Space Network Emulation System Based on a User-Space Network Stack

LEI Jianzhe[1], ZHAO Kanglian[1], HOU Dongxu[2], ZHOU Fenlin[2]
(1. Nanjing University, Nanjing 210023, China;
2. ZTE Corporation, Shenzhen 518057, China)

DOI: 10.12142/ZTECOM.202502003
https://kns.cnki.net/kcms/detail/34.1294.TN.20250514.1835.002.html, published online May 15, 2025
Manuscript received: 2025-02-22

Abstract: This paper presents a space network emulation system based on a user-space network stack named Nos to solve space networks' unique architecture and routing issues and kernel stacks' inefficiency and development complexity. Our low Earth orbit satellite scenario emulation verifies the dynamic routing function of the protocol stack. The proposed system uses technologies like Open vSwitch (OVS) and traffic control (TC) to emulate the space network's highly dynamic topology and time-varying link characteristics. The emulation results demonstrate the system's high reliability, and the user-space network stack reduces development complexity and debugging difficulty, providing convenience for the development of space network protocols and network functions.

Keywords: network emulation; space network; user-space network stack; network function virtualization

Citation (Format 1): LEI J Z, ZHAO K L, HOU D X, et al. Space network emulation system based on a user-space network stack [J]. ZTE Communications, 2025, 23(2): 11 - 19. DOI: 10.12142/ZTECOM.202502003
Citation (Format 2): J. Z. Lei, K. L. Zhao, D. X. Hou, et al., "Space network emulation system based on a user-space network stack," *ZTE Communications*, vol. 23, no. 2, pp. 11 - 19, Jun. 2025. doi: 10.12142/ZTECOM.202502003.

1 Introduction

1.1 Development of Space Network and Its Emulation Methods

In recent years, with the continuous iteration of communication technologies and the growing demands for information perception, satellite telemetry, and global network integration, the traditional terrestrial Internet based on the Open Systems Interconnection (OSI) protocol stack can no longer meet the increasingly diverse and expanding network service needs. As a new network service model, the space Internet offers a broader service coverage while ensuring transmission bandwidth. It effectively overcomes challenges such as user access limitations due to terrestrial factors.

Satellite networks have become indispensable in various fields, including military security, aerospace, civilian networks, and remote sensing exploration. However, a series of emerging and evolving network algorithms, protocol systems, and network management models have also surfaced alongside its rapid development. Implementing a new technology, from theoretical development to practical deployment, requires a series of complex validation processes, such as performance evaluations and network throughput tests. As a communication network deployed in unique environments, the satellite network particularly requires systematic network emulation methods and verification platforms to support technological validation. Network emulation methods are generally categorized into four types: network theoretical model construction, physical platform setup, network simulation, and network emulation[1].

- Network theoretical model construction: This method involves network research through modeling, theoretical analysis, and algorithm design. It provides a theoretical foundation for the design and implementation of network technology.
- Physical platform setup: This approach aims to replicate the network scenario to the greatest extent, offering high authenticity. However, it is challenging to deploy, needs more scalability and reconfigurability, and has high hardware requirements for network equipment, limiting its use for large-scale deployments.
- Network simulation: This software-based method simulates existing network scenarios, protocols, and services, offering relatively simple, cost-effective and easily extendable experimental environment. However, it does not support real traffic loads transmission, leading to less accurate results.
- Network emulation: Combining the advantages of physical platforms and network simulation, network emulation supports

This work was supported by the National Natural Science Foundation of China under Grant No. 62131012 and ZTE Industry-University-Institute Cooperation Funds under Grant No. IA20230712005.

real protocols and data flow transmission. It thus offers high fidelity, flexibility, easy deployability, and scalability.

A network emulation platform is generally constructed through virtualization technology. Virtualization reallocates and isolates physical hardware resources on real physical devices, abstracting resources from computer hardware to the network operating system and network applications, to create an emulation environment. Traditional virtualization technology builds virtual machine managers on the host system, where each virtual unit requires its own operating system. In large-scale satellite network scenarios, where the network applications between nodes are similar, traditional virtualization solutions result in considerable resource redundancy, leading to inefficiency. Therefore, container-based network solutions have been proposed. Based on the host server's operating system, containerized networking implements process-level virtualization, which minimizes emulation node overhead and maximizes the use of the host server's physical resources.

Given the unique structure of the space network, it differs significantly from traditional terrestrial networks in terms of transmission conditions, node deployment, and information compatibility. For instance, network signals are significantly impacted by factors such as cosmic electromagnetic interference and terrestrial atmospheric activity during transmission. This results in high bit error rates or temporary link interruptions. Additionally, due to the large distances between satellite nodes, network signal transmission experiences high latency and time jitter. The high-speed movement of satellites further leads to highly dynamic network topologies, causing periodic changes in link relationships between nodes[2]. These factors restrict satellite network service to some extent. When constructing an emulation system for space network, these characteristics must be considered and incorporated into the design to best replicate the space network environment.

Existing studies have led to the design and implementation of several mature and stable network emulation systems, including NS3, OMNeT++[3], STK[4], and EmuStack[5]. While these tools provide valuable insights into space network behavior, they have notable limitations:

• Limited real-time protocol testing: Many tools focus on theoretical simulations, which limits their ability to validate real-world protocol implementations.

• Inefficiency in handling dynamic topologies: The frequent changes in space network topologies, such as those seen in low Earth orbit (LEO), are not well supported by traditional simulation platforms.

• High computational overhead: Some platforms require significant computational resources, making them less scalable for large-scale emulations.

• Dependence on kernel-based network stacks: These systems often rely on kernel-level networking, leading to inefficiencies due to context-switching and limited real-time performance.

