Docker)		
	LeoEM [4]		Mininet	Route calculation	✓
	StarryNet [16]		Container (Docker)	Routing software installed on containers	Partially
	Pan *et al.* in [27]		Container (Docker)		×
	OpenSN		Container (*Better Manager*)		✓

Table 2: Performance comparison of SN emulators

SN Emulator	Emulation Efficiency	System Scalability	Function Extensibility
LeoEM	Efficient for SN emulation thanks to the lightweight virtualization of Mininet	Good vertical scalability on single machine, not horizontally scalable to more machines	Not able to emulate SN running distributed software (e.g., routing) due to the lightweight virtualization
StarryNet	Inefficient due to direct interaction with Docker CLI in SN emulation	Good horizontal scalability to multiple machines, but not vertically scalable due to the direct interaction with Docker CLI	Allow for distributed software (e.g, routing), but not flexible to modify images, topology, etc
OpenSN	Efficient due to our improvement on the process of using Docker CLI in our managers	Good vertical and horizontal scalability due to our improvement on the process of using Docker CLI in SN emulation	Allow for distributed software (e.g, routing), and flexible to extend (due to separation of user configuration from container network management)

adopts StarPerf's solution for constellation construction and dynamic route calculation. The authors leverage LeoEM to evaluate their proposed congestion control mechanisms.

Container-based: Mukerjee *et al.* in [24] present the design of Etalon (available at [8]), i.e., an open source reconfigurable datacenter network emulator developed upon Docker container. The authors establish a testbed on four servers to emulate a datacenter network, but Etalon does not support for SN emulation. Lai *et al.* in [16] develop StarryNet, i.e., a container-based emulator for SN. It is also implemented upon Docker, and emulates the distributed routing process by loading BIRD [3] on each node. Moreover, StarryNet incorporates some real orbital data, and is partially available at [31]. Pan *et al.* in [27] introduce a similar container-based SN emulator, using another routing software Quagga [28].

OpenSN is also a container-based emulator for SN, which exhibits advantages in emulation efficiency, system scalability, and function extensibility. We summarize the comparison in Table 2, and introduce the detailed design in Section 3.

3 OPENSN FRAMEWORK

This section presents the design of OpenSN. Specifically, we provide an overview in Section 3.1. We then introduce our improvements on emulation efficiency, system scalability, and function extensibility in Sections 3.2-3.4, respectively.

3.1 Overview and Architecture

OpenSN is a virtual-network emulator for LEO satellite networks, which leverages containers and virtual links to emulate SN. Figure 1 shows the architecture of OpenSN, including three key components: User-defined Configurator, Container Network Manager, and Key-value Database.

(1) User-defined Configurator allows user to specify the emulation parameters (e.g., constellation, connectivity, nodes, etc) and the emulation rules (e.g., ISL failure model, GSL handover policy, trajectory udpate, etc) in OpenSN. To enhance extensibility, the corresponding configuration data will not be directly pass to Container Network Manager, but is recorded in Key-value Database.

(2) Key-value Database records the necessary information of SN emulation. Some are user-specified emulation parameters and rules from User-defined Configurator. Some are the status data of SN emulation obtained from Container Network Manager in each emulation environment.

(3) Container Network Manager controls the major functionality of SN emulation, which is conducted on multiple machines connected by VXLAN. According to the information from Key-value Database, Container Runtime Manager is responsible for container creation/destruction and single-node resource management. Virtual Link Manager will construct the virtual links for each ISL and GSL in SN. Moreover, Message Forwarder is used to transfer necessary information (e.g., real-time topology, IP address of each interface, and user's command) between Key-value Database and Multi-Machine Emulation Environment.