1.2 User-Space Network Stack

The network interface subsystem, as the most complex module in the Linux operating system kernel, has undergone decades of development and evolution, achieving a high level of reliability and stability. However, while the kernel network stack is widely used, it has also faced criticism for its high debugging and development costs, as well as its relatively low packet forwarding speeds[6]. To improve the performance and scalability of the network stack, developers have been looking for ways to abandon the kernel network stack solution and migrate the entire functionality of the network stack to user space. With the continuous development and iteration of high-performance network I/O technologies such as Data Plane Development Kit (DPDK) and Netmap, the user-space network stack can bypass the operating system kernel, thereby directly delivering the received packets from the network interface card to the user space. This avoids the significant overhead caused by frequent context switching, memory copying, and other factors, thus improving the performance of the network stack[7]. Moreover, for network development personnel, a network stack located in user space is more straightforward to debug and maintain, which is beneficial for the development of space network technologies that require extensive validation work. Therefore, the kernel network stack is not well-suited for real-world space network environments.

Building upon existing user-space network stacks (e. g., mTCP[8], IX[9], and Arrakis[10]), this paper introduces a non-open-source, high-performance commercial solution specifically designed for next-generation space network routing technologies. Unlike other user-space network stacks, Nos not only offers exceptional data processing efficiency but also demonstrates excellent topological adaptability. Furthermore, it can be integrated with Docker container technology to operate in lightweight virtual environments. Designing and implementing an emulation system based on Nos allows for more effective debugging and development, thereby providing enhanced space network routing services.

This paper proposes an emulation system based on the user-space network stack Nos. The system overcomes most of the limitations by providing high-performance data processing, better topology adaptability, and scalability in lightweight virtual environments. This approach offers a more efficient platform for validating space network protocols.

2 Design of Space Network Emulation System

2.1 Design of General Emulation System

2.1.1 Node Emulation Solution

As a virtual system, the space network emulation system is built on the virtualization and reallocation of emulation server hardware resources. These resources are then abstracted into independent emulation units. Among these components, the

emulation node serves as the core element of the scenario. In emulation experiments, Docker container technology is commonly used to abstract hardware resources and manage the emulation nodes in a unified manner.

Docker is an open-source application container engine that provides a unified runtime environment for applications. It packages applications and their runtime environments into lightweight, portable container images, which can be deployed on any Linux machine. Meanwhile, Docker containers share the host system's operating system and hardware resources, managed by the Docker engine. This allows for fast startup and execution speeds, as well as high hardware resource utilization, making Docker container technology ideal for the unified orchestration of emulation nodes. It offers potent portability, quick startup, and high resource utilization[11].

The underlying principle of Docker networking is Linux "namespaces", a core mechanism enabling container networking. Namespaces can isolate various resources of a container, such as process IDs (PIDs), filesystem mount points, hostnames, and other system resources. The network namespace, in particular, logically provides independent network functionalities for different containers, including network devices, routing tables, Address Resolution Protocol (ARP) tables, iptables, firewalls, and sockets. Additionally, virtual devices such as veth, a virtual Ethernet device pair, can be used to interconnect containers. Emulation nodes can support different network protocols by deploying and running the corresponding network applications in Docker containers.

2.1.2 Link Emulation Solution

Connection between emulation containers is established through Linux's veth and Open vSwitch (OVS). Specifically, a veth network interface is created between the container and the OVS bridge, with OVS managing the link connectivity between nodes.

The emulation system provides an interface to control the link status. Users can upload a configuration file that stores the link connectivity information, and the main control program will import the relevant data into the MySQL database. Once the emulation experiment starts, the main control program continuously polls the database and, at time points where link events such as link up or down occur, calls the OVS processing function. It adds or deletes the corresponding flow entries in the bridge to represent the occurrence of the link event.

At the start of the emulation experiment, custom network applications run in the containers, while a set of threads are submitted by the main control service. When link characteristics such as delay, packet loss rate, and bandwidth change, these threads read the corresponding link configurations from the database and forward them to the network application in the container. The application then configures the appropriate traffic control (TC) queuing discipline for the container's veth interface to represent the occurrence of this particular link event. Thus, in the emulation experiment, dynamic topology and link characteristics control are abstracted as adding or deleting specific network flow entries in the OVS bridge and configuring TC queuing discipline in the container's virtual network interfaces.

The network applications running in the container can either be custom network programs that perform specific network configuration functions or open-source network programs. For example, after configuring the network topology in the main control program, a Quagga process can be run in the container to calculate the routing rules for the emulation scenario dynamically.

2.1.3 Emulation Architecture Design

The emulation system architecture, as shown in Fig. 1, is designed and implemented. The system can be abstracted from three dimensions: service call, emulation logic, and emulation scenarios.

Service call refers to how developers call the functions of the emulation system. At the engineering implementation level, the emulation system is built as a Maven project integrated with Spring Boot. The frontend page provides a corresponding web graphical user interface (GUI), allowing developers to invoke the system's backend through the relevant interfaces. The frontend program is deployed on an Nginx server, and its GUI provides rich functional interfaces. It also visually displays the topological relationships of the emulation scenarios, supporting complex scenarios consisting of ground stations, LEO satellites, deep space satellites, and lunar exploration probes. The backend server (Center Server) of the system performs operations such as scenario construction, link configuration, and service processing according to specific web requests. The data interaction between the frontend and backend is typically achieved through HTTP requests and responses. The frontend sends requests using JavaScript, and the backend receives and processes these requests and returns JSON data to the frontend. The backend's request processing often involves significant database access, as the database stores all experiment-related information, including experiment status, node configurations, and link details. The database and backend program are deployed on the same server, enabling local and high-speed database access operations. The backend main control program uniformly orchestrates the container nodes and builds a star-shaped topology with an OVS bridge at the center, as defined in the "Emulation Logic" module. As a virtual switch supporting the OpenFlow protocol and flow entry distribution, OVS provides support for dynamic topology control in the emulation system.

From a general perspective, the service call module serves as the interface through which the emulation system directly interacts with the user. User actions are transmitted via frontend requests to the backend, where they undergo a series of processing steps and database interactions. This process ulti-

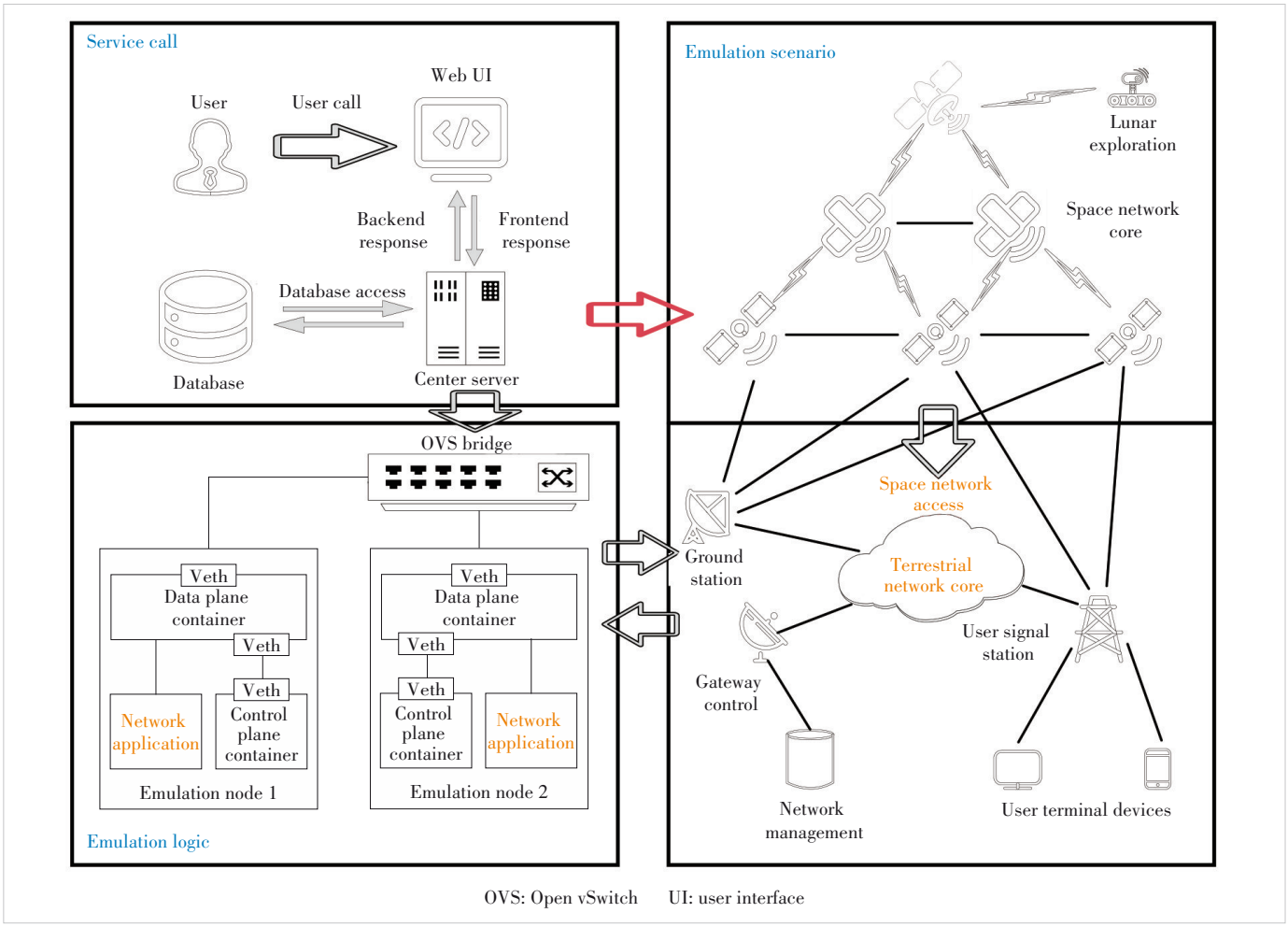

Figure 1. Emulation system architecture

mately constructs the container network as depicted in the Emulation Logic module. The elements in this container network are directly mapped to the corresponding elements in the Emulation Scenario. The Emulation Logic and Emulation Scenario represent two topological frameworks for the emulation experiment: the former reflects the actual network configuration, while the latter serves as an abstract model of the former. These three core modules effectively demonstrate the structure of a general emulation system.

In designing the general network emulation system, it is crucial to consider the platform that best supports the performance and scalability requirements of space network emulation. The system performance is influenced by multiple factors, including network topology dynamics, packet processing efficiency, resource allocation, and system scalability. These factors collectively determine the overall effectiveness of the emulation.

The system is deployed on a general-purpose x86, 64-bit server and utilizes a combination of Nos and Docker containerization to achieve high performance and flexibility. Nos enables efficient packet processing and supports dynamic topology adaptation, while Docker containers provide a lightweight, scalable environment for running emulation nodes.

The use of Docker as the platform ensures efficient resource utilization, minimizing computational overhead and allowing for the emulation of large-scale satellite constellations with high fidelity. This choice of platform addresses the limitations of traditional kernel-based approaches, such as high computational costs and reduced scalability, making it an ideal solution to emulating space networks in a real-time, dynamic environment.