3.2 Efficient Network Emulation

One of the major challenges and performance bottlenecks of emulating SN lies in the network emulation. On the one hand, the

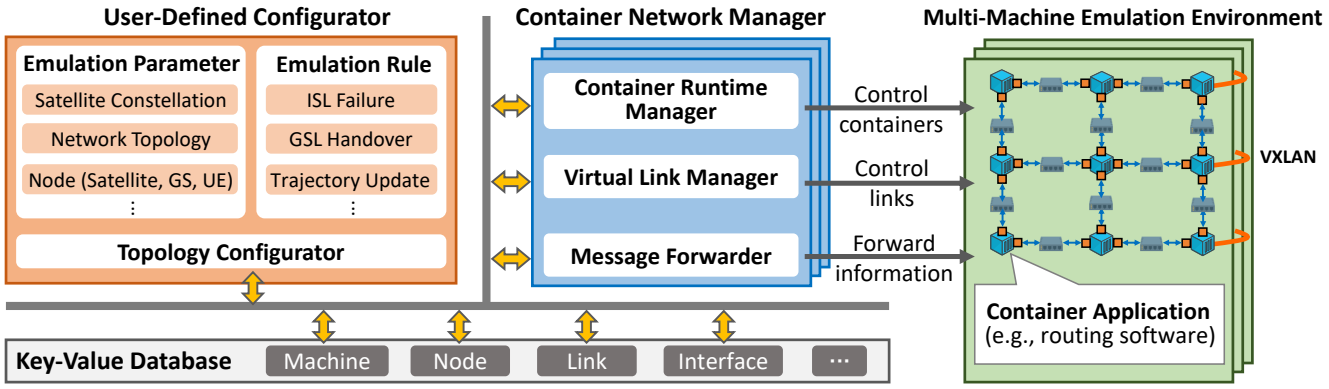


Figure 1: System architecture of OpenSN.

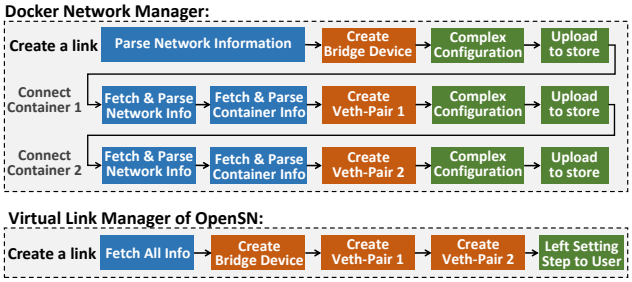


Figure 2: Progress of creating link under Docker Network Manager and Virtual Link Manager of OpenSN

number of involved nodes, ISLs and GSLs is huge for LEO mega-constellations. On the other hand, the link state changes (e.g., ISL failure/recover and GSL handover) are quite frequent due to the mobility nature of LEO satellites. The two aspects above require an efficient network manager. Existing container-based SN emulators (e.g., StarryNet) rely on Docker Network Manager, and have two drawbacks. First, Docker network controller is not developed for emulating large-scale networks with frequently changing states. The concurrent network state changes may block the task queue, thus reduce the response speed of network emulation. Second, Docker network manager exhibits some unnecessary processes in container and link management (e.g., repeatedly interacting with data store). This will become a significant overhead when the constellation scales up. Based on the above observations, OpenSN's Container Network Manager tends to improve the emulation efficiency via Virtual Link Manager and Container Runtime Manager.

Virtual Link Manager of OpenSN is developed to directly manage the Linux virtual network devices and the network namespace. This manager has the highest priority. It will immediately create a coroutine to implement rapid response once the command of link-state change is detected. Furthermore, OpenSN's Virtual Link Manager streamlines the progress of creating a virtual link. Figure 2 shows the event sequence of creating a virtual link under Docker Network Manager and OpenSN's Virtual Link Manager. Note that our solution could establish a virtual link within a single call. However, Docker Network Manager establishes a virtual link via three actions, thus increases the operation overhead.