2.2 Integration of Nos

The space network emulation platform described above is designed to integrate Nos. In this design, two Docker containers run in a single emulation node, as shown in Fig. 2. The network control plane functions are consolidated in the routing processor (RP) container, which is responsible for processing routing packets and dynamically calculating routing rules based on the real-time network topology. The network data plane functions are consolidated in the line processor (LP) container, which performs high-performance forwarding based on the routing information table of Nos. The two containers are connected through a veth pair and communicate with each

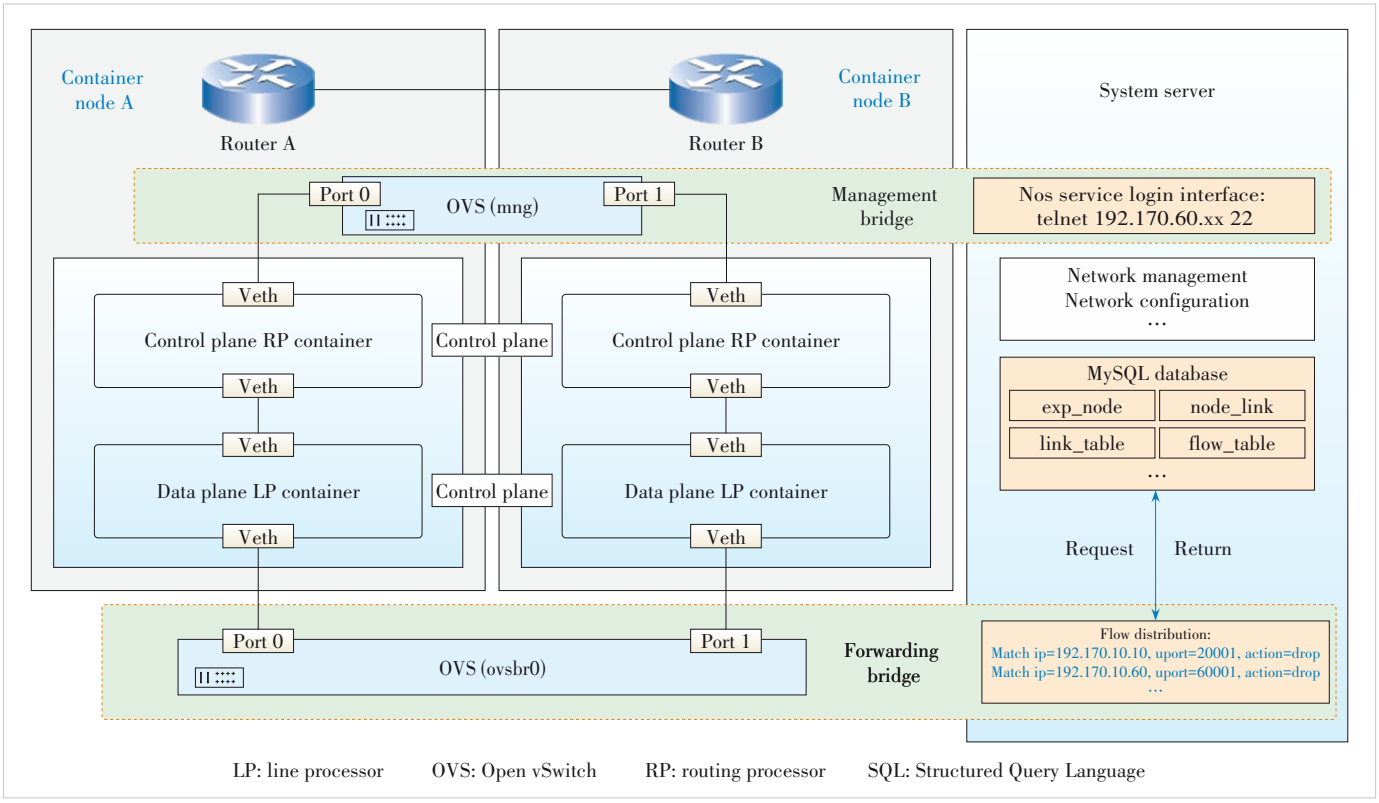

Figure 2. Emulation system architecture with integrated Nos

other directly, collectively forming an emulation node.

Each emulation node's LP container is connected to the OVS bridge "ovsbr0" via a southbound veth interface, with all emulation traffic being exchanged through the ovsbr0 bridge. The RP container of the emulation node is connected to the OVS bridge "mng" via a northbound veth interface. The mng bridge, as a management bridge, ensures communication between the main control server and the emulation node. Developers can log into the RP container's reserved port 22 via this management bridge and access the user management interface of the network stack.

2.3 Soft Forwarding Interface Configuration and Link Mapping

Unlike terrestrial networks, the space network often experiences link interruptions and handovers. For example, in a polar orbit constellation scenario, when a satellite enters the polar region, the link between satellites of adjacent orbits within the same latitude range will be temporarily interrupted and resume once the satellite exits the polar region[12]. Additionally, the satellite connected to a given ground station will change over time. In such high dynamic topologies, a unified emulation strategy is adopted. That is, all possible link resources are reserved during the scenario construction. When a link temporarily fails, the corresponding flow entries are added to the ovsbr0 bridge, matching all packets from the two end nodes of the link and discarding them, thus emulating link up/down and handover events.

However, the apparent disadvantage of this emulation strategy is that reserving resources for all possible links in advance can result in substantial waste, especially in scenarios where link handovers occur frequently, as shown in Fig. 3.

A ground station may only be connected to several satellites at any given time, while the links with all the other potential satellites are temporarily interrupted. These interrupted links,

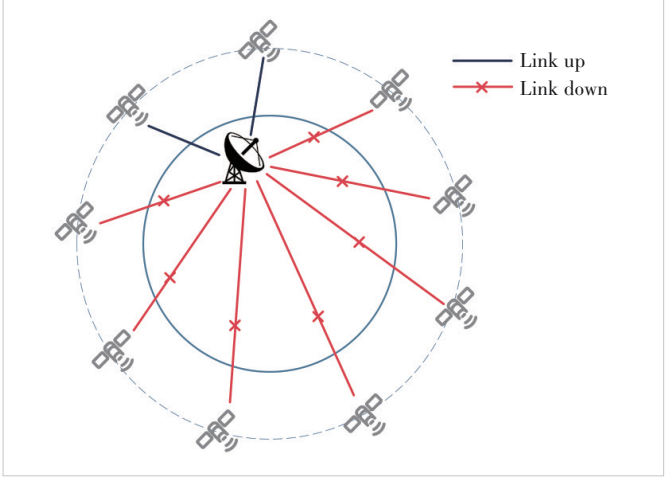

Figure 3. Link handovers occur frequently

however, could take significant system resources. In large-scale constellation scenarios, such scale of resource waste is unacceptable. Nos uses a virtual network interface called "fei" for software forwarding. This interface is implemented in software within the network stack and does not incur additional system resource overhead. All packets forwarded through the fei interface are first handed to the southbound veth interface of the LP container and then forwarded by the ovsbr0 bridge.

The LP container has only one southbound veth interface, and all emulation nodes are connected to the OVS bridge through a single veth interface. All traffic from the emulation node is transmitted through this interface. As shown in Fig. 4, in any given emulation node, the fei interface in the forwarding plane LP container is a virtual interface created in software, existing within the user-space network stack. An IP address needs to be assigned to it for end-to-end forwarding in the emulation experiment. The IP address configured on the veth interface connecting all LP containers to OVS is specified within a particular subnet (such as the subnet 192.170.10.0/24 in Fig. 4). That is, all LP container veth interfaces are in the same subnet.

To ensure that all packets passing through the veth interface are correctly matched with the software forwarding fei interface, a User Datagram Protocol (UDP) port number (uport) is introduced, and a mapping relationship from "veth IP + uport" to "fei IP" is established. Within any given emulation node, the fei IP address maps to the veth IP + uport of the LP container. For all neighboring nodes of a particular node, the link endpoints' fei IP can be mapped as a four-tuple: "IP local, uport local, IP remote, uport remote".

Before the experiment starts, the link mapping relationships between any node and its neighbors are saved in the network stack's startup configuration file "soft_forward.xml". Upon starting, the network stack reads this configuration file to establish local link mappings with all neighbors. On the emulation layer, any packet is forwarded through the fei interface of the LP container after the network stack finishes encapsulating it. However, at the implementation level, when the fei interface receives a packet, it cannot forward it directly. Instead, it must first use the local link mapping information and encapsulate an additional layer, combining the local veth IP address and local UDP port with the corresponding remote veth IP and remote UDP port. The packet is then handed to the southbound veth interface of the LP container and forwarded by the OVS bridge (Fig. 5). When an emulation node receives a packet, the same process occurs: the outer IP and UDP port numbers are decapsulated first, and then the packet is passed to the fei interface for processing.

In the emulation architecture mentioned above, the emulation of link characteristics also requires a corresponding design. First, for the OVS bridge ovsbr0, which is responsible for forwarding all traffic generated by the emulation nodes, flow entries can be added in it to match specific packets and take corresponding actions. For example, when emulating a node failure, a flow entry can be added to match the source or destination IP address of the node's southbound veth interface and drop the packet. This effectively emulates the temporary isolation of that node in the emulation scenario. Simply deleting