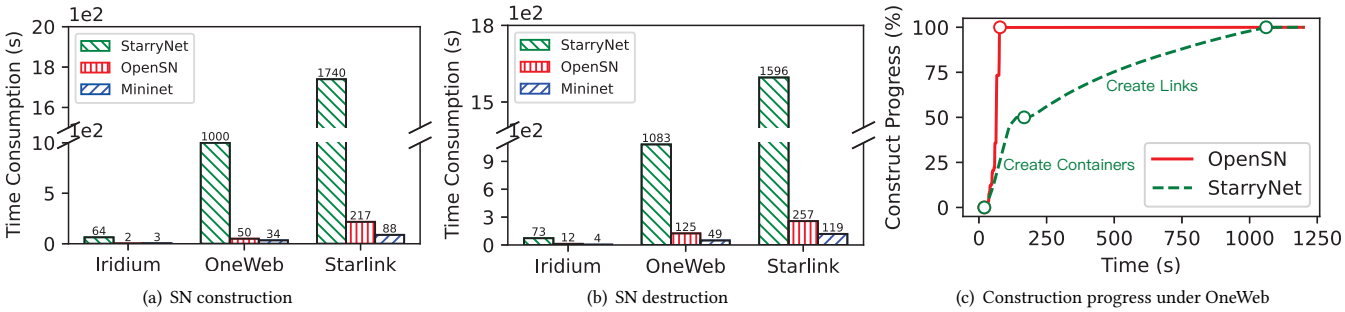
Container Runtime Manager of OpenSN leverages the official Docker SDK to connect with Docker daemon, instead of relying on Docker Command Line Interface (CLI) like StarryNet. This enables OpenSN to skip many unnecessary processes (e.g., starting sh and Docker-client) when creating containers or executing command inside containers, thus significantly reduces the operation overhead. Furthermore, the Container Runtime Managers of OpenSN are implemented on each machine in a distributed manner. Consequently, the efficiency of container management will be improved under a parallel architecture.

3.3 Multi-Machine Extension

The virtual-network emulation for LEO mega-constellation requires a lot of computing resources. Although OpenSN has made significant improvements for network emulation in Section 3.2, the single-machine deployment pattern still cannot afford to emulate satellites that run resource-consuming applications. Hence OpenSN provides the capability of multi-machine extension. Next we introduce how the control plane and data plane work.

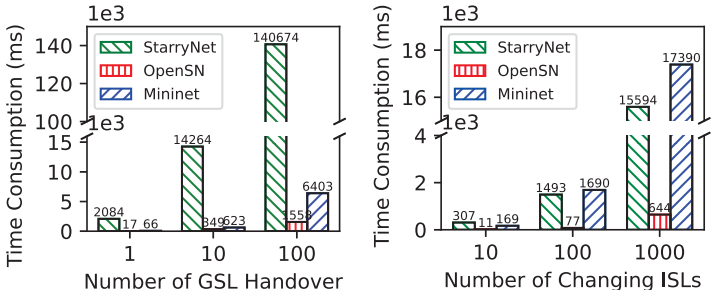
Control Plane: To achieve multi-machine extension, each machine in the emulation cluster should deploy a Container Network Manager as the control plane. Moreover, these managers need to share their machine data (e.g., ingress address) and monitor the entity data change to create/destroy containers or links. OpenSN leverages etcd [9] to achieve data exchange on the control plane, which is the primary data exchange component of container management tools. Specifically, etcd enables client to watch the status data (e.g., link delay and container status) at a low cost and with high flexibility. It enables OpenSN daemon in multiple machines to respond as soon as a state-updating command is given.

Data Plane: To achieve multi-machine extension, the data plane of OpenSN should be able to emulate the ISLs or GSLs when the corresponding containers are deployed on different emulation machines. OpenSN leverages VXLAN technology [22] to achieve cross-machine data transfer. This is because most links of SN are point-to-point, and VXLAN naturally supports such a network structure and can flexibly connect containers cross machines.



(a) SN construction (b) SN destruction (c) Construction progress under OneWeb

Figure 3: Time consumption of creating/deconstructing SN under OpenSN, StarryNet, and Mininet.



(a) GSL connection update (b) ISL latency update

Figure 4: Efficiency of link state update under OneWeb

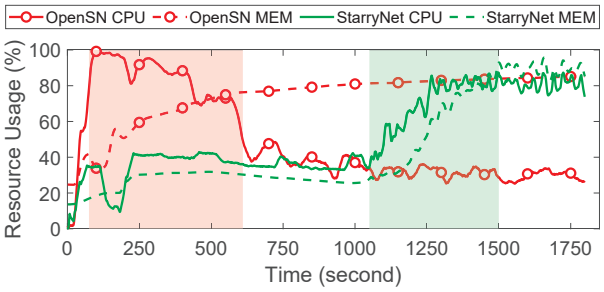


Figure 5: Resource usage of emulating OneWeb across three periods: network construction, routing convergence (*shaded*), and stable operation.