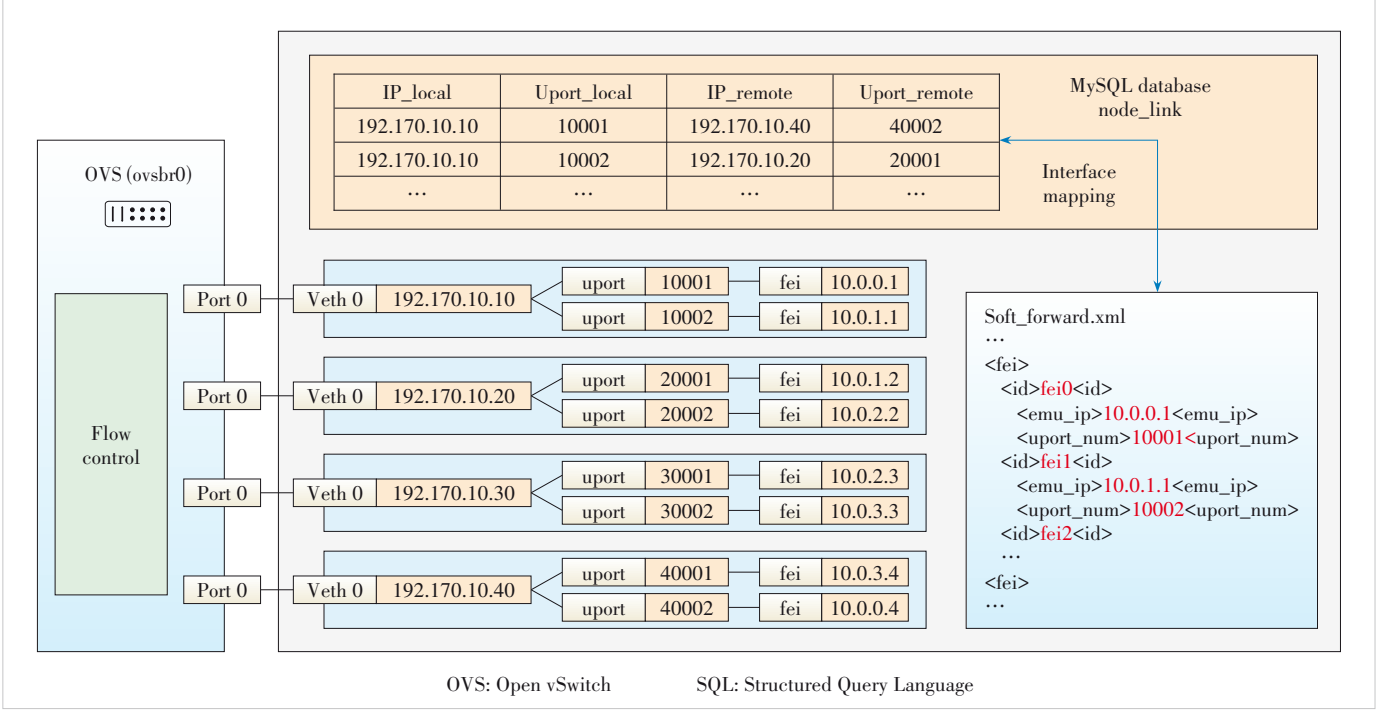

Figure 4. Virtual fei interface configuration for soft forward and its link-mapping rules

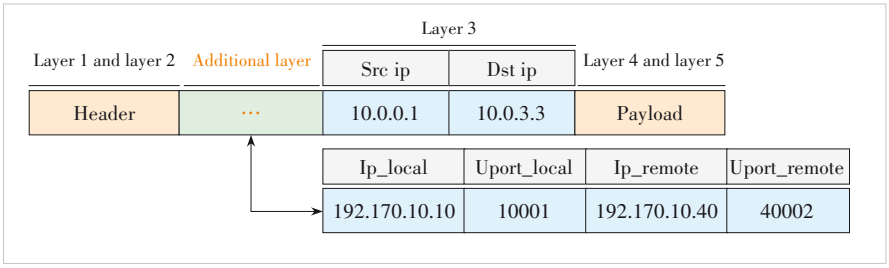

Figure 5. Additional packet encapsulation

the flow entry will suffice to restore the node's state. Similarly, when emulating the failure of a link, a flow entry can be added to match the source IP + UDP port number or the destination IP + UDP port number, and take the "drop" action.

Secondly, link characteristics such as delay and packet loss rate can be emulated by constructing a TC queue discipline tree (Fig. 6) on the southbound veth interface of the LP container. This will enable traffic flowing through the interface to be split. Packets enter the filter from the root queue, and the filter will match the destination IP address and UDP port number of the packets, directing them into different leaf classes[13]. By configuring the appropriate queue settings under the leaf classes, these link characteristics can be emulated.

3 Low Earth Orbit Constellation Emulation

A small LEO constellation scenario, as shown in Fig. 7, is constructed in the emulation system. The scenario consists of two polar satellite orbital planes, each with four satellites, along with two ground station terminals. All ten emulation nodes are created through the frontend GUI interface, and all link configurations are imported.

The satellite motion model used in this experiment is based on real-world orbital dynamics, with satellite positions and movements derived from real-world data exported via Satellite Toolkit (STK). The orbital parameters, such as satellite speed, orbital inclination, and orbital altitude, are extracted from STK's high-fidelity models, ensuring accurate representation of the satellite's behavior in LEO over time. These orbital parameters directly affect link availability and inter-satellite communication.

The link quality is computed based on various factors, including propagation delay, signal strength, and bit error rate. These factors are influenced by the relative distance between satellites, atmospheric conditions, and the satellite's position. The link quality model reflects the real-time variations caused by satellite motion, orbital perturbations, and environmental factors, ensuring that the emulation results accurately represent the dynamic nature of space networks.

After the emulation experiment starts, the RP container within Nos reads the routing protocol configuration file and calculates the routing relationships for the nodes. After a certain period, the routing converges, and all nodes obtain routing entries for all subnets in the emulation scenario.

The threads submitted by the emulation system access the link up/down information in real time and synchronize the control of the emulated links. For example, when a satellite enters the polar region, communication between adjacent-orbital sat-

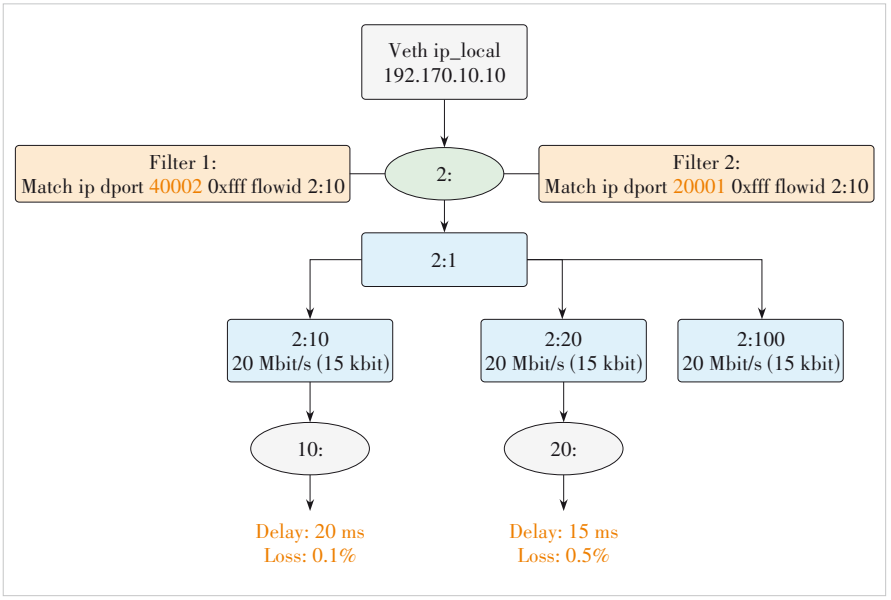

Figure 6. Traffic control (TC) queuing discipline tree

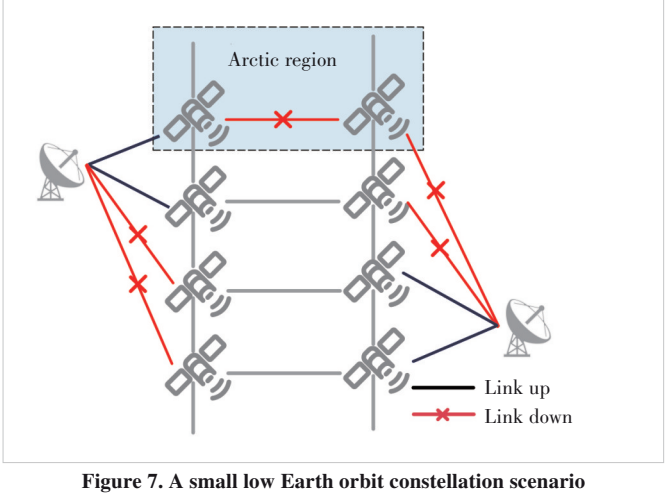

Figure 7. A small low Earth orbit constellation scenario

ellites in the same latitude zone will be temporarily interrupted until the satellite exits this region. When the control plane of Nos detects such network topology changes, it recalculates and updates the routing. The TC thread accesses the database to obtain real-time inter-satellite distances. Using these measurements, it calculates the inter-satellite propagation delay for the current time slice with a fixed duration and subsequently updates the TC queue discipline tree to propagate the delay information.

Fig. 8 shows the changes in delay and packet loss rates between two adjacent-orbital satellites in the same latitude zone. The delay exhibits a certain periodicity over time. When the two satellites move from high-latitude regions to low-latitude regions, the inter-satellite delay gradually increases; when moving from low-latitude regions to high-latitude regions, the inter-satellite delay gradually decreases. When the satellite's latitude becomes too high and it enters the polar region, the inter-satellite link is disconnected. In this case, communication between the two satellites must rely on inter-satellite links with satellites in their respective lower-latitude orbits, resulting in significantly higher delays.

The delay and packet loss rate variations between the two ground station terminals are shown in Fig. 9. At the same time, throughput and bandwidth utilization tests were conducted on a ground station terminal, and the results are shown in Fig. 10. The end-to-end delay remains generally stable, with minor fluctuations caused by satellite movement. Due to the impact of bottleneck links in the satellite network, the throughput of the ground station ranges approximately from 450 kbit/s to 700 kbit/s, with bandwidth utilization reaching over 75%. However, when a link is interrupted or involves a satellite-ground link switch, there is a certain waiting time for the routing information in Nos to converge again. During this time, the two terminals are temporarily unable to communicate.

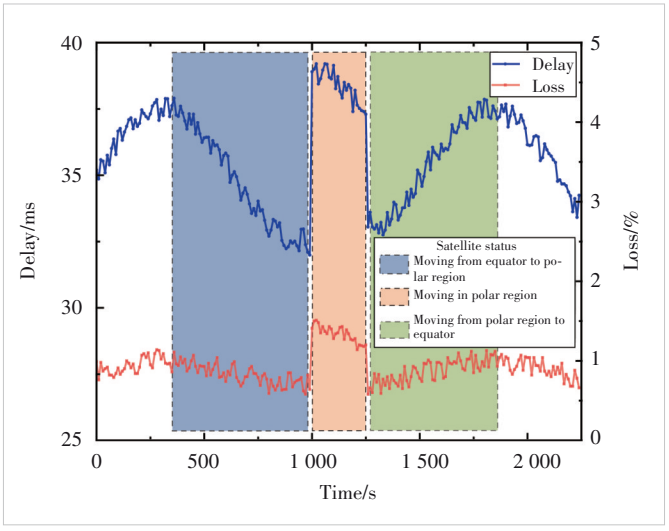

Figure 8. Delay and packet loss between two adjacent-orbital satellites in the same latitude region