3.4 Enhanced Extensibility

To facilitate systematic performance evaluation of new studies, the emulator of SN should exhibit good extensibility in terms of emulation functions. OpenSN achieves this goal via a separation architecture shown in Figure 1. Specifically, OpenSN separates the user-defined configuration from the container network management via a key-value database. Such a key-value database records the necessary information of SN emulation, and acts as a data forwarder between User-Defined Configurator and Container Network Manager. This way, users can specify and change their emulation configuration without being affected by the implementation details of the container network.

4 PERFORMANCE EVALUATION

This section evaluates the performance of OpenSN in terms of the emulation efficiency and system scalability. Moreover, we will also compare to existing open-source SN emulators, i.e., StarryNet and LeoEM. Recall that LeoEM is developed by adding trajectory calculation and route calculation to Mininet, but does not support distributed routing software. Hence the performance of LeoEM is similar to Mininet. Moreover, the absence of distributed routing software makes LeoEM and Mininet more efficient than StarryNet and OpenSN in certain aspects.

4.1 SN Construction/Destruction

We first evaluate the efficiency of network construction under OpenSN, StarryNet, and LeoEM (or Minimet).

Figure 3(a) and Figure 3(b) plot the time consumption of constructing and deconstructing SN, respectively. Each sub-figure contains the results of three constellations, i.e., Iridium (6×11), OneWeb (18×40), and Starlink (72×22). Note that LeoEM (or Mininet) achieves the best performance, since it does not run distributed routing software. OpenSN slightly increases the construction/destruction time compared to LeoEM (Mininet). Compared to the container-based StarryNet, OpenSN significantly reduces the network construction/destruction time (e.g., 5X-10X reduction).

Figure 3(c) shows the progress of constructing OneWeb under OpenSN and StarryNet. The markers on each curve represent some critical time slots. At the beginning of SN construction (e.g., 0-100s), StarryNet has a similar speed as OpenSN, since StarryNet uses Docker Swarm to create containers in a parallel manner. However, the construction speed of StarryNet significantly slows down during the link creation period. The reasons are two-fold. First, StarryNet uses Docker Network Controller to create links, which involves more actions. Second, StarryNet adopts asynchronous interactions with Docker CLI, which generates many Docker client processes in the operation system. These processes take up a lot pf computing resource and slow down the link creation speed. The above issues have been addressed by the Virtual Link Manager in OpeSN.

4.2 Link State Update

Next we evaluate the efficiency of executing operations of link state update under OpenSN, StarryNet, and LeoEM. Figure 4(a) shows the time consumption of configuring GSL handovers. Note that OpenSN significantly outperforms StarryNet. Moreover, OpenSN is 2× faster than Mininet for configuring 10 GSL handovers and 4× faster for configuring 100 GSL handovers. This is because OpenSN uses concurrency technology when creating and destroying virtual links. Figure 4(b) shows the time consumption of configuring ISL latency update. Overall, OpenSN is more efficient than StarryNet and Mininet. This is because OpenSN adopts concurrency execution and direct Netlink interaction. Both StarryNet and Mininet rely on command-line operations for updating link latency, which is less efficient than our adopted Netlink communication method in OpenSN.

4.3 Runtime Resource Cost

Figure 5 shows real-time resource usage of emulating OneWeb under OpenSN and StarryNet. There are three periods, i.e., network construction, routing convergence, and stable operation. We have the following observations.

Network Construction (*before the shaded area*): OpenSN makes full use of resources to speed up network construction. In contrast, the resource usage level of StarryNet is low in this period, thus it takes a longer time to construct the network.

Routing Convergence (*within the shaded area*): Both OpenSN and StarryNet will take up more memory due to the growth of routing table. During the routing convergence periods, OpenSN's CPU usage is gradually decreasing, while StarryNet's CPU usage is increasing. This is because StarryNet takes the ping command as the daemon process of each container, which will frequently trigger soft-interruptions.

Stable Operation (*after the shaded area*): OpenSN incurs a much lower CPU usage than StarryNet. This is because OpenSN develops a lightweight daemon application for containers, which enters idle state after routing convergence. This advantage allows OpenSN to maintain a more stable emulation environment for large-scale LEO constellations.

5 CONCLUSION

This paper presents OpenSN, i.e., an open-source library for emulating large-scale LEO satellite networks. OpenSN adopts container-based virtualization, thus allows for running distributed software (e.g., routing) on each node, and achieves a better horizontal scalability than Mininet-based SN emulator LeoEM. Compared to other container-based SN emulators (e.g., StarryNet), OpenSN streamlines the interaction with the Docker command line interface and significantly reduces unnecessary operation of creating virtual links. These modifications improve emulation efficiency and vertical scalability on a single machine. Furthermore, OpenSN separates user-defined configuration from container network management via a key-value database that records the necessary information of SN emulation. Such a separation architecture enhances the function extensibility. Experimental results validate the performance.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their useful comments that have helped improve this article. Moreover, the authors would like to thank the students (e.g., Kai Shen, Sichao Chen, Huanyu Meng, Yuxuan Lu, Yifei Chen, and Pengfei Gao) for helping developing OpenSN.

REFERENCES

[1] Debopam Bhattacherjee, Waqar Aqeel, Ilker Nadi Bozkurt, Anthony Aguirre, Balakrishnan Chandrasekaran, P Brighten Godfrey, Gregory Laughlin, Bruce Maggs, and Ankit Singla. 2018. Gearing up for the 21st century space race. In *Proceedings of ACM Workshop on Hot Topics in Networks (HotNets)*. 113–119.
[2] Debopam Bhattacherjee and Ankit Singla. 2019. Network topology design at 27,000 km/hour. In *Proceedings of International Conference on Emerging Networking Experiments And Technologies (CoNEXT)*. 341–354.
[3] BIRD: Internet Routing Daemon. [n. d.]. https://bird.network.cz.
[4] Xuyang Cao and Xinyu Zhang. 2023. SaTCP: Link-Layer Informed TCP Adaptation for Highly Dynamic LEO Satellite Networks. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*.
[5] Nan Cheng, Wei Quan, Weisen Shi, Huaqing Wu, Qiang Ye, Haibo Zhou, Weihua Zhuang, Xuemin Shen, and Bo Bai. 2020. A comprehensive simulation platform for space-air-ground integrated network. *IEEE Wireless Communications* 27, 1 (2020), 178–185.
[6] Docker. [n. d.]. https://www.docker.com.
[7] Eylem Ekici, Ian F Akyildiz, and Michael D Bender. 2001. Network layer integration of terrestrial and satellite IP networks over BGP-S. In *Proceedings of IEEE Global Telecommunications Conference (GLOBECOM)*, Vol. 4. IEEE, 2698–2702.
[8] Etalon: A reconfigurable datacenter network emulator. [n. d.]. https://github.com/mukerjee/etalon.
[9] etcd. [n. d.]. https://etcd.io.
[10] Giacomo Giuliari, Tobias Klenze, Markus Legner, David Basin, Adrian Perrig, and Ankit Singla. 2020. Internet backbones in space. *ACM SIGCOMM Computer Communication Review* 50, 1 (2020), 25–37.
[11] Mark Handley. 2018. Delay is not an option: Low latency routing in space. In *Proceedings of ACM Workshop on Hot Topics in Networks (HotNets)*. 85–91.
[12] Mark Handley. 2019. Using ground relays for low-latency wide-area routing in megaconstellations. In *Proceedings of ACM Workshop on Hot Topics in Networks (HotNets)*. 125–132.
[13] Iridium Next. [n. d.]. https://www.iridium.com/network/.
[14] Simon Kassing, Debopam Bhattacherjee, André Baptista Águas, Jens Eirik Saethre, and Ankit Singla. 2020. Exploring the Internet from space with Hypatia. In *Proceedings of ACM Internet Measurement Conference (IMC)*.
[15] Tobias Klenze, Giacomo Giuliari, Christos Pappas, Adrian Perrig, and David Basin. 2018. Networking in Heaven as on Earth. In *Proceedings of ACM Workshop on Hot Topics in Networks (HotNets)*. 22–28.