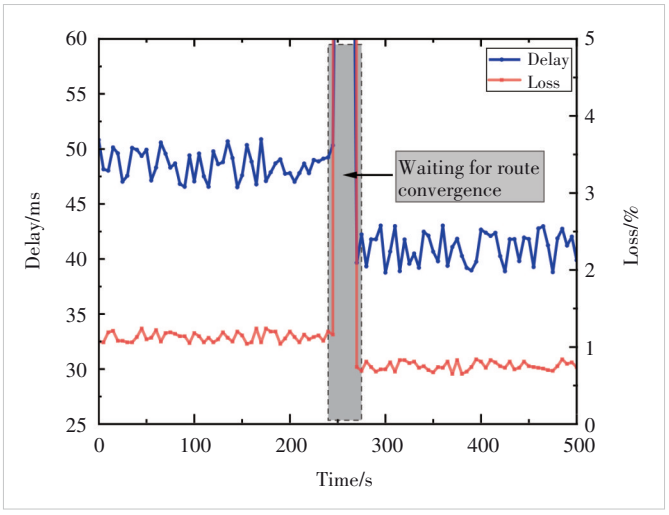

Figure 9. Delay and packet loss between two ground station terminals

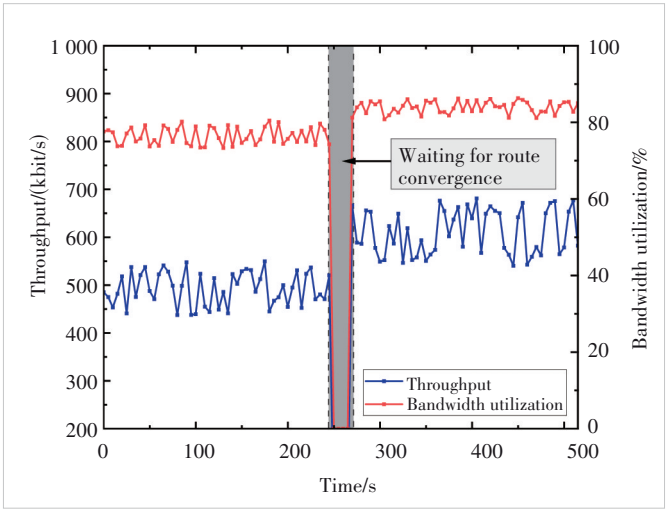

Figure 10. Throughput and bandwidth utilization of one ground station terminal

4 Conclusions

The construction of a space network emulation system is more complex than that of a ground network. We propose a space network emulation system based on Nos, a high-performance user-space network stack, in this paper. This emulation system facilitates the development and debugging of protocol systems and network functions. The separation of control and forwarding in the Nos architecture improves the overall stability of the emulation system. By constructing an LEO satellite constellation scenario, the routing and forwarding functions of Nos are validated, and the dynamic topology and time-varying link characteristics of the satellite network are realistically and reliably emulated. Nos has a rich set of functionalities. Therefore, this emulation system provides a reliable means for applying many network concepts and technologies to space communication.

There are still some areas in the space network emulation

that need to be fully considered. For example, inter-satellite or satellite-ground link delays are determined not only by distance, but also by many other factors such as atmospheric cloud cover. Moreover, the protocol model provided by Nos is primarily designed for the terrestrial network, and when emulating deep-space communication scenarios, protocols like Delay-Tolerant Network (DTN) are not supported. Future iterations and optimizations of the emulation platform should focus on enhancing system realism, stability, and network functionality completeness.

References

[1] ZHOU L W. Research on simulation and evaluation platform of large-scale satellite network based on container technology [D]. Xi'an: Xidian University, 2022. DOI: 10.27389/d.cnki.gxadu.2022.000914
[2] LI H W, WU Q, XU K, et al. Progress and tendency of space and earth integrated network [J]. Science & technology review, 2016, 34(14): 95 – 106. DOI: 10.3981/j.issn.1000-7857.2016.14.011
[3] VARGA A, HORNIG R. An overview of the OMNeT++ simulation environment [C]//Proc. 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops. ICST, 2008: 1 – 10
[4] COOK P, SCAVONE G. The synthesis ToolKit (STK) [EB/OL]. [2024-09-30]. http://hdl.handle.net/2027/spo.bbp2372.1999.366
[5] LI H F, ZHOU H C, ZHANG H K, et al. EmuStack: an openstack-based DTN network emulation platform [C]//Proc. International Conference on Networking and Network Applications (NaNA). IEEE, 2016: 387 – 392. DOI: 10.1109/NaNA.2016.24
[6] MARINOS I, WATSON R N M, HANDLEY M. Network stack specialization for performance [J]. ACM SIGCOMM computer communication review, 2015, 44(4): 175 – 186. DOI: 10.1145/2740070.2626311
[7] RIZZO L. Revisiting network I/O APIs: the netmap framework [J]. Communications of the ACM, 2012, 55(3): 45 – 51. DOI: 10.1145/2093548.2093565
[8] JEONG E Y, WOO S, JAMSHED M, et al. MTCP: a highly scalable user-level TCP stack for multicore systems [C]//11th USENIX Symposium on Networked Systems Design and Implementation. USENIX, 2014: 489 – 502
[9] BELAY A, PREKAS G, KLIMOVIC A, et al. IX: a protected dataplane operating system for high throughput and low latency [C]//11th USENIX Symposium on Networked Systems Design and Implementation. USENIX, 2014: 49 – 65
[10] PETER S, LI J, ZHANG I, et al. Arrakis: the operating system is the control plane [J]. ACM transactions on computer systems (TOCS), 2016, 33 (4): 1 – 30. DOI: 10.1145/2812806
[11] POTDAR A M, NARAYAN D G, KENGOND S, et al. Performance evaluation of docker container and virtual machine [J]. Procedia computer science, 2020, 171: 1419 – 1428. DOI: 10.1016/j.procs.2020.04.152
[12] FOSSA C E, RAINES R A, GUNSCH G H, et al. An overview of the IRIDIUM (R) low Earth orbit (LEO) satellite system [C]//Proc. IEEE 1998 National Aerospace and Electronics Conference. IEEE, 1998: 152 – 159. DOI: 10.1109/NAECON.1998.710110
[13] ALMESBERGER W. Linux network traffic control—implementation overview [EB/OL]. (1999-4-23) [2024-09-30]. https://www.almesberger.net/cv/papers/tcio8.pdf

Biographies

LEI Jianzhe is a student at the School of Electronic Science and Engineering, Nanjing University, China. His research focuses on space network emulation and network forwarding technologies.

ZHAO Kanglian (zhaokanglian@nju.edu.cn) received his PhD degree from Nanjing University, China. He is currently working at Nanjing University as a professor and doctoral supervisor. His main research interests include space intelligent information networks and 6G integrated terrestrial and space networks.

HOU Dongxu received his PhD degree from the School of Electronic Science and Engineering, Nanjing University, China in 2022. He is working at ZTE Corporation. His research interests include satellite network routing technologies and bearer networks.

ZHOU Fenlin is a Chief Engineer for Future Network Research at ZTE Corporation. His research directions include computing networks, deterministic networks, intelligent computing networks, non-terrestrial networks, and future network architectures.