[16] Zeqi Lai, Hewu Li, Yangtao Deng, Qian Wu, Jun Liu, Yuanjie Li, Jihao Li, Lixin Liu, Weisen Liu, and Jianping Wu. 2023. StarryNet: Empowering Researchers to Evaluate Futuristic Integrated Space and Terrestrial Networks. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.
[17] Zeqi Lai, Hewu Li, and Jihao Li. 2020. Starperf: Characterizing network performance for emerging mega-constellations. In *Proceedings of IEEE International Conference on Network Protocols (ICNP)*.
[18] Zeqi Lai, Hewu Li, Yikun Wang, Qian Wu, Yangtao Deng, Jun Liu, Yuanjie Li, and Jianping Wu. 2023. Achieving Resilient and Performance-Guaranteed Routing in Space-Terrestrial Integrated Networks. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*.
[19] LeoEM: a Real-Time LEO Satellite Network Emulator. [n. d.]. https://github.com/XuyangCaoUCSD/LeoEM.
[20] Yuanjie Li, Hewu Li, Lixin Liu, Wei Liu, Jiayi Liu, Jianping Wu, Qian Wu, Jun Liu, and Zeqi Lai. 2021. "Internet in Space" for Terrestrial Users via Cyber-Physical Convergence. In *Proceedings of ACM Workshop on Hot Topics in Networks (HotNets)*. 163–170.
[21] Teng Liang, Zhongda Xia, Guoming Tang, Yu Zhang, and Beichuan Zhang. 2021. NDN in large LEO satellite constellations: a case of consumer mobility support. In *Proceedings of ACM Conference on Information-Centric Networking (ICN)*.
[22] Mallik Mahalingam, Dinesh Dutt, Kenneth Duda, Puneet Agarwal, Lawrence Kreeger, T Sridhar, Mike Bursell, and Chris Wright. 2014. *Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks*. Technical Report.
[23] Mininet. [n. d.]. https://mininet.org.
[24] Matthew K Mukerjee, Christopher Canel, Weiyang Wang, Daehyeok Kim, Srinivasan Seshan, and Alex C Snoeren. 2020. Adapting TCP for reconfigurable datacenter networks. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[25] ns-3. [n. d.]. https://www.nsnam.org.
[26] OMNeT++. [n. d.]. https://omnetpp.org.
[27] Tian Pan, Xingchen Li, Wenhao Xue, Zizhen Bian, Tao Huang, and Yunjie Liu. 2022. A Docker-based LEO Satellite Network Testbed. *Chinese Journal of Computers* 45, 9 (2022), 2029–2046.
[28] Quagga. [n. d.]. https://www.nongnu.org/quagga.
[29] Qian Shan, Zhiyuan Wang, Shan Zhang, Qingkai Meng, and Hongbin Luo. 2023. Routing in LEO Satellite Networks: How Many Link-State Updates Do We Need?. In *Proceedings of IEEE International Conference on Satellite Computing (Satellite).*
[30] Starlink Internet With Space Lasers. [n. d.]. https://cordcuttersnews.com/spacex-ramps-up-starlink-internet-speeds-with-thousands-of-space-lasers/.
[31] StarryNet for the emulation of satellite Internet constellations. [n. d.]. https://github.com/SpaceNetLab/StarryNet.
[32] Systems Tool Kit (STK). [n. d.]. https://www.agi.com/products/stk.
[33] Fei Yan, Hongbin Luo, Shan Zhang, Zhiyuan Wang, and Peng Lian. 2023. A comparative study of IP-based and ICN-based link-state routing protocols in LEO satellite networks. *Peer-to-Peer Networking and Applications* 16, 6 (2023), 3032–3046.
[34] Fei Yan, Zhiyuan Wang, Shan Zhang, Qingkai Meng, and Hongbin Luo. 2024. Logic Path Identified Hierarchical Routing for Large-Scale LEO Satellite Networks. *IEEE Transactions on Network Science and Engineering* (2024).
[35] Hefan Zhang, Zhiyuan Wang, Shan Zhang, Qingkai Meng, and Hongbin Luo. 2023. Optimizing Link-Identified Forwarding Framework in LEO Satellite Networks. In *Proceedings of International Symposium on Modeling and Optimization in Mobile, Ad hoc, and Wireless Networks (WiOpt).*
[36] Yaoying Zhang, Qian Wu, Zeqi Lai, and Hewu Li. 2022. Enabling low-latency-capable satellite-ground topology for emerging LEO satellite networks. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM).* 1329–1338.